Kombinatorická optimalizace
(mgr. předmět 460-4091)

Petr Jančar
katedra informatiky, FEI VŠB-TU Ostrava
www.cs.vsb.cz/jancar

13. května 2016

Poznámky k textu. Tento text vzniká v průběhu kursu v letním semestru 2015/16 modifikováním textu z běhu kursu v roce 2014/15. Bude postupně přibývat, aktuálnost verze bude zřejmá z data uvedeného výše, a také z obsahu, který bude členěn po jednotlivých týdnech v semestru. Jedná se o podpůrný pracovní text, který bude také dostupný z web-stránky předmětu

http://www.cs.vsb.cz/jancar/KOMB-OPT/komb-opt.htm.

Jako základní referenční kniha může pro nás sloužit [5]:

> Bernhard Korte, Jens Vygen: Combinatorial Optimization (Theory and Algorithms), Springer (5th edition 2012)

(viz také stránku jednoho z autorů, `http://www.or.uni-bonn.de/~vygen/co.html`, kde jsou mj. některé aktualizace/opravy).

Budeme také používat pěknou knihu menšího rozsahu [9]:

> Jon Lee: A First Course in Combinatorial Optimization, Cambridge texts in applied mathematics (2004, reprinted 2011).

Ke kombinatorické optimalizaci lze pochopitelně nalézt mnohé materiály také na webu ...

Některé (jednoduché) pojmy z kombinatoriky a teorie grafů předpokládáme. Na FEI jsou k této oblasti k dispozici pěkná skripta [8, 7, 6], přístupná např. ze

stránky Petra Kováře `http://homel.vsb.cz/~kov16/predmety.php`.

(Je tam mj. také pojednáno o minimální kostře, nejkratších cestách, tocích v sítích, maximálním párování, ... , a dalších tématech, s nimiž se zde také stručně setkáme.)

Zde také předpokládáme elementární znalosti z lineární algebry a složitosti algoritmů a výpočetních problémů (k nimž lze snadno také najít materiály na FEI a jinde; ke složitosti algoritmů a problémů např. v [4]).

Týden 1 (od 8.2.)

Hladové algoritmy a matroidy.

Připomněli jsme si nejprve hladový (greedy) algoritmus nalézající kostru maximální váhy na ohodnoceném grafu; použili jsme Exercise (Maximum-weight spanning tree) str. 58 v [9].

Na tomto problému jsme také ilustrovali, co se myslí kombinatorickou optimalizací. Připomněli jsme, že koster úplného grafu s n vrcholy je n^{n-2} (s tím, že kdyby někdo chtěl najít a předvést nějaký pěkný důkaz tohoto Cayleyho tvrzení, je vítán); náš hladový algoritmus je tak výrazně lepší než probírání všech možností ....

Pak jsme definovali jednoduchý problém z oblasti rozvrhování (scheduling); použili jsme Exercise (Scheduling) str. 59 v [9]. Zde nám nebylo hned zřejmé, zda a jaký hladový algoritmus dojde k optimálnímu řešení.

Definovali jsme pojem matroidu, jako struktury M se základní, či nosnou, množinou (ground set) $E(M)$ a množinou $\mathcal{I}(M) \subseteq 2^{E(M)}$ tzv. nezávislých množin (independent sets) splňující axiomy $I1, I2, I3$:

$I1$: $\emptyset \in \mathcal{I}(M)$;

$I2$: $X \subseteq Y, Y \in \mathcal{I}(M) \Rightarrow X \in \mathcal{I}(M)$;

$I3$: $X, Y \in \mathcal{I}(M), |X| < |Y| \Rightarrow \exists e \in Y \setminus X : X + e \in \mathcal{I}(M)$.

(Tak jako [9] píšeme $X + e$ místo $X \cup \{e\}$.) Jako příklad $E(M)$ jsme vzali množinu hran daného grafu, prvky $\mathcal{I}(M)$ jsou pak lesy, tedy množiny hran bez cyklů.

Pak jsme upravili "algoritmus pro kostru" na *hladový algoritmus pro matroid s váhou*, tedy pro $M = (E(M), \mathcal{I}(M), c)$, kde $c : E \to \mathbb{R}$ (obecně váhu, nebo též cenu, $c(e)$ uvažujeme jako reálné číslo, byť v našich příkladech se jedná o čísla celá); uvedli jsme tento algoritmus

$S_0 := \emptyset$; $k := 1$; $U := E(M)$;
while $U \neq \emptyset$ **do**
  choose $s_k \in U$ of maximum weight; $U := U - s_k$;
  if $S_{k-1} + s_k \in \mathcal{I}(M)$ **then**
    $S_k := S_{k-1} + s_k$; $k := k + 1$
  end if
end while

a dokázali jsme (pozn.: dáno jako úkol!), že tento algoritmus nalezne nezávislou množinu s největší váhou; více konkrétněji, dokázali jsme, že pro každé $k \in \{0, 1, \ldots, r_M(E(M))\}$ algoritmus sestrojí nějakou nezávislou množinu S_k, která má maximální váhu mezi nezávislými množinami s k prvky. Číslo $r_M(E(M))$ lze chápat jako "dimenzi" či "hodnost" (rank) matroidu M, tj. počet prvků největší nezávislé množiny.

Ukázali jsme si (úkol!), že tento postup *nemůžeme* aplikovat na hledání největší nezávislé množiny v grafu; systém těchto nezávislých množin nesplňuje axiom I3.

Dali jsme si několik úkolů na příště; jsou vyjmenovány níže. **Takové úkoly mohou reprezentovat typy otázek/příkladů u zápočtu a zkoušky.** (To budeme upřesňovat. Je možné, že v některých případech si student bude moci zvolit mezi "více informatickou" a "více matematickou" verzí.) Předpokládám, že tyto úkoly vždy dořešíme za aktivní spoluúčasti studentů (počítá se se zápočtovými body za příslušnou aktivitu).

1. Ukažte rozumnou implementaci hladového algoritmu pro kostru, speciálně vysvětlete způsob, jak testovat, zda přidáním hrany nevznikne cyklus. (A to i v případě, že postupujume podle obecného algoritmu, při němž vznikající množina hran může tvořit více souvislých komponent!)

2. Dokažte, že lesy na grafu (podgrafy neorientovaného grafu, které neobsahují cyklus) splňují [jakožto množiny hran] axiomy I1,I2,I3; speciálně jde o I3, tedy o fakt, že když les $\mathcal{F}_1$ má více hran než $\mathcal{F}_2$, tak lze jednu hranu z $\mathcal{F}_1$, která není v $\mathcal{F}_2$, přidat k $\mathcal{F}_2$, aniž v něm vznikne cyklus.

3. Dokažte, že uvedený hladový algoritmus pro matroid s váhou opravdu dělá to, co se v textu tvrdí.

4. Ukažte, jak lze na "jobech" v našem scheduling problému přirozeně definovat matroid, z něhož pak vyplyne optimalita hladového algoritmu (toho, který vznikne instancí obecného "matroidového algoritmu").

5. Popište rámcově implementaci hladového algoritmu pro náš "scheduling".

6. Uvažujme matici A typu $m \times n$ nad $\mathbb{R}$ (nad tělesem reálných čísel). Chápejme ji jako reprezentaci matroidu M, kde $E(M)$ je množina sloupců matice A a $\mathcal{I}(M)$ obsahuje právě množiny vzájemně nezávislých sloupců (chápaných jako vektorů v $\mathbb{R}^m$). Ověřte, že se opravdu jedná o matroid.

7. Připomeňme, že množině vrcholů grafu řekneme *nezávislá množina*, jestliže mezi žádnými jejími dvěma vrcholy není hrana. Uvažme následující algoritmus, jehož vstupem je graf $G = (V, E)$:

   $S := \emptyset$; $U := V$;
   while $U \neq \emptyset$ **do**
   choose (any) $v \in U$; $U := U - v$;
   if $S + v$ je nezávislá množina **then**
   $S := S + v$
   end if
   end while

   Ukažte, že daný algoritmus nesestrojí vždy největší nezávislou množinu v daném grafu. Systém nezávislých množin na daném grafu tedy netvoří matroid; co chybí?

Týden 2 (od 15.2.)

Sešli jsme se jen ve čtvrtek 18.2. (jak bylo avízováno v prvním týdnu). Hlavní náplní byly úkoly z minulého týdne. Některé nebyly dodělány, dokončili jsme je na přednášce 22.2. (což ale zachycuji již v zápisu zde jako "doděláno později").

1. Ukažte rozumnou implementaci hladového algoritmu pro kostru, speciálně vysvětlete způsob, jak testovat, zda přidáním hrany nevznikne cyklus. (A to i v případě, že postupujume podle obecného algoritmu, při němž vznikající množina hran může tvořit více souvislých komponent!)

   V případě budování kostry tak, že postupně vznikající les je stále souvislý (tedy je stromem), stačí u každého vrcholu udržovat informaci, zda je již napojený.

   To ovšem nestačí při aplikaci obecného hladového algoritmu (pro matroidy) budujícího lesy s maximální vahou postupně pro počet hran $k = 1, 2, \ldots, n-1$ (kde n je počet vrcholů grafu). Jako datovou strukturu vhodnou pro testování příslušnosti koncových vrcholů hrany k různým (souvislým) komponentám (dosud vybudovaného grafu) a k jejich úpravám jsme si připomněli Union-Find.

   > Popis této datové struktury najdete snadno na Internetu, i se zajímavými souvislostmi ohledně výpočetní složitosti. Zde jen stručně neformálně strukturu připomeneme. Každý vrchol grafu vybavíme ukazatelem na svého *rodiče*; nejprve každý vrchol ukáže na sebe, je svým vlastním rodičem a *reprezentantem* své komponenty. Navíc má každý reprezentant, tj. každý vrchol se "smyčkou" (odkazující sám na sebe), přiřazenu jistou *váhu* (jistou horní mez pro hloubku), na začátku 0. Když máme *spojit dvě komponenty*, ukáže reprezentant té s menší váhou na reprezentanta druhé; když jsou váhy stejné, zvolíme jednoho reprezentanta a připojíme na druhého, přičemž ten druhý zvýší váhu o jedna. Když máme k vrcholu *najít reprezentanta*, postupujeme po ukazatelích až ke smyčce; cestou projdené vrcholy si přitom pamatujeme a posléze jim zacílíme ukazatel na ten vrchol se smyčkou, čímž dochází k zlepšení struktury, tj. k zplošťování stromů tvořených ukazateli.

   Závěr je, že v principu je časová složitost příslušného hladového algoritmu stejná jako složitost setřídění hran podle jejich váhy ...

2. Dokažte, že lesy na grafu G (podgrafy neorientovaného grafu, které neobsahují cyklus) splňují [jakožto množiny hran] axiomy I1,I2,I3; speciálně jde o I3, tedy o fakt, že když les $\mathcal{F}_1$ má více hran než $\mathcal{F}_2$, tak lze jednu hranu z $\mathcal{F}_1$, která není v $\mathcal{F}_2$, přidat k $\mathcal{F}_2$, aniž vznikne cyklus.

   Na cvičení kolegové navrhli převod na lineární matroid (o němž je řeč níže), korektnost převodu nebyla ale prokázána.

   Později jsme dokončili přímý důkaz: Kdyby každá hrana z $\mathcal{F}_1$ měla oba konce v jedné komponentě grafu $G.\mathcal{F}_2$ (kde $G.\mathcal{F}$ značí podgraf grafu G se stejnou množinou vrcholů, ale s množinou hran $\mathcal{F}$), tak by každá komponenta grafu $G.\mathcal{F}_2$ musela obsahovat alespoň jednu celou komponentu grafu $G.\mathcal{F}_1$ a počet komponent $G.\mathcal{F}_1$ by byl stejný nebo větší než počet komponent $G.\mathcal{F}_2$ (připomeňme, že izolovaný vrchol je také komponentou). Ovšem počet komponent grafu $G.\mathcal{F}$ (pro les $\mathcal{F}$) je roven $|V(G)| - |\mathcal{F}|$ (podle snadné

indukce), čímž dostaneme spor (jelikož počet komponent $G.\mathcal{F}_1$ je menší než počet komponent $G.\mathcal{F}_2$).

Dokázali jsme tak, že když pro lesy $\mathcal{F}_1$, $\mathcal{F}_2$ platí $|\mathcal{F}_1| > |\mathcal{F}_2|$, tak v $\mathcal{F}_1$ existuje hrana, která spojuje dvě různé komponenty grafu $G.\mathcal{F}_2$.

3. Dokažte, že uvedený hladový algoritmus pro matroid s váhou opravdu dělá to, co se v textu tvrdí.

   Dodělali jsme později. Pomocí axiomů matroidu jsme přivedli ke sporu předpoklad, že k algoritmem sestrojené nezávislé množině $S_{k+1} = \{e_1, e_2, \ldots, e_{k+1}\}$ existuje nějaká nezávislá množina $S = \{f_1, f_2, \ldots, f_{k+1}\}$ taková, že $c(S) > c(S_{k+1})$ ....

4. Ukažte, jak lze na "jobech" v našem scheduling problému přirozeně definovat matroid, z něhož pak vyplyne optimalita hladového algoritmu (toho, který vznikne instancí obecného "matroidového algoritmu").

   Na cvičení jsme diskutovali následující algoritmus.

   Opakuj následující krok:

   > mezi joby v seznamu s nejvyšším deadlinem vyjmi job s největší váhou a ten zařaď k vykonání na konec; počátek provedení tohoto jobu určuje nový nejvyšší deadline d_{max}; deadliny těch jobů ze seznamu, které mají deadline vyšší než d_{max}, sniž na d_{max}.

   Diskutovali jsme správnost algoritmu, ale všimli si, že to *není* instance obecného algoritmu (pro konstrukci nezávislých množin s max. váhou v matroidu).

   > Diskutovali jsme i tento postup, který se ukázal jako chybný:
   > opakuj následující krok:
   >
   > > mezi joby v seznamu s nejmenším deadlinem vezmi job s největší váhou a ten zařaď k vykonání; vyřaď ze seznamu ty joby, které již díky jejich deadlinu nelze přidat.

   Později jsme si uvědomili, že jako systém nezávislých množin lze vzít ty množiny jobů, které jsou proveditelné (v rámci svých deadlines). Speciálně jsme si ověřili, že množina $S \subseteq Jobs$ je proveditelná (a tedy nezávislá v našem matroidu) právě tehdy, když pro každé $d \in \{1, 2, \ldots, d_{max}\}$ (kde d_{max} je největší deadline u všech jobů) platí, že v S je nejvýše d jobů s deadlinem $\leq d$.

5. Popište rámcově implementaci hladového algoritmu pro náš "scheduling".

   Implementace je zřejmá z výše uvedeného.

6. Uvažujme matici A typu $m \times n$ nad $\mathbb{R}$ (nad tělesem reálných čísel). Chápejme ji jako reprezentaci matroidu M, kde $E(M)$ je množina sloupců matice A a $\mathcal{I}(M)$ obsahuje právě množiny vzájemně nezávislých sloupců (chápaných jako vektorů v $\mathbb{R}^m$). Ověřte, že se opravdu jedná o matroid.

   To se zdálo všem jasné díky elementární lineární algebře.

7. Připomeňme, že množině vrcholů grafu řekneme *nezávislá množina*, jestliže mezi žádnými jejími dvěma vrcholy není hrana. Uvažme následující algoritmus, jehož vstupem je graf $G = (V, E)$:

$S := \emptyset$; $U := V$;
while $U \neq \emptyset$ **do**
    choose (any) $v \in U$; $U := U - v$;
    if $S + v$ je nezávislá množina **then**
        $S := S + v$
    end if
end while

Ukažte, že daný algoritmus nesestrojí vždy největší nezávislou množinu v daném grafu. Systém nezávislých množin na daném grafu tedy netvoří matroid; co chybí?

Ukázali jsme si, že např. graf ve tvaru hvězdy (střed s paprsky) ukazuje, že axiom I3 není splněn. V budoucnu se vrátíme k tomu, že problém maximální nezávislé množiny v tomto smyslu je NP-těžký (a špatně aproximovatelný).

Týden 3 (od 22.2.)

Na prvním setkání v týdnu jsme dokončili diskusi úkolů z minulého týdne a pak jsme se také zamysleli nad tím, že ke kostře maximální váhy lze dojít i z "druhé strany": odebíráním hran s nejnižší váhou, dokud graf zůstává souvislý. (To je jistě lepší postup v případě velkého souvislého grafu, v němž je počet hran jen o málo větší než počet vrcholů.)

Duální matroid. Obecněji jsme si uvědomili, že ke každému matroidu M, danému množinami $E(M)$ a $\mathcal{I}(M) \subseteq 2^{E(M)}$, existuje *duální matroid* M^*. Před jeho definicí připomeňme, že *báze matroidu* M je každá množina $B \in \mathcal{I}(M)$ pro niž $r_M(B) = r_M(E(M))$; zobrazení (rank) $r_M : 2^{E(M)} \to \mathbb{N}$ je přitom definováno takto: $r_M(S) = \max\{|S'|; S' \subseteq S, S' \in \mathcal{I}(M)\}$.

Pro duální matroid M^* pokládáme $E(M^*) = E(M)$ a

$$\mathcal{I}(M^*) = \{S \subseteq E(M) \mid E(M) \setminus S \text{ obsahuje bázi matroidu } M\}.$$

Splnění axiomu I1 ($\emptyset \in \mathcal{I}(M^*)$) a I2 ($X \subseteq Y$, $Y \in \mathcal{I}(M^*) \implies X \in \mathcal{I}(M^*)$) je zřejmé. Pro I3 ($X, Y \in \mathcal{I}(M^*), |X| < |Y| \implies \exists e \in Y \setminus X : X + e \in \mathcal{I}(M^*)$), lze provést tento důkaz (nejlépe znázorněný obrázkem):

> Nechť $X, Y \in \mathcal{I}(M^*), |X| < |Y|$. Napišme $E(M) = X \cup B \cup C$, kde X, B, C jsou vzájemně disjunktní a B je báze matroidu M. Jestliže $Y \cap C \neq \emptyset$, pak každé $e \in Y \cap C$ splňuje tvrzení. Nechť nyní $Y = Y_1 \cup Y_2$, kde $Y_1 \subseteq X$ a $Y_2 \subseteq B$; nutně $|Y_2| > |X \setminus Y_1|$. Nechť $B' \subseteq E(M) \setminus Y$ je báze matroidu M; můžeme psát $B' = B'_1 \cup B'_2$, kde $B'_1 \subseteq X \setminus Y_1$ a $B'_2 \subseteq (B \setminus Y_2) \cup C$. Díky platnosti axiomu I3 pro M můžeme k B'_2 (což je nezávislá množina v M) postupně přidat $|B'_1|$ prvků z B tak, že výsledná množina B'' je bází M. Jelikož $|B'_1| < |Y_2|$, existuje $e \in Y_2$ takové, že $e \notin B''$; pro něj tedy platí $B'' \subseteq (E(M) \setminus (X + e))$.

Nové úkoly.

1. **Grafový matroid je vlastně i lineární matroid.** Uvažujme ke grafu $G = (V(G), E(G))$ matici typu $|V(G)| \times |E(G)|$, kde j-tý sloupec, odpovídající j-té hraně, má jedničky v řádcích odpovídajících koncovým vrcholům j-té hrany a jinak nuly.

   Ověřte, že grafový matroid grafu G (nezávislé množiny jsou lesy) odpovídá lineárnímu matroidu reprezentovanému onou maticí, *interpretujeme-li* ji jako matici nad tělesem $\{0, 1\}$ (s binárním sčítáním, tedy $1 + 1 = 0$).

2. Proveďte řešení naší instance scheduling problému (z knihy [9]) hladovým algoritmem v duálním matroidu. Také se zamyslete nad implementací tohoto algoritmu.

3. Zamyslete se nad případem, kdy deadlines u našich jobů mohou být racionální (a joby stále trvají všechny stejně, tedy jednu jednotku času).

4. Připomeňte si problém nejkratších cest v grafu, asi jste se s ním už všichni setkali (viz např. kapitola 2 v [9]). Zformulujte známý problém "převozník, vlk, koza, zelí" jako problém hledání nejkratší cesty v grafu.

Týden 4 (od 29.2.)

Dokončili jsme diskusi dřívějších úkolů, speciálně pak implemetaci řešení našeho "scheduling problému" hladovým algoritmem v duálním matroidu. Uvědomili jsme si, že k zjištění toho, zda množina jobů je nezávislá v duálním matroidu, tedy zda její doplněk obsahuje bázi původního matroidu, stačí udržovat pro každé $d \in \{0, 1, 2, \ldots, d_{max}-1\}$ počet jobů $n[d]$ v doplňku, které mají deadline větší než d. (Číslo $n[d]$ při postupné konstrukci začínající prázdnou množinou, jejímž doplňkem je množina všech jobů, nesmí podklesnout pod $d_{max}-d$, resp. nesmí vůbec klesnout, pokud je již na začátku menší než $d_{max}-d$).

Průnik matroidů (a bipartitní párování).

Připomněli jsme si pojem *párování* (*matching*) v (neorientovaném) grafu. Jedná se o množinu S hran, v níž žádné dvě různé hrany nejsou incidentní; každý vrchol má tedy stupeň ≤ 1 vzhledem k S (tedy v podgrafu majícím S jako množinu hran). Připomněli jsme si také *bipartitní grafy*, tedy grafy typu $G = (V(G), E(G))$, kde $V_G = V_1 \uplus V_2$ (symbol $\uplus$ znamená disjunktní sjednocení, množiny V_1, V_2 jsou tedy disjunktní) a každá hrana má jeden konec ve V_1 a druhý ve V_2.

Snadno jsme si všimli, že k nalezení maximálního párování v bipartitním grafu, čímž se rozumí párování s největším počtem hran (jedná se tedy o maximal-cardinality matching), nestačí hladový algoritmus (spočívající v přidávání hran, dokud to jde).

Dokázali jsme tak de facto, že máme-li dva matroidy M_1, M_2 se stejnou nosnou množinou $E = E(M_1) = E(M_2)$, tak systém množin

$$\mathcal{I}(M_1) \cap \mathcal{I}(M_2) = \{S \subseteq E \mid S \in \mathcal{I}(M_1), S \in \mathcal{I}(M_2)\}$$

obecně nesplňuje axiomy matroidu (konkrétně axiom $I3$):

Mějme bipartitní graf $G = (V_1 \uplus V_2, E)$. Pak systém

$$\mathcal{I}_1 = \{S \subseteq E \mid \text{každý } v \in V_1 \text{ má stupeň} \leq 1 \text{ vzhledem k } S\}$$

splňuje axiomy matroidu (ověřte!); tyto axiomy splňuje i systém

$$\mathcal{I}_2 = \{S \subseteq E \mid \text{každý } v \in V_2 \text{ má stupeň} \leq 1 \text{ vzhledem k } S\}.$$

Párování v G očividně odpovídají množinám z množiny $\mathcal{I}_1 \cap \mathcal{I}_2$. Kdyby systém $\mathcal{I}_1 \cap \mathcal{I}_2$ splňoval axiomy matroidu, pak by hladový (matroidový) algoritmus musel najít maximální párování (ovšem my jsme ukázali, že obecně nenajde).

Pro maximální párování tedy nemůžeme použít (primitivní) hladový algoritmus; příště si ale mj. připomeneme, že k nalezení maximálního párování v bipartitním grafu existují (jiné) rychlé algoritmy. Rychlé algoritmy, tedy algoritmy s časovou složitostí omezenou polynomem nízkého stupně, existují dokonce i pro nalezení maximálního váženého párování v obecném grafu (maximum-weight matching); zde každá hrana má váhu (reálné, či raději racionální, číslo) a úkolem je nalézt párování S s maximálním součtem vah hran v S.

Alespoň přitom zaznamenáme, že existují rychlé algoritmy, které pro dva matroidy M_1, M_2 se stejnou nosnou množinou, v níž je každý prvek opatřený váhou, naleznou v $\mathcal{I}(M_1) \cap \mathcal{I}(M_2)$ množinu s maximální váhou.

Nejkratší cesty.

Teď jsme se ovšem věnovali problému nejkratších cest v orientovaných grafech $G = (V, E, c)$ s váhou (cenou) na hranách ($c : E \to \mathbb{R}$; každá hrana má přiřazeno číslo reprezentující její

hodnotu, váhu, délku ...), který je stavebním blokem pro mnohé algoritmy kombinatorické optimalizace a který se vyskytuje v praxi v nejrůznějších situacích. (Nejen v situaci nalezení nejkratší cesty z města A do města B.) Opírali jsme se hlavně o kapitolu 2 knihy [9] a prováděli algoritmy detailně na příkladě, včetně diskuse rozumné implementace. (Diskutovali jsme i problém negativních cyklů.)

1. Bellman-Ford: počítáme vzdálenosti z v_0 k ostatním vrcholům, přičemž postupně povolujeme $k = 0, 1, 2, \ldots, |V|-2$ vnitřních vrcholů na příslušných cestách (paths), tj. povolujeme cesty s maximálně $1, 2, \ldots, |V|-1$ hranami; složitost je $O(|V| \cdot |E|)$ a tedy $O(|V|^3)$.

2. Floyd-Warshall: počítáme matici vzdáleností každého ke každému; vrcholy máme seřazeny $v_1, v_2, \ldots, v_n$ a používáme iterace $i = 0, 1, \ldots, |V|$, přičemž v iteraci i povolujeme jako vnitřní vrcholy příslušných cest jen vrcholy $v_1, v_2, \ldots, v_i$. Složitost $O(|V|^3)$.

3. Dijkstra: v případě *nezáporných vah* počítáme vzdálenosti z v_0 k ostatním vrcholům; používáme nanejvýš $|V|$ iterací, přičemž v každé iteraci přidáme alespoň jeden vrchol do P (permanent) – nejkratší cesta z v_0 k němu vede přes permanentní – a upravíme "vzdálenost" u každého dočasného (temporary) w tak, aby odpovídala nejkratší cestě z v_0 do w, v níž jsou jako vnitřní vrcholy povoleny jen ty permanentní. Složitost $O(|V|^2)$.

Nové úkoly.

1. Projděte si detailně uvedené algoritmy pro hledání nejkratších cest.

   Speciálně navrhněte přehledně datovou strukturu a její naplnění (nemusíte přitom používat konkrétní programovací jazyk), která po skončení Floyd-Warshallova algoritmu umožní snadno vyhledat nejkratší cestu mezi libovolnou dvojicí vrcholů.

2. Podejte důkaz korektnosti Dijkstrova algoritmu. (Nejde o detailní formality, ale o jasné argumenty, proč jím nalezené cesty jsou opravdu ty nejkratší.)

3. Předběžně se zamyslete se nad metodou zvanou *lineární programování (LP)*, na niž se brzy podíváme podrobněji. (Pěkný úvod do LP lze nalézt např. v [2]. Na úvodní texty lze odkázat samozřejmě i jinam, např. do [1].)

   Stručně: jedná se o varianty problému

   $$\text{maximalizuj } c^T x \text{ za podmínky } Ax \leq b,$$

   kde A je reálná matice $m \times n$, $b \in \mathbb{R}^m$, $c \in \mathbb{R}^n$. (Hledáme tedy $x \in \mathbb{R}^n$ splňující $Ax \leq b$, tedy tzv. *přípustné řešení*, pro něž je $c^T x$ maximální, pokud takové x existuje; pokud existuje, jedná se o *optimální řešení*.) (Indexem ^{T} označujeme operaci transpozice. Vektory c, x, b chápeme jako sloupcové, vektor c^T je tedy řádkový.)

   Zformulujte náš "scheduling problém" jako instanci problému LP.

4. ("Nepovinně.")

   Zformulujte problém nejkratších cest jako problém lineárního programování.

   K uvedené formulaci problému nejkratších cest jako problému LP můžete použít Problem (Minimum-weight dipaths by linear programming), str. 77 v [9]. (POZOR: V [9]

je uvedeno $\max \sum_{e \in E(G)} c(e)x_e$. Z kontextu je ale vidět, že byla zamýšlena varianta $\min \sum_{e \in E(G)} c(e)x_e$ [tedy $\max \sum_{e \in E(G)} -c(e)x_e$].)

Ukažte, že orientovaná kostra, která je produktem Bellman-Fordova či Dijkstrova algoritmu na našem příkladu přirozeně poskytuje přípustné řešení uvedeného LP-problému.

5. ("Nepovinně.")

   Podívejte se na Example (Directed Hamiltonian tours) na str. 89 v [9]. Na základě toho zkuste vysvětlit, jak bychom (jakýmkoli) algoritmem A pro hledání maximální (cardinality-maximum) množiny v průniku $\mathcal{I}(M_1) \cap \mathcal{I}(M_2) \cap \mathcal{I}(M_3)$, kde M_1, M_2, M_3 jsou tři matroidy se stejnou nosnou množinou, vyřešili nalezení hamiltonovské cesty v zadaném grafu (tj. orientovaného cyklu procházejícího každým vrcholem právě jednou).

   Na základě vašich znalostí o výpočetní složitosti problémů odhadněte, zda je znám rychlý algoritmus A.

Týden 5 (od 7.3.)

Připomněli jsme si látku minulého týdne, speciálně základní tvar problému lineárního programování.

Maximální (vážené) párování v (bipartitních) grafech.

Pouvažovali jsme nad algoritmem, který k zadanému (neorientovanému) grafu $G = (V, E, c)$, kde $c : E \to \mathbb{R}$ přiřazuje každé hraně e její váhu (cenu) $c(e)$, sestrojí pro každé $k = 0, 1, \ldots, r$ nějaké párování $S_k \subseteq E$ (žádné dvě různé hrany v S_k tedy nejsou incidentní) takové, že $|S_k| = k$ a hodnota $c(S_k) = \sum_{e \in S_k} c(e)$ je maximální; je tedy

$$c(S_k) = \max\{c(S) \mid S \text{ je párování s } k \text{ hranami }\}. \tag{1}$$

Zde r je počet hran v maximálním párování (maximum-cardinality matching), čímž rozumíme největší párování z hlediska počtu hran. Číslo r náš algoritmus také zjistí.

> Jako konkrétní příklad jsme vzali instanci problému "Assignment problem" řešeného v Example (Kuhn's Assignment Algorithm) na str. 123 v [9].

Speciálním případem je případ, kde $c(e) = 1$ pro každou hranu e; algoritmus pak prostě sestrojí nějaké maximální párování.

Takový algoritmus jistě existuje, stačí použít hrubou sílu (brute-force), otázkou je nalezení efektivních algoritmů (z hlediska časové složitosti). My jsme ukázali polynomiální algoritmus pro případ bipartitních grafů; nejprve jsme ale udělali důležité pozorování pro obecné grafy:

Mějme v obecném grafu $G = (V, E, c)$ párování S_k s k hranami, které je optimální pro k, tedy splňuje (1), a snažme se vytvořit nějaké párování S_{k+1} s $k{+}1$ hranami, které je optimální pro $k{+}1$. Představme si hrany v S_k jako modré, taky s nimi incidentní vrcholy budou modré, a ostatní hrany budou červené; vrcholy neincidentní s modrými hranami budou červené.

Připomeňme, že cestou (path) z vrcholu v do w rozumíme posloupnost na sebe navazujících hran s jedním volným koncem ve v a druhým ve w, kde se žádný vrchol (a tedy ani žádná hrana) neopakuje; pokud $v = w$, tak ony "volné konce" jsou spojeny, jedná se o (jednoduchý) cyklus, který také formálně chápeme jako cestu z v do v.

Cestu z v do w v našem modro-červeném grafu nazveme *alternující*, jestliže se v ní modré a červené hrany střídají. Dvě modré po sobě být stejně nemohou (modré tvoří párování a proto na sebe nemohou navazovat), dvě incidentní červené se v alternující cestě mohou vyskytovat jen v případě, že se jedná o cestu z v do v (tedy cyklus) začínající a končící červenou hranou; ten cyklus pak má lichou délku.

Hodnotou alternující cesty p rozumíme $\sum_{\text{červené } e \in p} c(e) - \sum_{\text{modré } e \in p} c(e)$. Speciálně si všimněme, že pokud je alternující cesta sudým cyklem, tak počty modrých a červených hran jsou v ní stejné a její hodnota nemůže být pozitivní. (Jinak by S_k nebylo optimální; vidíte proč?)

Tvrzení. *Jako kýžené S_{k+1} (optimální pro $k{+}1$) můžeme vzít toto: v modro-červeném grafu vytvořeném z G pro S_k nalezneme alternující cestu z červeného vrcholu do jiného červeného vrcholu, která má nejvyšší možnou hodnotu. Pak v této cestě přebarvíme modré hrany na červené a červené na modré. Pokud taková cesta neexistuje, pak v G neexistuje (vůbec žádné) párování s $k{+}1$ hranami.*

Důkaz. Uvažujme nejprve, že existuje nějaké S_{k+1}, které je optimální pro $k{+}1$ (a které nemuselo vzniknout navrženým způsobem); zafixujme ono S_{k+1} a obarvěme jeho hrany zeleně (vrcholy nepřebarvujeme).

Jak vypadají (souvislé) komponenty grafu vzniklého z G tak, že ponecháme jen modré a zelené hrany? Žádné dvě různé modré hrany nejsou incidentní, žádné dvě různé zelené hrany nejsou incidentní … Každá komponenta je tedy očividně

- buď jedna zelená hrana vzniklá přebarvením modré (jak S_k, tak S_{k+1} obsahují tu hranu);
- nebo se jedná o alternující cestu (v níž byly červené hrany přebarveny na zelené); pokud je to cyklus, nutně obsahuje stejný počet modrých a zelených hran a hodnota tohoto cyklu je nula (jelikož S_k je optimální pro k a S_{k+1} je optimální pro k+1).

Jelikož zelených hran je o jedna víc než modrých, alespoň jedna komponenta je alternující cestou z červeného vrcholu do jiného červeného vrcholu (proč?). Ve zbylých komponentách jsou celkově počty modrých a zelených hran stejné a součty jejich cen musí být také stejné! (Jinak by buď S_k nebylo optimální pro k nebo S_{k+1} by nebylo optimální pro k+1.)

Důkaz tvrzení je tím vlastně hotov … (domyslete …). □

Problémem je, že v obecném grafu je ono hledání alternující cesty z červeného vrcholu do jiného červeného vrcholu s nejvyšší hodnotou zapeklitý problém. Problémy dělají cykly liché délky. V případě "maximum-cardinality matching" (cena každé hrany je jedna) se to dá chytře vyřešit (ještě se příště k tomu vrátíme), případ "maximum-weight matching" je technicky výrazně komplikovanější a nebudeme se jím zde dále zabývat.

> Aspoň zaznamenáme, že "maximum-weight matching" patří mezi "nejtěžší" problémy kombinatorické optimalizace, pro něž jsou známy polynomiální algoritmy (viz např. [5]).

