Algoritmy pro těžké problémy

(mgr. předmět KMI/ALS2, dříve nazývaný "Algoritmy a složitost 2")

Petr Jančar

22. března 2025

Poznámky k textu. Tento text slouží jako podpůrný materiál k přednáškám. Podává přehled o tom, o čem jsme diskutovali, a odkazuje na další zdroje. (Přehled cvičení zpracovává Jakub Juračka.)

Text je zhruba členěn po jednotlivých týdnech v semestru. Pro přehlednost zápis k jednotlivému týdnu začíná vždy na nové straně.

V textu se explicitně odkazujeme na následující zdroje, které jsou přístupné z web-stránky předmětu.

- [Sli] Slidy k předchozímu běhu kurzu (přístupné ze stránky J. Juračky pro cvičení).
- [KO] Text ke kombinatorické optimalizaci.
- [VP] Text k vybraným partiím teoretické informatiky (anglicky).

Týden 1

Připomněli jsme si dále zmíněné pojmy.

Algoritmus. Turingův stroj jako reprezentace algoritmu. Časová složitost algoritmu (neboli Turingova stroje). Prostorová složitost algoritmu.

(Výpočetní) problém, rozhodovací (neboli ANO/NE) problém, optimalizační problém.

Třídy složitosti; jejich prvky jsou problémy (v základní verzi rozhodovací problémy, nebo též příslušné jazyky).

Speciálně jsme připomněli polynomiální převeditelnost mezi problémy a pojem NP-úplnosti.

Připomněli jsme si, že u problému minimální kostry (MinSpanTree) funguje tzv. hladový algoritmus a problém je v PTIME.

Uvědomili jsme si detailně, co je to optimalizační problém $\mathcal{O}$ a co je jeho rozhodovací verze $\mathcal{O}_{dec}$.

Připomněli jsme, že problém vrcholového pokrytí (MinVertexCover) je NP-úplný, přesněji řečeno, že jeho rozhodovací verze $\text{MinVertexCover}_{dec}$ je NP-úplným problémem.

Zavedli jsme pojem *aproximačního algoritmu* pro daný optimalizační problém $\mathcal{O}$. Je to polynomiální algoritmus (tedy algoritmus s polynomiální časovou složitostí), který k instanci problému $\mathcal{O}$ vydá přípustné řešení, které však nemusí být optimální. (Typicky se takové algoritmy navrhují pro NP-těžké optimalizační problémy.)

Zavedli jsme pojem *aproximačního poměru* daného aproximačního algoritmu.

Ukázali jsme si, že přirozený hladový algoritmus pro MinVertexCover nemá konstantní aproximační poměr. Na druhou stranu jednoduchý algoritmus sestavení (jakéhokoli) množinově-maximálního párování (množiny vzájemně neincidentních hran, která nelze zvětšit) ukazuje, že MinVertexCover patří do třídy APX(2), tj. do třídy (optimalizačních) problémů, pro něž existují algoritmy s aproximačním poměrem nejvýš 2.

Studijní podklady:

V [Sli] najdete část nazvanou "Complexity review"a také "Introduction to approximation algorithms".

V [KO] je podobná látka obsažena v částech "Týden 11"a "Týden 12".

V [VP] se jedná o kapitolu 6 (Approximation algorithms).

Ve všech zmíněných materiálech jsou i odkazy na další zdroje.

Pro pohodlí čtenáře jsou níže zkopírovány příslušné pasáže z [KO].

==========

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 typicky 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 $\varphi(x_1, x_2, x_3, x_4, x_5)$ (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 $\varphi(x_1, \ldots, x_n)$ je totiž splnitelná právě tehdy, když formule $\exists x_1 \cdots \exists x_n \varphi(x_1, \ldots, x_n)$ 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 $\varphi(x_1, \ldots, x_n)$ 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 $\mathsf{PTIME}^{\mathcal{O}}$ je pak restrikcí třídy $\mathrm{FP}^{\mathcal{O}}$ na rozhodovací problémy.

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

(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 $\mathcal{P}_2 \in \mathsf{PTIME}$, pak $\mathcal{P}_1 \in \mathsf{PTIME}$. (Tedy jestliže $\mathcal{P}_1 \preceq_p \mathcal{P}_2$ a $\mathcal{P}_1 \notin \mathsf{PTIME}$, pak $\mathcal{P}_2 \notin \mathsf{PTIME}$.)

NP-těžké, NP-úplné a NP-ekvivalentní problémy.
Rozhodovací problém $\mathcal{P}$ je NP-*těžký*, jestliže pro každý problém $\mathcal{P}' \in \mathsf{NPTIME}$ platí $\mathcal{P}' \preceq_p \mathcal{P}$; jestliže navíc platí $\mathcal{P} \in \mathsf{NPTIME}$, 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 $\mathcal{P} \in \mathsf{NPTIME}$, 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 $\mathcal{P}' \in \mathsf{NPTIME}$ platí $\mathcal{P}' \preceq_p^T \mathcal{P}$. Řekneme, že $\mathcal{P}$ je NP-*lehký*, jestliže existuje problém $\mathcal{P}' \in \mathsf{NPTIME}$ 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 $v^1_1, \ldots, v^1_{2m}$. 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 $v^1_1, \ldots, v^1_{2m}$ přidáme (hrany tvořící) opačnou cestu z $\neg x_1$ do x_1 přes vrcholy $v^1_{2m}, \ldots, v^1_1$. 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 $v^2_1, \ldots, v^2_{2m}$ 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 [3]).

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

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

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

Problém 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).

Týden 2

Připomeňme si, jak lze definovat optimalizační problémy formálně:

Optimalizační problém $\mathcal{O}$ chápeme jako pětici $(\mathrm{IN}, \mathrm{SOL}, \mathrm{AS}, v, \textsc{goal})$, kde

- IN je (typicky nekonečná spočetná) množina vstupů (neboli instancí) problému,
- SOL je množina možných řešení (solutions),
- funkce $\mathrm{AS} : \mathrm{IN} \to 2^{\mathrm{SOL}}$ (admissible solutions) přiřazuje každému vstupu $x \in \mathrm{IN}$ množinu přípustných řešení $\mathrm{AS}(x) \subseteq \mathrm{SOL}$

  (připomeňme, že obecně 2^M označuje množinu všech podmnožin množiny M; ještě obecněji, B^C označuje množinu všech zobrazení typu $C \to B$; jelikož při budování čísel v teorii množin je 2 chápáno jako množina $\{0,1\}$, je možné vidět 2^M i jako množinu všech zobrazení typu $M \to \{0,1\}$, která je v bijektivním vztahu s množinou $\{X \mid X \subseteq M\}$),
- účelová (či cílová, kriteriální, nákladová) funkce (objective function, cost function) $v : \mathrm{IN} \times \mathrm{SOL} \to \mathbb{R}_+$ přiřazuje každé dvojici $x \in \mathrm{IN}$, $s \in \mathrm{AS}(x)$ kladné reálné číslo (value) $v(x,s)$,
- $\textsc{goal} \in \{\textsc{min}, \textsc{max}\}$ určuje, zda se jedná o minimalizaci či maximalizaci.