My jsme si ale všimli, že u *bipartitního* grafu $G = (V_1 \cup V_2, E, c)$ to umíme: Pro zkonstruované optimální S_k dáme hranám orientaci: červené "zleva doprava", tedy z V_1 do V_2, modré (tedy elementy S_k) "zprava doleva", tedy z V_2 do V_1; pro každou modrou e teď obrátíme cenu, položíme tedy (dočasně) $c(e) := -c(e)$. A hledáme prostě nejdelší cestu z červeného vrcholu do jiného červeného vrcholu; ta cesta je díky orientaci hran v našem bipartitním grafu automaticky alternující! (Navíc nutně začíná ve V_1 a končí ve V_2.) Můžeme tedy použít např. algoritmus "Floyd-Warshall"; hledáme nejdelší cesty, ale pozitivní cykly tam nejsou (díky optimalitě S_k, jak jsme již diskutovali), takže ok … Ukázali jsme tak

polynomiální algoritmus konstruující "maximum-weight matching" v bipartitních grafech.

Poznamenejme, že námi uvedený algoritmus je koncepčně jednoduchý: je založen na jednoduchém pozorování a aplikaci hledání nejkratších cest (přesně řečeno hledání nejdelších cest v situaci, kdy nejsou pozitivní cykly …). Algoritmus uvedený v [9] je založen na myšlenkách duality v lineárním programování, ke kterým se ještě dostaneme …

Maximální toky, minimální řezy.

Téma jsme jen stručně připomněli. (Základy najdete např. ve zmiňovaných skriptech Petra Kováře [7] a samozřejmě i v našich referenčních knihách [5, 9].) Všímali jsme si hlavně plynulé návaznosti na náš algoritmus pro maximum-weight matching, který byl postaven na zlepšujících cestách.

> Problém jsme ilustrovali na jednoduchém příkladu Exercise (Edmonds-Karp labeling) na s. 144 v [9].

Ještě se k tématu stručně vrátíme.

Uvědomili jsme si také mj., že problém maximálního toku není striktně vzato problémem kombinatorické optimalizace (ale speciálním případem lineárního programování); na druhé straně duální problém minimálního řezu je "čistým" problémem kombinatorické optimalizace.

Nové úkoly.

1. Připomeňme si instanci problému nejkratších cest ze s. 77 v [9] a dále si připomeňme orientovanou kostru s kořenem ve vrcholu a, kterou vytvořil Dijkstrův algoritmus.

   Vysvětlete, jak tato kostra odpovídá řešení příslušné instance LP (lineárního programu) navržené v Problem (Minimum-weight dipaths by linear programming) také na s. 77 v [9]. (V knize chybí znaménko $-$; ma být $\max \sum_{e\in E(G)} -c(e)x_e$, neboli $\min \sum_{e\in E(G)} c(e)x_e$.)

2. Vysvětlete, co to je "Assignment problem" řešený v Example (Kuhn's Assignment Algorithm) na str. 123 v [9] a aplikujte na uvedenou konkrétní instanci ("osoby" $1, 2, 3, 4$, úkoly "a,b,c,d") polynomiální algoritmus k vyřešení. (Předpokládám, že použijete ten "náš" algoritmus, odvozený výše.)

3. Připomeňme, že pro (neorientovaný) graf $G = (V, E)$ je množina $C \subseteq V$ *vrcholovým pokrytím* (vertex cover), jestliže každá hrana (v E) je incidentní alespoň s jedním vrcholem z C. (Když nasadíme "hlídače" do každého vrcholu v C, tak každá hrana je [alespoň na jednom konci] "hlídaná", či "pokrytá".)

   Když S je párování a C je vrcholové pokrytí (v daném grafu), tak $|S| \leq |C|$ (proč?).

   Představte si teď, že jste řešili maximum-cardinality matching v *bipartitním* grafu a skončili jste tedy v situaci s nějakým párováním S_k a s modrými a červenými (orientovanými) hranami tak, že neexistuje cesta z červeného do jiného červeného vrcholu. Navrhněte vrcholové pokrytí C daného grafu tak, že $|C| = |S_k|$.

   Podařilo se? Pak jste dokázali tzv. *Königovu větu*:

   v bipartitním grafu se velikost maximálního párování rovná velikosti minimálního vrcholového pokrytí.

   (Platí to i pro nebipartitní grafy, např. kliku K_3 ?)

4. Zformulujte standardní verzi problému maximálního toku jako problému LP (lineárního programování).

5. V dohledné době si zkuste najít nějaký volně použitelný "Linear programming solver" (jedna možnost je GLPK (GNU Linear Programming Kit), https://www.gnu.org/software/glpk/) a zkuste v něm naformulovat a vyřešit nějakou úlohu dle vlastního výběru ...

Týden 6 (od 14.3.)

Připomněli jsme si mj. Königovu větu (v bipartitním grafu se velikost maximálního párování rovná velikosti minimálního vrcholového pokrytí). Důkaz vyplynul z našeho algoritmu pro konstrukci maximálního párování (maximum-cardinality matching): po jeho skončení vybereme do (minimálního) vrcholového pokrytí C modré body takto: z každé modré hrany, která je dosažitelná z nějakého červeného bodu (vzpomeňme, že červené hrany jdou zleva doprava, z V_1 do V_2, a modré hrany zprava doleva, z V_2 do V_1), zařadíme do C pravý vrchol (ten z V_2), a ze zbylých modrých hran zařadíme do C levé vrcholy (ty z V_1) .... Žádná hrana nemohla zůstat nepokrytá (proč?).

Maximální tok, minimální řez (pokračování). Připomněli jsme si (využitím obrázků ze s. 142, 143 v cite [9]) algoritmus postupného zlepšování přípustného toku, nalézáním cest ze zdroje s do stoku t po "nenasycených" hranách: po hraně (u, v) lze jít "po směru" (z u do v), jestliže aktuálně máme $x_e < c(e)$, anebo "proti směru" (z v do u), jestliže máme $x_e > 0$. (Předpokládáme nulové dolní meze.) Na zafixované zlepšující cestě upravíme x_e tak, abychom tok co nejvíce zvětšili (na hranách e "po směru" zvětšíme x_e o Δ, na hranách e "proti směru" zmenšíme x_e o Δ); hrany, které brání většímu zvětšení, tedy ty, které se změnou nasytí (x_e bude $c(e)$ v případě hran "po směru" a 0 v případě hran "proti směru"), nazveme *kritické* (v dané iteraci).

Připomněli jsme, že jednoduchá strategie, která ke zlepšení bere vždy nějakou *nejkratší* zlepšující cestu (nejkratší z hlediska počtu hran, nalezenou např. prohledáváním do šířky), zkonstruuje maximální tok v polynomiálně mnoha iteracích.

> Důkaz je např. v [9]. Jedná se o vcelku přímočarý rozbor toho, že při zvolené strategii nejkratších zlepšujících cest po každé iteraci, tedy po každé úpravě aktuálního toku, vzdálenost "zdroj $s \to$ vrchol u" ani vzdálenost "vrchol $u \to$ stok t" nemůže klesnout, pro každý jednotlivý vrchol u; vzdálenosti přitom měříme počtem hran v příslušných nejkratších cestách užívajících jen nenasycené hrany. Také se snadno odvodí, že když se konkrétní hrana e objeví jako kritická hrana ve zvolené zlepšující cestě v iteraci i a později v iteraci $j > i$, tak délka zlepšující cesty v iteraci j je aspoň o dvě hrany větší než délka zlepšující cesty v iteraci i. Pro každou hranu e je tedy $|V|/2$ horní mez pro počet iterací, kdy je e kritická. V každé iteraci alespoň jedna hrana kritická je, takže $|E| \cdot |V|/2$ je horní mezí pro celkový počet iterací.

Konstrukce maximálního toku tak skončí v případě, že množina $S \subseteq V$ vrcholů, které jsou dosažitelné ze zdroje po nenasycených hranách, neobsahuje stok t. (Zdroj s patří do S triviálně, je dosažitelný z s cestou nulové délky.)

Kapacita řezu $(S, V \setminus S)$, tedy $C(S) = \sum_{e \in \delta^+(S)} c(e)$ (kde $\delta^+(S)$ je množina hran, jež mají počátek v S a konec mimo S) je tedy zcela naplněna: pro všechny $e \in \delta^+(S)$ je $x_e = c(e)$; přitom $x_e = 0$ pro všechny hrany $e \in \delta^-(S)$, tedy pro hrany mající počátek mimo S a konec v S.

Snadno ověříme, že pro libovolnou $S \subseteq V$, kde $s \in S$ a $t \in V \setminus S$, je kapacita $C(S)$ horní mezí pro velikost jakéhokoli (tedy i maximálního) přípustného toku (tj. pro bilanci zdroje, tj. pro hodnotu $\delta^+(s) - \delta^-(s)$). Dokázali jsme tím známé tvrzení stručně vyjádřené jako

maximální tok je roven minimálnímu řezu.

Všimněme si tzv. *dobré charakterizace* v tomto případě: rovnost řešení dvou "duálních" úloh zde prokazuje optimalitu obou z nich; navíc, když jedna má optimální řešení, tak má optimální řešení i duální úloha a hodnoty těchto řešení se rovnají.

Zformulovali jsme problém maximálního toku jako problém LP (lineárního programování). (Uvažovali jsme i dolní omezení ℓ pro přípustné toky. V tom případě jako kapacitu řezu bereme $C(S) = \sum_{e\in\delta^+(S)} c(e) - \sum_{e\in\delta^-(S)} \ell(e)$.)

Maximální párování v obecných grafech.

> Jedná se o zobecnění konstrukce "maximum-cardinality matching" z bipartitních na obecné grafy. K tomu jsme se na setkání nedostali. Necháme tuto sekci k "nepovinnému" promyšlení zájemcům.

Naznačme alespoň myšlenku polynomiálního algoritmu, který k danému párování S_k s k hranami (v obecném neorientovaném grafu G) nalezne párování S_{k+1} s $k+1$ hranami, pokud takové párování existuje.

Připomeňme, že hrany v S_k považujeme za modré, i vrcholy s nimi incidentní jsou modré; ostatní hrany a vrcholy jsou červené. (Hrany jsou zde neorientované.) Ukázali jsme si již, že S_{k+1} existuje právě tehdy, když existuje alternující cesta z červeného do červeného vrcholu (v níž jsou tedy barvy hran

červená - modrá - červená - $\cdots$ - modrá - červená).

V takové cestě je o jednu červenou hranu navíc oproti modrým; když v té cestě přebarvíme červené hrany na modré a modré na červené, dostaneme S_{k+1}.

Představme si, že začneme hledat takovou alternující cestu z nějakého červeného vrcholu v_0 hledáním do šířky. Předpokládejme, že příslušnou cestu nenajdeme, ale narazíme na cyklus

v_0- červená modrá $\cdots$ červená modrá $-v_1-$ červená modrá $\cdots$ červená modrá červená $-v_1$.

Teď je klíčové pozorování:

Na úseku "v_0 červená modrá $\cdots$ červená modrá v_1" prohodíme obarvení hran; tím dostaneme jiné párování S'_k s k hranami. (Žádné dvě různé modré hrany se nemohly stát incidentními.) Vrchol v_1 se takto stane červeným a máme pro něj lichý alternující cyklus, který má o jednu červenou hranu navíc oproti modrým. Z cyklu ven nevede žádná modrá hrana; uzavřeme ten cyklus do "bubliny" a díváme se na bublinu jako na jeden červený vrchol (provedli jsme kontrakci několika vrcholů do jednoho; každá hrana, která vedla do nějakého vrcholu v cyklu teď vede do onoho jednoho vrcholu). Ve vzniklém menším grafu jsme tím žádnou alternující cestu z červeného do červeného nemohli ztratit. Když nalezneme v menším grafu cestu z červeného do červeného, tak v té cestě přebarvíme hrany (čímž se počet modrých hran zvětší o jedna); pak zase "rozbalíme" bublinu a pokud po našem přebarvení teď vstupuje nějaká modrá hrana do bubliny (samozřejmě je nanejvýš jedna), tak prostě obarvení hran v cyklu v bublině "otočíme" tak, aby vrchol, do něhož vstupuje modrá hrana zvenku, měl obě hrany v cyklu červené ....

Lineární programování.

Seznámíme se jen krátce se základy lineárního programování. Připomněl jsem, že pěkný stručný úvod lze nalézt např. v [2].

Jako konkrétní instanci "problému diety" z [2] jsme vzali případ, kdy uvažujeme tři potraviny F_1, F_2, F_3 (F jako "food") a dvě živiny N_1, N_2 (N jako "nutrient"). V jednotce (např. jednom kg) F_1 jsou obsaženy 1 jednotka (např. mg) N_1 a 2 jednotky N_2. V jednotce F_2 jsou obsaženy 4 jednotky N_1 a 1 jednotka N_2. V jednotce F_3 jsou obsaženy 4 jednotky N_1 a 3 jednotky N_2. Cena za jednotku F_1 je 6 (např. desetikorun), cena za jednotku F_2 je 9 a cena za jednotku F_3 je 10. Minimální denní doporučená dávka živiny N_1 je 3 jednotky; u živiny N_2 jsou to 4 jednotky. Úkolem je navrhnout nákup potravin F_1, F_2, F_3 tak, aby nakoupené potraviny obsahovaly (alespoň) minimální denní doporučenou dávku živin a přitom cena nákupu byla co nejmenší.

Hledané množství nákupu potraviny F_i jsme si označili y_i (pro $i = 1, 2, 3$); jedná se o nezáporné reálné číslo. (Předpokládáme, že potraviny není nutno kupovat v celých jednotkách.) Definovali jsme si matici A typu $m \times n = 3 \times 2$, kde prvek a_{ij} (v i-tém řádku a j-tém sloupci) je počet jednotek živiny N_j v potravině F_i. Vektor b (typu $m \times 1$) jsme definovali tak, že b_i je cena za jednotku F_i. Vektor c (typu $n \times 1$) jsme definovali tak, že c_j je minimální doporučená denní dávka živiny N_j.

Máme tedy minimalizovat součin $y^T b$

(kde y^T je transponovaný vektor y, tedy $y^T = (y_1, y_2, y_3)$ je typu $1 \times m$)

za podmínek $y^T A \geq c^T$, $y \geq \mathbf{0}$

(kde $\mathbf{0}$ je nulový vektor příslušného rozměru).

Zauvažovali jsme také o duálním problému

$$\max c^T x \text{ za podmínek } Ax \leq b,\ x \geq \mathbf{0}.$$

Zde (hypotetický) "prodejce pilulek s živinami" hledá cenu x_1 za jednotku živiny N_1 a cenu x_2 za jednotku N_2 tak, aby maximalizoval cenu denní doporučené dávky, ale zároveň "nebyl dražší" než příslušné potraviny.

Nové úkoly.

1. Standardně se jako *primální problém* nebo též *primální úloha* (anglicky "primal", česky též "primární" či "prvotní") chápe problém LP ve tvaru

   $$\max c^T x \text{ za podmínek } Ax \leq b, x \geq \mathbf{0}.$$

   Zapište tuto úlohu s konkrétními čísly z našeho "dietního problému", znázorněte geometricky množinu přípustných řešení (tedy množinu vektorů $x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$ splňujících $Ax \leq b, x \geq \mathbf{0}$) a nalezněte (nějak "logicky") optimální řešení v tomto konkrétním případě.

2. Ukažte detailně, proč z podmínek $Ax \leq b, x \geq 0, y^T A \geq c^T, y \geq 0$ plyne

   $$c^T x \leq y^T A x \leq y^T b.$$

   Speciálně ukažte, jak využijeme asociativitu operace násobení matic.

3. K (standardní primální) úloze

$$\max c^T x \text{ za podmínek } Ax \leq b, x \geq \mathbf{0}$$

je standardně definována *duální úloha*

$$\min y^T b \text{ za podmínek } y^T A \geq c^T, y \geq \mathbf{0}.$$

Co lze vyvodit z předchozího příkladu pro případ, kdy máme konkrétní přípustné řešení (označené) x^* primální úlohy a konkrétní přípustné řešení y^* duální úlohy?

4. K (primálnímu) problému $Ax \leq b, x \geq 0, \max c^T x$ jsme jako duální problém definovali $y^T A \geq c^T, y \geq 0, \min y^T b$. Ten se ovšem dá přirozeně vyjádřit také jako standardní maximalizační problém: $-A^T y \leq -c, y \geq 0, \max -b^T y$ (ověřte). Zkonstruujte k posledně uvedenému problému duální problém (s proměnnými označenými x) a ověřte, že je ekvivalentní formulaci $Ax \leq b, x \geq 0, \max c^T x$. (Tedy dvojnásobnou aplikací operátoru duality dostaneme původní problém.)

5. Uvažme obecný problém LP, kdy máme omezení ve formě nerovnic i rovnic a na některé proměnné klademe podmínku nezápornosti zatímco na jiné ne. Navrhněte, jak můžeme takový problém snadno převést do tvaru standardního maximalizačního problému.

6. Zamyslete se nad tím, zda každá úloha LP musí mít přípustné řešení, a zda každá úloha s přípustným řešením musí mít i optimální řešení.

   Úloha s přípustným řešením, která nemá optimální řešení, se nazývá neomezená. Jak je to s duální úlohou v tomto případě? (Může mít přípustné řešení?)

7. Zkuste se zamyslet nad následujícím tvrzením:

   Existuje-li optimální řešení úlohy

   $$\max c^T x \text{ za podmínek } Ax \leq b, x \geq \mathbf{0},$$

   tak existuje i takové optimální řešení x^*, pro něž sloupce matice A odpovídající nenulovým složkám vektoru x^* jsou nezávislé.

Týden 7 (od 21.3.)

Lineární programování (pokračování).

Navázali jsme na předchozí jistou rekapitulací shrnutou také níže. Zadání konkrétního příkladu (2) je převzato z [1].

Připomněli jsme, že mnohé problémy kombinatorické optimalizace lze přirozeně vyjádřit jako problémy *celočíselného* lineárního programování (Integer LP, ILP), kde proměnné x_i (složky vektoru proměnných x) nabývají jen celočíselných hodnot, často jen z množiny $\{0, 1\}$; stejně tak koeficienty v nerovnicích jsou celočíselné.

> Pokud jsou koeficienty racionální, lze samozřejmě převést na celočíselné vynásobením společným jmenovatelem.

Obecně je ovšem ILP NP-těžký problém (jak ještě připomeneme později), zatímco jeho *relaxace*, tedy připuštění všech reálných hodnot v daném intervalu (např. v intervalu $[0, 1]$), vede na úlohu LP, která je polynomiální (jak ještě také budeme diskutovat).

> Byť je LP obecně polynomiální, řešení pro velké instance má pořád velkou časovou náročnost z praktického hlediska. Ilustrovali jsme si to připomenutím problému maximálního toku. Ač tento problém snadno vyjádříte jako problém LP (že ano), pro řešení se v praxi používají specializované algoritmy [využívající techniku zlepšujících cest], které jsou výrazně efektivnější. Ovšem např. problém *multikomoditních toků* [kde máme více dvojic zdroj-stok s předepsanými velikostmi toků mezi nimi a hledáme "mix" všech toků při dodržení kapacit jednotlivých hran, případně ještě za minimální cenu] se umí optimálně řešit v polynomiálním čase jen přes LP. (V praxi pak lze upřednostnit efektivnější algoritmy, které jen aproximují optimální řešení. O aproximačních algoritmech budeme také ještě hovořit.)

Jako ilustrační příklad LP jsme vzali následující instanci (s reálnými proměnnými x_i):

$$\begin{aligned} &\text{maximalizuj} && 3x_1 + x_2 + 2x_3 && \text{za podmínek} \\ & && x_1, x_2, x_3 \geq 0, \\ & x_1 &+ x_2 &+ 3x_3 &\leq 30, \\ & 2x_1 &+ 2x_2 &+ 5x_3 &\leq 24, \\ & 4x_1 &+ x_2 &+ 2x_3 &\leq 36. \end{aligned} \tag{2}$$

Standardní maximalizační problém a standardní minimalizační problém. Náš příklad je příkladem *standardní maximalizační úlohy*, tedy problému typu

$$Ax \leq b, x \geq 0, \max c^T x.$$

> Připomínáme, že matice A je typu $m \times n$, vektory bereme primárně jako sloupcové, tedy b je typu $m \times 1$ a c je typu $n \times 1$; transponovaný c^T je tedy řádkový vektor, typu $1 \times n$. Pro konkrétní x je Ax sloupcovým vektorem, lineární kombinací sloupců matice A, s koeficienty $x_1, x_2, \ldots, x_n$. (Částečné) uspořádání vektorů je definováno po složkách; $Ax \leq b$ tedy znamená, že i-tá složka vektoru Ax je menší nebo rovna b_i, pro $i = 1, 2, \ldots, m$. Symbol 0 zde znamená vektor s nulovými složkami; jeho rozměr a "sloupcovost" či "řádkovost" jsou vždy zřejmé z kontextu. Někdy se znak transpozice vynechává, pokud nehrozí nedorozumění. Např. se často píše cx (skalární součin vektorů c, x) místo $c^T x$, apod.

Funkce $f(x) = c^T x$ se nazývá *účelová funkce* (objective function, česky také cílová funkce či kriteriální funkce).

Před řešením naší konkrétní úlohy jsme nejdříve diskutovali *standardní minimalizační problém*, tj. problém typu

$$y^T A \geq c^T, y \geq 0, \min\, y^T b.$$

(Místo sloupců bereme nezáporné lineární kombinace řádků; místo shora jsou tyto kombinace omezeny zdola, a místo maximalizování zde minimalizujeme.)

Dualita. Je přirozené brát $y^T A \geq c^T, y \geq 0, \min\, y^T b$ jako *duální problém* k problému $Ax \leq b, \max\, c^T x$, který se pak nazývá *primální problém* (anglicky "primal", česky též "primární" či "prvotní").

> Správně bychom měli rozlišovat pojem "problém", např. problém LP, a pojem "instance problému", např. $Ax \leq b, x \geq 0, \max c^T x$ pro konkrétní matici A a vektory b, c. Toto ale většinou necháváme na kontext, z něhož je jasné, co máme na mysli. Všimněme si také, že $y^T A \geq c^T, y \geq 0, \min\, y^T b$ lze podle standardních pravidel pro transpozici matic psát $A^T y \geq c, y \geq 0, \min\, b^T y$, nebo ekvivalentně $-A^T y \leq -c, y \geq 0, \max\, -b^T y$.

K naší konkrétní úloze (2) je tedy duální úlohou tato úloha:

$$\text{minimalizuj} \qquad 30y_1 + 24y_2 + 36y_3 \qquad \text{za podmínek} \tag{3}$$

$$y_1, y_2, y_3 \geq 0,$$

$$\begin{array}{rcrcrcl} y_1 & + & 2y_2 & + & 4y_3 & \geq & 3, \\ y_1 & + & 2y_2 & + & y_3 & \geq & 1, \\ 3y_1 & + & 5y_2 & + & 2y_3 & \geq & 2. \end{array}$$

Jako úkol jsme ponechali odvození (očekávaného) faktu, že duálním problémem k duálnímu problému je zase primální problém.

Přípustným vektorem (feasible vector), též *přípustným řešením*, úlohy $Ax \leq b, x \geq 0, \max c^T x$ rozumíme libovolný vektor x splňující $Ax \leq b, x \geq 0$. Podobně *přípustným řešením* úlohy $y^T A \geq c^T, y \geq 0, \min y^T b$ rozumíme libovolný vektor y splňující $y^T A \geq c^T$, $y \geq 0$. (Někdy má smysl uvažovat např. [nepřípustné] řešení splňující $Ax \leq b$, které není nezáporné.)

> Jak je obvyklé, používáme např. x jako symbol pro vektor proměnných, ale někdy také pro konkrétní vektor z $\mathbb{R}^n$. Konkrétní užití by mělo být vždy jasné z kontextu; někdy užijeme "dekoraci", např. x^*, když chceme zdůraznit, že máme na mysli konkrétní vektor.

Snadno jsme odvodili (už mezi úkoly), že podmínky

$$Ax \leq b, x \geq 0, y^T A \geq c^T, y \geq 0$$

implikují

$$c^T x \leq y^T A x \leq y^T b.$$

Máme-li tedy (jakékoli) přípustné řešení x^* primální úlohy a přípustné řešení y^* duální úlohy, tak platí $c^T x^* \leq (y^*)^T b$. Pokud tedy nastává případ $c^T x^* = (y^*)^T b$, tak obě řešení x^*, y^* jsou ve svých úlohách optimální!

Proveďme důkaz ještě detailně. Bude se nám to v budoucnu ještě hodit k jinému účelu (u "complementary slackness").

Zapišme si nejdříve $y^T Ax$ podrobněji:

$$\big(y_1, y_2, \ldots, y_m\big) \cdot \begin{pmatrix} a_{11}, a_{12}, \ldots, a_{1n} \\ a_{21}, a_{22}, \ldots, a_{2n} \\ \cdots \\ a_{m1}, a_{m2}, \ldots, a_{mn} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ \cdots \\ x_n \end{pmatrix}$$

Násobení matic je asociativní, je tedy $(y^T A)x = y^T(Ax)$; důkaz de facto provedeme níže. Rozepíšeme-li výraz $(y^T A)x$, dostáváme

$$\left(\big(y_1, y_2, \ldots, y_m\big) \cdot \begin{pmatrix} a_{11} \\ a_{21} \\ \cdots \\ a_{m1} \end{pmatrix}, \big(y_1, y_2, \ldots, y_m\big) \cdot \begin{pmatrix} a_{12} \\ a_{22} \\ \cdots \\ a_{m2} \end{pmatrix}, \ldots, \big(y_1, y_2, \ldots, y_m\big) \cdot \begin{pmatrix} a_{1n} \\ a_{2n} \\ \cdots \\ a_{mn} \end{pmatrix} \right) \cdot \begin{pmatrix} x_1 \\ x_2 \\ \cdots \\ x_n \end{pmatrix}$$

neboli $z^T \cdot x$, kde

$z^T = y_1 \cdot \big(a_{11}, a_{12}, \ldots, a_{1n}\big) + y_2 \cdot \big(a_{21}, a_{22}, \ldots, a_{2n}\big) + \cdots + y_m \cdot \big(a_{m1}, a_{m2}, \ldots, a_{mn}\big)$.

Tedy

$$(y^T A)\, x = \tag{4}$$

$$\begin{aligned} &= (y_1 a_{11} + y_2 a_{21} + \cdots + y_m a_{m1}) \cdot x_1 + \\ &+ (y_1 a_{12} + y_2 a_{22} + \cdots + y_m a_{m2}) \cdot x_2 + \\ &\ldots\ldots \\ &+ (y_1 a_{1n} + y_2 a_{2n} + \cdots + y_m a_{mn}) \cdot x_n\,. \end{aligned}$$

Rozepíšeme-li výraz $y^T(Ax)$, dostáváme

$$\big(y_1, y_2, \ldots, y_m\big) \cdot \left(\big(a_{11}, a_{12}, \cdots, a_{1n}\big) \cdot \begin{pmatrix} x_1 \\ x_2 \\ \cdots \\ x_n \end{pmatrix}, \ldots, \big(a_{m1}, a_{m2}, \cdots, a_{mn}\big) \cdot \begin{pmatrix} x_1 \\ x_2 \\ \cdots \\ x_n \end{pmatrix} \right)^T$$

neboli

$$\big(y_1, y_2, \ldots, y_m\big) \cdot \left(\begin{pmatrix} a_{11} \\ a_{21} \\ \cdots \\ a_{m1} \end{pmatrix} \cdot x_1 + \begin{pmatrix} a_{12} \\ a_{22} \\ \cdots \\ a_{m2} \end{pmatrix} \cdot x_2 + \cdots + \begin{pmatrix} a_{1n} \\ a_{2n} \\ \cdots \\ a_{mn} \end{pmatrix} \cdot x_n \right).$$

Tedy

$$y^T(Ax) = \tag{5}$$

$$\begin{aligned} &= y_1 \cdot (a_{11} x_1 + a_{12} x_2 + \cdots + a_{1n} x_n) + \\ &+ y_2 \cdot (a_{21} x_1 + a_{22} x_2 + \cdots + a_{2n} x_n) + \\ &\ldots\ldots \\ &+ y_m \cdot (a_{m1} x_1 + a_{m2} x_2 + \cdots + a_{mn} x_n)\,. \end{aligned}$$

Z rovností (4) a (5) vidíme, že
$(y^T A)x = \sum_{j=1}^{n}(\sum_{i=1}^{m} y_i a_{ij}) \cdot x_j$ a
$y^T(Ax) = \sum_{i=1}^{m} y_j \cdot (\sum_{j=1}^{n} a_{ij} x_j)$.
V obou případech (standardními úpravami pro sčítání a násobení) dostáváme $\sum_{i=1}^{m}\sum_{j=1}^{n} y_i a_{ij} x_j$; tedy $(y^T A)x = y^T(Ax)$.
Teď si ještě všimneme:

- Z rovnice (4) ihned plyne, že pokud $(y_1 a_{1j} + y_2 a_{2j} + \cdots + y_m a_{mj}) \geq c_j$ a $x_j \geq 0$ pro vš. $j = 1, 2, \ldots, n$, tak $y^T Ax \geq c^T x$ (kde $c^T x = c_1 x_1 + c_2 x_2 + \cdots + c_n x_n$).
- Z rovnice (5) plyne, že pokud $y_i \geq 0$ a $(a_{i1} x_1 + a_{i2} x_2 + \cdots + a_{in} x_n) \leq b_i$ pro vš. $i = 1, 2, \ldots, m$, tak $y^T Ax \leq y^T b$ (kde $y^T b = y_1 b_1 + y_2 b_2 + \cdots + y_m b_m$).

Dokázali jsme tedy následující tvrzení.

Tvrzení 1 *(Slabá dualita) Nechť x^* je přípustným řešením problému $Ax \leq b, x \geq 0, \max c^T x$ a y^* je přípustným řešením problému $y^T A \geq c^T, y \geq 0, \min y^T b$. Pak $c^T x^* \leq (y^*)^T b$.*
Když platí $c^T x^* = (y^*)^T b$, tak x^* a y^* jsou optimálními řešeními pro své příslušné problémy.

Pokud nám tedy někdo dodá např. řešení $x^* = (8, 4, 0)$ a $y^* = (0, \frac{1}{6}, \frac{2}{3})$ u našich konkrétních problémů (2) a (3), pak zjištěním, že obě splňují své omezující podmínky a navíc $c^T x^* = 3 \cdot 8 + 1 \cdot 4 + 2 \cdot 0 = 28$ a $(y^*)^T b = 0 \cdot 30 + \frac{1}{6} \cdot 24 + \frac{2}{3} \cdot 36 = 28$, máme ověřeno, že x^* i y^* jsou optimální (ve svých příslušných problémech).

> Připomněli jsme si, že když nám někdo ukáže tok v síti a zároveň řez, jehož kapacita se rovná velikosti toku, víme, že se jedná o maximální tok a minimální řez. Podobně, když nám někdo pro graf G dodá párování S a vrcholové pokrytí C, kde $|S| = |C|$, tak se jedná o maximální párování a minimální vrcholové pokrytí.

Řekneme, že problém $Ax \leq b, x \geq 0, \max c^T x$ pro konkrétní A, b, c je

a/ *řešitelný* [či konzistentní] (feasible, F), když existuje přípustné řešení, tedy x takové, že $Ax \leq b, x \geq 0$; zde rozlišíme dva podpřípady:

  i/ *řešitelný omezený* (feasible bounded, FB), když existuje $d \in \mathbb{R}$ tak, že $c^T x \leq d$ pro každé přípustné x,

  ii/ *řešitelný neomezený* (feasible unbounded, FU), když pro každé $d \in \mathbb{R}$ existuje přípustné x splňující $c^T x > d$.

b/ *neřešitelný* [či nekonzistentní] (infeasible, I), když neexistuje přípustné řešení.

Analogicky rozčleníme minimalizační problémy. Slabá dualita mj. implikuje, že když je standardní (primální) úloha P řešitelná neomezená (FU), tak k ní duální úloha D je neřešitelná (a naopak).

Teprve později se dostaneme k důkazu následující věty:

Věta 2 *(Silná dualita) Když je konkrétní úloha lineárního programování řešitelná omezená (FB), tak její duální úloha je také řešitelná omezená (FB), obě úlohy mají optimální řešení a hodnoty příslušných účelových funkcí pro ta optimální řešení se rovnají.*

Když tedy je např. úloha $Ax \leq b, x \geq 0, \max c^T x$ řešitelná a omezená, tak má optimální řešení x^* a duální úloha $y^T A \geq c^T, y \geq 0, \min y^T b$ má (optimální) řešení y^* takové, že $c^T x^* = (y^*)^T b$.

Z vět o dualitě plyne, že když označíme jako P konkrétní (primální) úlohu a D k ní duální úlohu, tak mohou nastávat pouze čtyři případy:

	1	2	3	4
P	FB	FU	I	I
D	FB	I	FU	I

V případě 1 (FB,FB) se navíc optimální hodnoty účelových funkcí rovnají (a jsou dosaženy pro konkrétní vektory x^*, y^*).

> Větu 2 bychom mohli dokázat např. užitím tzv. Fourierovy-Motzkinovy eliminace. To by nám ale ještě nedalo algoritmický návod k řešení úloh LP. My teď zabijeme "dvě mouchy jednou ranou" tím, že ukážeme tzv. simplexový algoritmus ...
> Předem je užitečné zamyslet se nad tím, že když má např. konkrétní úloha $Ax \leq b, x \geq 0, \max c^T x$ optimální řešení, tedy příslušnou lineární kombinaci sloupců matice A, tak má i takové optimální řešení, které s nenulovými (tedy pozitivními) koeficienty kombinuje nějakou množinu nezávislých sloupců matice A.

Konkrétní běh simplexového algoritmu. Vraťme se k řešení naší konkrétní úlohy (2).