Cvičení. Formulujte v uvedené formě problémy MinSpanTree a MaxClique.

Rozhodovací verze optimalizačního problému $\mathcal{O} = (\mathrm{IN}, \mathrm{SOL}, \mathrm{AS}, v, \textsc{goal})$ je rozhodovací (tj. ANO/NE) problém označovaný $\mathcal{O}_{dec}$, u nějž je vstupem dvojice (x, ℓ), kde $x \in \mathrm{IN}$ a $\ell \in \mathbb{Q}_+$ (kladné racionální číslo), a otázkou je, zda existuje $s \in \mathrm{AS}(x)$ takové, že $v(x,s) \leq \ell$ v případě $\textsc{goal} = \textsc{min}$ a $v(x,s) \geq \ell$ v případě $\textsc{goal} = \textsc{max}$.

Cvičení. Formulujte v uvedené formě problémy $\mathrm{MinSpanTree}_{dec}$ a $\mathrm{MaxClique}_{dec}$.

Třídy PO a NPO.

Třídou PO rozumíme třídu optimalizačních problémů, které jsou řešitelné polynomiálními algoritmy. Problém $\mathcal{O} = (\mathrm{IN}, \mathrm{SOL}, \mathrm{AS}, v, \textsc{goal})$ tedy patří do PO, jestliže existuje algoritmus A s polynomiální časovou složitostí, který k dané instanci $x \in \mathrm{IN}$ vydá (nějaké) *optimální řešení* $s_A(x)$, tedy přípustné řešení $s_A(x) \in \mathrm{AS}(x)$, pro které platí

$$v(x, s_A(x)) = \textsc{goal}\,\{\, v(x,s) \mid s \in \mathrm{AS}(x) \,\}$$

(tedy $v(x, s_A(x)) = \textsc{min}\,\{\, v(x,s) \mid s \in \mathrm{AS}(x) \,\}$ v případě minimalizačního problému a $v(x, s_A(x)) = \textsc{max}\,\{\, v(x,s) \mid s \in \mathrm{AS}(x) \,\}$ v případě maximalizačního problému), nebo zjistí, že optimální řešení neexistuje.

Cvičení. Připomeňte si (hladový) algoritmus prokazující, že MinSpanTree patří do PO. Zároveň si připomeňte, proč přirozený hladový algoritmus pro MinVertexCover obecně nenalézá optimální řešení.

Všimněme si, že pro $\mathcal{O} \in$PO nutně existuje polynom $p_1 : \mathbb{N} \to \mathbb{N}$ takový, že pro každé $x \in \mathrm{IN}$, pro nějž existuje optimální řešení, existuje alespoň jedno optimální řešení $s \in \mathrm{AS}(x)$, pro nějž $|s| \leq p_1(|x|)$. (Proč?)

(Výrazem $|o|$, nebo též $size(o)$, obecně označujeme velikost objektu o, tedy např. délku binárního řetězce, který objekt o reprezentuje v nějakém přirozeném kódování zřejmém z kontextu.)

Bez újmy na obecnosti budeme pro problém $\mathcal{O} \in$PO požadovat existenci polynomu $p_1 : \mathbb{N} \to \mathbb{N}$ takového, že $|s| \leq p_1(|x|)$ pro každé $x \in \mathrm{IN}$ a $s \in \mathrm{AS}(x)$. (Jako přípustná řešení pro x prostě prohlásíme jen ta $s \in \mathrm{AS}(x)$, která splňují $|s| \leq p_1(|x|)$, pokud je zaručeno, že v případě existence optimálních řešení alespoň jedno optimální řešení tu podmínku splňuje.)

(Tento požadavek jsme dodali proto, abychom i formálně zaručili inkluzi $\mathrm{PO} \subseteq \mathrm{NPO}$.)

Do třídy NPO (N označuje nedeterminismus) patří problém $\mathcal{O} = (\mathrm{IN}, \mathrm{SOL}, \mathrm{AS}, v, \textsc{goal})$, jestliže

- existuje polynom $p_1 : \mathbb{N} \to \mathbb{N}$ takový, že $|s| \leq p_1(|x|)$ pro každé $x \in \mathrm{IN}$ a $s \in \mathrm{AS}(x)$,
- a dále existuje polynomiální algoritmus V, který pro dané $x \in \mathrm{IN}$ a $s \in \mathrm{SOL}$ pozná, zda $s \in \mathrm{AS}(x)$, a v kladném případě vydá hodnotu $v(x, s)$. (Časová složitost algoritmu V je tedy omezena polynomem p_2, neboli $T_V(n) \leq p_2(n)$, kde $n = |x| + |s|$.)

Cvičení. Ukažte, proč MinSpanTree patří do PO.

Cvičení. Vysvětlete, proč pro $\mathcal{O} \in$NPO platí $\mathcal{O}_{dec} \in \mathsf{NP}$.

Problém TSP a jeho NP-obtížnost.

Problém TSP (Traveling Salesman Problem) je optimalizační problém

$$\mathrm{TSP} = (\mathrm{IN}, \mathrm{SOL}, \mathrm{AS}, v, \textsc{goal}),$$

kde instance $x \in \mathrm{IN}$ je úplný graf $G = (V, E, c)$ s hranami ohodnocenými kladnými celými čísly ("cost" c je tedy funkce typu $E \to \mathbb{N}_+$); lze předpokládat, že $V = \{1, 2, \ldots, n\}$ pro nějaké $n \in \mathbb{N}_+$ a v tom případě $\mathrm{AS}(x) = \{\pi \mid \pi \text{ je permutace množiny } \{1, 2, \ldots, n\}\}$. Pro permutaci $\pi = (i_1, i_2, \ldots, i_n)$ pak máme

$$v(x, \pi) = c(\{i_1, i_2\}) + c(\{i_2, i_3\}) + \cdots + c(\{i_{n-1}, i_n\}) + c(\{i_n, i_1\}).$$

Jedná se o minimalizační problém, takže $\textsc{goal} = \textsc{min}$.

Cvičení. Ukažte, že TSP patří do NPO (a speciálně tedy, že TSP_{dec} patří do NP).

Ukázali jsme myšlenky důkazu, že TSP_{dec} je NP-těžký (a tudíž NP-úplný):

Připomněli jsme ideu důkazu Cookovy věty (problém SAT je NP-úplný) a sérii polynomiálních redukcí

$$\mathrm{SAT} \preceq_p \text{3-SAT} \preceq_p \mathrm{HC} \preceq_p \mathrm{HK} \preceq_p \mathrm{TSP}_{dec}.$$

Aproximační algoritmy pro MinVertexCover.