Přidáním tzv. *doplňkové proměnné* (slack variable, česky též přídatná proměnná, volná proměnná, proměnná zachycující volnost, rozdílová proměnná apod.) ke každé nerovnici změníme nerovnice na rovnice; v našem případě přidáme x_4, x_5, x_6. Zároveň zavedeme speciální proměnnou z, která bude zachycovat hodnotu účelové funkce. Místo hledání trojice (x_1, x_2, x_3) nezáporných hodnot, které splňují nerovnosti v (2) a pro něž bude $3x_1 + x_2 + 2x_3$ maximální, řešíme tedy následující ekvivalentní úlohu (promyslete si tu ekvivalenci): hledáme konkrétní vektor (sedmici reálných čísel) $(x_1, x_2, x_3, x_4, x_5, x_6, z)$ splňující rovnice

$$\begin{array}{rcrcrcrcrcrcrcl} x_1 & + & x_2 & + & 3x_3 & + & x_4 & & & & & & & = & 30 \\ 2x_1 & + & 2x_2 & + & 5x_3 & & & + & x_5 & & & & & = & 24 \\ 4x_1 & + & x_2 & + & 2x_3 & & & & & + & x_6 & & & = & 36 \\ -3x_1 & - & x_2 & - & 2x_3 & & & & & & & + & z & = & 0 \end{array} \tag{6}$$

přičemž $x_1, x_2, x_3, x_4, x_5, x_6 \geq 0$ a z je maximální možné. (Pro z nezápornost obecně nevyžadujeme; v našem příkladu je $c \geq 0$, proto z vychází také nezáporné.) Soustavu (6) můžeme stručně reprezentovat maticí (s označenými sloupci) znázorněnou následující tabulkou:

x_1	x_2	x_3	x_4	x_5	x_6	z	
1	1	3	1	0	0	0	30
2	2	5	0	1	0	0	24
$\boxed{4}$	1	2	0	0	1	0	36
-3	-1	-2	0	0	0	1	0

(7)

Rámečku kolem prvku a_{31} (tedy prvku v 3. řádku a 1. sloupci) si teď nevšímejme. Všimněme si ale jednotkové matice 3×3 tvořenou sloupci u proměnných x_4, x_5, x_6, když pomineme poslední řádek matice. Při uvažování i posledního řádku vidíme jednotkovou matici 4×4,

když uvažujeme i sloupec u z (který se následujícími úpravami nebude měnit). Proměnným x_4, x_5, x_6 říkáme v této reprezentaci *bázové proměnné* (basic variables, česky také základní proměnné, ale snažíme se zdůraznit vztah k bázi). Proměnné x_1, x_2, x_3 jsou teď *nebázové*. Máme tedy teď (B...bázové, N...nebázové), v "0-té iteraci",

$$B_0 = \{x_4, x_5, x_6\} \text{ a } N_0 = \{x_1, x_2, x_3\}.$$

Soustavu (6) můžeme také zapsat ekvivalentně takto:

$$\begin{array}{rcrcrcrcr} x_4 & = & 30 & - & x_1 & - & x_2 & - & 3x_3 \\ x_5 & = & 24 & - & 2x_1 & - & 2x_2 & - & 5x_3 \\ x_6 & = & 36 & - & 4x_1 & - & x_2 & - & 2x_3 \\ z & = & 0 & + & 3x_1 & + & x_2 & + & 2x_3 \end{array} \tag{8}$$

Explicitně tím zdůrazňujeme, že hodnoty bázových proměnných (x_4, x_5, x_6) a hodnota proměnné z jsou jednoznačně určeny hodnotami nebázových proměnných (x_1, x_2, x_3). Tzv. *bázové řešení* (basic solution) dostaneme, když položíme nebázové proměnné rovny nule; zde tedy je bázovým řešením vektor $(x_1, x_2, x_3, x_4, x_5, x_6, z) = (0, 0, 0, 30, 24, 36, 0)$. Protože v našem případě máme $b = (30, 24, 36) \geq 0$, dostáváme přípustné řešení, tedy hodnoty všech x_i jsou nezáporné. (Z technických důvodů zde nepíšeme vektory sloupcově, ač bychom měli.)

> Později uvidíme, co dělat, když některá b_i jsou záporná a bázové řešení tak není přípustným řešením.
> Jinak také poznamenejme, že jako bázové řešení bychom striktně vzato měli označit jen $(x_1, x_2, x_3, x_4, x_5, x_6) = (0, 0, 0, 30, 24, 36)$; jemu pak přísluší hodnota účelové funkce, v našem případě 0. Zde ale bereme hodnotu z jako součást bázového řešení.

Z rovnice pro z v (8) vidíme, že např. zvětšením x_1 (vzhledem k bázovému řešení, kde $x_1 = x_2 = x_3 = 0$) zvětšíme hodnotu z, o jejíž maximalizaci nám jde. Při zvětšení x_1 musíme dát pozor, abychom stále měli přípustné řešení. Z rovnice pro x_4 v (8) je zřejmé, že x_1 nemůžeme zvětšit víc než o 30, rovnice pro x_5 dává omezení 12 ($= \frac{24}{2}$) a rovnice pro x_6 dává omezení 9 ($= \frac{36}{4}$). Řídíme se nejtvrdším omezením, v našem případě je minimum z omezení 9, což odpovídá proměnné x_6. Zajímáme se tedy o (přípustné) bázové řešení vzhledem k

$$B_1 = \{x_1, x_4, x_5\} \text{ a } N_1 = \{x_2, x_3, x_6\}.$$

Bázi tedy opouští x_6 a vstoupí do ní x_1. To je znázorněno rámečkem okolo prvku a_{31} v (7); v 3. řádku má totiž z bázových proměnných jedničku x_6 a x_1 označuje 1. sloupec. Rovnici pro x_6 v (8) přepíšeme jako $x_1 = -\frac{1}{4}x_2 - \frac{2}{4}x_3 - \frac{1}{4}x_6 + \frac{36}{4}$, tedy

$$x_1 = 9 - \frac{1}{4}x_2 - \frac{1}{2}x_3 - \frac{1}{4}x_6,$$

a pravou stranu pak dosadíme za x_1 v ostatních rovnicích. Soustavu (8) tedy můžeme ekvivalentně zapsat takto:

$$\begin{array}{rcrcrcrcr} x_4 & = & 30 & -(9 - \frac{1}{4}x_2 - \frac{1}{2}x_3 - \frac{1}{4}x_6) & - & x_2 & - & 3x_3 \\ x_5 & = & 24 & -2(9 - \frac{1}{4}x_2 - \frac{1}{2}x_3 - \frac{1}{4}x_6) & - & 2x_2 & - & 5x_3 \\ x_1 & = & & 9 - \frac{1}{4}x_2 - \frac{1}{2}x_3 - \frac{1}{4}x_6 & & & & \\ z & = & 0 & +3(9 - \frac{1}{4}x_2 - \frac{1}{2}x_3 - \frac{1}{4}x_6) & + & x_2 & + & 2x_3 \end{array} \tag{9}$$

Úpravou rovnic upravíme i matici (7). To je totéž, jako když použijeme následující kroky známé z Gaussovy eliminační metody. (Vzpomeňte si, že vynásobením [obou stran] rovnice nenulovým číslem se množina řešení nezmění; množina řešení se také nezmění, nahradíme-li jednu rovnici tak, že od ní odečteme násobek jiné rovnice.) Zarámečkované $a_{31} = 4$ nazveme *pivot* a matici upravíme tak, aby jednotkovou matici nově vytvořily sloupce u x_4, x_5, x_1 a z. Hodnotu pivotu (je nenulová!) označíme p, řádek a sloupec pivotu označíme r (row) a c (column, nepleťme si s vektorem c); máme tedy $a_{rc} = p$, v našem případě $a_{31} = 4$.

Nejprve vydělme řádek r pivotem; dostaneme tedy $\hat{d}_{rj} = \frac{d_{rj}}{p}$ pro každý sloupec j; obecně symbol d_{ij} značí hodnotu v tabulce před úpravou (v i-tém řádku a j-tém sloupci), symbol $\hat{d}_{ij}$ pak hodnotu po úpravě. Dostaneme tedy

x_1	x_2	x_3	x_4	x_5	x_6	z		
1	1	3	1	0	0	0	|	30
2	2	5	0	1	0	0	|	24
$\boxed{1}$	$\frac{1}{4}$	$\frac{1}{2}$	0	0	$\frac{1}{4}$	0	|	9
-3	-1	-2	0	0	0	1	|	0

(10)

Nyní od 1. řádku odečteme d_{1c}-násobek nového řádku r (pokud $r \neq 1$), tedy $\hat{d}_{1j} = d_{1j} - d_{1c}\frac{d_{rj}}{p}$ (v našem případě odečítáme od 1. řádku původní 3. řádek vynásobený $\frac{1}{4}$); dostaneme:

x_1	x_2	x_3	x_4	x_5	x_6	z		
0	$\frac{3}{4}$	$\frac{5}{2}$	1	0	$-\frac{1}{4}$	0	|	21
2	2	5	0	1	0	0	|	24
$\boxed{1}$	$\frac{1}{4}$	$\frac{1}{2}$	0	0	$\frac{1}{4}$	0	|	9
-3	-1	-2	0	0	0	1	|	0

(11)

Obecně pro $i \neq r$ máme $\hat{d}_{ij} = d_{ij} - d_{ic}\frac{d_{rj}}{p}$; po patřičné úpravě 2. řádku dostáváme

x_1	x_2	x_3	x_4	x_5	x_6	z		
0	$\frac{3}{4}$	$\frac{5}{2}$	1	0	$-\frac{1}{4}$	0	|	21
0	$\frac{3}{2}$	4	0	1	$-\frac{1}{2}$	0	|	6
$\boxed{1}$	$\frac{1}{4}$	$\frac{1}{2}$	0	0	$\frac{1}{4}$	0	|	9
-3	-1	-2	0	0	0	1	|	0

(12)

Analogicky upravíme 4. řádek a dostáváme

x_1	x_2	x_3	x_4	x_5	x_6	z		
0	$\frac{3}{4}$	$\frac{5}{2}$	1	0	$-\frac{1}{4}$	0	|	21
0	$\frac{3}{2}$	4	0	1	$-\frac{1}{2}$	0	|	6
1	$\frac{1}{4}$	$\frac{1}{2}$	0	0	$\frac{1}{4}$	0	|	9
0	$-\frac{1}{4}$	$-\frac{1}{2}$	0	0	$\frac{3}{4}$	1	|	27

(13)

Soustava (9), a tedy i (8), je tak ekvivalentní soustavě

$$\begin{array}{rcrcrcrcr} x_4 & = & 21 & - & \frac{3}{4}x_2 & - & \frac{5}{2}x_3 & + & \frac{1}{4}x_6 \\ x_5 & = & 6 & - & \frac{3}{2}x_2 & - & 4x_3 & + & \frac{1}{2}x_6 \\ x_1 & = & 9 & - & \frac{1}{4}x_2 & - & \frac{1}{2}x_3 & - & \frac{1}{4}x_6 \\ z & = & 27 & + & \frac{1}{4}x_2 & + & \frac{1}{2}x_3 & - & \frac{3}{4}x_6 \end{array} \tag{14}$$

Bázové řešení je zde $(x_1, x_2, x_3, x_4, x_5, x_6, z) = (9, 0, 0, 21, 6, 0, 27)$. Je vidět, že např. zvýšení x_3 z nuly může pomoci zvýšit z. Nabízí se tedy vyměnit x_3 za některou proměnnou v bázi. Než to uděláme, udělejme si jisté zjednodušení naší notace. Když máme určeny bázové proměnné v jistém pořadí, např. na začátku se jedná o (x_4, x_5, x_6), zajímá nás tvar matice, kde sloupce u x_4, x_5, x_6 a z, v daném pořadí, vytvoří jednotkovou matici. Odkazujeme-li k takové matici, např. k (7), domluvíme se na stručnější notaci; matici (7) zapíšeme stručněji takto:

	x_1	x_2	x_3		
x_4	1	1	3		30
x_5	2	2	5		24
x_6	$\boxed{4}$	1	2		36
z	-3	-1	-2		0

(15)

Proměnnými označujícími řádky rozumíme aktuální bázové proměnné (a z); původní tvar bychom dostali, kdybychom přidali příslušnou jednotkovou matici. K prvkům matice odkazujeme jako k a_{ij}, vyjma posledního řádku a a posledního sloupce. K prvkům posledního sloupce odkazujeme jako k b_i (kromě posledního), k prvkům posledního řádku jako k $-c_j$ (kromě posledního). Prvek posledního řádku a posledního sloupce (tedy v pravém dolním rohu) označíme v (value, hodnota účelové funkce). Toto značení pro jednoduchost používáme i po úpravách, kdy už se jedná o jiné hodnoty než jsou původní A, b, c. *Pivot jsme zvolili takto*:

i/ vybrali jsme jeden sloupec, kde $c_j > 0$ (tedy $-c_j < 0$); j bude sloupcem pivotu;

ii/ našli jsme řádky i, kde $a_{ij} > 0$, a spočetli u nich $\frac{b_i}{a_{ij}}$; jako řádek pivotu jsme vzali (některé) i, kde $\frac{b_i}{a_{ij}}$ je minimální.

> Jeden úkol žádá, ať vysvětlíte, proč je úloha neomezená (feasible unbounded), když ve všech řádcích máme $a_{ij} \leq 0$.

V našem případě jsme prohodili x_1 a x_6. Dostali jsme matici (13), ve stručné podobě zapsané

	x_6	x_2	x_3		
x_4	$-\frac{1}{4}$	$\frac{3}{4}$	$\frac{5}{2}$		21
x_5	$-\frac{1}{2}$	$\frac{3}{2}$	4		6
x_1	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{2}$		9
z	$\frac{3}{4}$	$-\frac{1}{4}$	$-\frac{1}{2}$		27

(16)

Nyní vyberme ke vstupu do báze x_3, protože $-c_3$ ve sloupci označeném x_3 je záporné; nový pivot bude tedy ve sloupci 3. Řádky v 3. sloupci dávají omezení postupně $\frac{21}{5/2} = \frac{42}{5}$, $\frac{6}{4} = \frac{3}{2}$, $\frac{9}{1/2} = 18$. Minimum je dosaženo v 2. řádku, proto prohodíme x_3 (vstoupí do báze) a x_5 (opustí bázi); nový pivot bude v 2. řádku. Budeme tedy mít

$$B_2 = \{x_4, x_3, x_1\} \text{ a } N_2 = \{x_6, x_2, x_5\}.$$

Označme si pivot:

	x_6	x_2	x_3		
x_4	$-\frac{1}{4}$	$\frac{3}{4}$	$\frac{5}{2}$		21
x_5	$-\frac{1}{2}$	$\frac{3}{2}$	$\boxed{4}$		6
x_1	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{2}$		9
z	$\frac{3}{4}$	$-\frac{1}{4}$	$-\frac{1}{2}$		27

(17)

Máme tedy $a_{rc} = p$, v našem případě $a_{23} = 4$. Snadno ověříme, že naše úpravy výše se dají schématicky zachytit takto (kde d_{ij} znamená dosavadní hodnotu v i-tém řádku a j-tém sloupci a $\hat{d}_{ij}$ označuje novou):

i/ $\boxed{\hat{d}_{rc} = \frac{1}{p}}$ (na místě pivotu se objeví jeho převrácená hodnota);

ii/ pro $j \neq c$: $\boxed{\hat{d}_{rj} = d_{rj}/p}$ (hodnota v řádku pivotu je vydělena pivotem);

iii/ pro $i \neq r$: $\boxed{\hat{d}_{ic} = -d_{ic}/p}$ (hodnota v sloupci pivotu je vydělena pivotem a znaménko se otočí);

iv/ pro $i \neq r$ a $j \neq c$: $\boxed{\hat{d}_{ij} = d_{ij} - d_{rj}d_{ic}/p}$ (od hodnoty mimo řádek a sloupec pivotu se odečte součin hodnot v příslušném řádku sloupce pivotu a příslušném sloupci řádku pivotu vydělený pivotem).

Provedeme-li to u matice (17), hodnoty se změní takto (ověřte); pro pohodlí opakujeme matici (17) jako matici (18) a výsledek po "pivotování" je zachycen v matici (19):

	x_6	x_2	x_3	
x_4	$-\frac{1}{4}$	$\frac{3}{4}$	$\frac{5}{2}$	21
x_5	$-\frac{1}{2}$	$\frac{3}{2}$	$\boxed{4}$	6
x_1	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{2}$	9
z	$\frac{3}{4}$	$-\frac{1}{4}$	$-\frac{1}{2}$	27

(18)

	x_6	x_2	x_5	
x_4	$\frac{1}{16}$	$-\frac{3}{16}$	$-\frac{5}{8}$	$\frac{69}{4}$
x_3	$-\frac{1}{8}$	$\frac{3}{8}$	$\frac{1}{4}$	$\frac{3}{2}$
x_1	$\frac{5}{16}$	$\frac{1}{16}$	$-\frac{1}{8}$	$\frac{33}{4}$
z	$\frac{11}{16}$	$-\frac{1}{16}$	$\frac{1}{8}$	$\frac{111}{4}$

(19)

Zde je tedy bázovým řešením $(x_1, x_2, x_3, x_4, x_5, x_6, z) = (\frac{33}{4}, 0, \frac{3}{2}, \frac{69}{4}, 0, 0, 27.75)$.

Nyní lze pro přechod do báze zvažovat pouze x_2 (se záporným $-c_j$). Protože $\frac{3/2}{3/8} < \frac{33/4}{1/16}$, bázi opustí x_3 (ač už mimo bázi byla). Budeme mít tedy

$$B_3 = \{x_4, x_2, x_1\} \text{ a } N_2 = \{x_6, x_3, x_5\}.$$

Úpravou podle pivotu v 2. řádku a 2. sloupci dostaneme

	x_6	x_3	x_5	
x_4	0	$\frac{1}{2}$	$-\frac{1}{2}$	18
x_2	$-\frac{1}{3}$	$\frac{8}{3}$	$\frac{2}{3}$	4
x_1	$\frac{1}{3}$	$-\frac{1}{6}$	$-\frac{1}{6}$	8
z	$\frac{2}{3}$	$\frac{1}{6}$	$\frac{1}{6}$	28

(20)

Původní soustavu (6) jsme tedy převedli na ekvivalentní soustavu (mající stejné sedmice $(x_1, x_2, x_3, x_4, x_5, x_6, x_7, z)$ jako řešení) v tomto tvaru:

$$\begin{array}{rcl} x_4 &=& 18 \quad\quad - \frac{1}{2}x_3 + \frac{1}{2}x_5 \\ x_2 &=& 4 + \frac{1}{3}x_6 - \frac{8}{3}x_3 - \frac{2}{3}x_5 \\ x_1 &=& 8 - \frac{1}{3}x_6 + \frac{1}{6}x_3 + \frac{1}{6}x_5 \\ z &=& 28 - \frac{2}{3}x_6 - \frac{1}{6}x_3 - \frac{1}{6}x_5 \end{array} \tag{21}$$

Přitom bázové řešení $(x_1, x_2, x_3, x_4, x_5, x_6, z) = (8, 4, 0, 18, 0, 0, 28)$ je přípustné a z nemůže být očividně větší pro jakékoli jiné přípustné řešení. (Proč?) Našli jsme tedy optimum!

Když jsme tedy dosáhli situace, kdy nejen poslední sloupec, ale i poslední řádek je nezáporný (s případnou výjimkou pravého dolního rohu) [k tomu došlo v tabulce 20], tak už nás "vnitřek nezajímá"; k přečtení optimálního řešení nám stačí

	x_6	x_3	x_5	
x_4				18
x_2				4
x_1				8
	$\frac{2}{3}$	$\frac{1}{6}$	$\frac{1}{6}$	28

(22)

Optimální řešení jsme sestavili takto: položili jsme (nebázové) $x_3 = x_5 = x_6 = 0$ a hodnoty bázových jsme si přečetli v pravém sloupci: $x_1 = 8$, $x_2 = 4$, $x_4 = 18$. V původním problému (2) máme tedy optimální řešení

$$x^* = (x_1^*, x_2^*, x_3^*) = (8, 4, 0), \text{ s hodnotou účelové funkce } 3x_1^* + x_2^* + 2x_3^* = 3 \cdot 8 + 4 = 28.$$

Nerovnosti jsme splnili takto

$x_1^* + x_2^* + 3x_3^* = 12 \leq 30$ (s rezervou 18 [slack $x_4 = 18$]),

$2x_1^* + 2x_2^* + 5x_3^* = 24 \leq 24$ (bez rezervy, tj. rezerva 0),

$4x_1^* + x_2^* + 2x_3^* = 36 \leq 36$ (bez rezervy).

Zároveň jsme (nějakým zázrakem?) z (22) přečetli i optimální řešení duálního problému (3): podívali jsme se na

x_4 (přídatná proměnná pro 1. řádek) jako na y_1,

na x_5 (přídatná proměnná pro 2. řádek) jako na y_2

a na x_6 (přídatná proměnná pro 3. řádek) jako na y_3.

Tabulku (22) jsme tedy přepsali do tvaru

	y_3	x_3	y_2	
y_1				18
x_2				4
x_1				8
	$\frac{2}{3}$	$\frac{1}{6}$	$\frac{1}{6}$	28

(23)

Položili jsme $y_1^* = 0$ (neboť jsme $y_1 = x_4$ nalezli v levém sloupci), a y_2^*, y_3^* jsme si přečetli dole v sloupcích odpovídajících $y_2 = x_5, y_3 = x_6$: tedy

$$y^* = (y_1^*, y_2^*, y_3^*) = (0, \tfrac{1}{6}, \tfrac{2}{3}), \text{ s hodnotou účelové funkce } y^* b = \tfrac{1}{6}24 + \tfrac{2}{3}36 = 28.$$

V souladu s naším značením pro vektory bychom správně měli psát $(y^*)^T = (y_1^*, y_2^*, y_3^*)$, $(y^*)^T b$, ale k nedorozumění by v takových případech nemělo docházet.

Nerovnosti v (3) jsme splnili takto:

$y_1^* + 2y_2^* + 4y_3^* = 3 \geq 3$ (bez rezervy),

$y_1^* + 2y_2^* + y_3^* = 1 \geq 1$ (bez rezervy),

$3y_1^* + 5y_2^* + 2y_3^* = \frac{13}{6} \geq 2$ (s rezervou).

Příště mj. prodiskutujeme, zda v našem případě došlo k ojedinělým "zázrakům" či k obecně platným zákonitostem ... Můžete zkusit zformulovat hypotézu o vztahu nulových složek optimálních řešení a plnění nerovnic s rezervou a bez rezervy ...

Nové úkoly.

1. Diskutovali jsme, jak obecný problém LP, kdy máme omezení ve formě nerovnic i rovnic a na některé proměnné klademe podmínku nezápornosti zatímco na jiné ne, lze snadno převést do tvaru standardního maximalizačního, či standardního minimalizačního, problému. Ještě si to promyslete.

2. Vysvětlete, že pro řešitelnou neomezenou úlohu musí být duální úloha neřešitelná.

3. Ukažte příklad problému LP, který je neřešitelný a pro nějž je i jeho duální problém neřešitelný.

4. Aplikujte simplexový algoritmus na náš dietní problém.

5. Když v simplexovém algoritmu narazíme na případ $-c_j < 0$ a $a_{ij} \leq 0$ pro všechny řádky i, úloha je neomezená; vysvětlete proč.

6. Už jste si vyzkoušeli nějaký (volně dostupný) LP solver?

Týden 8 (od 28.3.)

(V týdnu odpadlo pondělní setkání kvůli velikonocům.)

Nejdříve jsme prošli dřívější úkoly; některé jsme řešili až v průběhu výkladu.

Simplexový algoritmus (dokončení).

Dořešíme několik věcí, které v náčrtu algoritmu minule nebyly dojasněny.

Simplexový algoritmus vyřeší zároveň duální problém. Připomeňme, že tabulkou typu

$$\begin{array}{c|cccc|c} & s_1 & s_2 & \cdots & s_n & \\ \hline t_1 & a_{11} & a_{12} & \cdots & a_{1n} & b_1 \\ t_2 & a_{21} & a_{22} & \cdots & a_{2n} & b_2 \\ & & \cdots & & & \\ t_m & a_{m1} & a_{m2} & \cdots & a_{mn} & b_m \\ \hline z & -c_1 & -c_2 & \cdots & -c_n & v \end{array} \tag{24}$$

jsme reprezentovali soustavu rovnic, kterou níže uvádíme ve dvou verzích:

$$\begin{array}{cc} t_1 + a_{11}s_1 + a_{12}s_2 + \cdots + a_{1n}s_n = b_1 & (t_1 = b_1 - a_{11}s_1 - a_{12}s_2 - \cdots - a_{1n}s_n) \\ t_2 + a_{21}s_1 + a_{22}s_2 + \cdots + a_{2n}s_n = b_2 & (t_2 = b_2 - a_{21}s_1 - a_{22}s_2 - \cdots - a_{2n}s_n) \\ \cdots & \\ t_m + a_{m1}s_1 + a_{m2}s_2 + \cdots + a_{mn}s_n = b_m & (t_m = b_m - a_{m1}s_1 - a_{m2}s_2 - \cdots - a_{mn}s_n) \\ z - c_1s_1 - c_2s_2 - \cdots - c_ns_n = v & (z = v + c_1s_1 + c_2s_2 + \cdots + c_ns_n) \end{array} \tag{25}$$

Řešením této soustavy rozumíme jakoukoli $(n+m+1)$-tici reálných čísel, které po dosazení postupně do proměnných $s_1, \ldots, s_n, t_1, \ldots, t_m, z$ splňují všechny rovnice. Na řešení *sol* (solution) je vhodné se dívat jako na zobrazení typu

$$sol : \{s_1, \ldots, s_n, t_1, \ldots, t_m, z\} \to \mathbb{R}.$$

Řešení nazýváme přípustným, jestliže hodnoty přiřazené proměnným $s_1, \ldots, s_n, t_1, \ldots, t_m$ jsou nezáporné. Optimální řešení je takové přípustné řešení, při němž je (hodnota přiřazená proměnné) z maximální. Jde nám tedy o řešení soustavy

$$t + As = b, z = v + c^T s,$$

kde t_i a s_j jsou nezáporné a z je maximální možné.

Pro úlohu $Ax \leq b, x \geq 0, \max c^T x$ na začátku (v "nulté" iteraci) sestavíme tabulku (24) tak, že symboly pro proměnné splňují $(s_1, \ldots, s_n) = (x_1, \ldots, x_n)$ a $(t_1, \ldots, t_m) = (y_1, \ldots, y_m)$, kde y_i reprezentují přídatné (či doplňkové) proměnné (slack variables); přitom symboly a_{ij}, b_i, c_j odpovídají zadání A, b, c a $v = 0$. V dalších iteracích (vždy po operaci odpovídající vybranému pivotu) jsou pak symboly x_j a y_i různě prohazovány (mezi řádkem nahoře a sloupcem vlevo) a hodnoty na místech odpovídajících a_{ij}, b_i, $-c_j$, v se příslušně mění. Přitom se ovšem *nemění množina řešení* soustav rovnic odpovídajících postupně vytvářeným tabulkám; připomeňme, že řešení zde stále chápeme jako zobrazení typu

$$\{x_1, \ldots, x_n, y_1, \ldots, y_m, z\} \to \mathbb{R}.$$

Aplikací operace "pivotování" podle pivotu $p = a_{rc} \neq 0$ obdržíme z tabulky (24) tabulku

$$\begin{array}{c|cccc|c} & \hat{s}_1 & \hat{s}_2 & \cdots & \hat{s}_n & \\ \hline \hat{t}_1 & \hat{a}_{11} & \hat{a}_{12} & \cdots & \hat{a}_{1n} & \hat{b}_1 \\ \hat{t}_2 & \hat{a}_{21} & \hat{a}_{22} & \cdots & \hat{a}_{2n} & \hat{b}_2 \\ & & \cdots & & & \\ \hat{t}_m & \hat{a}_{m1} & \hat{a}_{m2} & \cdots & \hat{a}_{mn} & \hat{b}_m \\ \hline z & -\hat{c}_1 & -\hat{c}_2 & \cdots & -\hat{c}_n & \hat{v} \end{array} \tag{26}$$

kde pro symboly proměnných máme

$(\hat{t}_1, \ldots, \hat{t}_m) = (t_1, \ldots, t_{r-1}, s_c, t_{r+1}, \ldots, t_m)$ a $(\hat{s}_1, \ldots, \hat{s}_n) = (s_1, \ldots, s_{c-1}, t_r, s_{c+1}, \ldots, s_n)$; symboly pro proměnné v řádku pivotu a ve sloupci pivotu se prohodily (a jinak zůstaly tyto symboly beze změny). Číselné hodnoty a_{ij}, b_i, $-c_j$, v (označené obecně d_{ij}) se změnily podle (již dříve uvedených) pravidel

$\boxed{\hat{d}_{rc} = \frac{1}{p}}$,

$\boxed{\hat{d}_{rj} = d_{rj}/p}$ (pro $j \neq c$),

$\boxed{\hat{d}_{ic} = -d_{ic}/p}$ (pro $i \neq r$),

$\boxed{\hat{d}_{ij} = d_{ij} - d_{rj}d_{ic}/p}$ pro $i \neq r$ a $j \neq c$.

Odvodili jsme si, že tímto "pivotováním" se množina řešení soustavy nemění. (Když konkrétní přiřazení $sol : \{s_1, \ldots, s_n, t_1, \ldots, t_m, z\} \to \mathbb{R}$ splňuje soustavu rovnic (25), pak splňuje i soustavu odpovídající tabulce (26), a naopak.)

Na tabulku (24) se ovšem můžeme podívat i jinak, totiž "duálně", tedy nikoli jako na reprezentaci úlohy

$$t + As = b,\ \max z = v + c^T s$$

(na začátku tedy $y + Ax = b$, $\max z = c^T x$), ale jako na reprezentaci úlohy $-s^T + t^T A = c^T$, $\min z' = v + t^T b$, kterou raději napíšeme ve tvaru

$$s + (-A)^T t = -c,\ \max -z' = -v - b^T t.$$

Takže tabulka (24) má ještě asociovanou tabulku

$$\begin{array}{c|cccc|c} & t_1 & t_2 & \cdots & t_m & \\ \hline s_1 & -a_{11} & -a_{21} & \cdots & -a_{m1} & -c_1 \\ s_2 & -a_{12} & -a_{22} & \cdots & -a_{m2} & -c_2 \\ & & \cdots & & & \\ s_n & -a_{1n} & -a_{2n} & \cdots & -a_{mn} & -c_n \\ \hline -z' & b_1 & b_2 & \cdots & b_m & -v \end{array} \tag{27}$$

kterou interpretujeme jako předtím; reprezentuje tedy soustavu $s + (-A)^T t = -c$, $-z' = -v - b^T t$, detailněji tedy

$$\begin{array}{c} s_1 - a_{11}t_1 - a_{21}t_2 - \cdots - a_{m1}t_m = -c_1 \\ s_2 - a_{11}t_1 - a_{22}t_2 - \cdots - a_{m2}t_m = -c_2 \\ \cdots \\ s_m - a_{m1}t_1 - a_{m2}t_2 - \cdots - a_{mn}t_m = -c_n \\ -z' + b_1 + b_2 + \cdots + b_m = -v \end{array} \tag{28}$$

Když jsme "pivotovali" podle pivotu $a_{rc} = p$ v tabulce (24), vznikla tabulka (26). Co se stane, když pivotujeme v tabulce (27), asociované k tabulce (24), podle odpovídajícího pivotu $-a_{rc} = -p$ (který je v tabulce (27) v r-tém sloupci a c-tém řádku)? Ukážeme, že vznikne přesně tabulka, která je asociovaná k (26), tedy tabulka

$$\begin{array}{c|cccc|c} & \hat{t}_1 & \hat{t}_2 & \cdots & \hat{t}_m & \\ \hline \hat{s}_1 & -\hat{a}_{11} & -\hat{a}_{21} & \cdots & -\hat{a}_{m1} & -\hat{c}_1 \\ \hat{s}_2 & -\hat{a}_{12} & -\hat{a}_{22} & \cdots & -\hat{a}_{m2} & -\hat{c}_2 \\ & & \cdots & & & \\ \hat{s}_n & -\hat{a}_{1n} & -\hat{a}_{2n} & \cdots & -\hat{a}_{mn} & -\hat{c}_n \\ \hline -z' & \hat{b}_1 & \hat{b}_2 & \cdots & \hat{b}_m & -\hat{v} \end{array} \tag{29}$$

To teď přímočaře ověříme rozborem případů. (Používáme font "c" pro označení pořadového čísla, ať se odliší od symbolu pro vektor "c".) Pivotovací pravidla nám totiž říkají:

i/ Na místo $\cdot_{\mathtt{rc}}$, tedy na místo pivotu, v tabulce (29) jde o r-tý sloupec a c-tý řádek, dej $\frac{1}{-p}$, což je $-\hat{a}_{\mathtt{rc}}$;

ii/ na místo $\cdot_{\mathtt{r}j}$ ($j \neq \mathtt{c}$, jde o r-tý sloupec a j-tý řádek) dej

a/ v případě $j \leq n$ hodnotu $-\frac{-a_{\mathtt{r}j}}{-p} = -\frac{a_{\mathtt{r}j}}{p} = -\hat{a}_{\mathtt{r}j}$;

b/ v případě $j = n{+}1$ (poslední řádek v (29)) hodnotu $-\frac{b_{\mathtt{r}}}{-p} = \frac{b_{\mathtt{r}}}{p} = \hat{b}_{\mathtt{r}}$;

iii/ na místo $\cdot_{i\mathtt{c}}$ ($i \neq \mathtt{r}$, jde o c-tý řádek a i-tý sloupec)) dej

a/ v případě $i \leq m$ hodnotu $\frac{-a_{i\mathtt{c}}}{-p} = \frac{a_{i\mathtt{c}}}{p} = -\hat{a}_{i\mathtt{c}}$;

b/ v případě $i = m{+}1$ (poslední sloupec v (29)) hodnotu $\frac{-c_{\mathtt{c}}}{-p} = \frac{c_{\mathtt{c}}}{p} = -\hat{c}_{\mathtt{c}}$;

iv/ na místo $\cdot_{ij}$ ($i \neq \mathtt{r}, j \neq \mathtt{c}$) dej

a/ v případě $i \leq m, j \leq n$ hodnotu $-a_{ij} - \frac{-a_{\mathtt{r}j}\cdot(-a_{i\mathtt{c}})}{-p} = -(a_{ij} - \frac{a_{\mathtt{r}j}\cdot a_{i\mathtt{c}}}{p}) = -\hat{a}_{ij}$;

b/ v případě $i = m{+}1$, $j \leq n$ hodnotu $-c_j - \frac{-c_{\mathtt{c}}\cdot(-a_{\mathtt{r}j})}{-p} = -c_j - \frac{-c_{\mathtt{c}}\cdot a_{\mathtt{r}j}}{p} = -\hat{c}_j$;

c/ v případě $i \leq m$, $j = n{+}1$ hodnotu $b_i - \frac{b_{\mathtt{r}}\cdot(-a_{i\mathtt{c}})}{-p} = b_i - \frac{b_{\mathtt{r}}\cdot a_{i\mathtt{c}}}{p} = \hat{b}_i$;

d/ v případě $i = m{+}1$, $j = n{+}1$ hodnotu $-v - \frac{b_{\mathtt{r}}\cdot(-c_{\mathtt{c}})}{-p} = -(v - \frac{b_{\mathtt{r}}\cdot(-c_{\mathtt{c}})}{p}) = -\hat{v}$.

Proto "pivotováním" udržujeme nejen stejnou množinu řešení původní úlohy, ale také se nemění množina řešení soustav rovnic, které odpovídají asociovaným tabulkám.

Když tedy dospějeme k tabulce typu (26), s asociovanou tabulkou (29), kde všechny $\hat{b}_i$ a $-\hat{c}_j$ jsou nezáporné, dostaneme snadno dvě "duální" přípustná řešení:

i/ zobrazení sol_1 definované vztahy

$$sol_1(\hat{s}_j) = 0,\ sol_1(\hat{t}_i) = \hat{b}_i,\ sol_1(z) = \hat{v}$$

je přípustným řešením soustavy $\hat{t} + \hat{A}\hat{s} = \hat{b}$, $z = \hat{v} + (\hat{c})^T\hat{s}$, a tedy

přípustným řešením pro $y + Ax = b$, $z = c^T x$ (v nulté iteraci);

ii/ zobrazení sol_2 definované vztahy

$$sol_2(\hat{t}_i) = 0,\ sol_2(\hat{s}_j) = -\hat{c}_j,\ sol_2(z') = \hat{v}$$

je přípustným řešením soustavy $(\hat{s})^T + (\hat{t})^T(-\hat{A}) = (-\hat{c})^T$, $-z' = -\hat{v} - (\hat{t})^T\hat{b}$,
tedy soustavy $(-\hat{s})^T + (\hat{t})^T\hat{A} = (\hat{c})^T$, $z' = \hat{v} + (\hat{t})^T\hat{b}$, a tedy

přípustným řešením pro $-x + y^T A = c^T$, $z' = y^T b$ (v nulté iteraci).