Připomněli jsme si dva tyto algoritmy a ukázali jsme příklad instance MinVertexCover na bipartitním grafu, kde (obecně aproximačně horší) hladový algoritmus najde optimum, zatímco (obecně aproximačně lepší) "párovací" algoritmus využije svůj aproximační poměr 2 "na maximum". (Uvažte graf, jehož hrany jsou vzájemně neincidentní.)

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

$$\mathrm{APX} = \bigcup_{c\in\mathbb{N}_+} \mathrm{APX}(c).$$

> **Cvičení**. Připomeňte si algoritmus, kterým jsme ukázali, že MinVertexCover patří do $\mathrm{APX}(2)$.

Špatná aproximovatelnost (obecného) TSP.
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 $\mathrm{APX}(1)$. 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.

Uvědomili jsme si, že výše diskutovaná redukce $\mathrm{HK}\preceq_p \mathrm{TSP}_{dec}$ se dá upravit tak, že použitím techniky mezer (*gap technique*) ukazuje špatnou aproximovatelnost TSP:

$$\mathrm{TSP}\notin \mathrm{APX}(2^n) \text{ (tedy také } \mathrm{TSP}\notin \mathrm{APX}\text{), za předpokladu P}\neq\mathsf{NP}.$$

Popišme to podrobněji. 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.

$\mathbf{TSP}^{\Delta}\in\mathbf{APX}(2)$.
Problém TSP omezený na instance, v nichž je splněna trojúhelníková nerovnost (metrický TSP), označujeme TSP^{Δ}. Už jsme ukázali, že i TSP^{Δ} je NP-těžký, ale teď ukážeme, že je docela dobře aproximovatelný.

Ukážeme, že pro TSP^{Δ} existuje 2-aproximační algoritmus (což později ještě zlepšíme).

Uvažujme úplný graf $G = (V, E, c)$ s kladnými cenami (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, $c(\text{MST}) \leq v_{opt}(G)$ (kde MST je minimální kostra).

Uvažujme algoritmus A_1, který pro úplný graf $G = (\{1, 2, \ldots, n\}, E, c)$ s kladnými váhami $c(e)$ na hranách $e \in E$ nejprve nalezne minimální kostru MST (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 $2 \cdot c(\text{MST})$ (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 $\frac{v(G, s_{A_1}(G))}{v_{opt}(G)} \leq 2$.

Studijní podklady:

Vedle dříve zmíněných částí [Sli], [KO], [VP] najdete v [Sli] také část "Traveling Salesman Problem".

Týden 3

$\mathbf{TSP}^{\Delta} \in \mathbf{APX}(\frac{3}{2})$.

V předcházející části jsme ukázali, že TSP^{Δ} patří do $\mathrm{APX}(2)$; demonstrovali jsme to algoritmem označeným A_1. Nyní ukážeme, 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.)

(Následující text přebíráme z [KO], s mírnými úpravami.)

Christofides (autor A_2) využil polynomiality problému nalezení perfektního párování s minimální váhou v grafu s (nezápornými) váhami na hranách.

> Připomeňme, že *párování* (matching) v (neorientovaném) grafu $G = (V, E)$ je množina hran $E' \subseteq E$, které jsou vzájemně disjunktní, tedy žádné dvě hrany z E' nemají společný vrchol (neboli nejsou incidentní). *Perfektní párování* (perfect matching) je párování, které je také *hranovým pokrytím* (edge cover), což znamená, že každý vrchol je incidentní s nějakou hranou v onom párování.
>
> Problém nalezení minimálního perfektního párování v grafu $G = (V, E, c)$ s ohodnocenými hranami ($c : E \to \mathbb{N}$) lze chápat jako speciální případ problému "maximum-weight matching", který patří do PO; to budeme diskutovat později.

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 $c(S) = \sum_{e \in S} c(e)$ (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 $e \in \mathrm{MST} \cap S$ 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 $c(\mathrm{MST}) + c(S)$.
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 $c(\mathrm{MST}) + c(S)$.
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 $v(G, s_{A_2}(G)) \leq c(\mathrm{MST}) + c(S)$ (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 $c(\textsc{mst}) + c(S) \leq \frac{3}{2}\, v_{opt}(G)$; jelikož již víme, že $c(\textsc{mst}) \leq v_{opt}(G)$, 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.

Týden 4

Nalezení minimálního perfektního párování, což jsme nechali bez vysvětlení v důkazu, že TSP^{Δ} je v APX(1.5), se dá chápat jako speciální případ nalezení maximálního váženého párování (v ohodnoceném grafu). Ukázali jsme si polynomiální algoritmus pro tento problém alespoň v případě bipartitních grafů. Využili jsme mj. známého polynomiálního algoritmu pro hledání nejkratších cest mezi vrcholy v ohodnoceném grafu bez negativních cyklů (v našem případě šlo o hledání nejdelších cest v grafech bez pozitivních cyklů).

(Problém nejkratších cest je připomenut na str. 8-9 v [KO]. V "Týden 5"v [KO] najdete popis algoritmu pro výše uvedený problém v bipartitních grafech. Pro pohodlí čtenáře jsou příslušné texty zkopírovány zde níže.)

Nejkratší cesty.

Připomeňme si problém 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 ...), jehož řešení 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.) Speciálně si uvědomme problém možných negativních cyklů.

Myšlenky tří standardních (polynomiálních) algoritmů pro řešení problému nejkratších cest jsou uvedeny níže.

> Můžeme odkázat např. na kapitolu 2 knihy [4] (ale totéž naleznete samozřejmě na mnoha jiných místech).

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 $O(|V|^2)$.

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.

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 celkem snadno, 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. [3]).

My jsme si ale všimli, že u *bipartitního* grafu $G = (V_1 \uplus V_2, E, c)$ 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.

Poznamenáme, že tento algoritmus není nejlepší z hlediska složitosti (jak se lze přesvědčit v literatuře), ale je koncepčně jednoduchý: je založen na jednoduchém pozorování a aplikaci nejkratších cest ...

3D-párování (3-dimensional matching).

Viděli jsme, že maximální (vážené) párování v bipartitních grafech lze nalézt rychlým algoritmem. V bipartitním grafu $G = (V, E)$ je množina vrcholů V rozdělena do dvou částí (partit), $V = V_1 \uplus V_2$, a hrana je zde množina obsahující jeden prvek V_1 a jeden prvek V_2, tedy $E \subseteq \{\{u, v\} \mid u \in V_1, v \in V_2\}$.

Teď si všimneme, že už speciální případ rozšíření problému na 3-partitní hypergrafy je NP-těžký. Definujme (rozhodovací) problém *3DM (3-dimensional matching)* takto:

> *Instance*: vzájemně disjunktní konečné množiny (vrcholů) V_1, V_2, V_3 se stejnou mohutností $q = |V_1| = |V_2| = |V_3|$ a množina "hyperhran" (či "trojúhelníků") $T \subseteq \{\{u, v, w\} \mid u \in V_1, v \in V_2, w \in V_3\}$.
>
> *Otázka*: existuje $T' \subseteq T$ tak, že $|T'| = q$ a každé dva různé trojúhelníky v T' jsou disjunktní (tedy nemají žádný společný vrchol) ?
> (Díky požadavku $|T'| = q$ taková T' pokrývá všechny vrcholy z $V_1 \cup V_2 \cup V_3$.)

Tvrzení. *Problém 3DM je NP-úplný.*

Důkaz. Příslušnost k NP je zřejmá. (Proč?)

Ukážeme teď, že SAT$\preceq_p$ 3DM. Mějme formuli φ v konjunktivní normální formě; nechť má n proměnných $x_1, x_2, \ldots, x_n$ a m klauzulí $C_1, C_2, \ldots, C_m$. (Každá klauzule je disjunkcí literálů, kde literál je x_i nebo $\neg x_i$.) Sestrojíme instanci V_1, V_2, V_3, T problému 3DM, která je pozitivní právě tehdy, když φ je splnitelná.

Pro proměnnou x_1 zařadíme do V_1 vrcholy $x_1^1, \bar{x}_1^1, x_1^2, \bar{x}_1^2, \ldots, x_1^m, \bar{x}_1^m$, do V_2 vrcholy $a_1^1, a_1^2, \ldots, a_1^m$ a do V_3 vrcholy $b_1^1, b_1^2, \ldots, b_1^m$; do T zařadíme trojúhelníky $\{x_1^1, a_1^1, b_1^1\}$, $\{\bar{x}_1^1, b_1^1, a_1^2\}$, $\{x_1^2, a_1^2, b_1^2\}$, $\{\bar{x}_1^2, b_1^2, a_1^3\}$, $\ldots$, $\{x_1^m, a_1^m, b_1^m\}$, $\{\bar{x}_1^m, b_1^m, a_1^{m+1}\}$, kde a_1^{m+1} chápeme jako další označení pro a_1^1. (Nakreslete si obrázek a uvědomte si, že vybereme-li z definovaných

trojúhelníků množinu vzájemně disjuktních trojúhelníků, která pokrývá všechny vrcholy $a_1^1, a_1^2, \ldots, a_1^m$ a $b_1^1, b_1^2, \ldots, b_1^m$, tak buď zůstanou nepokryty "pozitivní" vrcholy $x_1^1, x_1^2, \ldots, x_1^m$ nebo negativní vrcholy $\bar{x}_1^1, \bar{x}_1^2, \ldots, \bar{x}_1^m$. Obecně pro každou proměnnou x_i dodáme příslušné vrcholy a trojúhelníky $\{x_i^1, a_i^1, b_i^1\}$, $\{\bar{x}_i^1, b_i^1, a_i^2\}$, $\ldots$, $\{x_i^m, a_i^m, b_i^m\}$, $\{\bar{x}_i^m, b_i^m, a_i^{m+1}\}$ (kde $a_i^{m+1} = a_i^1$).

Nyní pro každou klauzuli C_j dodáme do V_2 vrchol c_j a do V_3 vrchol d_j; pro každý literál x_i v klauzuli C_j dodáme do T trojúhelník $\{x_i^j, c_j, d_j\}$ a pro každý literál $\neg x_i$ v klauzuli C_j dodáme do T trojúhelník $\{\bar{x}_i^j, c_j, d_j\}$.

Uvědomme si, že ke každému pravdivostnímu ohodnocení splňujícímu φ existuje nějaká množina (dosud definovaných) vzájemně disjunktních trojúhelníků, které pokrývají všechny vrcholy a_i^j, b_i^j, c_j, d_j pro všechna $i \in \{1, \ldots, n\}$ a $j \in \{1, \ldots, m\}$; naopak každá taková množina trojúhelníků přirozeně určuje pravdivostní ohodnocení splňující φ.

> **Cvičení**. Zmíněná množina vzájemně disjunktních trojúhelníků, které pokrývají všechny vrcholy a_i^j, b_i^j, c_j, d_j pro všechna $i \in \{1, \ldots, n\}$ a $j \in \{1, \ldots, m\}$, nechá nepokryto $nm - m$ vrcholů z V_1. Zároveň dosud máme $|V_1| = 2nm$ a $|V_2| = |V_3| = nm + m$. Navrhněte, jak dodat $nm - m$ vrcholů do V_2 a $nm - m$ vrcholů do V_3 a doplnit T tak, aby naše konstrukce prokazovala kýžené SAT$\preceq_p$ 3DM.

□

Steinerův strom (Steiner tree).

Na cvičení se budete mj. zabývat návrhem aproximačních algoritmů pro problém nalezení minimálního Steinerova stromu. V naší verzi to je podobné nalezení minimální kostry v souvislém ohodnoceném grafu; zde ovšem jen vybrané vrcholy grafu jsou *povinné*, takže se hledá minimální strom (tedy strom s minimálním součtem ohodnocení hran), který zahrnuje všechny povinné vrcholy (a nemusí zahrnovat ty ostatní, nepovinné). Na rozdíl od problému minimální kostry, pro toto zobecnění není znám polynomiální algoritmus. Polynomiálním převodem z 3DM se snadno ukáže, že tento problém je NP-těžký. Uvedeme stručně myšlenku této redukce.

K instanci 3DM, jak je uvedena výše, sestrojíme instanci problému minimálního Steinerova stromu takto: Vrcholy vytvářeného grafu G očíslujme $1, 2, \ldots, k$, kde $k = 3q + |T| + 1$, a definujme bijekce

$$b_1 \colon V_1 \to \{1, 2, \ldots, q\},\ b_2 \colon V_2 \to \{q+1, q+2, \ldots, 2q\},\ b_3 \colon V_3 \to \{2q+1, 2q+2, \ldots, 3q\}$$

a $b \colon T \to \{3q+1, 3q+2, \ldots, 3q+|T|\}$. Pro každý trojúhelník $t = \{u, v, w\} \in T$ dodejme do G hrany $(b(t), b_1(u))$, $(b(t), b_2(v))$, $(b(t), b_3(w))$ a $(b(t), k)$. Váha každé hrany je 1 a povinné jsou vrcholy z množiny $\{1, 2, \ldots, 3q\} \cup \{k\}$. Dá se přímočaře ověřit, že v G existuje Steinerův strom s váhou (nanejvýš) $4q$ právě tehdy, když výchozí instance problému 3DM je pozitivní.