Když tedy definujeme sol_1' jako restrikci sol_1 na proměnné $x_1, \ldots, x_n, z$ (hodnoty pro přídatné proměnné y nás teď nezajímají), tak máme přípustné řešení soustavy $Ax \leq b$ s hodnotou účelové funkce $c^T x = \hat{v}$. Podobně sol_2' vzniklé restrikcí sol_2 na proměnné $y_1, \ldots, y_m, z'$ je přípustným řešením soustavy $y^T A \geq c^T$ s hodnotou účelové funkce $y^T b = \hat{v}$.

Takže sol_1' je optimálním řešením úlohy $Ax \leq b$, $\max c^T x$ a sol_2' je optimálním řešením duální úlohy $y^T A \geq c$, $\min y^T b$.

Ilustrovali jsme to jiným konkrétním příkladem, konkrétně příkladem 1 z Exercises na konci 4. kapitoly v [2]. Jeden z úkolů žádá řešení dalších příkladů.

Nalezení iniciálního přípustného řešení.

V případě, že řešíme úlohu $Ax \leq b$, $x \geq 0$, $\max c^T x$, kde není $b \geq 0$, tak iniciální přípustné řešení nemusí být nasnadě. (V tom případě také řešení $y + Ax = b$, kde $x = 0$, není nezáporné.)

Vzpomeňme si na problém maximálního toku, když jsme přidali podmínku dolních mezí na protékající množství v jednotlivých úsecích [hranách]; nulový tok zde není obecně přípustný. My jsme nejprve hledali nějaký přípustný tok původní úlohy řešením problému maximálního toku v jiné úloze [bez dolních mezí]. Zde jsme v podobné situaci: k nalezení iniciálního přípustného řešení původní úlohy LP, nebo ke zjištění, že původní úloha přípustné řešení nemá, dospějeme aplikací simplexového algoritmu na trochu jinou úlohu.

Myšlenka je jednoduchá: k proměnným $x_1, x_2, \ldots, x_n$ přidáme (také nezápornou) proměnnou x_0 a místo $Ax \leq b$, $x \geq 0$, $\max c^T x$ budeme nejdříve řešit úlohu

$$A'x' \leq b,\ x' \geq 0,\ \max -x_0,$$

kde $x' = (x_0, x_1, \ldots, x_n)$ a A' vznikne z A přidáním sloupce "minus-jedniček" vlevo. Hledáme tedy nezáporné řešení soustavy

$$\begin{aligned} -x_0 + a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &\leq b_1 \\ -x_0 + a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &\leq b_2 \\ &\cdots \\ -x_0 + a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n &\leq b_m \end{aligned}$$

ve kterém je $-x_0$ co největší, tedy x_0 co nejmenší.

Zde je přípustné řešení nasnadě:
položme $x = (x_1, x_2, \ldots, x_n) = (0, 0, \ldots, 0)$ a $x_0 = -\min\{b_i \mid i \in \{1, 2, \ldots, n\}\}$. Když úlohu vyřešíme a nalezneme optimální řešení $(x_0^*, x_1^*, \ldots, x_n^*)$ [které je samozřejmě nezáporné], tak jsou dvě možnosti:

i/ $x_0^* = 0$ (tedy maximální hodnota účelové funkce $-x_0$ je 0); v tom případě $(x_1^*, \ldots, x_n^*)$ je přípustným řešením pro $Ax \leq b$, $x \geq 0$;

ii/ $x_0^* > 0$ (tedy maximální hodnota účelové funkce $-x_0$ je záporná); v tom případě očividně neexistuje přípustné řešení $Ax \leq b$, $x \geq 0$; původní úloha $Ax \leq b$, $x \geq 0$, $\max c^T x$ tedy vůbec nemá přípustné řešení.

Podívejme se, jak vypadá řešení úlohy v naší verzi simplexového algoritmu. Vezměme opět jeden příklad z [1]:

$$\begin{array}{lll} \text{maximalizuj} & 2x_1 - x_2 & \text{za podmínek} \\ & x_1, x_2 \geq 0, & \\ & \begin{array}{rcrcl} 2x_1 & - & x_2 & \leq & 2, \\ x_1 & - & 5x_2 & \leq & -4. \end{array} & \end{array} \tag{30}$$

Nejprve tedy budeme řešit úlohu

$$\begin{array}{lll} \text{maximalizuj} & -x_0 & \text{za podmínek} \\ & x_0, x_1, x_2 \geq 0, & \\ & \begin{array}{rcrcrcl} -x_0 & + & 2x_1 & - & x_2 & \leq & 2, \\ -x_0 & + & x_1 & - & 5x_2 & \leq & -4. \end{array} & \end{array} \tag{31}$$

Po přidání doplňkových (přídatných) proměnných (slack variables) y_1, y_2 vytvoříme tabulku (nejdříve bez zarámovaného pivotu)

	x_0	x_1	x_2	
y_1	-1	2	-1	2
y_2	-1	1	-5	-4
z	1	0	0	0

(32)

Bázové řešení $(x_0, x_1, x_2, y_1, y_2, z) = (0, 0, 0, 2, -4, 0)$ zde není přípustným, ale to se změní po tomto pivotování: do báze pošleme proměnnou x_0 (bez ohledu na to, že poslední řádek v jejím sloupci není záporný) a bázi opustí některá z proměnných, v jejímž řádku b_i nabývá minima; v našem případě to bude proměnná y_2, a proto jsme zarámečkovali pivot v 2. řádku a 1. sloupci (bez ohledu na to, že je negativní). Po provedení příslušné transformace tabulky dostáváme

	y_2	x_1	x_2	
y_1	-1	1	4	6
x_0	-1	-1	5	4
z	1	1	-5	-4

(33)

a bázové řešení $(x_0, x_1, x_2, y_1, y_2, z) = (4, 0, 0, 6, 2, 0, -4)$ je již přípustné! (Vzpomeňme si, že na hodnotu účelové funkce z podmínku nezápornosti neklademe.)

Pokračujeme již standardně (udržujeme se tedy v přípustných řešeních). V našem případě stačí jen jedna další iterace, protože bázi opouští (sloupec označený) x_2 (v posledním řádku máme $-5 < 0$) a do báze se vrací x_0 (jelikož $\frac{4}{5} < \frac{6}{4}$). Pivot je tedy v 2. řádku a 3. sloupci (hodnota je 5), a po příslušné transformaci dostáváme

	y_2	x_1	x_0	
y_1	$-\frac{1}{5}$	$\frac{9}{5}$	$-\frac{4}{5}$	$\frac{14}{5}$
x_2	$-\frac{1}{5}$	$-\frac{1}{5}$	$\frac{1}{5}$	$\frac{4}{5}$
z	0	0	1	0

(34)

Skončili jsme tedy s optimální hodnotou $x_0 = 0$ a vektor $(x_1, x_2, y_1, y_2) = (0, \frac{4}{5}, \frac{14}{5}, 0)$ je přípustným řešením úlohy (30), když v ní doplníme doplňkové proměnné y_1, y_2. Soustava (30) s doplňkovými proměnnými je zároveň ekvivalentní soustavě

$$y_1 = \tfrac{14}{5} + \tfrac{1}{5}y_2 - \tfrac{9}{5}x_1$$
$$x_2 = \tfrac{4}{5} + \tfrac{1}{5}y_2 + \tfrac{1}{5}x_1$$

kde maximalizovat máme $2x_1 - x_2$. (V soustavě dané tabulkou (34) jsme vypustili poslední řádek a proměnnou x_0.) (Původní) účelovou funkci $2x_1 - x_2$ vyjádříme pomocí aktuálně nebázových proměnných y_2, x_1 takto:

$$2x_1 - x_2 = 2x_1 - (\tfrac{4}{5} + \tfrac{1}{5}y_2 + \tfrac{1}{5}x_1) = -\tfrac{4}{5} - \tfrac{1}{5}y_2 + \tfrac{9}{5}x_1$$

Standardním simplexovým algoritmem budeme tedy teď pokračovat s tabulkou

	y_2	x_1	
y_1	$-\frac{1}{5}$	$\frac{9}{5}$	$\frac{14}{5}$
x_2	$-\frac{1}{5}$	$-\frac{1}{5}$	$\frac{4}{5}$
z	$\frac{1}{5}$	$-\frac{9}{5}$	$-\frac{4}{5}$

(35)

Zamezení možnému zacyklení algoritmu.

Simplexový algoritmus jsme definovali trochu nepřímo a v některých částech jsme nechali určitou volnost: pro pivot jsme mohli zvolit libovolný sloupec j splňující $-c_j < 0$ (pokud jich bylo více) a v něm pak kterýkoli řádek i splňující $a_{ij} > 0$, pro nějž $\frac{b_i}{a_{ij}}$ nabývalo minima. To minimum může být 0 díky $b_i = 0$, takže ne každá iterace vede ke zvětšení hodnoty účelové funkce (ta se nezmenší, ale může několik iterací zůstat stejná). Dají se zkonstruovat příklady, kdy nevhodná posloupnost pivotů vede k cyklu, tedy k navrácení se ke stejné bázi.

Blandovo pravidlo. Jedno z pravidel, které takovému cyklu zamezí se jmenuje Blandovo (*Bland's rule*), také *pravidlo nejmenšího indexu*: uspořádej si proměnné (např. v pořadí $x_1, \ldots, x_n, y_1, \ldots, y_m$ od nejmenší po největší) a pro pivot vyber vždy ten sloupec j s $-c_j < 0$, který je označen nejmenší proměnnou; pokud v něm je příslušné minimum dosaženo ve více řádcích, zvol pro pivot ten řádek, který je označen nejmenší proměnnou.

Důkaz toho, že se algoritmus při daném pravidle nezacyklí, není těžký, ale je trochu technicky komplikovaný. Jednodušeji se dá ukázat nezacyklení např. pro tzv. *lexikografické pravidlo*, které teď objasníme.

> Pravidlo se v praxi nepoužívá, zde ho uvedeme hlavně proto, abychom opravdu dokončili důkaz věty 2 o silné dualitě.

Lexikografické pravidlo. Standardní iniciální tabulku doplníme m sloupci vpravo takto

(přidáme jednotkovou matici a nulový řádek):

$$\begin{array}{c|cccc|ccccc} & x_1 & x_2 & \cdots & x_n & & & & & \\ \hline y_1 & a_{11} & a_{12} & \cdots & a_{1n} & b_1 & 1 & 0 & \ldots & 0 \\ y_2 & a_{21} & a_{22} & \cdots & a_{2n} & b_2 & 0 & 1 & \ldots & 0 \\ & & \cdots & & & & & & & \\ y_m & a_{m1} & a_{m2} & \cdots & a_{mn} & b_m & 0 & 0 & \ldots & 1 \\ \hline z & -c_1 & -c_2 & \cdots & -c_n & v & 0 & 0 & \ldots & 0 \end{array} \tag{36}$$

"Pivotujeme" pořád jen podle pivotů v základní tabulce, přitom příslušně prohazujeme y_i a x_j, ale měníme podle "pivotovacích pravidel" všechna čísla v tabulce, včetně těch v dodaných sloupcích vpravo. Čísla v tabulce po konkrétní iteraci algoritmu označme $\hat{a}_{ij}$, $-\hat{c}_j$, $\hat{b}_i$, $\hat{v}$. Symbolem $\hat{\mathbf{b}}_i$ označíme vektor $(\hat{b_i}, ., \ldots, .)$ na konci (aktuálního) i-tého řádku (tedy řádkový vektor rozměru $1{+}m$). (Na začátku, když po "nulté iteraci" máme tabulku (36), máme tedy $\hat{\mathbf{b}}_i = (b_i, 0, \ldots, 0, 1, 0, \ldots, 0)$ s jedničkou v $(1{+}i)$-té komponentě.) Podobně $\hat{\mathbf{v}}$ označuje vektor odpovídající konci posledního řádku $(\hat{v}, ., \ldots, .)$.

Lexikografické uspořádání vektorů (omezíme se na vektory stejného rozměru) označíme $\mathbf{u} \preceq \mathbf{w}$; položíme $\mathbf{u} \prec \mathbf{w}$, jestliže první složka, ve které se vektory liší je menší ve vektoru $\mathbf{u}$. (Máme tedy např. $(5, -17, 13, 285, 428) \prec (5, -17, 15, -45, 0)$.) Uspořádání je úplné (taky říkáme lineární): libovolné dva vektory jsou si buď rovny, nebo je jeden menší než druhý.

Lexikografické pravidlo u simplexového algoritmu vypadá takto: sloupec j splňující $-\hat{c}_j < 0$ si zase můžeme vybrat libovolně, ale řádek v něm zvolíme podle lexikograficky nejmenšího vektoru $\hat{\mathbf{b}}_i/\hat{a}_{ij}$, kde $\hat{a}_{ij} > 0$.

V našem případě bude tedy řádek vždy určen jednoznačně (k danému vybranému sloupci). Ukážeme totiž, že vektory $\hat{\mathbf{b}}_1, \hat{\mathbf{b}}_2, \ldots, \hat{\mathbf{b}}_m$ jsou lineárně nezávislé, a žádný z nich tedy nemůže být násobkem jiného (a tedy podíly dvou různých vektorů se nemohou rovnat):

> Na začátku (pro tabulku (36)) je příslušná nezávislost zřejmá (díky dodané jednotkové matici). Předpokládejme nyní, že po aktuální iteraci jsou $\hat{\mathbf{b}}_1, \hat{\mathbf{b}}_2, \ldots, \hat{\mathbf{b}}_m$ lineárně nezávislé, což jinými slovy znamená, že
>
> $$\textstyle\sum_{i=1}^m \lambda_i \hat{\mathbf{b}}_i = \lambda_1 \hat{\mathbf{b}}_1 + \cdots + \lambda_m \hat{\mathbf{b}}_m = 0 \text{ implikuje } \lambda_1 = \cdots = \lambda_m = 0$$
>
> (pro libovolné reálné koeficienty λ_i). Po transformaci podle pivotu $\hat{a}_{\mathtt{rc}} > 0$ dostaneme nové hodnoty, označené $\hat{\mathbf{b}}'_1, \hat{\mathbf{b}}'_2, \ldots, \hat{\mathbf{b}}'_m$, takto:
>
> $$\hat{\mathbf{b}}'_{\mathtt{r}} = \frac{1}{\hat{a}_{\mathtt{rc}}} \hat{\mathbf{b}}_{\mathtt{r}} \text{ a } \hat{\mathbf{b}}'_i = \hat{\mathbf{b}}_i - \frac{\hat{a}_{i\mathtt{c}}}{\hat{a}_{\mathtt{rc}}} \hat{\mathbf{b}}_{\mathtt{r}} \text{ pro } i \neq r. \tag{37}$$
>
> Předpokládejme teď, že $\sum_{i=1}^m \lambda_i \hat{\mathbf{b}}'_i = 0$; to můžeme rozepsat
>
> $$(\tfrac{\lambda_{\mathtt{r}}}{\hat{a}_{\mathtt{rc}}} - \textstyle\sum_{i \neq \mathtt{r}} \lambda_i \frac{\hat{a}_{i\mathtt{c}}}{\hat{a}_{\mathtt{rc}}}) \hat{\mathbf{b}}_{\mathtt{r}} + \sum_{i \neq \mathtt{r}} \lambda_i \hat{\mathbf{b}}_i = 0,$$
>
> z čehož díky nezávislosti $\hat{\mathbf{b}}_1, \hat{\mathbf{b}}_2, \ldots, \hat{\mathbf{b}}_m$ plyne, že všechny koeficienty jsou nulové, tedy $\lambda_i = 0$ pro všechna $i \neq \mathtt{r}$ a $\frac{\lambda_{\mathtt{r}}}{a_{\mathtt{rc}}} - \sum_{i \neq \mathtt{r}} \lambda_i \frac{a_{i\mathtt{c}}}{a_{\mathtt{rc}}} = 0$, tedy $\frac{\lambda_{\mathtt{r}}}{a_{\mathtt{rc}}} = 0$ a tedy $\lambda_{\mathtt{r}} = 0$. Vektory $\hat{\mathbf{b}}'_1, \hat{\mathbf{b}}'_2, \ldots, \hat{\mathbf{b}}'_m$ jsou tedy také lineárně nezávislé.

Snadno také ověříme, že vektory $\hat{\mathbf{b}}_1, \hat{\mathbf{b}}_2, \ldots, \hat{\mathbf{b}}_m$ jsou vždy lexikograficky pozitivní, což znamená, že první nenulová složka je v nich pozitivní. Na začátku je to zřejmé (počítáme s vektorem $b \geq 0$), a udržování této vlastnosti plyne z (37) [což máte ověřit v jednom úkolu].

To také znamená, že vektor v "pravém dolním rohu" se transformací podle pivotu vždy lexikograficky zvětší, neboť $\hat{\mathbf{v}}' = \hat{\mathbf{v}} - \frac{-c_c}{a_{rc}}\hat{\mathbf{b}}_\mathbf{r}$ (k $\hat{v}$ se tedy přičte lexikograficky pozitivní vektor).

Protože konkrétní rozdělení proměnných $x_1, \ldots, x_n, y_1, \ldots, y_m$ na bázové a nebázové určuje celou tabulku jednoznačně (umíte vysvětlit proč?), algoritmus se při dodržování lexikografického pravidla nemůže zacyklit.

Shrnutí simplexového algoritmu.

Vstup: matice A, vektory b, c (rozměrů $m \times n$, $m \times 1$, $n \times 1$) s reálnými (resp. racionálními) prvky.

Výstup: odpověď, zda úloha $\max\{c^T x \mid x \geq 0, Ax \leq b\}$ (což je jiný zápis dřívějšího $Ax \leq b, x \geq 0, \max c^T$) je řešitelná omezená (FB), řešitelná neomezená (FU), či neřešitelná (I); v případě FB algoritmus také vydá (nějaké) optimální řešení x^* a hodnotu $c^T x^*$ (a navíc vydá optimální řešení y^* pro duální úlohu $\min\{y^T b \mid y \geq 0, y^T A \geq c^T\}$).

Postup:

1. Sestav tabulku

$$\begin{array}{c|cccc|c} & x_1 & x_2 & \cdots & x_n & \\ \hline y_1 & a_{11} & a_{12} & \cdots & a_{1n} & b_1 \\ y_2 & a_{21} & a_{22} & \cdots & a_{2n} & b_2 \\ & & \cdots & & & \\ y_m & a_{m1} & a_{m2} & \cdots & a_{mn} & b_m \\ \hline z & -c_1 & -c_2 & \cdots & -c_n & 0 \end{array} \tag{38}$$

   a/ Jestliže $b_i < 0$ pro nějaké i, tak proveď $InitSimplex$ (definován níže). Vrátí-li "neřešitelná" (I), vrať "neřešitelná" a skonči. Jinak $InitSimplex$ vrátí tabulku

$$\begin{array}{c|cccc|c} & \hat{s}_1 & \hat{s}_2 & \cdots & \hat{s}_n & \\ \hline \hat{t}_1 & \hat{a}_{11} & \hat{a}_{12} & \cdots & \hat{a}_{1n} & \hat{b}_1 \\ \hat{t}_2 & \hat{a}_{21} & \hat{a}_{22} & \cdots & \hat{a}_{2n} & \hat{b}_2 \\ & & \cdots & & & \\ \hat{t}_m & \hat{a}_{m1} & \hat{a}_{m2} & \cdots & \hat{a}_{mn} & \hat{b}_m \\ \hline z & -\hat{c}_1 & -\hat{c}_2 & \cdots & -\hat{c}_n & \hat{v} \end{array} \tag{39}$$

   kde $\{\hat{t}_1, \ldots, \hat{t}_m, \hat{s}_1, \ldots, \hat{s}_n\} = \{y_1, \ldots, y_m, x_1, \ldots, x_n\}$ a $\hat{b}_i \geq 0$ pro všechna i. (Jedná se o rovnost množin, nemusí samozřejmě platit $\hat{t}_i = y_i$ apod.)

   b/ Jestliže $b_i \geq 0$ pro všechna i, vezmi jako tabulku (39) původní tabulku (38).

2. a/ Pokud neexistuje sloupec j, kde $-\hat{c}_j < 0$, skonči s odpovědí FB a vydej řešení $x^* = (x_1^*, \ldots, x_n^*)$, kde každé x_j^* je sestrojené takto: jestliže x_j označuje sloupec (je tedy v horním řádku), polož $x_j^* = 0$; jestliže x_j označuje řádek r (je tedy v levém sloupci), polož $x_j^* = \hat{b}_r$. Jako hodnotu $c^T x^*$ vydej $\hat{v}$.

   (Pro duální případ vydej $y^* = (y_1^*, \ldots, y_m^*)$, kde každé y_i^* je sestrojené takto: jestliže y_i označuje řádek (je tedy v levém sloupci), polož $y_i^* = 0$; jestliže y_i označuje sloupec s (je tedy v horním řádku), polož $y_i^* = -\hat{c}_s$. Jako hodnotu $(y^*)^T b$ vydej $\hat{v}$.)

   b/ Jinak vyber sloupec s, kde $-c_s < 0$, např. užitím Blandova pravidla.

3. a/ Pokud $\hat{a}_{is} \leq 0$ pro všechna i, skonči s odpovědí "řešitelná neomezená".

b/ Jinak vyber řádek r, kde $\hat{a}_{rs} > 0$ a hodnota $\frac{\hat{b}_r}{\hat{a}_{rs}}$ je nejmenší možná. (Např. opět užitím Blandova pravidla.)

c/ Proveď transformaci tabulky podle pivotu $\hat{a}_{rs}$; nové hodnoty opět označ (tj. přiřaď do proměnných našeho algoritmu) $\hat{a}_{ij}$, $\hat{b}_i$, $-\hat{c}_j$, $\hat{v}$. Jdi na bod 2.

Algoritmus *InitSimplex*.

Vstup: tabulka (38), kde $b_i < 0$ pro nějaké i.

Výstup: odpověď "neřešitelná" nebo tabulka (39), v níž $\hat{b}_i \geq 0$ pro všechna i.

Postup: Vytvoř tabulku

$$\begin{array}{c|ccccc|c} & x_0 & x_1 & x_2 & \cdots & x_n & \\ \hline y_1 & -1 & a_{11} & a_{12} & \cdots & a_{1n} & b_1 \\ y_2 & -1 & a_{21} & a_{22} & \cdots & a_{2n} & b_2 \\ & & \cdots & & & & \\ y_m & -1 & a_{m1} & a_{m2} & \cdots & a_{mn} & b_m \\ \hline z & 1 & 0 & 0 & \cdots & 0 & 0 \end{array} \tag{40}$$

Proveď transformaci podle pivotu v prvním sloupci a v řádku s nejmenším b_i; tím se v posledním sloupci (mimo pravého dolního rohu) objeví nezáporné hodnoty.

V získané tabulce pokračuj podle (hlavního) simplexového algoritmu. Skončíš-li se záporným optimem, vydej "neřešitelná". Skončíš-li s optimem 0, vrať poslední tabulku, kterou upravíš tak, že odstraníš x_0 s příslušným sloupcem (řádkem) a nahradíš poslední řádek (označený z) řádkem, který by vznikl z posledního řádku vstupní tabulky stejnými průběžnými úpravami (tedy ve vyjádření původní účelové funkce nahradíš bázové proměnné jejich vyjádřením pomocí nebázových, přičemž rozdělení na bázové a nebázové určuje poslední tabulka).

Komplementarita omezujících podmínek (complementary slackness).

Máme-li A, b, c, pro vektory x, y řekneme, že jsou komplementární, jestliže

$$y^T(b - Ax) = 0 \text{ a } (y^T A - c^T)x = 0;$$

pokud platí $x \geq 0$, $y \geq 0$, $(b - Ax) \geq 0$, $(y^T A - c^T) \geq 0$ (pokud tedy všechny složky těchto vektorů jsou nezáporné), tak to znamená, že

a/ pro každé $i \in \{1, \ldots, m\}$ máme $y_i \cdot (b_i - (a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n)) = 0$

b/ a pro každé $j \in \{1, \ldots, n\}$ máme $x_j \cdot (a_{1j}y_1 + a_{2j}y_2 + \cdots + a_{mj}y_m - c_j) = 0$.

V tom případě tedy $a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n < b_i$ ("s rezervou") implikuje $y_i = 0$, a $y_i > 0$ implikuje $a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n = b_i$ ("bez rezervy"). Podobně $a_{1j}y_1 + a_{2j}y_2 + \cdots + a_{mj}y_m > c_j$ implikuje $x_j = 0$, a $x_j > 0$ implikuje $a_{1j}y_1 + a_{2j}y_2 + \cdots + a_{mj}y_m = c_j$.

V našich příkladech jsme již částečně vypozorovali následující fakt:

Věta 3 *Nechť x^* je přípustným řešením úlohy $Ax \leq b$, $x \geq 0$, $\max c^T x$ a y^* je přípustným řešením úlohy $y^T A \geq c$, $y \geq 0$, $\min y^T b$. Pak x^*, y^* jsou optimálními řešeními svých úloh právě tehdy, když jsou komplementární.*

Důkaz: Nechť x^*, y^* jsou přípustnými řešeními příslušných úloh; je tedy $x^* \geq 0$, $y^* \geq 0$, $(b - Ax^*) \geq 0$, $((y^*)^T A - c^T) \geq 0$. Platí tedy

$$c^T x^* \leq (y^*)^T A x^* \leq (y^*)^T b.$$

Víme, že x^*, y^* jsou optimální právě tehdy, když $c^T x^* = (y^*)^T b$, tedy právě tehdy, když

$$c^T x^* = (y^*)^T A x^* \text{ a } (y^*)^T A x^* = (y^*)^T b. \tag{41}$$

Konjunkci (41) ovšem můžeme ekvivalentně napsat jako

$$((y^*)^T A - c^T)x^* = 0 \text{ a } (y^*)^T (b - Ax^*) = 0. \tag{42}$$

□

Aplikujme větu na velmi jednoduchém příkladu, který už známe a bude se nám ještě hodit. Vzpomeňme si na následující bipartitní graf se čtyřmi vrcholy a třemi hranami (který jsme použili při ilustraci konstrukce konvexního obalu množiny přípustných řešení):

$G = (V, E)$, kde $V = \{v_1, v_2\} \cup \{v_3, v_4\}$ a $E = \{e_1, e_2, e_3\}$, $e_1 = \{v_1, v_3\}, e_2 = \{v_1, v_4\}, e_3 = \{v_2, v_4\}$.

Každé párování v G je dáno vektorem $(x_1, x_2, x_3) \in \{0, 1\}^3$, který splňuje podmínky

$$\begin{array}{ccccccc} x_1 & + & x_2 & & & \leq & 1 \\ & & x_2 & + & x_3 & \leq & 1 \end{array}$$

Zkusme problém "relaxovat", uvažujme tedy x_1, x_2, x_3 jako reálné proměnné splňující navíc podmínku nezápornosti: $x_1 \geq 0, x_2 \geq 0, x_3 \geq 0$. Při hledání maximálního párování (maximum-cardinality matching) maximalizujeme účelovou funkci $x_1 + x_2 + x_3$.

Představme si, že někdo tvrdí, že $(x_1^*, x_2^*, x_3^*) = (0, 1, 0)$ je optimem. Pak by ovšem pro duální úlohu

minimalizuj $y_1 + y_2$ za podmínek $y_1, y_2 \geq 0$ a

$$\begin{array}{ccccccc} y_1 & & & & & \geq & 1 \\ y_1 & + & y_2 & & & \geq & 1 \\ & & y_2 & & & \geq & 1 \end{array}$$

muselo existovat optimální řešení (y_1^*, y_2^*) splňující druhé omezení, tj. $y_1 + y_2 \geq 1$, bez rezervy (protože $x_2^* > 0$); muselo by tedy být $y_1^* + y_2^* = 1$. To je ovšem spor s omezeními $y_1 \geq 1$ a $y_2 \geq 1$.

Když se naopak domníváme, že $(x_1^*, x_2^*, x_3^*) = (1, 0, 1)$ je optimem, s hodnotou účelové funkce 2, tak to klade podmínku, že první a třetí omezení jsou splněna bez rezervy, tedy $y_1^* = 1$ a $y_2^* = 1$. Řešení $(y_1^*, y_2^*) = (1, 1)$ je ovšem přípustným řešením a hodnota účelové funkce je pro něj také 2; takže jsme naši domněnku potvrdili!

> Jistě jste si u tohoto příkladu vzpomněli na Königovu větu; příště ji mj. elegantně dokážeme přes dualitu lineárního programování a tzv. totálně unimodulární matice. (Nelekejte se, je to jednoduché.)

Nové úkoly.

1. Vyřešte si další příklady z Exercise na konci 4. kapitoly v [2], ať jste si jisti, že simplexovému algoritmu rozumíte.

2. Zformulujte obecně první "pivotování" v "pomocné" úloze maximalizace $-x_0$ ($InitSimplex$) a objasněte, proč po této první transformaci je bázové řešení přípustné.

3. Detailně argumentujte, že $Ax \leq b$, $x \geq 0$ má přípustné řešení právě tehdy, když úloha $A'x' \leq b$, $x' \geq 0$, $\max -x_0$ (vytvořená dodáním proměnné x_0 jak je uvedeno v textu) má optimální řešení s hodnotou účelové funkce 0.

4. (Nepovinně.) Ukažte, že při dodržování lexikografického pravidla v simplexovém algoritmu udržujeme vektory $\hat{\mathbf{b}}_i$ lexikograficky pozitivní (začínáme-li s $b \geq 0$).

Týden 9 (od 4.4.)

Výuka v tomto týdnu probíhala samostudiem a samostatným řešením příkladů studenty; jednalo se o dokončení látky zapsané v minulém týdnu a o látku, která je zde zapsána v následujícím týdnu.

Týden 10 (od 11.4.)

Prošli jsme (některé) dřívější úkoly (i letmo v průběhu výkladu); speciálně jsme si připomněli běh simplexového algoritmu na konkrétním příkladu. Navázali jsme hned ilustrací metody *řezajících rovin* (cutting planes) k řešení úloh ILP (Integer LP), tedy úloh *celočíselného* lineárního programování.

> Jak už jsme diskutovali, úlohy kombinatorické optimalizace se typicky dají vyjádřit jako úlohy ILP; řešení "relaxace" (tedy úlohy LP bez podmínek celočíselnosti) není obecně dostačující.

Metoda řezajících rovin (cutting planes) pro ILP.
Vyřešili jsme nejprve geometricky tuto jednoduchou úlohu LP:

$$\text{maximalizuj } x_2 \text{ za podmínek } x_2 \le x_1, x_2 \le -x_1 + 1, x_1 \ge 0, x_2 \ge 0. \tag{43}$$

Snadno jsme zjistili, že zde existuje jediné optimum, totiž $(x_1^*, x_2^*) = (\frac{1}{2}, \frac{1}{2})$, s hodnotou účelové funkce $\frac{1}{2}$.

Pak jsme úlohu převedli do standardního tvaru $\max\{c^T x \mid Ax \le b, x \ge 0\}$; v našem případě jsme dostali

$$c^T = \begin{pmatrix} 0 & 1 \end{pmatrix}, \quad x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \quad A = \begin{pmatrix} -1 & 1 \\ 1 & 1 \end{pmatrix}, \quad b = \begin{pmatrix} 0 \\ 1 \end{pmatrix}. \tag{44}$$

> Sestavili jsme také duální úlohu $\min\{y^T b \mid y^T A \ge c^T, y \ge 0\}$, reprezentovali ji geometricky (na rozdíl od primální úlohy je zde oblast řešitelnosti [feasible region] neomezená, je to neomezený polyedr, tedy nikoli polytop), a přesvědčili se, že optimem je zde $(y_1^*, y_2^*) = (\frac{1}{2}, \frac{1}{2})$, s hodnotou účelové funkce samozřejmě $\frac{1}{2}$.

Iniciální tabulka simplexového algoritmu odpovídající zadání (44) je tedy

	x_1	x_2	
y_1	-1	$\boxed{1}$	0
y_2	1	1	1
	0	-1	0

(45)

Již jsme v ní vyznačili pivot; všimněme si, že díky nule na konci řádku pivotu *se hodnota účelové funkce* (a celého pravého sloupce) *nezmění*, jen změníme bázové proměnné a dostáváme násl. tabulku (i s vyznačením násl. pivotu):

	x_1	y_1	
x_2	-1	1	0
y_2	$\boxed{2}$	-1	1
	-1	1	0

(46)

Z následující tabulky, lze již (obě) optimální řešení snadno "přečíst":

$$\begin{array}{c|cc|c} & y_2 & y_1 & \\ \hline x_2 & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\ x_1 & \frac{1}{2} & -\frac{1}{2} & \frac{1}{2} \\ \hline & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \end{array} \tag{47}$$

Tabulka (47) odpovídá soustavě rovnic

$$\begin{array}{rcrcrcl} x_2 & + & \frac{1}{2}y_2 & + & \frac{1}{2}y_1 & = & \frac{1}{2} \\ x_1 & + & \frac{1}{2}y_2 & - & \frac{1}{2}y_1 & = & \frac{1}{2} \\ \hline z & + & \frac{1}{2}y_2 & + & \frac{1}{2}y_1 & = & \frac{1}{2} \end{array} \tag{48}$$

(kde z zachycuje hodnotu účelové funkce).

Našli jsme tedy optimální řešení (x_1^*, x_2^*) úlohy (43), které ale není celočíselné. Když k úloze (43) přidáme podmínku $x_1, x_2 \in \mathbb{Z}$ (v našem případě tedy $x_1, x_2 \in \mathbb{Z}_+$, kde $\mathbb{Z}_+$ označuje množinu celých nezáporných čísel), máme očividně jen dvě přípustná řešení, totiž $(0,0)$ a $(1,0)$; obě jsou v tom případě optimální, s hodnotou účelové funkce 0. Jde teď o to, jak takové celočíselné optimální řešení najít (co "nejrozumnějším") obecným algoritmem.

Klíčová idea užití řezající roviny. Připomeňme nejdříve, že pro $a \in \mathbb{R}$ označuje $\lfloor a \rfloor$ *celou část* čísla a, tedy největší celé číslo, které je menší nebo rovno a. (Např. $\lfloor 13.9 \rfloor = \lfloor 13.1 \rfloor = \lfloor 13 \rfloor = 13$ a $\lfloor -13.9 \rfloor = \lfloor -13.1 \rfloor = -14$.) Jako *zlomkovou část* (Fractional Part) čísla a chápeme reálné číslo $\mathrm{FP}(a) = a - \lfloor a \rfloor$. Je tedy

$$a = \lfloor a \rfloor + \mathrm{FP}(a) \quad \text{a} \quad 0 \leq \mathrm{FP}(a) < 1;$$

tedy $\mathrm{FP}(a)$ je vždy nezáporné číslo.