Týden 5

Diskutovali jsme problém MinSetCover, kterému je věnována část "Set Cover" ve [Sli]. Instanci problému tvoří konečná množina U, množina $\mathcal{F}$ obsahující některé podmnožiny množiny U (tedy $\mathcal{F} \subseteq 2^U$) a váha (nebo též cena) $w : \mathcal{F} \to \mathbb{N}$. Přípustným řešením je každý soubor $\mathcal{C} \subseteq \mathcal{F}$ takový, že $\bigcup_{S\in\mathcal{C}} S = U$ (tedy $\{x \mid x \in S \text{ pro nějakou množinu } S \in \mathcal{C}\} = U$); jde přitom o minimalizaci hodnoty $\sum_{S\in\mathcal{C}} w(S)$.

Nejdříve jsme diskutovali redukci problému VertexCover na SetCover, která ukazuje, že SetCover je stejně nebo hůře aproximovatelný než VertexCover.

Pak jsme ukázali, že aproximační poměr hladového algoritmu (ber vždy množinu, která má nejmenší poměr její ceny ku počtu dosud nepokrytých prvků, které pokrývá) je omezen číslem $H(\max\{|S|; S \in \mathcal{F}\})$, kde $H(n)$ je harmonické číslo $1 + \frac{1}{2} + \frac{1}{3} + \cdots \frac{1}{n}$; z toho plyne, že aproximační poměr je omezen číslem $1 + \ln n$ (neboť snadno odvodíme $H(n) \leq 1 + \int_1^n \frac{1}{x}\,\mathrm{d}x$).

Náčrt důkazu:

Algoritmus vydá postupně množiny $S_1, S_2, \ldots, S_k$ tvořící soubor $\mathcal{C}$ (kde $\bigcup_{S\in\mathcal{C}} S = U$). Cena tohoto řešení je $\sum_{S\in\mathcal{C}} w(S)$, která je nutně stejná nebo větší než $\sum_{S\in\mathcal{C}^*} w(S)$, kde $\mathcal{C}^*$ je optimální řešení.

Každému prvku $x \in U$ přiřaďme váhu

$$c_x = \frac{w(S_i)}{|S_i \setminus (S_1 \cup S_2 \cup \cdots S_{i-1})|} \text{ pro to } i, \text{ pro nějž } x \in S_i \setminus (S_1 \cup S_2 \cup \cdots S_{i-1}).$$

Je tedy $\sum_{x\in U} c_x = \sum_{S\in\mathcal{C}} w(S)$. Všimněme si, že

$$\sum_{S\in\mathcal{C}^*} w(S) \leq \sum_{S\in\mathcal{C}} w(S) = \sum_{x\in U} c_x \leq \sum_{S\in\mathcal{C}^*} \sum_{x\in S} c_x.$$

Pokud dokážeme, že pro každou $S \in \mathcal{F}$ platí $\sum_{x\in S} c_x \leq w(S) \cdot H(|S|)$, budeme hotovi, neboť v tom případě

$$\sum_{S\in\mathcal{C}^*} \sum_{x\in S} c_x \leq H(\max\{|S|; S \in \mathcal{F}\}) \cdot \sum_{S\in\mathcal{C}^*} w(S).$$

My jsme indukcí podle mohutnosti $|S'|$ dokázali silnější tvrzení:
pro každou $S' \subseteq S \in \mathcal{F}$ platí $\sum_{x\in S'} c_x \leq w(S) \cdot H(|S'|)$. Zde je ten důkaz:

1. Pro $S' = \emptyset$ jasné (nerovnost má zde tvar $0 \leq 0$).
2. Nechť $|S'| = n > 0$. Vezměme nějaké $y \in S' \cap (S_i \setminus (S_1 \cup S_2 \cup \cdots \cup S_{i-1}))$ pro nejmenší možné i; tedy $S' \cap (S_1 \cup \cdots \cup S_{i-1}) = \emptyset$ a proto $S' \subseteq S \setminus (S_1 \cup S_2 \cup \cdots \cup S_{i-1})$. Podle hladové volby S_i máme

$$c_y = \frac{w(S_i)}{|S_i \setminus (S_1 \cup S_2 \cup \cdots \cup S_{i-1})|} \leq \frac{w(S)}{|S \setminus (S_1 \cup S_2 \cup \cdots \cup S_{i-1})|} \leq \frac{w(S)}{|S'|} = \frac{w(S)}{n}$$

a tedy při zápisu $S' = \{y\} \cup S''$ (kde $|S''| = n-1$) máme (využitím indukčního předpokladu pro S'')

$$\sum_{x\in S'} c_x = c_y + \sum_{x\in S''} c_x \leq \frac{w(S)}{n} + w(S) \cdot H(n-1) = w(S) \cdot H(n)\,.$$

Týden 6

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 problému TSP_{dec} (při převodu problému hamiltonovské kružnice na TSP_{dec} jsme v konstruovaných instancích nepotřebovali velká čísla, dokonce jako "váhy" stačily jedničky a dvojky), tak je zřejmé, že TSP (i TSP^{Δ}) je silně NP-těžký.

> 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, který je tzv. *slabě NP-těžký*, tj. v případě P≠NP není silně NP-těžký, je problém batohu (knapsack).

Problém Knapsack budeme uvažovat v této (standardní) verzi:

a/ Instance x 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 x je každá množina $I \subseteq \{1, 2, \ldots, n\}$ (množina [indexů] vybraných předmětů) splňující $\sum_{i \in I} w_i \leq B$ (nosnost nesmí být překročena).

c/ Účelová funkce je $v(x, I) = \sum_{i \in I} c_i$ (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, který používá metodu tzv. dynamického programování. Postup pro konkrétní instanci x se dá načrtnout např. takto:

> Pro $i = 0, 1, 2, \ldots, n$ vyplňuj pole $C[j]$ pro $j \in \{0, 1, \ldots, \sum_{i=1}^{n} c_i\}$ tak, že
>
> po i-tém běhu cyklu bude v $C[j]$ poznačeno, zda j je cena nějakého přípustného řešení $I \subseteq \{1, 2, \ldots, i\}$, a v kladném případě bude poznačena i nejmenší možná váha takového řešení.
>
> (Rozmyslete si návrh i-tého běhu cyklu a také doplnění algoritmu tak, aby na konci vrátil optimální řešení příslušné instance problému.)

Už speciální podproblém problému Knapsack$_{dec}$ je NP-těžký. Jedná se o

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

Dá se ukázat redukce 3-SAT $\preceq_p$ SubsetSum, která využívá polynomiálně dlouhé dekadické zápisy (exponenciálně velkých) čísel.

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 APX* = $\cap_{\varepsilon>0}$APX$(1+\varepsilon)$. Existence PTAS či dokonce FPTAS pro problém $\mathcal{O}$ znamená ještě silnější typ aproximovatelnosti než pouhou podmínku $\mathcal{O} \in$ APX*.

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

Plně polynomiální aproximační schéma (FPTAS, Fully Polynomial Time Approximation Scheme) pro optimalizační problém $\mathcal{O}$ je algoritmus A, který pro zadanou instanci x problému $\mathcal{O}$ a zadané $\varepsilon > 0$ vypočte nějaké přípustné řešení $s_A(x, \varepsilon)$ s aproximačním poměrem maximálně $1+\varepsilon$ a navíc má polynomiální časovou složitost vzhledem k $size(x)$ i vzhledem k $\frac{1}{\varepsilon}$ (jeho časová složitost je tedy v $O((size(x))^{c_1}(\frac{1}{\varepsilon})^{c_2})$ 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.

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

PTAS for MinLoadScheduling.

We will show a ptas for the following problem MinLoadScheduling (for 2 identical processors): an instance x is a list of tasks $1, 2, \ldots, n$ with times (durations) $d_1, d_2, \ldots, d_n$ and we want to distribute them to 2 processors so that the computation time is minimal. In other words, we search for a subset $I \subseteq \{1, 2, \ldots, n\}$ such that $\sum_{i \in I} d_i$ is minimal but satisfies $\sum_{i \in I} d_i \geq \frac{\ell}{2}$ where (the overall load) $\ell = d_1 + d_2 + \cdots + d_n$. (The tasks from I are then scheduled for one processor, the rest is scheduled for the other processor.)

We suggest the following ptas A for MinLoadScheduling. The algorithm A, given $d_1, d_2, \ldots, d_n$ and ε, classifies all tasks with $d_i \geq \varepsilon\ell$ as *large* (where $\ell = d_1 + d_2 + \cdots + d_n$); the tasks with $d_i < \varepsilon\ell$ are *small*. We note that there are at most $\lfloor \frac{1}{\varepsilon} \rfloor$ large tasks.

A then checks successively each of the at most $2^{\lfloor \frac{1}{\varepsilon} \rfloor}$ distributions of large tasks (to 2 processors); each particular distribution is completed by distributing small tasks 'greedily', i.e., each small task is then successively scheduled for the currently less loaded processor. A best solution is finally returned as the result.

Exercise. Check that the complexity of A is $O(2^{\lfloor \frac{1}{\varepsilon} \rfloor} \cdot n)$, and that the approximation ratio is $\leq (1+\varepsilon)$. (Hint. An optimal solution distributes large tasks in a certain way, and this way was also considered by A. Either all small tasks are in that case scheduled for one processor whose load is not greater than the load of the other, and A has thus found an optimal solution, or $\frac{\ell}{2} \leq v_{opt}(x) \leq v(x, s_A(x, \varepsilon)) \leq \frac{\ell}{2} + \frac{\varepsilon\ell}{2} = (1 + \varepsilon)\frac{\ell}{2}$.)

FPTAS pro Knapsack.

Věnovali jsme se návrhu "fptas" pro Knapsack. (Ilustrovali jsme si tak podstatu myšlenek v části "Approximation Schemes, Knapsack Problem" v [Sli].) Diskutovali jsme tuto verzi:

Instance (vstup):

předměty $1, 2, \ldots, n$, jejich váhy $w_1, w_2, \ldots, w_n \in \mathbb{N}$, kapacita $B \in \mathbb{N}$ (předpokládáme, že $w_i \leq B$ pro všechna $i \in [1, n]$) a ceny $c_1, c_2, \ldots, c_n \in \mathbb{N}$; navíc je dáno $\varepsilon > 0$.
(Např. $n = 100$, $\varepsilon = \frac{1}{10}$.)

Cíl:

najít $I \subseteq \{1, 2, \ldots, n\}$ tak, že $\sum_{i \in I} w_i \leq B$ a $\sum_{i \in I} c_i \geq (1 - \varepsilon) \cdot \sum_{i \in I^*} c_i$, kde $\sum_{i \in I^*} w_i \leq B$ a $\sum_{i \in I^*} c_i$ je maximální (tedy I^* je optimální řešení).

Algoritmus (aproximační schéma):

Položme $C = \max\{c_i \mid i \in [1, n]\}$ a pro vš. $i \in [1, n]$ definujme

$$c'_i = \left\lfloor \frac{c_i}{C} \cdot \frac{1}{\varepsilon} \cdot n \right\rfloor, \text{ tedy } c'_i = \left\lfloor \frac{c_i}{K} \right\rfloor, \text{ kde } K = \frac{\varepsilon C}{n} \text{ (a } \lfloor x \rfloor \text{ je největší celé číslo} \leq x).$$

Úlohu pro c'_i vyřešíme algoritmem založeném na dynamickém programování; stačí tedy řešit pro celkové ceny $0, 1, 2, \ldots, n \cdot \lfloor \frac{1}{\varepsilon} \cdot n \rfloor \leq \frac{n^2}{\varepsilon}$. (V našem konkrétním příkladu je $\frac{n^2}{\varepsilon} = 100000$.)

Vydáme příslušnou $I \subseteq \{1, 2, \ldots, n\}$ (optimální řešení v pozměněné instanci).

Polynomialita algoritmu jak v parametru n, tak v parametru $\frac{1}{\varepsilon}$, je zřejmá.
Dokažme, že skutečně platí $\sum_{i \in I} c_i \geq (1 - \varepsilon) \cdot \sum_{i \in I^*} c_i$.
Díky optimalitě I pro ceny c'_i, způsobu zaokrouhlení a faktu $\sum_{i \in I^*} c_i \geq C$ máme

$$\sum_{i \in I} c_i \geq K \cdot \sum_{i \in I} c'_i \geq K \cdot \sum_{i \in I^*} c'_i \geq \left(\sum_{i \in I^*} c_i \right) - nK = \left(\sum_{i \in I^*} c_i \right) - \varepsilon C \geq (1 - \varepsilon) \cdot \sum_{i \in I^*} c_i.$$

Poznámka. Ukázali jsme tedy, že pro zadanou instanci x a zadané ε uvedený algoritmus najde přípustné řešení $s_A(x, \varepsilon)$ splňující $\frac{v_{opt}(x)}{v(x, s_A(x,\varepsilon))} \leq \frac{1}{1-\varepsilon}$. Neplatí sice $\frac{1}{1-\varepsilon} \leq 1 + \varepsilon$, ale existenci fptas i podle standardní definice jsme dokázali, neboť pro zadané x, ε můžeme uvedený algoritmus spustit se vstupem $x, \frac{\varepsilon}{2}$; uvědomme si, že $\frac{1}{1-\frac{\varepsilon}{2}} \leq 1 + \varepsilon$ (pro $0 < \varepsilon < 1$), neboť $1 \leq (1 + \varepsilon)(1 - \frac{\varepsilon}{2}) = 1 + \frac{\varepsilon}{2} - \frac{\varepsilon^2}{2}$.