Obecně pro $x = (x_1, x_2, \ldots, x_n) \in \mathbb{R}_+^n$ můžeme nerovnici

$$a_1 x_1 + a_2 x_2 + \cdots + a_n x_n \leq b \ \ (\text{zde } b \in \mathbb{R}) \tag{49}$$

vyjádřit ekvivalentně jako

$$\lfloor a_1 \rfloor x_1 + \mathrm{FP}(a_1) x_1 + \lfloor a_2 \rfloor x_2 + \mathrm{FP}(a_2) x_2 + \cdots + \lfloor a_n \rfloor x_n + \mathrm{FP}(a_n) x_n \ \leq \ \lfloor b \rfloor + \mathrm{FP}(b).$$

Jelikož sčítance $\mathrm{FP}(a_j) x_j$ jsou nezáporné (máme totiž podmínku $x_j \in \mathbb{R}_+$, tj. $x_j \geq 0$), jejich vypuštěním nerovnost jistě neporušíme; tedy nerovnice (49) *implikuje* nerovnici

$$\lfloor a_1 \rfloor x_1 + \lfloor a_2 \rfloor x_2 + \cdots + \lfloor a_n \rfloor x_n \ \leq \ \lfloor b \rfloor + \mathrm{FP}(b).$$

Za předpokladu *celočíselnosti* $x_1, \ldots, x_n$ je poslední nerovnice ekvivalentní nerovnici s vypuštěnou $\mathrm{FP}(b)$ (proč?). Za předpokladu celočíselnosti proměnných tedy nerovnice (49) implikuje nerovnici

$$\lfloor a_1 \rfloor x_1 + \lfloor a_2 \rfloor x_2 + \cdots + \lfloor a_n \rfloor x_n \ \leq \ \lfloor b \rfloor \,. \tag{50}$$

Definice. Řekneme, že nerovnice (50) je *řezající rovina generovaná nerovnicí* (49). Také řekneme, že *rovnice* vzniklá z (49) nahrazením symbolu "$\leq$" symbolem "$=$" *generuje řezající rovinu* (50).

Pozorování. Když nezáporné celočíselné hodnoty $x_1, \ldots, x_n$ splňují nerovnici (49), tedy $\sum_{j=1}^n a_j x_j \leq b$, či speciálně rovnici $\sum_{j=1}^n a_j x_j = b$, tak splňují také příslušnou generovanou řezající rovinu (50), tedy $\sum_{j=1}^n \lfloor a_j \rfloor x_j \leq \lfloor b \rfloor$.

Technický pojem "řezající rovina" zde tedy používáme (možná ne úplně šťastně) pro příslušnou nerovnici určující poloprostor.

V našem konkrétním příkladu, v soustavě (48), např. druhá rovnice $x_1 + \frac{1}{2}y_2 - \frac{1}{2}y_1 = \frac{1}{2}$ (implikující nerovnici $x_1 + \frac{1}{2}y_2 - \frac{1}{2}y_1 \leq \frac{1}{2}$) generuje řezající rovinu $x_1 + 0 \cdot y_2 - 1 \cdot y_1 \leq 0$, tj. nerovnici $x_1 - y_1 \leq 0$. Když ji chceme vyjádřit jen pomocí proměnných x_1, x_2, lze využít rovnici $y_1 = x_1 - x_2$ ze soustavy odpovídající tabulce (45), a dostáváme $x_2 \leq 0$. (Zde je to totéž jako řezající rovina generovaná první rovnicí $x_2 + \frac{1}{2}y_2 + \frac{1}{2}y_1 = \frac{1}{2}$.)

Vidíme tedy, že požadavek celočíselnosti proměnných v (43) implikuje nerovnici $x_2 \leq 0$; když ji přidáme k původním nerovnicím, oblast řešitelnosti (feasible region) pro relaxovaný problém je již jen úsečka s krajními body $(0, 0)$ a $(1, 0)$. Takže vyřešení relaxované verze příslušné omezenější úlohy již dá celočíselné optimální řešení původního problému. Načrtněme teď obecný algoritmus, který opakuje řešení úloh LP postupně pro omezenější a omezenější oblasti řešitelnosti, až dojde k celočíselnému optimu (existuje-li).

Geometricky je oblast řešitelnosti průnikem poloprostorů, tedy polyedrem, který může mít vrcholy s neceločíselnými souřadnicemi. Vydá-li nám algoritmus pro LP takový vrchol, odřežeme ho přidanou nerovnicí [řezající rovinou], která zachová uvnitř nového menšího polyedru všechny původní body s celočíselnými souřadnicemi; a pokračujeme hledáním nového optimálního vrcholu pomocí (relaxovaného) LP ...

Gomoryho metoda řezajících rovin (Gomory's Cutting-Plane Method). Podáme obecný náčrt tohoto algoritmu, řešícího ILP:

Vstup: matice A, vektory b, c (rozměrů $m \times n$, $m \times 1$, $n \times 1$) s celočíselnými prvky.

Výstup: odpověď, zda úloha $\max\{c^T x \mid Ax \leq b, x \in (\mathbb{Z}_+)^n\}$ je řešitelná omezená (FB), řešitelná neomezená (FU), či neřešitelná (I); v případě FB algoritmus také vydá (nějaké) optimální (samozřejmě celočíselné) řešení x^* (a hodnotu $c^T x^*$).

Postup:

1. Vyřeš relaxovanou úlohu $\max\{c^T x \mid Ax \leq b, x \geq 0\}$ (v níž požadavek celočíselnosti není kladen); řekněme simplexovým algoritmem s Blandovým (či lexikografickým) pravidlem.

   Když jsi dospěl k závěru I (infeasible), vrať I (a skonči).

   Když relaxovaná úloha nemá žádné řešení, tak pochopitelně nemá ani celočíselné řešení.

   Když jsi dospěl k závěru FU (feasible unbounded), tak: jestliže původní úloha má alespoň jedno celočíselné řešení (úloha ILP je "feasible"), vrať FU; jinak vrať I (a skonči).

   Jak jsme snadno ilustrovali, relaxovaná úloha může být řešitelná neomezená, přičemž nemá celočíselné řešení (např. $\max\{\, x_2 \mid \frac{1}{3} \leq x_1 \leq \frac{2}{3} \,\}$). O (NP-těžkém) problému zjištění, zda daná úloha ILP je řešitelná, se budeme bavit později. Není těžké ukázat, že jestliže relaxovaná úloha je řešitelná neomezená a celočíselná úloha má řešení, pak je i celočíselná úloha řešitelná neomezená. [Idea geometricky: u neomezené úlohy s pouze celými, či obecněji s racionálními, čísly v zadání, roste hodnota účelové funkce podél nějaké (polo)přímky p, s racionální směrnicí, v příslušném polyedru; když polyedr obsahuje bod s celočíselnými souřadnicemi, tak jím procházející přímka, která je rovnoběžná s p, obsahuje nekonečně mnoho celočíselných bodů, na nichž účelová funkce roste také nade všechny meze.]

2. Nechť x^* je získané optimální řešení (primální úlohy).
Když x^* je celočíselné, vrať FB a x^* (a skonči); jinak pokračuj dalším bodem, přičemž aktuální tabulka, která mj. rozděluje proměnné (včetně doplňkových) na bázové a nebázové, je ta poslední simplexová tabulka, kterou dosavadní běh algoritmu sestrojil.

3. V aktuální tabulce nalezni řádek i, v němž prvek v nejpravějším sloupci (na místě $\hat{b}_i$) není celočíselný a přitom proměnná označující řádek je nejmenší možná (v zafixovaném uspořádání proměnných; jde o analogii Blandova pravidla).

   i/ Na abstraktnější úrovni lze zakončit popis algoritmu takto:
   řezající rovinu generovanou rovnicí v i-tém řádku vyjádři (výhradně) pomocí proměnných x (tedy jen původními proměnnými $x_1, \ldots, x_n$, bez použití doplňkových proměnných) a přidej ji jako další nerovnici k systému $\max\{c^T x \mid Ax \leq b, x \geq 0\}$ v bodě 1; pak začni znovu bodem 1 s tímto novým systémem (s větším počtem nerovnic a s menší oblastí řešitelnosti).

   ii/ Rozumné implementaci bližší je tento popis:
   k nerovnici odpovídající řezající rovině generované rovnicí v i-tém řádku přidej novou doplňkovou proměnnou, čímž vznikne nová rovnice;
   tu novou doplňkovou proměnnou přidej k bázovým a vyjádři ji pomocí nebázových;
   pak doplň příslušný řádek do poslední tabulky a aplikuj tzv. *duální simplexovou metodu* (s Blandovým či lexikografickým pravidlem) na tuto doplněnou tabulku.

   > Duální simplexová metoda bude osvětlena níže; z konkrétního příkladu vysvitne i důvod pro její použití.

   Skončila-li tato duální simplexová metoda s odpovědí FU, vrať I (a skonči).

   > Jelikož jsme začínali s přípustným řešením duální úlohy, tato úloha je řešitelná (feasible); pokud zjistíme, že je řešitelná neomezená (FU), tak primální úloha musí být neřešitelná (I, infeasible), což plyne z věty o slabé dualitě.

   Jinak (tedy když jsme tuto fázi skončili nalezením optimálních řešení y^* a x^* aktuální duální a primální úlohy), jdi na bod 2.

Ilustrujme metodu na našem konkrétním příkladu (43), kde navíc klademe podmínku celočíselnosti proměnných.

Bod 1 vede k tabulce (47); díky neceločíselnosti x^* přejdeme na bod 3. Proměnné máme uspořádány jako $x_1, x_2, y_1, y_2, \ldots$, proto vybereme rovnici v řádku odpovídajícím x_1, tedy $x_1 + \frac{1}{2}y_2 - \frac{1}{2}y_1 = \frac{1}{2}$; k ní přísluší řezající rovina $x_1 - y_1 \leq 0$.

> Při abstraktnější verzi 3.i/ bychom $x_1 - y_1 \leq 0$ vyjádřili jako $x_2 \leq 0$
>
> > (z poslední tabulky to odvodíme z rovnic $x_1 + \frac{1}{2}y_2 - \frac{1}{2}y_1 = \frac{1}{2}$ a $x_2 + \frac{1}{2}y_2 + \frac{1}{2}y_1 = \frac{1}{2}$, z nichž plyne $y_1 = 2x_1 + y_2 - 1$ a $y_2 = -2x_2 - y_1 + 1$, tedy $2y_1 = 2x_1 - 2x_2$, a tedy $y_1 = x_1 - x_2$),
>
> přidali k původním omezením, a pokračovali bodem 1 (s původním systémem doplněným o nerovnici $x_2 \leq 0$). Zde pokračujeme konkrétnější verzí 3.ii/.

Nerovnici $x_1 - y_1 \leq 0$ změníme na rovnici $x_1 - y_1 + y_3 = 0$ (přidali jsme novou, dosud nepoužitou, doplňkovou proměnnou y_3), a y_3 vyjádříme pomocí aktuálně nebázových y_1, y_2: použijeme-li pouze aktuální tabulku, dosadíme $x_1 = \frac{1}{2} - \frac{1}{2}y_2 + \frac{1}{2}y_1$ a získáme

$$y_3 = -x_1 + y_1 = -(\tfrac{1}{2} - \tfrac{1}{2}y_2 + \tfrac{1}{2}y_1) + y_1 = -\tfrac{1}{2} + \tfrac{1}{2}y_2 + \tfrac{1}{2}y_1.$$

K soustavě (48) tedy přibude rovnice $y_3 - \frac{1}{2}y_2 - \frac{1}{2}y_1 = -\frac{1}{2}$ a tabulku (47) doplníme na

	y_2	y_1	
x_2	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$
x_1	$\frac{1}{2}$	$-\frac{1}{2}$	$\frac{1}{2}$
y_3	$-\frac{1}{2}$	$\boxed{-\frac{1}{2}}$	$-\frac{1}{2}$
	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$

(51)

(kde si zatím rámečku nevšímáme).

> Tím jsme prostor přípustných řešení relaxovaného problému ořezali, ale tak, že jsme *zachovali všechna celočíselná přípustná řešení.*

Máme pokračovat aplikací simplexové metody na výslednou tabulku; ale pozor: bázové řešení teď není přípustné, protože v něm je y_3 záporné. Místo spuštění inicializační fáze (primální) simplexové metody využijeme toho, že pořád máme k dispozici přípustné řešení duální úlohy, a proto spustíme

Duální simplexový algoritmus. Je to jen přímočará analogie "normálního" postupu, jejíž korektnost plyne z úvah, které jsme již udělali. (Simplexová tabulka reprezentuje, přes "asociovanou tabulku", i duální úlohu, což pivotování zachovává.) Nyní jde o to, ať se pivotováním udržujeme v přípustných řešeních duální úlohy (až do okamžiku, kdy se zároveň objeví přípustné bázové řešení primální úlohy, nebo kdy zjistíme, že duální úloha je neomezená).

Pro simplexovou tabulku, v jejímž spodním řádku (bez pravého dolního rohu) jsou jen nezáporná čísla (tedy $-\hat{c} \geq 0$) pro pivot zvolíme

i/ řádek r se záporným $\hat{b}_i$ [a nejmenší proměnnou, když je jich více]; pokud takový řádek není, máme optimum;

ii/ sloupec s, pro nějž $\hat{a}_{rs}$ je *záporné* a poměr $\frac{-\hat{c}_s}{\hat{a}_{rs}}$ je nejblíže nule [v případě více možností volíme zase nejmenší proměnnou]; pokud $\hat{a}_{rj}$ je nezáporné pro vš. j, tak je duální úloha neomezená (a primální úloha je tedy neřešitelná).

Pivot v našem příkladu (51) je orámován. Transformací podle toho pivotu dostaneme

	y_2	y_3	
x_2	0	1	0
x_1	1	-1	1
y_1	1	-2	1
	0	1	0

(52)

Teď už jsou bázová řešení duální úlohy i primální úlohy přípustná, a tedy optimální. (Jinak bychom pivotovali dále.) Optimální řešení primální úlohy $(x_1^*, x_2^*) = (1, 0)$ je celočíselné; proto jej vracíme jako celočíselné optimum výchozího problému (43) a končíme.

> Můžeme si všimnout, že pivotem prohazujícím v tabulce (51) y_1 a y_2 bychom skončili v druhém optimu $(0, 0)$.

Poznámka. Uvedli jsme si kompletní důkaz, že simplexový algoritmus skončí (nezacyklí se), jestliže je používáno lexikografické pravidlo. Pro Blandovo pravidlo jsme důkaz neprovedli a neprovádíme zde ani (o něco složitější) důkaz konečnosti Gomoryho algoritmu.

Výpočetní složitost LP a ILP. Jen stručně: simplexový algoritmus pro LP není polynomiální, protože byla ukázána konstrukce (umělých) instancí vyžadujících navštívení exponenciálně mnoha bází (odpovídajících vrcholům příslušného polyedru). Přesto v praxi je simplexový algoritmus velmi úspěšný (samozřejmě s různými implementačními "vychytávkami", které se mj. také snaží vyhýbat příliš malým pivotům a jiným příčinám numerických problémů ...)

Významnou teoretickou hodnotu má elipsoidová metoda (Khachiyan 1979) prokazující, že LP patří do třídy PTIME; existuje tedy polynomiální algoritmus řešící všechny instance LP, byť v praxi se díky vyššímu stupni příslušného polynomu nepoužívá. Později byly navrženy i jiné verze polynomiálních algoritmů ...

Problém ILP je ovšem NP-těžký, už jen rozhodování, zda je daná úloha ILP řešitelná (feasible), je NP-těžké (jak ještě budeme diskutovat); pro ILP tedy žádný polynomiální algoritmus není znám (a vypadá to, že ani neexistuje).

> Jistě jste slyšeli o otázce PTIME$\stackrel{?}{=}$NPTIME (stručněji P$\stackrel{?}{=}$NP). Přes velké výzkumné úsilí se ji doposud nepodařilo zodpovědět; převládá ovšem obecné přesvědčení, že NP je vlastní nadtřídou P, z čehož by mj. plynulo, že pro tzv. NP-těžké problémy polynomiální algoritmy neexistují.

Proto je také jasné, že Gomoryho algoritmus pro ILP obecně vyžaduje (exponenciálně) dlouhé sekvence přidávání řezajících rovin a řešení meziproblémů LP.

(Nepovinně.)**(Exponenciálně) dlouhé sekvence řezání.** Ilustraci tohoto faktu jsme jen naznačili na příkladu. Úlohu (43) jsme upravili na tuto instanci problému ILP:

$$\text{maximalizuj } x_2 \text{ za podmínek } x_2 \leq 2x_1, x_2 \leq -2x_1 + 2, \text{ přičemž } x_1, x_2 \in \mathbb{Z}_+. \tag{53}$$

> Kniha [9] uvádí na s.154 obecnější problém (Chvátal-Gomory cutting planes) $x_2 \leq 2kx_1$, $x_2 \leq -2kx_1 + 2k$; naše instance (53) je speciálním případem s parametrem $k = 1$.

Když zde řešíme relaxovanou úlohu (bod 1 Gomoryho algoritmu), skončíme s tabulkou

	y_1	y_2	
x_1	$-\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{2}$
x_2	$\frac{1}{2}$	$\frac{1}{2}$	1
	$\frac{1}{2}$	$\frac{1}{2}$	1

(54)

Vybíráme teď tedy první rovnici $x_1 - \frac{1}{4}y_1 + \frac{1}{4}y_2 = \frac{1}{2}$ (s neceločíselnou pravou stranou); ta generuje řezající rovinu $x_1 - y_1 \leq 0$ jako minule, ale tentokrát tato nerovnice neodpovídá nerovnici $x_2 \leq 0$, ale nerovnici $-x_1 + x_2 \leq 0$ (v původních proměnných).

> Přidáním omezení $x_2 \leq x_1$ k (53) zase uřežeme vrchol, tentokrát $(\frac{1}{2}, 1)$, ale nezbyde pouhá úsečka. Snadno lze (geometricky) nahlédnout, že na zmenšeném polyedru [v našem případě polytopu ve tvaru trojúhelníku] je optimem vrchol $(\frac{2}{3}, \frac{2}{3})$.

Pokračováním v Gomoryho postupu dodáme doplňkovou proměnnou y_3 a rovnici $x_1 - y_1 + y_3 = 0$, kterou pomocí nebázových proměnných vyjádříme jako $y_3 - \frac{3}{4}y_1 - \frac{1}{4}y_2 = -\frac{1}{2}$. Dostáváme tedy následující tabulku, kde hned zarámujeme pivot.

	y_1	y_2	
x_1	$-\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{2}$
x_2	$\frac{1}{2}$	$\frac{1}{2}$	1
y_3	$\boxed{-\frac{3}{4}}$	$-\frac{1}{4}$	$-\frac{1}{2}$
	$\frac{1}{2}$	$\frac{1}{2}$	1

(55)

Použitím duální simplexové metody se dostaneme do optima

	y_3	y_2	
x_1	$-\frac{1}{3}$	$\frac{1}{3}$	$\frac{2}{3}$
x_2	$\frac{2}{3}$	$\frac{1}{3}$	$\frac{2}{3}$
y_1	$-\frac{4}{3}$	$\frac{1}{3}$	$\frac{2}{3}$
	$\frac{2}{3}$	$\frac{1}{3}$	$\frac{2}{3}$

(56)

Zde vybereme ke generování další řezající roviny rovnici $x_1 - \frac{1}{3}y_3 + \frac{1}{3}y_2 = \frac{2}{3}$. Nová řezající rovina tedy bude $x_1 - y_3 \leq 0$; jak se můžeme snadno přesvědčit, v původních proměnných je to naše známá $x_2 \leq 0$. Takže pokračování Gomoryho algoritmu skončí v následující iteraci (dodáním y_4, vyjádřením rovnice $x_1 - y_3 + y_4 = 0$ pomocí nebázových proměnných y_3, y_2 [a nové bázové y_4] a vyřešením příslušného relaxovaného problému duální simplexovou metodou; dodělejte sami).

Bázové řešení pomocí determinantů.

Jak brzy uvidíme, jsou prakticky se vyskytující úlohy ILP, kde Gomoryho postup nepotřebujeme, protože vrcholy polyedru řešení mají celočíselné souřadnice a celočíselná úloha je tedy hned vyřešena vyřešením relaxovaného problému.

K pochopení důležitého případu tohoto typu je vhodné se znovu zamyslet nad tím, jak vypadá optimální řešení úlohy LP nalezené simplexovým algoritmem. Uvažujme úlohu

$$\max\{c^T x \mid Ax = b, x \geq 0\}$$

(vzniklé ze standardní maximalizační úlohy s nerovnostmi dodáním doplňkových proměnných, čímž se nerovnosti změní na rovnosti). V matici A můžeme předpokládat (tj. rychlým algoritmem docílit) nezávislost řádků; máme tedy m nezávislých řádků a $n \geq m$ sloupců.

Existuje-li optimální řešení, pak je to bázové řešení odpovídající nějaké bázi sloupcového prostoru (column space) matice A, tvořenou m nezávislými sloupci. Když

$$\beta = (j_1, j_2, \ldots, j_m)$$

je uspořádáná m-tice nezávislých slopuců, označíme A_β čtvercovou matici $(m \times m)$, která je sestavena právě ze sloupců $j_1, j_2, \ldots, j_m$ matice A. Matice A_β je tedy regulární (neboli nesingulární, neboli invertibilní).

> K regulární matici B označuje B^{-1} matici inverzní; ta tedy splňuje vztahy $B^{-1} \cdot B = B \cdot B^{-1} = I$, kde I označuje jednotkovou matici příslušného rozměru.

Bázové řešení $x^\beta = (x_1^\beta, x_2^\beta, \ldots, x_n^\beta)$ příslušné k bázi β vznikne tak, že položíme $x_j^\beta = 0$ pro všechna $j \notin \beta$ a vyřešíme rovnici

$$A_\beta \cdot \begin{pmatrix} x_{j_1} \\ x_{j_2} \\ \cdot \\ \cdot \\ x_{j_m} \end{pmatrix} = b, \text{ takže dostáváme } \begin{pmatrix} x_{j_1}^\beta \\ x_{j_2}^\beta \\ \cdot \\ \cdot \\ x_{j_m}^\beta \end{pmatrix} = A_\beta^{-1} b.$$

Vzpomeňme si teď na Cramerovo pravidlo pro řešení $Ax = b$, kde A je regulární matice typu $m \times m$. Řešení $x^* = A^{-1}b$ splňuje

$$x_j^* = \frac{\det(A[j \leftarrow b])}{\det(A)} \text{ pro } j = 1, 2, \ldots, m;$$

$\det(A)$ označuje determinant matice A (který je nenulový, protože A je regulární), a výrazem $A[j \leftarrow b]$ označujeme matici, která vznikne z A tak, že j-tý sloupec nahradíme sloupcem b.

> Jen bokem si pro úplnost můžeme připomenout důkaz Cramerova pravidla.
> Mějme regulární matici A typu $m \times m$. Pro $j = 1, 2, \ldots, m$ definujme funkce $f_j : \mathbb{R}^m \to \mathbb{R}$ tak, že $f_j(y_1, \ldots, y_m) = \det(A[j \leftarrow y])$, kde $y = (y_1, \ldots, y_m)^T$. Standardním rozvojem determinantu matice $A[j \leftarrow y]$ podle j-tého sloupce dostáváme, že
>
> $$f_j(y_1, \ldots, y_m) = C_1^j y_1 + C_2^j y_2 + \cdots + C_m^j y_m$$
>
> pro jisté konstanty C_i^j (kde C_i^j je determinant matice vzniklé z A vypuštěním sloupce j a řádku i, případně vynásobený -1). Když řádkovým vektorem $(C_1^j, \ldots, C_m^j)$ vynásobíme matici A, dostaneme vektor $(0, \ldots, 0, \det(A), 0, \ldots, 0)$, kde $\det(A)$ je na j-té pozici. (Vynásobíme-li totiž vektor $(C_1^j, \ldots, C_m^j)$ a sloupec matice A, který je jiný než j-tý, dostaneme determinant matice s dvěma stejnými sloupci; ten je díky singularitě této matice nulový.)
>
> Položme $x^* = A^{-1}b$, tedy $Ax^* = b$. Z rovnice $(C_1^j, \ldots, C_m^j) \cdot A = (0, \ldots, 0, \det(A), 0, \ldots, 0)$ (kde $\det(A)$ je na j-té pozici) plyne také
>
> $$(C_1^j, \ldots, C_m^j) \cdot A \cdot x^* = (0, \ldots, 0, \det(A), 0, \ldots, 0) \cdot x^*$$
>
> Levá strana se ovšem rovná $(C_1^j, \ldots, C_m^j) \cdot b = C_1^j b_1 + \cdots + C_m^j b_m = \det(A[j \leftarrow b])$ a pravá strana se rovná $\det(A) \cdot x_j^*$. Proto $x_j^* = \frac{\det(A[j \leftarrow b])}{\det(A)}$.

Pro naše bázové řešení x^β to speciálně znamená, že když prvky matice A a vektoru b jsou celočíselné (což je vyžadováno u všech instancí ILP) a $\det(A_\beta)$ je 1 nebo -1, tak x^β je celočíselné.

Totálně unimodulární matice a celočíselnost řešení úloh LP.

Výše jsme ukázali, že instance problému ILP, tedy úloha

$$\max\{c^T x \mid Ax \leq b, x \in (\mathbb{Z}_+)^n\}, \text{ kde prvky matice } A \text{ a vektorů } b, c \text{ jsou celočíselné}$$

je jistě vyřešena, pokud je nalezeno optimální řešení relaxovaného problému odpovídající bázi β, pro niž je $\det(A_\beta)$ roven 1 nebo -1.

> Celočíselná čtvercová matice, jejíž determinant je 1 nebo -1, se nazývá *unimodulární*.

Tato pro nás žádoucí vlastnost (totiž, že $\det(A_\beta) \in \{-1, 1\}$) je jistě zaručena, pokud je matice A v zadání úlohy totálně unimodulární:

Definice (Obecná obdélníková) celočíselná matice A se nazývá *totálně unimodulární*, jestliže každá její čtvercová podmatice (vzniklá z A vypuštěním nějakých sloupců a řádků) má determinant -1, 1, nebo 0. (Mj. tedy každý prvek matice musí patřit do množiny $\{-1, 0, 1\}$, neboť tvoří podmatici typu 1×1.)

Věta Je-li v instanci ILP $\max\{c^T x \mid Ax \leq b, x \in \mathbb{Z}^n\}$ matice A totálně unimodulární, pak všechna bázová řešení relaxovaného problému jsou celočíselná.

Důkaz jsme již vlastně provedli. Stačí si jen ještě uvědomit, že dodáním jednotkové matice (odpovídající doplňkovým proměnným) nic nezkazíme. Když totiž A je totálně unimodulární, tak také $(A\ I)$ je totálně unimodulární.

> To plyne z očividného faktu, že když k totálně unimodulární matici přidáme sloupec (nebo řádek), který obsahuje (nanejvýš) jednu 1 a jinak nuly, tak výsledná matice je zase totálně unimodulární. (Řekněte si explicitně proč.)
> Když už jsme u toho, tak si všimneme, že např. vynásobením řádku nebo sloupce -1 také totální unimodularitu nepokazíme.
> Také si všimneme, že A je totálně unimodulární právě tehdy, když A^T je totálně unimodulární (jelikož pro každou čtvercovou matici B platí $\det(B) = \det(B^T)$).

Důkaz Königovy věty přes dualitu a totální unimodularitu.

Připomněli jsme si, že jsme již dříve (algoritmicky) dokázali, že velikost maximálního párování (maximum-cardinality matching) v bipartitním grafu se rovná velikosti minimálního vrcholového pokrytí (minimum-cardinality vertex cover).

Teď jsme přímočaře k uvedenému problému maximálního párování sestavili lineární program

> (tj. úlohu LP, kde proměnné odpovídaly hranám grafu; maximalizovali jsme $\sum_{e \in E} x_e$ za podmínek nezápornosti $x_e \geq 0$ a splnění $\sum_{e \in \delta(v)} x_e \leq 1$ pro všechny vrcholy v)

a ukázali jsme, že příslušná matice je totálně unimodulární.

Tím pádem je optimální bázové řešení celočíselné. Stejně tak optimální bázové řešení duální úlohy je celočíselné. Uvědomili jsme si, že celočíselné řešení duální úlohy přirozeně odpovídá vrcholovému pokrytí, přičemž účelová funkce počítá kardinalitu tohoto pokrytí.

> Každý sloupec matice odpovídá jedné hraně; jsou v něm právě dvě jedničky (jinak nuly), a sice v řádcích odpovídajících vrcholům incidentním s danou hranou. Díky bipartitnosti našeho grafu můžeme vynásobit řádky odpovídající vrcholům první partity -1 a docílit tak matici (obecně odpovídající orientovanému grafu), kde každý sloupec obsahuje právě jednu 1 a jednu -1. (Důkaz totální unimodularity této matice je snadný; žádá to jeden úkol.) Když máme celočíselné řešení duální úlohy (tj. nezápornou celočíselnou lineární kombinaci řádků), pro každý sloupec musí mít v řešení nenulový koeficient alespoň jeden jeho "jedničkový" řádek; množina řádků s nenulovými koeficienty odpovídá tedy množině vrcholů, která tvoří vrcholové pokrytí.
>
> Víme, že přímé zobecnění Königovy věty na obecné grafy neplatí. Nebipartitní graf (zde K_3) může vést např. na matici
>
> $$\begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix}$$

kde kýžené vlastnosti nedocílíme. Matice totiž není totálně unimodulární: její determinant je (rozvojem podle prvního řádku)

$$1 \cdot \begin{vmatrix} 0 & 1 \\ 1 & 1 \end{vmatrix} - 1 \cdot \begin{vmatrix} 1 & 1 \\ 0 & 1 \end{vmatrix} = -2\,.$$

Nové úkoly.

1. Zkuste exaktně dokázat, že když je úloha ILP řešitelná a její relaxovaná varianta je řešitelná neomezená, tak i původní celočíselná úloha je řešitelná neomezená. (Připomeňte si, jak skončí simplexový algoritmus, když je relaxovaná varianta řešitelná neomezená.)

2. Vyřešte Gomoryho metodou (těžkou :-)) úlohu

$$\text{maximalizuj } x_1 \text{ za podmínek } -3x_1 \leq -1 \text{ a } 3x_1 \leq 2, \text{ přičemž } x_1 \in \mathbb{Z}_+.$$

3. Vysvětlete, proč matice, v níž každý sloupec obsahuje právě jednu 1 a právě jednu -1 a jinak nuly, je totálně unimodulární. (Nápověda: postupujte indukcí podle rozměru čtvercových podmatic.)

4. Připomeňte si "Assignment problem" řešený v Example (Kuhn's Assignment Algorithm) na str. 123 v [9]. To je také úloha kombinatorické optimalizace vedoucí přirozeně na úlohu celočíselného lineárního programování. Sestavte obecné zadání pro příslušný relaxovaný problém a ukažte, že příslušná matice je totálně unimodulární. (Co z toho vyvodíme?)

Týden 11 (od 18.4.)

Nejprve jsme řešili dřívější úkoly. Speciálně jsme detailně připomněli Gomoryho postup řešení úloh ILP na úkolu 2 z minula. Také jsme diskutovali problém hlášení I (infeasible), FU (feasible unbounded) v našem popisu Gomoryho algoritmu v minulém týdnu.

NP-obtížnost problémů (kombinatorické optimalizace).

Jako hlavní nové téma jsme diskutovali fakt, že pro mnohé problémy kombinatorické optimalizace nejsou známy (a "pravděpodobně" ani neexistují) polynomiální algoritmy. Ukazuje se, že problémy kombinatorické optimalizace se dají (velmi) zhruba rozdělit na dvě skupiny: polynomiální a NP-těžké (což typicky znamená NP-ekvivalentní). Připomeňme si základní pojmy potřebné k této diskusi.

Výpočetní problémy a rozhodovací problémy. *Výpočetní problém*, stručně jen *problém*, je dán množinou vstupů (neboli instancí), množinou výstupů a zobrazením, které každému vstupu přiřazuje výstup; v základní definici je přiřazen každému vstupu právě jeden výstup, my ale budeme rovnou počítat s problémy, kde příslušných výstupů může být více.

> Přiřazení více možných výstupů k jednomu vstupu je u optimalizačních problémů běžné. Např. problém minimální kostry grafu (označený třeba MST, minimal spanning tree) má jako instance neorientované grafy s váhami na hranách a koster s minimální váhou může existovat k danému grafu více. (Problém MST tedy obecně přiřazuje k danému vstupu více výstupů; u příslušného řešícího algoritmu nám ovšem stačí, že vrátí jeden z nich.)

Množina vstupů je typicky spočetná a součástí zadání problému je i konkrétní prezentace vstupů a výstupů; každý vstup či výstup de facto odpovídá jisté konečné posloupnosti bitů, tedy řetězci nul a jedniček.

> Konkrétní reprezentace (neboli "kódování") vstupů a výstupů je většinou implicitní; v našich úvahách prostě předpokládáme nějakou přirozenou reprezentaci, ale detaily jsou v našem kontextu většinou nedůležité. (Např. graf lze reprezentovat incidenční maticí, matici pak sekvencí řádků s oddělovači; jiná, v našem kontextu ekvivalentní, prezentace je sekvence hran grafu zapsaných jako dvojice vrcholů, apod.)
> Důležitá konkrétnost, na níž někdy spočívá výpočetní složitost problému, je způsob *reprezentace čísel*. Standardně předpokládáme binární zápis, případně dekadický; hodnota čísla je tak exponenciálně velká vzhledem k délce jeho zápisu. (Jelikož $\log_2 n = (\log_2 10) \cdot (\log_{10} n)$, platí $\log_2 n \in \Theta(\log_{10} n)$, tj. $\log_2 n \in O(\log_{10} n)$ a $\log_{10} n \in O(\log_2 n)$, a proto je pro účely analýzy složitosti algoritmů nepodstatné, zda používáme binární či dekadický zápis. Pokud bychom ovšem použili unární zápis, v němž je číslo prezentováno počtem "čárek", složitost algoritmu jakožto funkce velikosti vstupu se rázem zmenší.)

Speciální třídu problémů (která hraje důležitou roli při charakterizaci výpočetní složitosti problémů) tvoří *rozhodovací problémy* (decision problems), u nichž je každému vstupu přiřazen právě jeden z výstupů ANO (neboli "true" neboli "1" neboli "accept") a NE (neboli "false" neboli "0" neboli "reject").

> Konkrétní rozhodovací problém lze tak přirozeně popsat zadáním instancí (vstupů) a zjišťovací otázkou, na niž je pro každý vstup odpověď ANO nebo NE.

Problém SAT. Výsadní postavení v našem kontextu má problém SAT (satisfiability), což je rozhodovací problém, u nějž jsou instancemi booleovské formule a otázkou je, zda je daná formule splnitelná (tedy zda existuje pravdivostní ohodnocení proměnných, při němž je formule

pravdivá). Příkladem formule ϕ, či podrobněji $\phi(x_1, x_2, x_3, x_4, x_5)$ (což označuje, že všechny proměnné vyskytující se ve ϕ jsou z množiny $\{x_1, x_2, x_3, x_4, x_5\}$), je

$$(x_1 \vee \neg x_2 \vee x_5) \wedge (\neg x_1 \vee x_3 \vee \neg x_4) \wedge (\neg x_2 \vee \neg x_3 \vee x_4);$$

tato formule je zároveň příkladem instance problému 3-SAT, protože je v *konjunktivní normální formě* (conjunctive normal form, CNF) a každá *klauzule* (tj. "konjunkt", který je disjunkcí literálů, kde *literál* je proměnná nebo její negace) obsahuje právě tři literály. V souladu se standardní praxí budeme u SAT předpokládat formule (rovnou) v CNF. (Uvedená formule je očividně splnitelná, jak demonstruje např. ohodnocení splňující $x_1 = 1, x_2 = 0, x_3 = 1$.)

> Popisujeme-li např. podmínky, které má splňovat nějaký systém (např. přípustné řešení úlohy kombinatorické optimalizace), jedná se často o konjunkci (mnoha) jednoduchých podmínek; konjunktivní normální forma je tak velmi přirozená. Jinak jistě víte, že každá booleovská formule se dá převést na ekvivalentní formuli v CNF; ovšem tento převod v některých případech nutně vede k exponenciálně větší formuli. Jeden úkol vás žádá o návrh polynomiálního algoritmu, který k obecné formuli ϕ sestrojí ϕ', která sice není ekvivalentní ϕ, ale je splnitelná právě tehdy, když ϕ je splnitelná.

Není těžké nahlédnout, že pro typický problém kombinatorické optimalizace lze přímočaře popsat množinu přípustných řešení jako konjunkci jednoduchých podmínek, konkrétněji tedy booleovskou formulí v CNF. (Úkoly také žádají demonstraci v konkrétních případech.) U popisu podmínky optimality (minimality či maximality účelové funkce pro dané řešení) je to komplikovanější.

> Přirozeným zobecněním problému SAT je problém Q-SAT (také označovaný QBF, Quantified Boolean Formulas), kde proměnné mohou být kvantifikovány. Pro zjednodušení úvah se často uvažuje jen případ, kdy jsou všechny proměnné kvantifikovány (tedy neexistují volné proměnné); v tom případě je splnitelnost formule totéž co pravdivost a nesplnitelnost totéž co nepravdivost. Problém SAT je opravdu speciálním případem Q-SAT: formule $\phi(x_1, \ldots, x_n)$ je totiž splnitelná právě tehdy, když formule $\exists x_1 \cdots \exists x_n \phi(x_1, \ldots, x_n)$ je pravdivá.
> Optimalitu řešení lze přímočaře popsat kvantifikovanou booleovskou formulí (kde binárně zapsaná čísla reprezentujeme sekvencemi booleovských proměnných). Ovšem s výpočetní složitostí (obecného) problému Q-SAT je to horší, jak se ještě zmíníme. Proto je zde vhodnější využít těsné výpočetní vazby na rozhodovací problémy, u nichž kvantifikátory nepotřebujeme.

Rozhodovací verze optimalizačních problémů. Každému optimalizačnímu problému $\mathcal{P}$ (žádajícímu pro daný vstup vydání přípustného výstupu, který je optimální vzhledem k zadané účelové funkci) odpovídá příslušný rozhodovací problém $\mathcal{P}_{dec}$ (*dec* jako "decision"): $\mathcal{P}_{dec}$ má jako vstup instanci problému $\mathcal{P}$ a číslo ℓ a otázkou je, zda existuje přípustné řešení s hodnotou účelové funkce alespoň ℓ (v případě maximalizačního problému) či nanejvýš ℓ (v případě minimalizačního problému). Problém $\mathcal{P}_{dec}$ je pak už vcelku snadné vyjádřit jako speciální případ problému SAT.

> Úkoly žádají také ukázat těsný "výpočetní" vztah mezi $\mathcal{P}$ a $\mathcal{P}_{dec}$. Stručně řečeno: umíme-li rychle řešit jeden z nich, umíme také rychle řešit druhý.

Výpočetní složitost algoritmů a problémů. Problém je *algoritmicky řešitelný*, jestliže existuje algoritmus, který ke každému vstupu "vypočte" příslušný výstup (nebo jeden z výstupů, které problém danému vstupu přiřazuje). U rozhodovacího problému říkáme také *algoritmicky rozhodnutelný*, nebo jen *rozhodnutelný*, místo "algoritmicky řešitelný".

Problémy kombinatorické optimalizace jsou v principu vždy algoritmicky řešitelné tzv. hrubou silou (brute force), kdy jsou systematicky probrána všechna přípustná řešení. Těch je ovšem typicky exponenciálně mnoho (podmnožin množiny s n prvky je totiž 2^n) a již pro vstupy, u nichž příslušná základní množina (třeba hran grafu) má několik desítek prvků, se výsledku výpočtu takového algoritmu nedočkáme.

Obecně se za nutnou podmínku praktické použitelnosti algoritmu považuje polynomialita: *algoritmus* $\mathcal{A}$ má polynomiální časovou složitost, neboli je *polynomiální*, neboli má složitost $O(n^k)$ pro nějaké $k \in \mathbb{N}$, jestliže existují konstanty $c, k \in \mathbb{N}$ takové, že výpočet algoritmu $\mathcal{A}$ na jakýkoli vstup velikosti n skončí v nanejvýše $c \cdot n^k$ krocích.

> Samozřejmě by bylo třeba upřesnit, co se rozumí kroky algoritmu. Tímto vyjádřením se odkazujeme k nějakému konkrétnímu "programovacímu jazyku", což může být jak skutečný programovací jazyk jako C či Java, tak matematický model typu Turingova stroje, stroje RAM, apod. Pro naše účely si stačí uvědomit, že pojem "polynomialita algoritmu" je dostatečně robustní, tedy nezávislý na tom, jaký konkrétní výpočetní model [v němž algoritmy "implementujeme"] máme na mysli.

O *problému* řekneme, že je *polynomiální*, jestliže existuje polynomiální algoritmus, který jej řeší.

Třídy složitosti P, FP, NP. Třída P (tj. PTIME) je třída (neboli množina) *rozhodovacích* problémů, které jsou řešitelné polynomiálními algoritmy (tj. algoritmy s polynomiální časovou složitostí). Třída FP (F ze slova Funkce, rozumí se funkce přiřazující vstupům výstupy, resp. množiny výstupů) je třída (obecných) problémů řešitelných polynomiálními algoritmy.

> Např. problém nejkratších cest, problém maximálního toku, problém minimální kostry, problém maximálního váženého párování v grafu, i obecný problém lineárního programování patří do FP. (Jak jsme již diskutovali, simplexový algoritmus není polynomiální, kvůli exponenciálnímu běhu na některých [byť řídce se v praxi vyskytujících] instancích, ale k problému LP existují jiné algoritmy, které polynomiální jsou.) Do FP samozřejmě patří i neoptimalizační problémy, jako je např. třídění a spousta dalších problémů.

Někdy se mezi třídami P a FP nerozlišuje; pak P (či PTIME) označuje de facto FP.
Třídou NP, tj. NPTIME (Nondeterministic Polynomial TIME), se rozumí (vždy jen) třída *rozhodovacích* problémů, které jsou řešitelné polynomiálními nedeterministickými algoritmy.

Nedeterministický algoritmus lze chápat jako standardní (deterministický) algoritmus obohacený o možnost instrukce typu ($x := 0$ [] $x := 1$), kde [] představuje nedeterministický výběr: konkrétní výpočet algoritmu provedením této instrukce přiřadí do x buď 0 nebo 1. Při použití takových instrukcí má příslušný algoritmus k danému vstupu více možných výpočtů.

Řekneme, že *nedeterministický algoritmus* $\mathcal{A}$ *řeší rozhodovací problém* $\mathcal{P}$, jestliže pro každý vstup problému $\mathcal{P}$, kterému problém přiřazuje výstup ANO, existuje alespoň jeden výpočet algoritmu $\mathcal{A}$, který vrátí ANO, a pro každý vstup problému $\mathcal{P}$, který má přiřazen výstup NE, vrátí všechny výpočty algoritmu $\mathcal{A}$ odpověď NE.

Polynomialita nedeterministického algoritmu $\mathcal{A}$ je definována stejně jako u deterministického: musí existovat c, k tak, že každý výpočet algoritmu $\mathcal{A}$ pro vstup velikosti n skončí za nanejvýš $c \cdot n^k$ kroků.

> Každý problém z P je tedy v NP (neboť deterministický algoritmus je de facto speciálním případem nedeterministického algoritmu); platí tak P$\subseteq$NP (ale otázka, zda se jedná o vlastní inkluzi, je dlouhodobě otevřená). Např. o problému SAT nevíme, zda patří do P, ale do NP patří očividně: příslušný algoritmus pro vstup $\phi(x_1, \ldots, x_n)$ provede sekvenci instrukcí $(x_1 := 0 \ [] \ x_1 := 1), \ldots, (x_n := 0 \ [] \ x_n := 1)$, pak vyhodnotí ϕ pro zvolené ohodnocení (přiřazené do "programových proměnných" $x_1, \ldots, x_n$) a výsledek vrátí (true jako ANO a false jako NE).
> Nedeterministický algoritmus v uvedeném smyslu není samořejmě nijak "praktický"; jedná se ale o pojem umožňující elegantní zachycení mnohých aspektů složitosti problémů.

Rozhodovací problémy $\mathcal{P}_{dec}$ příslušné problémům kombinatorické optimalizace $\mathcal{P}$ jsou v zásadě všechny v NP.

> Používá se i definice třídy FNP. Problém, přiřazující každému vstupu množinu výstupů (třeba prázdnou), patří do FNP, jestliže existuje nedeterministický polynomiální algoritmus, pro nějž platí
>
> 1. každý výpočet pro vstup w skončí vydáním nějakého výstupu z množiny přiřazené vstupu w nebo odpovědí NE;
> 2. jestliže množina výstupů přiřazená vstupu w není prázdná, pak alespoň jeden výpočet pro w vydá výstup z této množiny.
>
> Do FNP tak patří např. tato modifikace problému SAT: vstup je booleovská formule ϕ a příslušná množina výstupů je množina všech pravdivostních ohodnocení proměnných ve ϕ, pro něž je ϕ pravdivá.

Polynomiální převeditelnost mezi problémy ($\mathcal{P} \preceq_p^T \mathcal{P}'$ a $\mathcal{P} \preceq_p \mathcal{P}'$). Při programování typicky používáme funkce (procedury) z nějaké knihovny. Představme si teď, že máme k dispozici proceduru, která rozhoduje jistý rozhodovací problém $\mathcal{O}$ (např. SAT), a že máme program (algoritmus) $\mathcal{A}$, který řeší problém $\mathcal{P}$, přičemž může (vícekrát) volat proceduru pro $\mathcal{O}$ (při každém volání jí předloží konkrétní vstup neboli parametr). Pokud je $\mathcal{A}$ deterministický polynomiální algoritmus za předpokladu, že každé volání funkce $\mathcal{O}$ se počítá jako (pouhý) jeden krok, tak jsme ukázali, že *problém $\mathcal{P}$* je *turingovsky polynomiálně převeditelný na problém $\mathcal{O}$*, což značíme $\mathcal{P} \preceq_p^T \mathcal{O}$ (symbol T odkazuje k "turingovsky", p odkazuje k "polynomiálně").

> Jinými slovy: pokud máme tzv. *orákulum* $\mathcal{O}$, tj. rozhodovací problém, jehož instance umíme (hypoteticky) řešit (tedy zodpovídat ANO/NE) v jednom kroku, tak $\mathcal{P} \preceq_p^T \mathcal{O}$ znamená, že $\mathcal{P}$ umíme řešit v polynomiálním čase, relativizovaném vzhledem k $\mathcal{O}$; v jiném značení napíšeme $\mathcal{P} \in \mathrm{FP}^{\mathcal{O}}$, kde $\mathrm{FP}^{\mathcal{O}}$ je třída problémů řešitelných (deterministickými) polynomiálními algoritmy, které mohou využívat orákula $\mathcal{O}$.
> Třída $\mathrm{P}^{\mathcal{O}}$ je pak restrikcí třídy $\mathrm{FP}^{\mathcal{O}}$ na rozhodovací problémy.

Pozorování. Když $\mathcal{P} \preceq_p^T \mathcal{O}$ a $\mathcal{O} \in$ P, tak $\mathcal{P} \in$ FP. Když $\mathcal{P} \preceq_p^T \mathcal{O}$ a $\mathcal{P} \notin$ FP, tak $\mathcal{O} \notin$ P.

(Uvědomte si důkaz, snadno plynoucí z faktu, že složení polynomů je polynom; tedy funkce $p_2(p_1(n))$ je polynom, jsou-li p_1, p_2 polynomy.)

Lze přirozeně definovat

$$\mathcal{P}_1 \preceq_p^T \mathcal{P}_2$$

i pro obecný problém $\mathcal{P}_2$ (nejen rozhodovací). Pak ovšem do práce příslušného polynomiálního algoritmu pro $\mathcal{P}_1$ (který může volat proceduru pro $\mathcal{P}_2$) musíme započítat i čtení výsledků, které volání procedury pro $\mathcal{P}_2$ vracejí; velikost těchto výsledků musí být tedy také polynomiálně omezená (vzhledem ke vstupu problému $\mathcal{P}_1$).

Relace $\preceq_p^T$ na množině všech problémů je kvazi-uspořádání, neboť je reflexivní a tranzitivní (pro každý problém $\mathcal{P}$ platí $\mathcal{P} \preceq_p^T \mathcal{P}$; ze vztahů $\mathcal{P}_1 \preceq_p^T \mathcal{P}_2$, $\mathcal{P}_2 \preceq_p^T \mathcal{P}_3$ plyne $\mathcal{P}_1 \preceq_p^T \mathcal{P}_3$ [protože složení polynomů je polynom]). Přirozeně tak dostáváme i ekvivalenci

$$\mathcal{P}_1 \equiv_p^T \mathcal{P}_2 \text{ definovanou konjunkcí } \mathcal{P}_1 \preceq_p^T \mathcal{P}_2 \wedge \mathcal{P}_2 \preceq_p^T \mathcal{P}_1,$$

která vystihuje vztah "problémy $\mathcal{P}_1$ a $\mathcal{P}_2$ jsou stejně těžké" z abstraktní perspektivy, jež vidí polynomialitu algoritmu jako "atom" (na jehož detaily, jako je stupeň polynomu apod., se nedíváme).

Jeden úkol žádá upřesnění dříve zmiňované úzké vazby výpočetní složitosti optimalizačního problému a jeho rozhodovací verze. Za rozumných (běžně splněných) podmínek platí pro optimalizační problém $\mathcal{O}$ vztah

$$\mathcal{O} \equiv_p^T \mathcal{O}_{dec}.$$

Speciálním případem turingovské polynomiální převeditelnosti je (standardní) *polynomiální převeditelnost* (také nazývaná "polynomial many-one reducibility") mezi rozhodovacími problémy: říkáme, že (rozhodovací problém) $\mathcal{P}_1$ je polynomiálně převeditelný na (rozhodovací problém) $\mathcal{P}_2$, což značíme

$$\mathcal{P}_1 \preceq_p \mathcal{P}_2$$

(tedy bez horního indexu T), jestliže existuje polynomiální algoritmus, který pro instanci I_1 problému $\mathcal{P}_1$ sestrojí instanci I_2 problému $\mathcal{P}_2$ tak, že odpověď (ANO/NE) přiřazená k I_1 v problému $\mathcal{P}_1$ je stejná jako odpověď přiřazená k I_2 v problému $\mathcal{P}_2$.

Pozorování. Jestliže $\mathcal{P}_1 \preceq_p \mathcal{P}_2$ a $\mathcal{P}_2 \preceq_p \mathcal{P}_3$, tak $\mathcal{P}_1 \preceq_p \mathcal{P}_3$ (tedy $\preceq_p$ je tranzitivní).
Jestliže $\mathcal{P}_1 \preceq_p \mathcal{P}_2$ a $\mathcal{P}_2 \in$ P, pak $\mathcal{P}_1 \in$ P. (Tedy jestliže $\mathcal{P}_1 \preceq_p \mathcal{P}_2$ a $\mathcal{P}_1 \notin$ P, pak $\mathcal{P}_2 \notin$ P.)

NP-těžké, NP-úplné a NP-ekvivalentní problémy.
Rozhodovací problém $\mathcal{P}$ je NP-*těžký*, jestliže pro každý problém $\mathcal{P}' \in$ NP platí $\mathcal{P}' \preceq_p \mathcal{P}$; jestliže navíc platí $\mathcal{P} \in$ NP, pak $\mathcal{P}$ je NP-*úplný*.

Připomněli jsme si tuto důležitou větu:

Věta(Cook). SAT je NP-úplný.

> Idea důkazu není těžká, jen technická: Když máme problém $\mathcal{P} \in$ NP, tak k němu existuje nedeterministický Turingův stroj M a konstanty c, k tak, že M rozhoduje problém $\mathcal{P}$ (tedy má alespoň jeden akceptující výpočet pro vstup w právě tehdy, když vstupu w přiřazuje $\mathcal{P}$ odpověď ANO) a pro každý vstup w velikosti n vykoná M nanejvýš $c \cdot n^k$ kroků, tedy také projde nejvýše $c \cdot n^k$ konfiguracemi, jež zahrnují paměť velikosti nejvýše $c \cdot n^k$. Posloupnost takových konfigurací lze přímočaře popsat booleovskou formulí, která mj. obsahuje jednu proměnnou pro každé políčko paměti v každém kroku výpočtu. Konjunkcí (polynomiálně mnoha) jednoduchých "lokálních" vztahů mezi proměnnými přímočaře popíšeme nutnou a postačující podmínku k tomu, aby splňující pravdivostní ohodnocení odpovídalo akceptujícímu výpočtu M pro vstup w.

Obecný (např. optimalizační) problém $\mathcal{P}$ je NP-*těžký*, jestliže pro každý problém $\mathcal{P}' \in$ NP platí $\mathcal{P}' \preceq_p^T \mathcal{P}$. Řekneme, že $\mathcal{P}$ je NP-*lehký*, jestliže existuje problém $\mathcal{P}' \in$ NP takový, že $\mathcal{P} \preceq_p^T \mathcal{P}'$. Když $\mathcal{P}$ je NP-těžký i NP-lehký, tak je NP-*ekvivalentní*.

> NP-úplnost se tedy vztahuje pouze k *rozhodovacím* problémům a (standardní) polynomiální převeditelnosti $\preceq_p$.
> K rozhodovacímu problému $\mathcal{P}$ lze přirozeně definovat *doplňkový problém* co-$\mathcal{P}$ (complement of $\mathcal{P}$): instance zůstávají stejné, ale výstupy ANO/NE jsou prohozeny. (Takže mj. platí co-(co-$\mathcal{P}$) = $\mathcal{P}$.) Např. u problému co-SAT (také označovaného UNSAT) se vlastně ptáme, zda daná formule je nesplnitelná.
> Všimněme si, že platí $\mathcal{P} \preceq_p^T$ co-$\mathcal{P}$ (a co-$\mathcal{P} \preceq_p^T \mathcal{P}$); ovšem $\mathcal{P} \preceq_p$ co-$\mathcal{P}$ obecně neplatí. Proto má smysl zkoumat i třídu co-NP (třídu doplňkových problémů pro problémy z NP), která také obsahuje třídu P, ale je možná různá od NP (za předpokladu P≠NP). Problém co-SAT možná není v NP, ale je ve smyslu našich definic NP-ekvivalentní.
> Poznamenejme ještě, že NP i co-NP jsou podmnožinami množiny PSPACE, která obsahuje problémy řešitelné algoritmy, jimž stačí k běhu polynomiálně omezená paměť (byť třeba dělají exponenciálně mnoho kroků). Např. dříve zmiňovaný problém Q-SAT je PSPACE-úplný (je v PSPACE a každý problém z PSPACE lze na něj polynomiálně převést). Kupodivu není ovšem dokázáno ani to, že PTIME je *vlastní* podtřídou PSPACE, ač to vypadá "hodně pravděpodobně".

Následující (jednoduché) tvrzení ukazuje, proč je zjištění NP-obtížnosti u konkrétních problémů důležité (nemá u nich pak smysl hledat polynomiální řešící algoritmus a musíme k praktickému řešení přistoupit jinak), a zároveň ukazuje způsob umožňující z NP-obtížnosti jednoho problému vyvodit NP-obtížnost jiného.

Tvrzení.

a/ Pokud P≠NP (což vypadá "velmi pravděpodobně"), tak žádný NP-těžký problém není polynomiální.

b/ Když $\mathcal{P}$ je NP-těžký a $\mathcal{P} \preceq_p \mathcal{P}'$, pak $\mathcal{P}'$ je NP-těžký.

Využitím NP-úplnosti (a tedy NP-obtížnosti) problému SAT jsme vyvodili NP-úplnost problému hamiltonovského cyklu, označeného HC: instancí je orientovaný graf a otázkou je, zda existuje (orientovaný) cyklus, který projde každým vrcholem právě jednou. Tento problém je očividně v NP (jak vypadá příslušný nedeterministický algoritmus?); jeho NP-obtížnost jsme vyvodili tak, že jsme ukázali

$$\mathrm{SAT} \preceq_p \mathrm{HC}$$

> K formuli ϕ v CNF s m klauzulemi $C_1, \ldots, C_m$ a n proměnnými $x_1, \ldots, x_n$, kde žádná C_i neobsahuje x_j i $\neg x_j$ (takovou případnou C_i prostě vypustíme), postupně zkonstruujeme orientovaný graf G_ϕ takto:
>
> Vytvoříme "startovací vrchol" s, vrcholy označené $x_1, \neg x_1, \ldots, x_n, \neg x_n$ a $C_1, \ldots, C_m$, a dodáme hrany (s, x_1), $(s, \neg x_1)$, (x_n, s), $(\neg x_n, s)$ a (x_i, x_{i+1}), $(x_i, \neg x_{i+1})$, $(\neg x_i, x_{i+1})$, $(\neg x_i, \neg x_{i+1})$ pro $i = 1, 2, \ldots, n-1$.
>
> Pak dodáme cestu z x_1 do $\neg x_1$ délky $2m+1$ přes nově přidané vrcholy $v_1^1, \ldots, v_{2m}^1$. Pokud se x_1 vyskytuje v klauzuli C_i, přidáme navíc hrany (v_{2i-1}, C_i), (C_i, v_{2i}) (to uděláme pro všechny C_i obsahující x_1).

K cestě z x_1 do $\neg x_1$ přes vrcholy $v_1^1, \ldots, v_{2m}^1$ přidáme (hrany tvořící) opačnou cestu z $\neg x_1$ do x_1 přes vrcholy $v_{2m}^1, \ldots, v_1^1$. Pokud se $\neg x_1$ vyskytuje v klauzuli C_i, přidáme navíc hrany (v_{2i}, C_i), (C_i, v_{2i-1}).

Toto zopakujeme pro x_2, dodáním nových vrcholů $v_1^2, \ldots, v_{2m}^2$ a přidáním příslušných hran, a pak pro $x_3, x_4, \ldots, x_n$.

Proveďte si detailně argumentaci, proč splnitelnost výchozí formule ϕ implikuje existenci hamiltonovského cyklu v zkonstruovaném grafu G_ϕ a proč je tomu i naopak (existence hamiltonovského cyklu v G_ϕ implikuje splnitelnost formule ϕ).

Úkoly nás dále vedou k zamyšlení, jak využít NP-obtížnosti problému HC k prokázání NP-obtížnosti problému HK, tj. problému hamiltonovské kružnice, tedy hamiltonovského cyklu v neorientovaném grafu. Problém HK lze pak jednoduše využít k prokázání toho, že problém TSP (Travelling Salesman Problem) je NP-ekvivalentní (a jeho rozhodovací verze TSP_{dec} je NP-úplná).

Problém TSP je jedním z centrálních problémů kombinatorické optimalizace, pro nějž není znám polynomiální algoritmus. My jsme si jej ilustrovali na jeho speciálním (také NP-ekvivalentním) případu optimálního vrtání děr do desky plošných spojů (jak je to také uvedeno na začátku knihy [5]).

Problém celočíselného lineárního programování (ILP) je NP-ekvivalentní. Jeden úkol žádá ukázat, že již problém řešitelnosti (feasibility) u úloh ILP je NP-těžký; to se dá ukázat snadným převodem ze SAT. (Vlastně tak ukážeme, že $\text{SAT} \preceq_p \text{ILP}_{dec}$, kde ILP_{dec} je rozhodovací verze problému ILP.)

Problém ILP je také NP-lehký, a tedy NP-ekvivalentní, je ale trochu odlišný od jiných NP-ekvivalentních problémů mj. tím, že ukázat horní omezení složitosti, tedy že problém je NP-lehký (což je ekvivalentní tomu, že ILP_{dec} je v NP), je náročnější.

Prokázání příslušnosti k NP je u problémů SAT, TSP_{dec} a spousty dalších snadné: navrhneme polynomiální algoritmus, který příslušné řešení nedeterministicky "uhodne" a pak deterministicky ověří. Nutnou podmínkou polynomiality algoritmu ovšem je, že velikost takového (uhodnutého) řešení je polynomiálně omezena vzhledem k velikosti vstupu, což u ILP_{dec} není zřejmé.

Máme-li rozhodnout, zda $Ax \leq b$ má celočíselné řešení (pro celočíselnou matici A a celočíselný vektor b), pak se nabízí uhodnout konkrétní vektor x a deterministicky ověřit příslušné nerovnosti. Máme ovšem zaručeno, že když nějaké řešení existuje, tak existuje také "malé řešení", tj. řešení, které zapíšeme polynomiálně omezeným počtem bitů (vzhledem k počtu bitů, jimiž jsou popsány A a b) ? Ano, je to zaručeno, a proto problém řešitelnosti ILP (a obecněji problém ILP_{dec}) patří do NP.

Důkaz tohoto faktu zde neprovedeme, jen naznačíme. Vzpomeňte si na obecnou definici determinantu $\det(A)$ čtvercové matice $m \times m$:

$$\det(A) = \sum_{\sigma \in \text{PERM}(m)} sgn(\sigma) \cdot \prod_{i=1}^{m} a_{i,\sigma(i)}$$

kde $\text{PERM}(m)$ je množina všech permutací množiny $\{1, 2, \ldots, m\}$ (tedy bijektivních zobrazení $\sigma : \{1, 2, \ldots, m\} \to \{1, 2, \ldots, m\}$), $sgn(\sigma)$ označuje znaménko permutace σ (tedy

číslo $(-1)^k$, kde k je počet inverzí, tedy dvojic (i,j) splňujících $i<j$ a $\sigma(i)>\sigma(j)$) a a_{ij} označuje prvek matice A v i-tém řádku a j-tém sloupci. Když d je maximální absolutní hodnota čísel v A, tak absolutní hodnotu $\det(A)$ můžeme jistě omezit číslem $m^m \cdot d^m$, k jehož zápisu nám jistě stačí polynomiálně mnoho bitů (vzhledem k počtu bitů popisujících A a b). Vzpomeneme-li si dále na Cramerovo pravidlo a úvahy o bázových řešeních, které jsme prováděli při zkoumání totální unimodularity, existence onoho "malého" řešení $Ax \leq b$ začíná být "vidět".

Nové úkoly.

1. Uvažujte *dopravní problém* v této formě: Máme M skladů, které obsahují aktuálně $s_1, s_2, \ldots, s_M$ množstevních jednotek nějaké komodity (např. tun uhlí). Dále máme N zákazníků, kteří žádají dovézt $d_1, d_2, \ldots, d_N$ jednotek. (Řekněme, že platí $\sum_{1\leq j\leq N} d_j \leq \sum_{1\leq i\leq M} s_i$.) Cena dovozu jednotky komodity ze skladu i zákazníkovi j je c_{ij}. Sestavte odpovídající lineární program minimalizující celkovou cenu převozu (při uspokojení požadavků zákazníků a respektování aktuálního množství ve skladech). Ověřte, že příslušná matice je totálně unimodulární (a vyvoďte odpovídající "praktický" závěr).

2. Ukažte, že varianta problému SAT, u nějž povolujeme jako instance obecné booleovské formule, je polynomiálně převeditelná na variantu, u níž povolujeme jen formule v CNF. (Nápověda. Syntaktický strom formule můžeme přirozeně chápat jako jakýsi logický obvod [vstupy obvodu jsou proměnné ve formuli a hradla odpovídají logickým spojkám; pro každé hradlo zavedeme novou proměnnou]. Stačí tedy popsat korektní ohodnocení obvodu, při němž je na výstupu 1.)

3. Zformulujte problém hamiltonovské kružnice (existuje v neorientovaném grafu cyklus procházející každým vrcholem právě jednou?), označený HK, jako speciální případ problému SAT. (Máte tedy popsat, jak k zadanému grafu sestrojíte [co nejpřímočařeji] booleovskou formuli, která je splnitelná právě tehdy, když graf obsahuje hamiltonovskou kružnici. Ukážete tak, že $\mathrm{HK} \preceq_p \mathrm{SAT}$.)

4. Zformulujte problém TSP (Travelling Salesman Problem), a jeho rozhodovací variantu, tedy problém TSP_{dec}. Pak načrtněte, jak bychom mohli instance problému TSP_{dec} přímočaře popsat jako instance problému SAT.
(Nápověda. Binárně zapsané přirozené číslo lze přímočaře reprezentovat sekvencí booleovských proměnných. Číselný predikát $PLUS(a,b,c)$, který je pravdivý právě když $a+b=c$, lze snadno definovat příslušnou množinou klauzulí [tedy (pod)formulí v CNF].)

5. Uvažujme optimalizační problém $\mathcal{P}$, pro nějž máme polynomiální algoritmus, který k zadané instanci spočítá dolní a horní odhad optima. (To je u všech "praktických" problémů splněno.) Je snadné nahlédnout, že $\mathcal{P}_{dec} \preceq_p^T \mathcal{P}$; ukažte to. Pak ukažte, že $\mathcal{P} \preceq_p^T \mathcal{P}_{dec}$. (Nápověda. Využijte binárního hledání (binary search) v možných hodnotách účelové funkce.)

6. Ukažte, že problém řešitelnosti u úloh ILP (který se ptá, zda pro danou celočíselnou matici A a celočíselný vektor b existuje celočíselný vektor x splňující $Ax \leq b$) je NP-těžký.

7. Navrhněte algoritmus, který má ukázat $\mathrm{HC} \preceq_p \mathrm{HK}$ (tedy polynomiální převeditelnost problému hamiltonovského cyklu v orientovaném grafu na analogický problém v neorientovaném grafu) a prokažte, že váš algoritmus je v tomto smyslu opravdu v pořádku.

8. Zamyslete se nad převodem $\mathrm{HK} \preceq_p \mathrm{TSP}_{dec}$; to by mělo být velmi jednoduché. Snadno byste tak měli i ukázat, že $\mathrm{HK} \preceq_p \mathrm{TSP}^{\Delta}_{dec}$, kde TSP^{Δ} (taky nazývaný metrický TSP) odkazuje k restrikci problému TSP na instance splňující trojúhelníkovou nerovnost (tj. standardní axiom metrických prostorů): při označení $c(\{u, v\})$ pro (nezápornou) cenu hrany $\{u, v\}$ (tj. "vzdálenost" vrcholů u, v) platí $c(\{u, v\}) \leq c(\{u, w\}) + c(\{w, v\})$ pro všechny vrcholy u, v, w.

Týden 12 (od 25.4.)

Neřešili jsme všechny dřívější úkoly (možná se k nim stručně vrátíme, ale každý by si určitě měl promyslet sám). Udělali jsme ale mj. toto:

- Detailně jsme připomněli převod $\mathrm{SAT} \preceq_p \mathrm{HC}$ (který není složitý, ovšem poté, co pochopíme hlavní myšlenku).
- Ukázali jsme snadný převod $\mathrm{HC} \preceq_p \mathrm{HK}$ (každý vrchol nahradíme neorientovanou cestou s třemi vrcholy, původní hranu vstupující do vrcholu pak nasměrujeme na začátek oné cesty a začátek hrany vystupující z vrcholu přesuneme na konec cesty; nakonec odstraníme orientaci hran).
- Ukázali jsme snadný převod $\mathrm{HK} \preceq_p \mathrm{TSP}_{dec}$, dokonce $\mathrm{HK} \preceq_p \mathrm{TSP}^{\Delta}_{dec}$ (v původním grafu dodáme každé hraně váhu 1, a pak doplníme na úplný graf s tím, že nové hrany dostanou váhu 2; jako limit délky okružní cesty položíme počet vrcholů).
- Také jsme probrali snadný převod $\mathrm{SAT} \preceq_p \mathrm{ILP}_{dec}$ (booleovské proměnné x_i prostě chápeme jako celočíselné proměnné s omezením $0 \leq x_i \leq 1$ a pro každou klauzuli dodáme přirozenou podmínku splněnosti: např. pro $(x_1 \vee \neg x_2 \vee \neg x_3)$ je to podmínka $x_1 + (1 - x_2) + (1 - x_3) \geq 1$; ptáme se na řešitelnost (feasibility), účelová funkce nehraje roli [formálně můžeme třeba maximalizovat účelovou funkci $f(x_1, \ldots, x_n) = x_1$ a položit spodní limit 0]).

NP-úplnost max. nezávislé množiny (či kliky) a min. vrcholového pokrytí. Uvedli jsme myšlenku převodu $\mathrm{SAT} \preceq_p \mathrm{MaxIndSet}_{dec}$, kde MaxIndSet (Maximum Independent Set) je optimalizační problém přiřazující neorientovanému grafu $G = (V, E)$ (jakožto vstupu problému) množinu největších nezávislých množin (jakožto množinu příslušných výstupů); množina $V' \subseteq V$ je nezávislá, jestliže mezi žádnými vrcholy $u, v \in V'$ nevede hrana (tedy $\forall u, v \in V' : \{u, v\} \notin E$).

> "Největší" je samozřejmě myšleno vzhledem k počtu prvků.
>
> V (částečném) uspořádání podle inkluze je maximální nezávislá množina každá taková nezávislá množina, která není vlastní podmnožinou jiné nezávislé množiny. Např. ve "hvězdicovém" grafu $(\{0, 1, 2, \ldots, n\}, \{\{0, 1\}, \{0, 2\}, \ldots, \{0, n\}\})$ je v tomto smyslu $\{0\}$ maximální nezávislou množinou. Nějakou maximální nezávislou množinu v tomto smyslu samozřejmě sestrojíme velmi rychle: postupně označujeme vrcholy tak, aby mezi označenými nebyla hrana, až už další vrchol označit nelze.
>
> Slovem "maximální" v našem kontextu se odkazujeme ke kvazi-uspořádání podle počtu prvků (relace kvazi-uspořádání je reflexivní a tranzitivní, ne nutně antisymetrická); proto bychom spíš měli používat "největší" (anglicky maximum-cardinality), ale k nedorozumění by nemělo docházet. (V hvězdicovém grafu je pro $n \geq 2$ jediná maximální množina v tomto smyslu, a sice $\{1, 2, \ldots, n\}$. Např. v úplném bipartitním grafu se stejně velkými partitami jsou maximální [tj. největší] nezávislé množiny dvě.)
>
> Připomeňme si, že systém nezávislých množin na grafu netvoří matroid, tedy přímočarý hladový algoritmus ke konstrukci maximální (tj. největší) nezávislé množiny nemáme. Jelikož si teď odvozujeme, že problém MaxIndSet je NP-ekvivalentní (problém $\mathrm{MaxIndSet}_{dec}$ je NP-úplný), tak to znamená, že nemáme ani jakýkoli jiný polynomiální algoritmus řešící tento problém, tedy vracející pro $G = (V, E)$ nějakou nezávislou množinu s největším možným počtem prvků.