Týden 7

Dokončili jsme diskusi témat předchozích týdnů. Několik dalších týdnů se budeme věnovat tzv. lineárnímu programování, jakožto obecnému nástroji pro řešení mnohých optimalizačních problémů.

Lineární programování

Seznámíme 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.

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

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

Zapsali jsme tuto úlohu s konkrétními čísly z našeho "dietního problému" a znázornili geometricky množinu přípustných řešení (tedy množinu vektorů $x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$ splňujících $Ax \leq b, x \geq \mathbf{0}$). Podařilo se nám nalézt optimální řešení v tomto konkrétním případě.

Úkoly pro cvičení

1. 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.

2. K (standardní primární) 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^* primární úlohy a konkrétní přípustné řešení y^* duální úlohy?

3. K (primárnímu) 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.)

4. 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.

5. 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?)

Týden 8

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).

Cvičení. Ukažte, že ILP je NP-těžký problém (např. převodem z 3-SAT). Jako rozhodovací verzi vezměte problém, zda pro danou instanci existuje alespoň jedno přípustné řešení.

> *Poznámka.* 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.)

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

$$\text{maximalizuj} \qquad 3x_1 + x_2 + 2x_3 \qquad \text{za podmínek} \tag{2}$$

$$x_1, x_2, x_3 \geq 0,$$

$$\begin{aligned} x_1 &+ x_2 &+ 3x_3 &\leq 30, \\ 2x_1 &+ 2x_2 &+ 5x_3 &\leq 24, \\ 4x_1 &+ x_2 &+ 2x_3 &\leq 36. \end{aligned}$$

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 *primární 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 primární 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^* primární 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

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

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

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

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

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

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 dostáváme (standardními úpravami pro sčítání a násobení) na pravých stranách $\sum_{i=1}^{m} \sum_{j=1}^{n} y_i a_{ij} x_j$; tedy $(y^T A)x = y^T(Ax)$.

Teď si ještě všimneme:

- Z rovnice (4) ihned plyne, že pokud $(y_1a_{1j} + y_2a_{2j} + \cdots + y_ma_{mj}) \geq c_j$ a $x_j \geq 0$ pro vš. $j = 1, 2, \ldots, n$, tak $y^TAx \geq c^Tx$ (kde $c^Tx = c_1x_1 + c_2x_2 + \cdots + c_nx_n$).
- 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 $y^TAx \leq y^Tb$ (kde $y^Tb = y_1b_1 + y_2b_2 + \cdots + y_mb_m$).

Dokázali jsme tedy následující tvrzení.

Tvrzení 1 *(Slabá dualita) Nechť x^* je přípustným řešením problému $Ax \leq b, x \geq 0, \max c^Tx$ a y^* je přípustným řešením problému $y^TA \geq c^T, y \geq 0, \min y^Tb$. Pak $c^Tx^* \leq (y^*)^Tb$.*
Když platí $c^Tx^* = (y^*)^Tb$, 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 $c^Tx^* = 3 \cdot 8 + 1 \cdot 4 + 2 \cdot 0 = 28$ a $(y^*)^Tb = 0 \cdot 30 + \frac{1}{6} \cdot 24 + \frac{2}{3} \cdot 36 = 28$, 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 $Ax \leq b, x \geq 0, \max c^Tx$ 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 $c^Tx \leq d$ pro každé přípustné x,

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

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

Analogicky rozčleníme minimalizační problémy. Slabá dualita mj. implikuje, že když je standardní (primární) 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 $Ax \leq b, x \geq 0, \max c^Tx$ řešitelná a omezená, tak má optimální řešení x^* a duální úloha $y^TA \geq c^T, y \geq 0, \min y^Tb$ má (optimální) řešení y^* takové, že $c^Tx^* = (y^*)^Tb$.

Z vět o dualitě plyne, že když označíme jako P konkrétní (primární) 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

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

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:

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

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 ...

Úkoly pro cvičení

1. Vysvětlete, že pro řešitelnou neomezenou úlohu musí být duální úloha neřešitelná.
2. Ukažte příklad problému LP, který je neřešitelný a pro nějž je i jeho duální problém neřešitelný.
3. Aplikujte simplexový algoritmus na náš dietní problém.
4. Když v simplexovém algoritmu narazíme na případ $-c_j < 0$ a $a_{ij} \leq 0$ pro všechny řádky i, úloha je neomezená; vysvětlete proč.

Poznámka. Můžete si samozřejmě vyzkoušet nějaký (volně dostupný) LP solver.

Týden 9

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 "$\mathtt{c}$" 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 $\mathtt{r}$-tý sloupec a $\mathtt{c}$-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 $\mathtt{r}$-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 $\mathtt{c}$-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]:

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

Nejprve tedy budeme řešit úlohu

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

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 do báze jde (sloupec označený) x_2 (v posledním řádku máme $-5 < 0$) a bázi opouští 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

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

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.)

Úkoly pro cvičení

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 10

Na simplexový algoritmus pro řešení úloh LP (Linear Programming) navazujeme 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 pro nalezení optimálního celočíselného řešení. Taky si připomeňme, že ILP-feasibility (problém, zda daná instance ILP má alespoň jedno přípustné řešení) je NP-těžký problém; proto nemůžeme čekat, že metoda řezajících rovin povede vždy rychle k cíli.

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 primární 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":

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

(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 $\text{FP}(a) = a - \lfloor a \rfloor$. Je tedy

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

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

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

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

vyjádřit ekvivalentně jako

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

Jelikož sčítance $\text{FP}(a_j) x_j$ 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

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

Za předpokladu *celočíselnosti* $x_1, \ldots, x_n$ je poslední nerovnice ekvivalentní nerovnici s vypuštěnou $\text{FP}(b)$ (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 zjisti, jestli původní úloha má alespoň jedno celočíselné řešení (úloha ILP je "feasible")

   > (to lze řešit rekurzivně, tedy stejnou metodou, ale tak, že místo $c^T x$ maximalizujeme $0x$, což nemůže skončit jako FU [připomeňme si ale, že ILP-feasability je NP-těžký problém]);

   v pozitivním případě vrať FU; jinak vrať I (a skonči).

   > Jak jsme snadno ilustrovali, relaxovaná úloha může být řešitelná neomezená, přičemž nemá celočíselné řešení (např. $\max\{x_2 \mid \frac{1}{3} \leq x_1 \leq \frac{2}{3}\}$). 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 (primární 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 primární 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 primární 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 $y_1 = x_1 - x_2$ [jak hned vidíme z tabulky (45)]),

přidali $x_2 \leq 0$ 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 (primární) 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 primární 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 primární 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 primární úlohy přípustná, a tedy optimální. (Jinak bychom pivotovali dále.) Optimální řešení primární 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_2 a y_3 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 lze ukázat konstrukci (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; 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 [4] 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, existují 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)

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

Úkoly pro cvičení.