Myšlenka převodu $\mathrm{SAT} \preceq_p \mathrm{MaxIndSet}_{dec}$ je jednoduchá:

> Ke každé klauzuli (ve výchozí formuli v CNF) sestrojíme úplný graf, jehož vrcholy odpovídají literálům v klauzuli; tyto grafy dáme dohromady (vezmeme jejich disjunktní sjednocení) a navíc spojíme hranou každé dva "vzájemně nekompatibilní" vrcholy, tj. takové vrcholy, kde jeden odpovídá literálu x_j a druhý literálu $\neg x_j$. Jako limit m vezmeme počet klauzulí, tedy počet oněch úplných podgrafů, z nichž každý může do nezávislé množiny přispět nanejvýš jedním vrcholem.

Důkaz korektnosti konstrukce:

i/ Když je výchozí formule splnitelná, tedy existuje pravdivostní ohodnocení, při němž každá klauzule obsahuje alespoň jeden pravdivý literál, tak v sestrojeném grafu očividně existuje nezávislá množina I velikosti m (z každého úplného podgrafu odpovídajícího jedné klauzuli do I zařadíme jeden vrchol odpovídající pravdivému literálu).

ii/ Naopak, když v sestrojeném grafu existuje nezávislá množina I velikosti m (pro každou klauzuli tedy její úplný podgraf přispěl jedním vrcholem), tak jistě existuje pravdivostní ohodnocení, při němž jsou všechny literály odpovídající vrcholům z I pravdivé; toto zobrazení také očividně splňuje výchozí formuli.

U problému MaxClique jde o *maximální kliku* v grafu (tedy úplný podgraf s největším možným počtem vrcholů). Je zřejmé, že když $V' \subseteq V$ je nezávislá množina v grafu $G = (V, E)$, tak V' je klika v "doplňkovém grafu"

$$G' = (V, E'), \text{ kde } E' = \{\{u, v\} \mid u \neq v \text{ a } \{u, v\} \notin E\},$$

a také naopak (když V' je klika v G, tak V' je nezávislá množina v G'). Je tedy očividné, že

$$\mathrm{MaxIndSet}_{dec} \equiv_p \mathrm{MaxClique}_{dec},$$

kde $\mathcal{P} \equiv_p \mathcal{P}'$ znamená, že $\mathcal{P} \preceq_p \mathcal{P}'$ a $\mathcal{P}' \preceq_p \mathcal{P}$.

Problém $\mathrm{MaxClique}_{dec}$ je tedy také NP-úplný. (Příslušnost k NP je jako obvykle zřejmá.)

> Standardně uvažujeme grafy bez smyček (kde smyčka je hrana, jejíž konce jsou stejným vrcholem). Takže smyčky nejsou ani v původním grafu, ani v doplňkovém (pokud někdy neřekneme jinak).
>
> Ještě poznamenejme, že explicitně nerozebíráme praktické problémy, které motivují uvedené pojmy, ale není těžké si souvislosti z praxe uvědomit. Např. pojem nezávislé množiny se přirozeně objeví, když máme sestavit např. nějaký výbor z osob, které nejsou ve vzájemném osobním vztahu (způsobujícím konflikt zájmů), apod.

O *vrcholovém pokrytí* jsme již mluvili; pro graf $G = (V, E)$ je $C \subseteq V$ vrcholovým pokrytím (vertex cover), jestliže každá hrana je incidentní s C (tedy $\forall \{u, v\} \in E : \{u, v\} \cap C \neq \emptyset$). Snadno nahlédneme:

Pozorování V grafu $G = (V, E)$ je $V' \subseteq V$ nezávislou množinou právě tehdy, když $V \setminus V'$ je vrcholovým pokrytím.

Z toho plyne, že v grafu s n vrcholy existuje nezávislá množina velikosti (alespoň) m právě tehdy, když v něm existuje vrcholové pokrytí velikosti (nanejvýš) $n-m$. Proto máme

$\text{MaxIndSet}_{dec} \equiv_p \text{MinVertexCover}_{dec}$; problém MinVertexCover přiřazuje grafu vrcholová pokrytí s nejmenším možným počtem vrcholů. Rozhodovací problém $\text{MinVertexCover}_{dec}$ je tedy také NP-úplný (a optimalizační problém MinVertexCover je NP-ekvivalentní).

Aproximační algoritmy a stupně aproximovatelnosti optimalizačních problémů.

Dalším tématem našeho kurzu (tématem, do kterého samozřejmě zase jen "povrchově zabrousíme") jsou aproximační algoritmy pro těžké (lze přesněji říci NP-těžké) optimalizační problémy.

Připomeňme, že *optimalizační problém* je určen

a/ množinou vstupů (instancí problému),

b/ přiřazením množiny *přípustných řešení* každému vstupu,

c/ účelovou funkcí, přiřazující vstupu a jemu příslušnému přípustnému řešení nějakou reálnou (často nezápornou) hodnotu,

d/ a informací, zda se jedná o maximalizaci či minimalizaci.

Zamýšlenými výstupy problému k danému vstupu jsou pak *optimální řešení*, tedy přípustná řešení, pro něž je hodnota účelové funkce maximální či minimální. Algoritmus řešící daný optimalizační problém má tedy k zadanému vstupu vrátit nějaké optimální řešení či sdělení, že takové řešení neexistuje.

> Optimální řešení neexistuje buď proto, že neexistuje žádné přípustné řešení, nebo proto, že existuje nekonečně mnoho přípustných řešení a účelová funkce na nich není omezená [shora pro maximalizaci a zdola pro minimalizaci]. Připomeňme, že u problémů kombinatorické optimalizace je množina přípustných řešení konečná, neboť tato řešení jsou jistými podmnožinami nějaké konečné množiny; při systematickém prohledávání všech řešení hrubou silou ovšem typicky dojde ke "kombinatorické explozi", což znamená, že počet řešení roste exponenciálně vzhledem k velikosti vstupu.

Již víme, že u NP-těžkých problémů nemůžeme doufat v polynomiální algoritmy nalézající optima. Jelikož tyto problémy musíme v praxi nějak řešit, je přirozené hledat např. aproximační algoritmy.

Aproximační algoritmus pro daný optimalizační problém je rychlý (rozuměj polynomiální) algoritmus, který pro každý vstup vrátí nějaké přípustné řešení; to nemusí být optimální, jen optimum nějak "aproximuje". Naší snahou samozřejmě je, aby algoritmus aproximoval co nejlépe, tedy aby se hodnoty účelové funkce pro jím vydaná přípustná řešení co nejvíce blížila optimu.

Aproximační poměr. Přirozená možnost, jak zachytit kvalitu aproximačního algoritmu, je zkoumání tzv. aproximačního poměru, což teď objasníme.

Mějme optimalizační problém $\mathcal{O}$ (třeba TSP) a aproximační algoritmus A pro problém $\mathcal{O}$ (třeba hladový algoritmus pro TSP: vždy jeď do nejbližšího města, které jsi dosud nenavštívil). Předpokládejme, že účelová funkce dává jen kladné hodnoty a optimum pro každý vstup existuje; hodnotu účelové funkce pro vstup x a jeho optimální řešení označíme $v_{opt}(x)$ (u TSP je to tedy délka nejkratší okružní cesty pro graf popsaný vstupem x).

Ke vstupu x problému $\mathcal{O}$ (v případě TSP je tedy x popisem úplného grafu s hranami ohodnocenými nezápornými [řekněme celými] čísly) vydá algoritmus A přípustné řešení $s_A(x)$

(v našem příkladu tedy nějakou permutaci $(i_1, i_2, \ldots, i_n)$ množiny $\{1, 2, \ldots, n\}$, kde n je počet [očíslovaných] měst, tedy vrcholů grafu). Tomuto řešení $s_A(x)$ přísluší hodnota účelové funkce, označená $v(x, s_A(x))$ (v našem příkladu je to $d(i_1, i_2) + d(i_2, i_3) + \cdots + d(i_{n-1}, i_n) + d(i_n, i_1)$ kde $d(i, j)$ je "vzdálenost měst" i, j, která je určena vstupem x).

Aproximační poměr (approximation ratio) $\text{AR}_{A,\mathcal{O}}(x)$ algoritmu A pro problém $\mathcal{O}$ a vstup x je

- $\frac{v(x,s_A(x))}{v_{opt}(x)}$, jestliže jde o minimalizaci, a
- $\frac{v_{opt}(x)}{v(x,s_A(x))}$, jestliže jde o maximalizaci.

Tento poměr je tedy nutně větší nebo roven 1 (poměr 1 znamená, že algoritmus pro x našel optimum). Pro funkci $r : \mathbb{N} \to \mathbb{R}_{\geq 1}$ (kde $\mathbb{R}_{\geq 1}$ je množina reálných čísel větších nebo rovných 1) řekneme, že aproximační algoritmus A pro problém $\mathcal{O}$ má aproximační poměr $r(n)$, říkáme také, že se jedná o

$$r(n)\text{-}\textit{aproximační algoritmus},$$

jestliže pro každý vstup x platí $\text{AR}_{A,\mathcal{O}}(x) \leq r(size(x))$ (kde $size(x)$ je jako obvykle délka x neboli počet bitů potřebných pro zápis vstupu x v dohodnuté reprezentaci).

> Přesněji bychom měli říkat "má aproximační poměr *nanejvýš* $r(n)$", neboť funkce $r(n)$ je horním odhadem skutečného poměru, který může být těžko analyzovatelnou funkcí. Speciálně důležitý je případ kdy $r(n)$ je konstantní funkcí, tedy $r(n) = c \in \mathbb{R}_{\geq 1}$.
>
> Mohlo by nás napadnout zkoumat i aproximační rozdíl $|v(x, s_A(x)) - v_{opt}(x)|$. Nedává to ale praktický smysl; kdyby např. existoval pro nějaký NP-těžký problém aproximační algoritmus s konstantním aproximačním rozdílem, tak by platilo P=NP (a ke každému NP-ekvivalentnímu problému by existoval polynomiální algoritmus nalézající optimum).

Třídy problémů $\mathbf{APX}(r(n))$, $\mathbf{APX}(c)$, $\mathbf{APX}$.
Výrazem $\text{APX}(r(n))$ označujeme třídu optimalizačních problémů, pro něž existují $r(n)$-aproximační algoritmy. Pro konstantní funkce tak dostáváme např. třídy $\text{APX}(\frac{3}{2})$, $\text{APX}(2)$, atd. Všechny problémy řešitelné aproximačními algoritmy s konstantními poměry shrnujeme do třídy

$$\text{APX} = \bigcup_{c \in \mathbb{N}} \text{APX}(c).$$

Obecný TSP je špatně aproximovatelný (TSP$\notin$ APX), ale TSP$^\Delta \in \mathbf{APX}(\frac{3}{2})$.
V našich dalších vyjádřeních někdy tiše předpokládáme, že P≠NP. Kdyby totiž platilo P=NP, pak by i každý NP-ekvivalentní optimalizační problém patřil do $\text{APX}(1)$.

Problém, který nepatří do APX (za předpokladu P≠NP), je považován za špatně aproximovatelný.

> Všechna taková vyjádření jsou samozřejmě velmi hrubá z hlediska praktického řešení; i u takto špatně aproximovatelného problému je samozřejmě možné, že běžně řešené instance problému jsou aproximovány dobře a rychle, např. některými heuristickými přístupy.

TSP $\notin \mathbf{APX}(2^n)$. Ukázali jsme si, že (obecný) TSP nejenže nepatří do APX, ale nepatří ani do $\text{APX}(2^n)$ (vše samozřejmě za předpokladu P≠NP):

Vzpomeňme si na převod $\mathrm{HK} \preceq_p \mathrm{TSP}_{dec}$, kde v původním grafu dodáme každé hraně váhu 1 a pak doplníme na úplný graf s tím, že nové hrany dostanou váhu 2. Když místo 2 dáme na nové hrany váhu $n \cdot 2^n$, kde n je počet vrcholů, tak jsme vytvořili tuto "mezeru" mezi případem s odpovědí ANO (hamiltonovská kružnice existuje) a případem s odpovědí NE:

a/ když v původním grafu existuje hamiltonovská kružnice, tak v zúplněném váženém grafu existuje hamiltonovská kružnice (okružní cesta) délky n;

b/ když v původním grafu neexistuje hamiltonovská kružnice, tak v zúplněném váženém grafu má nejkratší hamiltonovská kružnice délku minimálně $(n-1) + n \cdot 2^n$.

Kdyby tedy pro TSP existoval 2^n-aproximační algoritmus, tak v případě a/ by pro výsledný úplný vážený graf (kde optimum účelové funkce je n) vydal přípustné řešení s hodnotou maximálně $n \cdot 2^n$; v případě b/ by vydal řešení s hodnotou větší než $n \cdot 2^n$ (když $n \geq 2$). Tento algoritmus bychom tak mohli používat k rychlému (tj. polynomiálnímu) rozhodování problému HK, což by implikovalo P=NP.

Pro onu "mezeru" (oddělení případů s hamiltonovskou kružnicí a bez ní) jsme využili nemetrického TSP; přiřazení váhy 2 novým hranám neporušuje trojúhelníkovou nerovnost, ale přiřazení váhy $n \cdot 2^n$ ji obecně porušuje.

Teď ukážeme, že metrický TSP, tedy TSP^{Δ}, dobře aproximovatelný (v našem smyslu) je.

$\mathrm{TSP}^{\Delta} \in \mathbf{APX}(2)$. Ukážeme, že pro TSP^{Δ} existuje 2-aproximační algoritmus (což později ještě zlepšíme).

Uvažujme úplný graf $G = (V, E, c)$ s nezápornými váhami (délkami) $c(e)$ hran $e \in E$, kde c je de facto metrika, tedy splňuje trojúhelníkovou nerovnost ($c(\{u,v\}) + c(\{v,w\}) \geq c(\{u,w\})$).

Speciálně to implikuje, že *vypouštění vrcholů z posloupnosti nezvětšuje její váhu*, čímž rozumíme toto: Pro posloupnost vrcholů $j_1, j_2, \ldots, j_k$ definujeme její váhu jako součet $c(\{j_1,j_2\}) + c(\{j_2,j_3\}) + \cdots + c(\{j_{k-1},j_k\})$ (pro pořádek doplníme, že váha posloupnosti s jedním nebo žádným prvkem je nula). Díky trojúhelníkové nerovnosti platí $c(\{j_\ell, j_{\ell+1}\}) + c(\{j_{\ell+1}, j_{\ell+2}\}) \geq c(\{j_\ell, j_{\ell+2}\})$ (pro libovolné $\ell \in \{1,2,\ldots,k-2\}$); tedy vypuštěním vrcholu z posloupnosti její váha nevzroste (klesne nebo zůstane stejná).

Všimněme si teď, že když z libovolné hamiltonovské kružnice v G vyřadíme jednu hranu, dostaneme kostru grafu. Délka nejkratší hamiltonovské kružnice (tedy "okružní cesty") je tedy větší nebo rovna váze minimální kostry grafu G; stručně vyjádřeno, $c(\textsc{mst}) \leq v_{opt}(G)$ (kde MST je minimální kostra).

Uvažujme algoritmus A_1, který pro úplný graf $G = (\{1,2,\ldots,n\}, E, c)$ s nezápornými váhami $c(e)$ na hranách $e \in E$ nejprve nalezne minimální kostru MST ("matroidovým" hladovým algoritmem) a pak vydá permutaci vrcholů $(i_1, i_2, \ldots, i_n)$ odpovídající jejich prvnímu navštívení při prohledání kostry MST do hloubky. Polynomialita algoritmu A_1 je zřejmá; ukážeme teď, že A_1 je 2-aproximační algoritmus pro TSP^{Δ}.

Strom MST má n vrcholů a $n-1$ hran. Konkrétní běh algoritmu "depth-first-search" na MST postupně navštěvuje vrcholy $j_1, j_2, \ldots, j_{2n-1}$, kde $j_1 = j_{2n-1}$ a každý vrchol se v této posloupnosti vyskytuje alespoň jednou (list MST je navštíven jen jednou, vnitřní vrchol je kromě prvního navštívení [z předchůdce] navíc navštíven i při návratech z jeho následníků). Váha posloupnosti $j_1, j_2, \ldots, j_{2n-1}$ se tedy rovná $2 \cdot c(\mathrm{MST})$ (tedy dvojnásobku váhy kostry, neboť váha každé hrany v MST se při běhu "depth-first-search" započte dvakrát). Když teď v posloupnosti $j_1, j_2, \ldots, j_{2n-1}$ pro každý vrchol i ponecháme pouze jeho první výskyt (a další jeho případné výskyty vypustíme), s výjimkou vrcholu j_1, pro nějž ponecháme i

jeho poslední výskyt j_{2n-1}, dostaneme jistou posloupnost $i_1, i_2, \ldots, i_n, i_1$, kde $i_1 = j_1 = j_{2n-1}$ a $(i_1, i_2, \ldots, i_n)$ je nějakou permutací množiny $\{1, 2, \ldots, n\}$; váha této posloupnosti je nanejvýš $2 \cdot c(\text{MST})$ (protože "vypouštění vrcholů nezvětšuje váhu") a je to vlastně hodnota $v(G, s_{A_1}(G))$. Máme tedy

$$c(\text{MST}) \leq v_{opt}(G) \leq v(G, s_{A_1}(G)) \leq 2 \cdot c(\text{MST}),$$

z čehož plyne, že $\frac{v(G,s_{A_1}(G))}{v_{opt}(G)} \leq 2$. Pro přesnost bychom měli připomenout, že jsme v zadání nevyloučili váhy nula; pokud $v_{opt}(G) = 0$ (a náš zlomek tedy nemá smysl), platí také $v(G, s_{A_1}(G)) = 0$ (A_1 najde optimum).

$\text{TSP}^\Delta \in \textbf{APX}(\frac{3}{2})$. Alespoň zaznamenejme, že algoritmus A_1 se dá doplnit na sofistikovanější algoritmus A_2, který je $\frac{3}{2}$-aproximačním algoritmem pro TSP^Δ. (A_2 má samozřejmě také polynomiální časovou složitost, ale omezenou polynomem s vyšším stupněm než je tomu u A_1.) V následující vsuvce algoritmus A_2 nastíníme (pro hlubší zájemce).

> Christofides (autor A_2) využil polynomiality problému nalezení perfektního párování s minimální váhou v grafu s (nezápornými) váhami na hranách.
>
> > My jsme dříve zmínili polynomialitu problému "maximum-weight matching"; dokázali jsme ji jen v případě bipartitních grafů a pro obecné grafy jsme ukázali jen polynomialitu problému "maximum-cardinality matching". Perfektní párování je párování, které je také hranovým pokrytím (edge cover), což znamená, že každý vrchol je incidentní s nějakou hranou v onom párování. Problémy "maximum-weight matching" (zahrnující verzi "nalezni párování s k hranami a maximální váhou") a "minimum-weight perfect matching" spolu úzce souvisejí a oba jsou polynomiální.
>
> Algoritmus A_2 dostane stejný vstup $G = (V, E, c)$ jako A_1 a postupuje takto:
>
> 1. Sestroj minimální kostru MST grafu G.
> 2. Najdi množinu vrcholů $L \subseteq V$, které mají ve stromu MST lichý stupeň (pro takový vrchol je počet hran v MST, které jsou s ním incidentní, lichý); počet prvků L je sudý (vidíte proč?).
> 3. Najdi na podgrafu grafu G indukovaném množinou vrcholů L perfektní párování S s minimální váhou $c(S)$ (to jistě existuje, protože graf G je úplný a počet prvků L je sudý).
> 4. Uvažuj podgraf G' grafu G s plnou množinou vrcholů $V = \{1, 2, \ldots, n\}$, ale jen s hranami z MST a z S; přitom každou případnou hranu $e \in \text{MST} \cap S$ nahraď dvěma kopiemi e', e'' (s váhou $c(e)$). (Dovolíme tak dvě různé hrany mezi stejnou dvojicí vrcholů.) Celková váha hran v grafu G' je $c(\text{MST}) + c(S)$.
> 5. Podle (snadno odvoditelné) Eulerovy věty v grafu G' (který je souvislý a v němž každý vrchol má sudý stupeň) existuje, a lze snadno sestrojit, cyklus procházející každou hranou právě jednou. Sestroj takový cyklus; tomu přísluší jistá posloupnost navštívených vrcholů $j_1, j_2, \ldots, j_k$, kde $j_1 = j_k$ a každý vrchol i má v posloupnosti alespoň jeden výskyt. Váha posloupnosti $j_1, j_2, \ldots, j_k$ je $c(\text{MST}) + c(S)$.
> 6. Vydej posloupnost $(i_1, i_2, \ldots, i_n)$ vzniklou z $j_1, j_2, \ldots, j_k$ tak, že pro každý vrchol i ponecháš v posloupnosti jen jeho první výskyt.
>
> Je tedy $v(G, s_{A_2}(G)) \leq c(\text{MST}) + c(S)$ (protože vypouštění nezvyšuje váhu díky trojúhelníkové nerovnosti a posloupnost $i_1, i_2, \ldots, i_n, i_1$ tak má nanejvýš takovou váhu jako $j_1, j_2, \ldots, j_k$).

K prokázání toho, že algoritmus A_2 je $\frac{3}{2}$-aproximačním algoritmem pro TSP^Δ tak stačí ukázat, že $c(\mathrm{MST})+c(S) \leq \frac{3}{2}\, v_{opt}(G)$; jelikož již víme, že $c(\mathrm{MST}) \leq v_{opt}(G)$, stačí ukázat, že $c(S) \leq \frac{1}{2}\, v_{opt}(G)$. Uvažujme tedy nějakou nejkratší okružní cestu, tedy hamiltonovskou kružnici v G s minimální váhou, tedy s váhou $v_{opt}(G)$. Vrcholy z L se na té kružnici vyskytují v nějakém pořadí $j_1, j_2, \ldots, j_{|L|}$ (kde $|L|$ je sudé); díky trojúhelníkové nerovnosti je součet

$$c(\{j_1, j_2\}) + c(\{j_2, j_3\}) + \cdots + c(\{j_{|L|-1}, j_{|L|}\}) + c(\{j_{|L|}, j_1\})$$

nanejvýš roven $v_{opt}(G)$. Tento součet je ovšem součtem vah dvou perfektních párování pro L (totiž párování $\{\{j_1, j_2\}, \{j_3, j_4\}, \cdots \{j_{|L|-1}, j_{|L|}\}\}$ a párování $\{\{j_2, j_3\}, \{j_4, j_5\}, \cdots \{j_{|L|-2}, j_{|L|-1}\}, \{j_{|L|}, j_1\}\}$). Jedno z těchto perfektních párování pro L má tedy váhu nanejvýš $\frac{1}{2}\, v_{opt}(G)$; to znamená, že i váha $c(S)$ perfektního párování pro L s minimální váhou je nanejvýš rovna $\frac{1}{2}\, v_{opt}(G)$.

Takže víme, že $\mathrm{TSP}^\Delta \in \mathrm{APX}(1.5)$; nevíme ovšem, zda to jde lépe, tedy zda $\mathrm{TSP}^\Delta \in \mathrm{APX}(c)$ pro nějaké $c < 1.5$; ani nevíme, zda by takové zlepšení implikovalo P=NP.

Znovu poznamenejme, že onen aproximační poměr 1.5 se vztahuje k nejhorším případům (worst-case), na konkrétních praktických instancích algoritmus může dávat (a zřejmě také dává) výsledky daleko bližší optimu.

MinVertexCover $\in$ APX(2). Pro (NP-ekvivalentní) problém minimálního vrcholového pokrytí (pro graf $G = (V, E)$ hledáme $C \subseteq V$ tak, že každá hrana $e \in E$ je pokryta C, tedy incidentní s C) nás asi jako první aproximační algoritmus napadne hladový algoritmus tohoto typu:

buduj C tak, že do něj postupně zařazuješ takové vrcholy, které pokrývají co nejvíce dosud nepokrytých hran.

Ukážeme nejprve, že tento algoritmus nemá konstantní aproximační poměr, neprokazuje tedy, že MinVertexCover $\in$ APX; teprve pak jiným algoritmem prokážeme, že MinVertexCover $\in$ APX(2).

Hladovost pro MinVertexCover není nejlepší.
Pro zvolené $m \in \mathbb{N}$ (představme si velké m) konstruujeme postupně jistý graf G_m. Vezmeme jako "základní vrcholy" $v_1, \ldots, v_m$ a dodáme vrcholy $v_1^1, \ldots, v_m^1$ a hrany $\{v_i^1, v_i\}$ (pro $i = 1, 2, \ldots, m$). Základní vrcholy tvoří teď (jedno) minimální vrcholové pokrytí (velikosti m) v dosud sestrojeném grafu.

Přidáme $v_1^3, v_2^3, \ldots, v^3_{\lfloor \frac{m}{3} \rfloor}$ a hrany $\{v_1^3, v_1\}, \{v_1^3, v_2\}, \{v_1^3, v_3\}$ (tedy v_1^3 je hranami spojen s první trojicí základních vrcholů), dále přidáme hrany $\{v_2^3, v_4\}, \{v_2^3, v_5\}, \{v_2^3, v_6\}$ (tedy v_2^3 je spojen s druhou trojicí), atd.

Pak dodáme $v_1^4, v_2^4, \ldots, v^4_{\lfloor \frac{m}{4} \rfloor}$ a hrany $\{v_1^4, v_1\}, \{v_1^4, v_2\}, \{v_1^4, v_3\}, \{v_1^4, v_4\}$ (tedy v_1^4 je spojen s první čtveřicí základních vrcholů), dále v_2^4 spojíme s druhou čtveřicí, atd. Takto pokračujeme s přidáváním $v_1^j, v_2^j, \ldots, v^j_{\lfloor \frac{m}{j} \rfloor}$ pro $j = 3, 4, \ldots, m$. Po dodání v_1^m a jeho spojení se všemi základními vrcholy je graf G_m hotov.

Vidíme, že základní vrcholy stále tvoří vrcholové pokrytí (velikosti m). Snadno ovšem ověříme, že náš hladový algoritmus by do C nejprve zařadil vrchol v_1^m, pak postupně všechny vrcholy v_i^{m-1} (s horním indexem $m-1$), pak všechny v_i^{m-2}, atd.; po zařazení všech v_i^3 do C by

algoritmus ještě zařadil posledních m vrcholů, třeba těch základních. Vrátil by tedy vrcholové pokrytí C velikosti

$$\left\lfloor \frac{m}{m} \right\rfloor + \left\lfloor \frac{m}{m-1} \right\rfloor + \cdots + \left\lfloor \frac{m}{3} \right\rfloor + m$$

ač optimální hodnota $v_{opt}(G)$ by byla m. Lze snadno nahlédnout, že aproximační poměr pro takto sestrojené instance je tedy $\Theta(H(m))$ kde $H(m)$ je funkce vracející m-té harmonické číslo pro argument m:

$$H(m) = \sum_{i=1}^{m} \frac{1}{i} = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{m}.$$

Aproximační poměr na těchto instancích je tedy $\Theta(\ln m)$; není tedy konstantní, ale roste s rostoucí velikostí instancí (byť pomalu, totiž jako přirozený logaritmus).

> Připomeňme, že $f(n) \in \Theta(g(n))$ (také někdy značeno $f(n) = \Theta(g(n))$) znamená, že $f(n) \in O(g(n))$ a $g(n) \in O(f(n))$. Neformálně řečeno to znamená, že $f(n)$ "roste stejně rychle jako" $g(n)$ (až na konstantní faktor).
>
> Fakt $H(m) \in \Theta(\ln m)$ lze nahlédnout z grafu funkce $f(x) = \frac{1}{x}$ a jejího integrálu, připomeneme-li si, že $\ln x$ je primitivní funkce k funkci $\frac{1}{x}$, tedy, že derivace $\ln x$ je $\frac{1}{x}$.

2-*aproximační algoritmus pro MinVertexCover.*
Algoritmus A prokazující, že MinVertexCover $\in$ APX(2), je jednoduchý:

> zkonstruuj jakékoli párování S, které je maximální vzhledem k inkluzi (nemusí tedy obsahovat největší možný počet hran, ale nedá se rozšířit na větší párování), což uděláme velmi snadno "hladově"; pak pro každou hranu z S zařaď oba její konce do C; množina C je pak jistě vrcholovým pokrytím velikosti $2 \cdot |S|$.

Optimální vrcholové pokrytí (tj. pokrytí s nejmenším možným počtem vrcholů) nutně obsahuje alespoň jeden konec každé hrany z S, takže jeho velikost je minimálně $|S|$. (Máme tedy $\frac{v(G,s_A(G))}{v_{opt}(G)} \leq 2$.)

> Kupodivu tento jednoduchý algoritmus dává nejlepší známý aproximační poměr. Bylo také ukázáno, že kdyby MinVertexCover patřil do APX(1.36), tak by platilo P=NP.
>
> Ještě také můžeme poznamenat, že příbuzný problém MinEdgeCover, tj. problém nejmenšího hranového pokrytí, v němž nám jde o nejmenší množinu hran S takovou, že každý vrchol je incidentní s nějakou hranou z S, je polynomiální. To ve zbytku této vsuvky prokážeme.
>
> Uvažujme graf $G = (V, E)$ bez izolovaných vrcholů (tedy každý vrchol je koncem nějaké hrany). Vzpomeňme si, že jsme si ukazovali polynomiální algoritmus, který k zadanému grafu sestrojí maximální párování M (maximum-cardinality matching), tedy párování s největším možným počtem hran. Použijme takovou množinu hran M jako základ a doplňme ji na hranového pokrytí S tak, že pro každý nepokrytý vrchol u (tj. neincidentní s M) přidáme jednu hranu incidentní s u. (Každá hrana incidentní s nepokrytým vrcholem má druhý konec nutně pokrytý nějakou hranou v M; připomeňme rovněž, že izolované vrcholy v našem grafu nejsou.)
>
> Dostali jsme tedy hranové pokrytí S velikosti $|M| + (|V| - 2|M|)$. Ověřme, že se jedná o minimální hranové pokrytí. Uvažujme tedy nějaké minimální hranové pokrytí $X \subseteq E$. Vezměme podmnožinu $Y \subseteq X$, která je párováním a má největší možný počet prvků.

Vrcholy neincidentní s Y jsou tedy pokryty hranami z X, které jsou napojeny na Y (mají jeden konec incidentní s Y). Kdyby Y nebyla maximálním párováním (maximum-cardinality matching) v G, tak by existovala alternující cesta P z červeného vrcholu v_1 do červeného vrcholu v_2, když hrany v Y a s nimi incidentní vrcholy chápeme jako modré a ostatní hrany a vrcholy chápeme jako červené (alternující cesta střídá červené a modré hrany); předpokládejme, že taková cesta P existuje. Když nyní z X odstraníme dvě hrany, které pokrývaly v_1 a v_2, a dále z X odstraníme modré hrany z oné alternující cesty P, a naopak přidáme červené hrany z P, tak počet hran ve výsledném X' je o jedna menší než v X, ale X' je rovněž hranovým pokrytím – spor.
Takže Y je maximálním párováním v G a náš algoritmus nutně našel párování M, kde $|M| = |Y|$, a pak dodal $|V| - 2|M|$ hran k pokrytí zbylých vrcholů; vytvořené hranové pokrytí S má tedy stejný počet hran jako ono uvažované optimální X.

2-aproximační algoritmus pro MinWeightVertexCover.
Využitím lineárního programování snadno navrhneme 2-aproximační algoritmus i pro MinWeightVertexCover; u tohoto problému má každý vrchol v vstupního grafu nezápornou váhu, či cenu, $c(v)$ a my hledáme vrcholové pokrytí s nejmenší váhou.

Uvažme tuto úlohu LP vytvořenou ke grafu $G = (V, E, c)$:

Každému vrcholu v dodáme proměnnou x_v s omezením $0 \leq x_v \leq 1$; pro každou hranu $\{u, v\} \in E$ dodáme nerovnici $x_u + x_v \geq 1$; minimalizujeme $\sum_{v \in V} c(v) x_v$.

Jako úloha ILP (celočíselného LP) to přesně vystihuje problém MinWeightVertexCover. ILP je ovšem NP-ekvivalentní, jak víme. Relaxovanou úlohu však můžeme vyřešit polynomiálním algoritmem pro LP. Takto dospějeme k optimálnímu řešení x^*, které nemusí být celočíselné, ale pro každou hranu $\{u, v\}$ máme jistě splněno $x^*_u \geq \frac{1}{2}$ nebo $x^*_v \geq \frac{1}{2}$. Pak ovšem

$$C = \{v \in V \mid x^*_v \geq \frac{1}{2}\}$$

je vrcholovým pokrytím a jeho váha $\sum_{v \in C} c(v)$ je očividně maximálně dvojnásobkem relaxovaného optima $\sum_{v \in V} c(v) x^*_v$, a tím spíše maximálně dvojnásobkem skutečného (celočíselného) optima.

Problém množinového pokrytí MinSetCover.
Problém MinSetCover je zobecněním problému MinVertexCover. Neformálně řečeno: máme množinu X "schopností" (skills) a množinu osob $\mathcal{F}$, z nichž každá má nějaké schopnosti; máme nalézt co nejmenší tým $C \subseteq \mathcal{F}$ tak, aby všechny schopnosti z X byly týmem pokryty.

Formálně: vstupem je konečná množina X a množina $\mathcal{F} \subseteq 2^X$, kde $\bigcup \mathcal{F} = X$; hledáme $C \subseteq \mathcal{F}$ tak, že $\bigcup C = X$ a $|C|$ je minimální.

Výrazem 2^X označujeme množinu všech podmnožin množiny X; ekvivalentní interpretace chápe 2^X jako množinu všech zobrazení množiny X do množiny 2, kde $2 = \{0, 1\}$ (což je standardní při definici čísel [či ordinálů] v teorii množin).

Napíšeme-li $\bigcup Y$, rozumí se tím, že prvky Y jsou množiny a $\bigcup Y$ označuje sjednocení všech množin v Y, tedy $\bigcup Y = \{x \mid x \in M \text{ pro nějaké } M \in Y\}$.

Analýza hladového algoritmu (ber vždy osobu, která má nejvíc schopností z těch dosud nepokrytých) je součástí jednoho referátu-projektu. Aproximační poměr zde vyjde $\Theta(\ln n)$

(konkrétněji $1+\ln n$), jako u hladového postupu pro MinVertexCover. U problému MinSetCover je ovšem známo, že nemůžeme čekat zásadní zlepšení; kdyby totiž MinSetCover patřil do $\mathrm{APX}(c \ln n)$ pro jisté $c > 0$, tak by platilo P=NP.

Nové úkoly.

1. Vysvětlete, co by musel splňovat algoritmus, který by byl 2-aproximačním algoritmem pro problém MaxClique.

2. Ukázali jsme těsný vztah mezi problémy MaxClique, MaxIndSet a MinVertexCover. Myslíte, že z existence 2-aproximačního algoritmu pro MinVertexCover nutně plyne existence 2-aproximačního algoritmu pro MaxClique? (Vysvětlete svůj názor.)

3. Ukázali jsme 2-aproximační algoritmus pro MinWeightVertexCover využitím lineárního programování. Při praktické implementaci je samozřejmě vhodné se poohlížet i po přímých algoritmech (nevyužívajících obecného "děla" v podobě LP). Zde je jeden jednoduchý 2-aproximační algoritmus:

   i/ Dostaneš zadaný graf $G = (V, E, c)$, kde každý vrchol v má přiřazenu nezápornou váhu (cenu) $c(v)$.

   ii/ Přiřaď pro začátek každé hraně $e \in E$ váhu $c(e) := 0$; platí tedy podmínka (invariant)
   $$\forall v \in V : c(v) \geq \sum_{e \in \delta(v)} c(e) \tag{57}$$
   (čili součet vah incidentních s kterýmkoli zvoleným vrcholem není větší než váha onoho vrcholu).

   iii/ Prober v nějakém pořadí všechny hrany a pro probíranou hranu e vždy maximálně zvyš její váhu $c(e)$ tak, aby jsi zachoval invariant (57).

   iv/ Nakonec vydej množinu C "nasycených" vrcholů, tj.
   $$C = \{v \in V \mid c(v) = \sum_{e \in \delta(v)} c(e)\}.$$

   Dokažte, že se opravdu jedná o 2-aproximační algoritmus pro MinWeightVertexCover.

4. Vysvětlete, proč je problém MinSetCover zobecněním problému MinVertexCover.

Týden 13 (od 2.5.)

Nejdříve jsme si podrobně připomněli pojmy potřebné k diskusi aproximačních algoritmů a (ne)aproximovatelnosti konkrétních optimalizačních problémů.

Při diskusi předchozích úkolů jsme si mj. uvědomili, že (hypotetický) 2-aproximační algoritmus pro problém MaxClique by v daném grafu vždy nalezl kliku alespoň poloviční velikosti (v počtu vrcholů) vzhledem k optimu, tedy vzhledem ke klice s největším možným počtem vrcholů.

Odvodili jsme si také, že přes těsnou souvislost problému MinVertexCover a MaxIndSet, a tedy i MaxClique, nám 2-aproximační algoritmus pro MinVertexCover nepomůže k vytvoření 2-aproximačního algoritmu pro problém MaxClique.

Pak jsme si jen uvedli, že je dokonce známo, že MaxClique nelze aproximovat s konstantním aproximačním poměrem, tedy že MaxClique $\notin$ APX (zase samozřejmě za předpokladu P$\neq$NP).

> Plánoval jsem alespoň nastínit důkaz faktu MaxClique $\notin$ APX, jelikož tento důkaz je zvlášť zajímavý. Nedostali jsme se k tomu, tak alespoň nastiňuji zde pro hlubší zájemce.
>
> Vzpomeňme si, že jsme celkem snadno dokázali TSP $\notin$ APX, a sice převodem z NP-úplného problému (konkrétně problému hamiltonovské kružnice), který vytvořil (dostatečnou) "mezeru" mezi optimem v případě, kdy odpověď pro výchozí instanci byla ANO, a optimem v případě, kdy tato odpověď byla NE.
>
> Ve známém důkazu MaxClique $\notin$ APX se také využívá techniky mezer (gap technique), ale v technicky nesmírně komplikovanějším hávu. Důkaz se totiž opírá o tzv. *PCP-teorém* (kde PCP lze číst jako Probabilistically Checkable Proofs), tj. tvrzení, že třída rozhodovacích problémů označovaná PCP$(\log n, 1)$ je rovna třídě NP.
>
> Rozhodovací problém $\mathcal{P}$ (např. SAT) patří do třídy PCP$(\log n, 1)$, jestliže existuje *polynomiální pravděpodobnostní* algoritmus V (Verifier), který při práci na dané instanci I (problému $\mathcal{P}$) velikosti n
>
> a/ předpokládá existenci nějakého "důkazu" P_I k dané instanci I, což je řetězec $2^{p(n)}$ bitů pro nějaký polynom p
>
> (velikost toho předpokládaného důkazu je tedy exponenciální vzhledem k velikosti I),
>
> b/ může využít $O(\log n)$ náhodných bitů
>
> (lze si představit, že pro jistou konstantu $c_1 \in \mathbb{N}$, algoritmus V nejprve vygeneruje náhodné bity $r_1, r_2, \ldots, r_{c_1 \log n}$ [přesněji bychom měli psát $r_{\lfloor c_1 \log n \rfloor}$], které pak mohou ovlivňovat další výpočet),
>
> c/ může se $O(1)$-krát (tedy c_2-krát, pro nějakou konstantu $c_2 \in \mathbb{N}$) zeptat na nějaký bit $\mathrm{P}_I[i]$ předpokládaného důkazu
>
> (adresa (index) i onoho bitu je tedy číslo z množiny $\{1, 2, \ldots, 2^{p(n)}\}$, které V vždy při dotazu zapíše binárně do vymezeného prostoru velikosti $p(n)$; tyto vyžádané bity (dodané nějakým vnějším "agentem" jménem "Prover") také samozřejmě mohou ovlivnit výpočet algoritmu V),
>
> d/ skončí (v polynomiálně omezeném počtu kroků) ve stavu "accept" nebo "reject".
>
> Navíc ovšem musí platit (pro prokázání toho, že $\mathcal{P} \in$ PCP$(\log n, 1)$), že
>
> - když I je pozitivní instance (tedy problém $\mathcal{P}$ přiřazuje I odpověď ANO), tak musí existovat důkaz (řetězec bitů) P, pro nějž všechny výpočty V na I skončí ve stavu accept (Verifier V tedy v tomto případě akceptuje I s pravděpodobností 1),

- když I je negativní instance, tak pro jakýkoli "důkaz" P', Verifier V akceptuje s pravděpodobností nejvýš $\frac{1}{2}$.

  Pro konkrétní instanci I a konkrétní řetezec P' je pro každou možnost hodnot náhodných bitů $r_1, r_2, \ldots, r_{c_1 \log n}$ (těch možností je $2^{c_1 \log n}$ tedy n^{c_1}) výpočet určen jednoznačně. Verifier zde tedy může akceptovat pro nanejvýš jednu polovinu z těch n^{c_1} možností.

Není vůbec vidět, že např. pro SAT nějaký takový polynomiální Verifier V s příslušnými důkazy P_I pro pozitivní instance I (tj. pro splnitelné formule) existuje.

> Přirozeným důkazem toho, že daná booleovská formula je splnitelná, je pravdivostní ohodnocení proměnných splňující formuli. Ovšem není vidět, jak takový důkaz algoritmu V pomůže, když se může zeptat jen na několik bitů důkazu, kde onen počet "několik" vůbec nezávisí na vstupní formuli!

Byl to překvapivý netriviální výsledek dosažený v 90. letech jako vyvrcholení série výzkumných prací v oblasti tzv. interaktivních protokolů, že SAT opravdu poskytuje možnost takového ověření; to je podstata výsledku $\mathrm{PCP}(\log n, 1) = \mathrm{NP}$.

> Když se chceme přesvědčit, zda daná (velká) booleovská formule v CNF s třemi literály v každé klauzuli je splnitelná (tedy je pozitivní instancí NP-úplného problému 3-SAT) a můžeme se zeptat jen na několik bitů z důkazu, který obsahuje splňující pravdivostní ohodnocení proměnných, tak se nabízí možnost zvolit náhodně klauzuli (k tomu nám stačí $\log m$ bitů, kde m je počet klauzulí) a zeptat se na hodnoty tří proměnných v klauzuli: když tyto hodnoty činí klauzuli splněnou, akceptujeme; jinak zamítneme. Takhle snadno to ovšem nefunguje, neboť ohodnocení může nesplňovat např. jen jedinou z m klauzulí a my se do ní trefíme jen se zanedbatelnou pravděpodobností. Celá technická (netriviální) finta se dá vyjádřit tak, že formuli vhodně "nafoukneme" tak, aby pravděpodobnost trefy do nesplněné klauzule u nesplňujícího ohodnocení nebyla zanedbatelná ....

Každý problém z $\mathrm{PCP}(\log n, 1)$ se dá snadno převést na problém MaxClique s "mezerou".

> Mějme příslušný algoritmus V s konstantami c_1, c_2 jako výše. Pro danou instanci I velikosti n sestrojíme graf G_I takto: Každý vrchol odpovídá jedné možnosti volby náhodných bitů $r_1, r_2, \ldots, r_{c_1 \log n}$ a jedné možnosti odpovědí $q_1, q_2, \ldots, q_{c_2}$ na dotazy na bity důkazu, při nichž algoritmus V pro instanci I akceptuje. (Těch vrcholů je tedy nanejvýš $2^{c_1 \log n} \cdot 2^{c_2}$ tj. $O(n^{c_1})$.)
> Dva vrcholy spojíme hranou, jestliže se liší jejich hodnoty náhodných vektorů a příslušné bity důkazu jsou konzistentní – tzn., že když se výpočty odpovídající oněm dvěma vrcholům ptají na stejný bit důkazu (mající stejný index neboli stejnou adresu), tak musí dostat stejnou odpověď.
> Celý graf G_I pro I očividně zkonstruujeme v polynomiálním čase.
> Snadno teď ověříme, že když I (velikosti n) je pozitivní, tak v sestrojeném grafu G_I existuje klika velikosti n^{c_1}; když je instance I negativní, tak maximální klika v G_I má velikost nejvýše $\frac{1}{2}n^{c_1}$.

Využitím faktu $\mathrm{SAT} \in \mathrm{PCP}(\log n, 1)$ (či obecněji $\mathrm{NP} \subseteq \mathrm{PCP}(\log n, 1)$) snadno vyvodíme, že $\mathrm{MaxClique} \notin \mathrm{APX}$ (za předpokladu $\mathrm{P} \neq \mathrm{NP}$).

> Volba $\frac{1}{2}$ pro omezení pravděpodobnosti, s jakou může Verifier V akceptovat negativní instanci, totiž není podstatná. Když vezmeme algoritmus V', který pro I vždy provede dva nezávislé výpočty algoritmu V a akceptuje jen tehdy,

když oba tyto výpočty akceptují a jsou konzistentní, tak srazíme pravděpodobnost přijetí negativní instance na maximálně $\frac{1}{2} \cdot \frac{1}{2} = \frac{1}{4}$. Využitím V' k převodu na problém MaxClique (tj. ke konstrukci grafu G_I k instanci I) tedy uděláme větší mezeru: maximální klika v případě negativní instance je nanejvýš čtvrtinová oproti případu pozitivní instance.
Takovým způsobem vyvrátíme hypotézu MaxClique $\in$ APX(c) pro libovolnou danou konstantu $c \in \mathbb{N}$.

Pseudo-polynomiální algoritmy. Silná NP-obtížnost.

Pobavili jsme se o číslech v instancích (optimalizačních) problémů, která udávají váhy, ceny, vzdálenosti, ... Typicky je zapisujeme binárně či dekadicky, takže hodnota těchto čísel je exponenciální vzhledem k délce jejich zápisu. U některých problémů je toto podstata jejich NP-obtížnosti, u mnohých ale ne.

Problémy typu SAT, MinVertexCover, MaxClique apod. žádná taková čísla v instancích nemají, takže u nich obtížnost jistě nespočívá v "exponencialitě" hodnot čísel. (Čísla u těchto problémů typicky používáme k očíslovaní vrcholů grafu, booleovských proměnných, apod., ale jejich hodnoty jsou malé vzhledem k přirozeně definované velikosti instance.)

V této souvislosti (při zkoumání zásadnosti vlivu binárního zápisu čísel na obtížnost problému) je užitečné zvážit variantu problému, při niž zapisujeme čísla unárně (např. číslo 245 tedy dvěstěčtyřicetipěti "čárkami").

Výpočetní složitost algoritmu je funkcí velikosti vstupu a proto se při unární reprezentaci čísel složitost algoritmu (a problému) formálně sníží (oproti normálnímu případu s binární reprezentací). Samozřejmě v praxi takovým "snížením" nic nezískáme, úvahy s unární reprezentací jsou ale dobré pro charakterizaci zdroje obtížnosti konkrétních problémů.

Přidejme si definice dvou pojmů:

- Máme-li algoritmus A řešící problém $\mathcal{P}$, řekneme, že *algoritmus* A je *pseudo-polynomiální*, jestliže je polynomiální (tj. má časovou složitost omezenou polynomem) v případě, že čísla v instancích $\mathcal{P}$ jsou reprezentována unárně.
- Řekneme, že problém $\mathcal{P}$ je *silně NP-těžký* (strongly NP-hard), jestliže je NP-těžký i při unárním zápisu čísel v instancích.

Připomeneme-li si, jak jsme prokazovali NP-obtížnost problémů jako je TSP či ILP (v převodech z jiných NP-těžkých problémů jsme v konstruovaných instancích nepotřebovali velká čísla, dokonce jako "váhy" stačily jedničky a dvojky), tak je zřejmé, že tyto problémy jsou silně NP-těžké.

NP-těžké problémy, v jejichž instancích se ("exponenciálně velká") čísla nevyskytují (jako je SAT, HC, MaxClique apod.), jsou podle naší definice také silně NP-těžké.

Standardním příkladem NP-těžkého problému, které není silně NP-těžký, je problém batohu (který jsme už v jisté verzi diskutovali u diskuse nejkratších cest v acyklických grafech).

Problém Knapsack budeme uvažovat v této (standardní) verzi:

a/ Instance I je dána sekvencemi kladných celých čísel $w_1, \ldots, w_n$ (váhy n předmětů) a $c_1, \ldots, c_n$ (ceny těchto předmětů), a dále číslem B ("Bound", nosnost batohu).

b/ Přípustným řešením k dané instanci I je každá množina $S \subseteq \{1, 2, \ldots, n\}$ (množina [indexů] vybraných předmětů) splňující $\sum_{i \in S} w_i \leq B$ (nosnost nesmí být překročena).

c/ Účelová funkce je $v(I, S) = \sum_{i \in S} c_i$ (cena vybraných předmětů).

d/ Hodnotu účelové funkce maximalizujeme. (Hledáme tedy výběr předmětů, který nepřekročí nosnost a má největší cenu.)

Problém Knapsack má pseudo-polynomiální řešící algoritmus (jak si připomeneme v úkolech).

Ukázali jsme si NP-obtížnost dokonce pro následující podproblém (rozhodovacího) problému Knapsack_{dec}:

> Problém *SubsetSum*:
>
> Instance: kladná celá čísla $c_1, c_2, \ldots, c_n$ a B.
>
> Otázka: existuje $S \subseteq \{1, 2, \ldots, n\}$, pro niž platí $\sum_{i \in S} c_i = B$?

Zmíněnou NP-obtížnost jsme ukázali převodem 3-SAT $\preceq_p$ SubsetSum.

> Připomeňme, že 3-SAT (v každé klauzuli jsou právě tři literály) je také NP-úplný; snadno lze totiž ukázat SAT $\preceq_p$ 3-SAT. (2-SAT je ovšem polynomiální.)

Stručně jsme se ještě dotkli dvou témat:

- Pojem (úplného) polynomiálního aproximačního schématu
  (PTAS ... Polynomial Time Approximation Scheme,
  FPTAS ... Fully Polynomial Time Approximation Scheme).

  O tom ještě také pohovoříme příště, přičemž se mj. vrátíme k problémům z oblasti "scheduling".

- Metoda větvení a mezí (branch-and-bound) jako další možnost, jak přistoupit k řešení konkrétních těžkých optimalizačních problémů (např. ILP). Této metodě je mj. věnována kapitola 7 v knize [9], která metodu uvádí srozumitelným způsobem na jednoduchém příkladě.

Nové úkoly.

1. Vysvětlete, proč za předpokladu P≠NP nemůže existovat problém, který je silně NP-těžký a přitom ho řeší nějaký pseudo-polynomiální algoritmus.

2. Připomeňte si dříve diskutovaný algoritmus pro řešení problému batohu, založený na dynamickém programování, a upravte ho pro naši verzi problému Knapsack.

   > Jedna možnost spočívá v postupném vyplňování "dvourozměrného pole" $C[i, j]$, kde $i \in \{0, 1, 2, \ldots, n\}$ a $j \in \{0, 1, 2, \ldots, B\}$; do konkrétního políčka $C[i, j]$ máme zapsat největší možnou (souhrnnou) cenu předmětů vybraných z množiny $\{1, 2, \ldots, i\}$, které mají (souhrnnou) váhu j (zapíšeme $-\infty$, když taková množina neexistuje).

Vysvětlete, proč je tento algoritmus pseudo-polynomiální, ale ne polynomiální.

3. Připomeňte si konstrukci v 3-SAT $\preceq_p$ SubsetSum; v konstruovaných instancích problému SubsetSum se vyskytovala "velká" čísla (s polynomiálně dlouhým zápisem, ale exponenciálně velkou hodnotou, vzhledem k velikosti výchozí instance 3-SAT).

   Technicky se nám hodil dekadický zápis k zvýšení průhlednosti konstrukce.

   Proč nelze čekat, že by se nám ta konstrukce podařila i bez oněch velkých čísel?

Týden 14 (od 9.5.)

Řekli jsme si pár věcí k zápočtové písemce a probrali její strukturu v ilustračním textu, který byl studentům zaslán. Připomněli jsme si některé základní pojmy, jejichž znalost se u zápočtové písemky předpokládá.

Např. to byly pojmy potřebné k objasnění vztahu $\mathcal{P} \equiv_p^T \mathcal{P}_{dec}$, který platí v zásadě pro všechny praktické problémy $\mathcal{P}$ kombinatorické optimalizace. Ten složitější převod jsme si ilustrovali na konkrétním případě, ukázali jsme totiž podrobně, že $\mathrm{TSP} \preceq_p^T \mathrm{TSP}_{dec}$.

> Pro danou instanci TSP lze v polynomiálním čase spočítat délku d_{opt} nejkratší okružní cesty, jestliže můžeme vícekrát volat proceduru pro rozhodovací problém TSP_{dec}, která odpovídá v jednotkovém čase; libovolná okružní cesta nám totiž určí horní odhad d pro d_{opt} a pak postupujeme metodou půlení intervalu $[1, \ldots, d]$ (binary search), přičemž využíváme volání TSP_{dec}. Když už známe d_{opt}, probíráme postupně hrany, přičemž pro každou hranu e uděláme toto: Zjistíme, zda při zvýšení váhy e (např. o 1) zůstává v grafu nějaká okružní cesta délky d_{opt} (to zjistíme příslušným voláním TSP_{dec}); když ano, ponecháme hraně e onu zvýšenou váhu (tím pádem e určitě není součástí žádné nejkratší okružní cesty v upraveném grafu), když ne, necháme hraně e původní váhu a označíme ji jako kritickou (každá nejkratší okružní cesta ji zaručeně obsahuje). Na závěr platí, že kritických hran je tolik, kolik je vrcholů, a vytvářejí (právě jednu) nejkratší cestu. (Proč?)

Připomněli jsme si z minulého týdne pojmy pseudo-polynomiálních algoritmů a silně NP-těžkých problémů.

> Knapsack je typický problém, který je NP-ekvivaletní (či NP-úplný v rozhodovací verzi), ale zároveň řešitelný pseudo-polynomiálním algoritmem. Není tedy silně NP-těžký (také se někdy říká, že je slabě NP-těžký), samozřejmě za předpokladu P≠NP.

PTAS a FPTAS.

Uvedli jsme si pojmy aproximačních schémat, které zachycují ještě silnější typ aproximovatelnosti (některých) NP-těžkých optimalizačních problémů než je aproximovatelnost s konstantním aproximačním poměrem (byť jakkoli malým).

> APX^* označuje třídu optimalizačních problémů, které jsou (polynomiálně) aproximovatelné s aproximačním poměrem $1+\varepsilon$ pro libovolně malé reálné $\varepsilon > 0$. Je tedy $\mathrm{APX}^* = \cap_{\varepsilon>0}\mathrm{APX}(1+\varepsilon)$. Existence PTAS či dokonce FPTAS pro problém $\mathcal{O}$ znamená ještě silnější typ aproximovatelnosti než pouhou podmínku $\mathcal{O} \in \mathrm{APX}^*$.

Polynomiálním aproximačním schématem (PTAS, Polynomial Time Approximation Scheme) pro optimalizační problém $\mathcal{O}$ rozumíme algoritmus A, který pro zadanou instanci I problému $\mathcal{O}$ a zadané $\varepsilon > 0$ (typicky ve formě $\varepsilon = \frac{1}{z}$, kde z je kladné celé číslo) vypočte nějaké přípustné řešení $s_A(I, \varepsilon)$ s aproximačním poměrem maximálně $1+\varepsilon$ (tedy $\frac{v(I,s_A(I,\varepsilon))}{v_{opt}(I)} \leq 1+\varepsilon$ v případě minimalizace a $\frac{v_{opt}(I)}{v(I,s_A(I,\varepsilon))} \leq 1+\varepsilon$ v případě maximalizace). Navíc pro každé zafixované ε musí mít příslušná verze algoritmu A (jejímž vstupem je tedy pouze I) polynomiální časovou složitost vzhledem k $size(I)$.

Plně polynomiální aproximační schéma (FPTAS, Fully Polynomial Time Approximation Scheme) pro optimalizační problém $\mathcal{O}$ je algoritmus A, který pro zadanou instanci I problému $\mathcal{O}$ a zadané $\varepsilon > 0$ vypočte nějaké přípustné řešení $s_A(I, \varepsilon)$ s aproximačním poměrem

maximálně $1+\varepsilon$ a navíc má polynomiální časovou složitost vzhledem k $size(I)$ i vzhledem k $\frac{1}{\varepsilon}$ (jeho časová složitost je tedy v $O((size(I))^{c_1}(\frac{1}{\varepsilon})^{c_2})$ pro nějaké konstanty c_1, c_2).

Např. pro problém Knapsack existuje FPTAS, což je silnější fakt než existence pseudo-polynomiálního algoritmu.

> Existence FPTAS pro problém $\mathcal{O}$ implikuje existenci pseudo-polynomiálního algoritmu pro $\mathcal{O}$, za jistých podmínek kladených na účelovou funkci, které jsou ovšem u přirozených optimalizačních problémů splněny.

Např. pro problém MinVertexCover^{planar} (minimální vrcholové pokrytí v rovinných grafech), který je rovněž NP-těžký, je známo PTAS, ale ne FPTAS.

My si níže uvedeme hlavní myšlenku FPTAS pro Knapsack, byť ve formě rozvrhovacího problému pro dva procesory. Přesněji řečeno, onen rozvrhovací problém, označený $P2|C_{max}$, je ekvivalentní problému MaxSubsetSum (jehož instancí jsou kladná celá čísla $c_1, c_2, \ldots, c_n$ a B a hledá se množina $J \subseteq \{1, 2, \ldots, n\}$, která maximalizuje součet $\sum_{i \in J} c_j$ za podmínky $\sum_{i \in J} c_j \leq B$).

Rozvrhovací problémy (scheduling).

Rozvrhovací problémy tvoří specifickou podoblast kombinatorické optimalizace. Tohoto tématu se zase jen letmo dotkneme. V tomto kurzu už jsme se setkali s jedním konkrétním problémem z této oblasti, který šel řešit "matroidovým hladovým algoritmem".

Existuje dohodnutá klasifikace speciální třídy problémů v této oblasti a dohodnuté značení problémů v této třídě. My se podíváme na tři vybrané problémy doplněné o známé informace o jejich výpočetní složitosti:

1/ problém $P2|res \circ \circ \circ, p_j = 1|C_{max}$ ..... polynomiální;

2/ problém $P2|C_{max}$ ..... NP-ekvivalentní, s FPTAS;

3/ problém $P3|res\, 1 \circ \circ, p_j = 1|C_{max}$ ..... NP-ekvivalentní, silně NP-těžký.

Nejprve se podíváme na druhý problém, u nějž prezentujeme výše avizované FPTAS.

Problém $P2|C_{max}$.

Značení $P2$ znamená, že předpokládáme dva paralelně běžící procesory (stroje, machines) M_1, M_2, které jsou stejné, tedy mají stejné vlastnosti (např. rychlost) ohledně zpracování úkolů (jobs). Instance problému obsahuje úkoly $J_1, J_2, \ldots, J_n$ s příslušnými dobami trvání (processing times) $p_1, p_2, \ldots, p_n$ (což jsou, řekněme, kladná celá čísla). Každý úkol má být bez přerušení proveden na jednom procesoru (tedy kterémkoli z M_1, M_2). Každý procesor může v každém okamžiku pracovat na nejvýš jednom úkolu. Přípustným řešením k instanci našeho optimalizačního problému $P2|C_{max}$ je jakékoli přiřazení úkolů procesorům; to de facto odpovídá podmnožině $\mathcal{J} \subseteq \{1, 2, \ldots, n\}$ obsahující indexy úkolů přiřazených procesoru M_1 (úkoly se zbývajícími indexy jsou přiřazeny procesoru M_2). Účelová funkce je daná značením C_{max} (maximum completion time); jde nám o *minimalizaci* času, kdy budou zpracovány všechny úkoly (minimalizujeme hodnotu C_{max}). V našem případě tedy minimalizujeme maximum ze součtů $\sum_{i \in \mathcal{J}} p_i$ a $\sum_{i \notin \mathcal{J}} p_i$.

> U obecného rozvrhovacího problému je přípustným řešením k dané isntanci konkrétní časový rozvrh zpracování jednotlivých úkolů na jednotlivých procesorech; přípustnost

spočívá v tom, že nedochází ke kolizím, tedy že v žádném okamžiku nemá žádný procesor zpracovávat víc než jeden úkol, atd. V našem konkrétním případě předpokládáme, že každý procesor zpracovává přidělené úkoly v libovolném pořadí, bez meziprodlev. (V složitějších problémech je mezi úkoly závislost, např. typu "úkol J_7 může být započat až po ukončení úkolů J_2 a J_4", což přidává další podmínky na přípustnost rozvrhu. Takovou závislost v námi uvažovaném problému $P2|C_{max}$ nemáme.)

Rozhodovací verze problému $P2|C_{max}$ je pochopitelně v NP a navíc z NP-obtížnosti problému SubsetSum snadno vyvodíme NP-obtížnost problému $P2|C_{max}$. (Jeden úkol žádá toto odvození.) Známe rovněž pseudo-polynomiální algoritmus pro $P2|C_{max}$ (neboť tento problém lze přirozeně chápat jako speciální případ problému Knapsack).

Všimněme si, že hladový přístup např. ve formě "seřaď joby od nejdelšího po nejkratší a postupně je dávej vždy tomu procesoru, který má aktuálně kratší 'náklad' (a, řekněme, procesoru M_1, když mají oba stejný náklad)" může vést k výrazně neoptimálnímu řešení.

Vezměme např. úkoly s délkami $51, 49, 34, 33, 33$. Optimum C_{max} je 100, hladový přístup dá 116.

Ukažme nejdříve PTAS pro problém $P2|C_{max}$. Pro danou instanci prezentovanou čísly $p_1, p_2, \ldots, p_n$ a ε spočteme $\ell = \sum_{i=1}^{n} p_i$ a prohlásíme každé číslo $p_i \geq \varepsilon\ell$ za "velké"; čísla $p_i < \varepsilon\ell$ jsou "malá". Velkých čísel je tedy nanejvýš $\frac{\ell}{\varepsilon\ell} = \frac{1}{\varepsilon}$. Vyzkoušejme všech $2^{\frac{1}{\varepsilon}}$ možností přiřazení velkých úkolů (jobů J_i s $p_i \geq \varepsilon\ell$) procesorům M_1, M_2, přičemž každou tuto možnost přiřazení velkých doplníme hladovým přiřazením malých. Tento algoritmus je tedy očividně polynomiální pro každé fixní ε, jelikož v tom případě je $2^{\frac{1}{\varepsilon}}$ bráno jako konstanta (nezávislá na vstupu).

Ověřme splnění podmínky kladené na aproximační poměr. V optimálním řešení jsou velké úkoly nějak rozděleny mezi M_1 a M_2. Toto rozdělení velkých úkolů náš algoritmus také zvažoval. Pak doplnil hladově všechny malé úkoly; pokud takto vzniklé řešení není optimální, tak náklad jednoho procesoru je méně než o $\varepsilon\ell$ delší než náklad druhého. Součet nákladů je ℓ, takže v případě neoptimality je nalezené C_{max} menší než $\frac{\ell}{2} + \frac{\varepsilon\ell}{2}$; máme tedy $\frac{\ell}{2} \leq v_{opt}(I) \leq v(I, s_A(I,\varepsilon)) < \frac{\ell}{2} + \frac{\varepsilon\ell}{2} = \frac{\ell}{2}(1+\varepsilon)$, a tedy $\frac{v(I,s_A(I,\varepsilon))}{v_{opt}(I)} \leq 1+\varepsilon$.

Problém $P2|C_{max}$ má ovšem i FPTAS; to je mj. předmětem jednoho referátu.

Idea se dá intutivně přiblížit takto: použijeme standardní pseudo-polynomiální algoritmus (založený na metodě dynamického programování), přičemž pracujeme se "zaokrouhlenými" čísly; zaokrouhlení musí být dostatečně velké k tomu, aby byl algoritmus polynomiální, ale takové, aby zaokrouhlovací chyby nenarušily žádaný aproximační poměr.

Problém $P2|res \circ \circ \circ, p_j = 1|C_{max}$**.**

Oproti předchozímu problému zde ve specifikaci přibyla neprázdná prostřední část. Výrazem $p_j = 1$ značíme, že všechny úkoly (joby) mají jednotkovou dobu trvání. Do hry ale navíc vstupují zdroje (resources); tři kolečka $\circ$ znamenají, že na ně nejsou kladeny další podmínky. Tím myslíme, že každá instance kromě úkolů $J_1, J_2, \ldots, J_n$ (s jednotkovou dobou trvání) obsahuje také zdoje $R_1, R_2, \ldots, R_\ell$; každý zdroj R_i má kapacitu s_i (kladné celé číslo) a každý úkol J_i je doplněn nároky $r_{1i}, r_{2i}, \ldots, r_{\ell i}$ na jednotlivé zdroje. V žádném okamžiku nemohou nároky právě prováděných úkolů překročit v souhrnu kapacitu některého zdroje.

Uvažme následující polynomiální řešení tohoto optimalizačního problému. K instanci (v značení uvedeném výše) sestrojíme graf, v němž vrcholy odpovídají úkolům $J_1, J_2, \ldots, J_n$.

Mezi dvěma různými vrcholy J_i a J_j vedeme hranu právě tehdy, když úkoly J_i a J_j jsou kompatibilní, tzn., že pro každý zdroj R_h máme $r_{hi} + r_{hj} \leq s_h$. Ve vzniklém grafu nalezneme maximální párování (maximum-cardinality matching) S; pro každou hranu $\{J_i, J_j\} \in S$ přiřadíme jeden konec procesoru M_1 a druhý konec procesoru M_2. Zbylé úkoly nakonec všechny přiřadíme M_1. Nalezené C_{max} je tedy $n - |S|$. (Jeden úkol žádá ověření,že se jedná o optimum.)

Problém $P3|res\,1 \circ \circ, p_j = 1|C_{max}$**.**

Oproti předchozímu problému zde předpokládáme tři (identické) procesory a jen jediný zdroj. Problém je zase očividně NP-lehký (tedy jeho rozhodovací varianta je v NP). Dá se také ukázat, že je NP-těžký, dokonce silně NP-těžký. Ukáže se to snadno převodem problému 3-Partition: instancí jsou kladná celá čísla $a_1, a_2, \ldots, a_{3t}$ (počet čísel je násobkem tří) a otázkou je, zda lze daná čísla rozdělit do trojic tak, že součty všech jednotlivých trojic jsou stejné (tedy že součet každé trojice je $\frac{\sum_{i=1}^{3t} a_i}{t}$).

Známá kniha [3] uvádí jako šest základních NP-úplných problémů problémy

3-SAT, 3-DM, MinVertexCover, MaxClique, HamiltonianCircuit, Partition.

Problém Partition je v zásadě totéž jako SubsetSum, ptáme se totiž, jestli daná množina čísel se dá rozdělit na dvě části tak, že součty obou částí jsou stejné.

Problém 3-DM (3-dimensional matching) je zobecněním otázky perfektního párování v bipartitním grafu: Máme teď tři "partity", tj. konečné vzájemně disjunktní množiny U, V, W, kde $|U| = |V| = |W| = k$, a množinu "trojhran" $E \subseteq U \times V \times W$. Ptáme se, zda existuje podmnožina $S \subseteq E$, $|S| = k$, jejíž prvky jsou vzájemně disjuktní (tedy "trojhrany" v S perfektně pokrývají U, V, W).

Jen v problému Partition vystupují čísla a tento problém je "slabě" NP-úplný, jak víme.

Problém 3-Partition najdeme v [3] jako sedmý základní NP-úplný problém. Ač je to také "číselný" problém, je silně NP-těžký. (Důkaz ukazuje převod z 3-DM a je dost technický.)

Problém 3-Partition převedeme polynomiálně na rozhodovací verzi problému $P3|res\,1\circ\circ, p_j = 1|C_{max}$ takto:

K instanci $a_1, a_2, \ldots, a_{3t}$ sestrojíme úkoly $J_1, J_2, \ldots, J_{3t}$ (s jednotkovou dobou trvání), přičemž a_i chápeme jako nárok úkolu J_i na zdroj (na onen jeden zdroj R_1). Jako kapacitu zdroje vezmeme $s_1 = \frac{\sum_{i=1}^{3t} a_i}{t}$. (Předpokládáme, že s_1 vyjde celé, jinak je jasné, že ona instance problému 3-Partition má odpověď NE.) Jako limit na C_{max} vezmeme t. (Důkaz korektnosti žádá jeden úkol.)

Nové úkoly.

1. Vyjděte z toho, že SubsetSum je NP-těžký a dokažte, že $P2|C_{max}$ je NP-těžký.
2. Ukažte, že výše navržený způsob řešení problému $P2|res\circ\circ\circ, p_j = 1|C_{max}$ vede zaručeně k optimu.
3. Ukažte, že výše navržená konstrukce skutečně prokazuje, že problém 3-Partition je polynomiálně převeditelný na rozhodovací verzi problému $P3|res\,1 \circ \circ, p_j = 1|C_{max}$.

Reference

[1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms (2nd edition).* The MIT Press, 2001.

[2] Thomas S. Ferguson. *Linear programming (A concise introduction).* UCLA, `http://www.math.ucla.edu/~tom/LP.pdf`.

[3] Michael R. Garey and David S. Johnson. *Computers and Intractability, A Guide to the Theory of NP-Completeness.* W.H.Freeman and Co. New York, 1979.

[4] Petr Jančar. *Teoretická informatika (*`www.cs.vsb.cz/jancar/TEORET-INF/teoret-inf.htm`*).* FEI VŠB - TUO, 2010.

[5] Bernhard Korte and Jens Vygen. *Combinatorial Optimization (Theory and Algorithms) 5th edition.* Springer, 2012.

[6] Petr Kovář. *Teorie grafů.* FEI VŠB - TUO, 2012.

[7] Petr Kovář. *Úvod do teorie grafů.* FEI VŠB - TUO, 2012.

[8] Michael Kubesa. *Základy diskrétní matematiky.* FEI VŠB - TUO, 2012.

[9] Jon Lee. *A First Course in Combinatorial Optimization.* Cambridge texts in applied mathematics, 2004 (second printing 2011).