1. **2-aproximační algoritmus pro MinWeightVertexCover.**
   Využitím lineárního programování snadno navrhneme 2-aproximační algoritmus 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.

2. (Nepovinně.) 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.)

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

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

4. Vysvětlete, proč (obecná obdélníková) 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. Důležitý bod je, že když ve čtvercové matici obsahuje každý sloupec právě jednu 1 a právě jednu -1, tak jsou řádky lineárně závislé (tj., existuje jejich nenulová lineární kombinace rovnající se nulovému vektoru) a determinant této matice je tedy 0.)

Týden 11

Uvedli jsme pravděpodobnostní algoritmy a jako příklad pravděpodobnostní řešení problému MaxSAT, včetně derandomizace vedoucí k deterministickému aproximačnímu algoritmu.

Pravděpodobnostní algoritmy

Existují rozhodovací (tedy ANO/NE) problémy, pro které neznáme rychlé (tj. polynomiální) algoritmy, a přesto je lze v praxi spolehlivě řešit nejen pro malé vstupy. Stačí slevit z absolutního nároku na řešící algoritmus, totiž z toho, aby algoritmus vždy vydal *zaručeně* (tedy stoprocentně) správnou odpověď. Přesněji řečeno, přestaneme vyžadovat, aby algoritmus nutně předepisoval *deterministický* proces, a připustíme náhodný prvek (házení kostkou). Opakované běhy algoritmu na tentýž vstup pak mohou vydávat různé výstupy – každý z nich s určitou pravděpodobností.

K formalizaci pojmu pravděpodobnostního algoritmu můžeme použít jednoduchou verzi *pravděpodobnostního Turingova stroje* (probabilistic Turing machine, PTM); takový stroj $PM = (Q, \Sigma, \Gamma, \delta, q_0, F)$ si můžeme představit jako standardní Turingův stroj s tím, že $\delta(q, a)$ může být nyní i dvouprvková množina – v tom případě se při výpočtu v konfiguraci $uqav$ volí jedna ze dvou možných bezprostředně následujících konfigurací náhodně – tak, že každá má pravděpodobnost zvolení $\frac{1}{2}$. (Můžeme si představit, že se zde při výpočtu hází mincí; v programovacích jazycích se typicky vyskytuje nějaká funkce RANDOM, založená na generátoru (pseudo)náhodných čísel.)

Podobně jako u nedeterministického Turingova stroje, odpovídá jednomu vstupu PTM strom výpočtů. Vrcholy stromu jsou ohodnoceny konfiguracemi; kořen je ohodnocen počáteční konfigurací a následníci vrcholu ohodnoceného konfigurací c jsou ohodnoceni právě bezprostředně následujícími konfiguracemi konfigurace c.

Každá hrana je přitom ohodnocena příslušnou pravděpodobností – hraně vycházející z vrcholu s jedním následníkem je přiřazena 1, každé ze dvou hran vycházejících z vrcholu s dvěma následníky je přiřazena $\frac{1}{2}$. Každému vrcholu v je přiřazena pravděpodobnost jeho dosažení – tj. součin ohodnocení hran na cestě od kořene k vrcholu v.

Pak lze např. přesně definovat pravděpodobnost jevu, že daný PTM vydá na daný vstup x odpověď ANO – tato pravděpodobnost je dána součtem pravděpodobností listů ve stromu výpočtu pro vstup x, které jsou ohodnoceny koncovými konfiguracemi znamenajícími odpověď ANO (tedy přijímajícími konfiguracemi).

Než se podíváme na konkrétní ANO/NE problém, u nějž má pravděpodobnostní algoritmus praktický smysl, podíváme se na případ, kdy je užitečné uvažovat pravděpodobnostní algoritmus, který ale vzápětí derandomizujeme, tedy nahradíme deterministickým algoritmem.

A use of (de)randomization; Max-3-Sat is in APX(8/7)

Let us consider the problem MAX-3-SAT. An instance is a boolean formula $\varphi(x_1, \ldots, x_n)$ in CNF, where each clause contains 3 literals; the task is to find a truth-assignment (for variables $x_1, \ldots, x_n$) which satisfies the maximal number of clauses. We assume $\varphi = c_1 \wedge c_2 \wedge \cdots \wedge c_m$; so m denotes the number of clauses.

> **Exercise** (non-compulsory). Show that MAX-3-SAT is NP-equivalent. (NP-hardness can be easily shown by SAT $\leq_T$ MAX-3-SAT; but can you also show MAX-3-SAT $\leq_T$ SAT ?)

There is a simple approximation algorithm A for MAX-3-SAT with $r_A \leq 2$ (so MAX-3-SAT belongs to the class APX(2)):

> Evaluate the number of satisfied clauses for the truth assignments $(0, 0, \ldots, 0)$ and $(1, 1, \ldots, 1)$ and return the one which satisfies more clauses (more precisely, at least as many clauses as the other assignment).

The desired property follows from the observation that if clause c_i is not satisfied by $(0, 0, \ldots, 0)$ then it is surely satisfied by $(1, 1, \ldots, 1)$; hence one of the assignments satisfies at least $\lceil \frac{m}{2} \rceil$ clauses, which guarantees

$$\lceil \tfrac{m}{2} \rceil \leq v(\varphi, s_A(\varphi)) \leq v_{opt}(\varphi) \leq m, \text{ and thus } \tfrac{v_{opt}(\varphi)}{v(\varphi, s_A(\varphi))} \leq 2.$$

> *Remark.* We have thus also illustrated a frequent scheme for estimating approximation ratios. In the case of maximization problems, we try to find a suitable upper bound for the optimal value (since we seldom are able to express this value exactly) and a lower bound for the computed approximation value. In the minimization problems, we look for a suitable lower bound for the optimum and an upper bound for the approximation.

Let us note that if we take a (uniformly distributed) *random truth assignment* (for $x_1, x_2, \ldots, x_n$) in φ, where we assume that all three variables in each clause are different, then the expected value $E(Y)$ of (the random variable counting) the number of satisfied clauses is $\frac{7}{8}m$ (why?).

> (Hint. Recall that the expected value is linear, so $E(Y) = E(Y_1 + Y_2 + \cdots + Y_m) = E(Y_1) + E(Y_2) + \cdots + E(Y_m)$ where $E(Y_i)$ is the expected value of the random variable which is 1 if the clause c_i is satisfied and 0 if c_i is not satisfied.)

Exercise. Try to derandomize the probabilistic algorithm based on this observation, i.e., suggest a deterministic polynomial algorithm approximating MAX-3-SAT with the ratio $\leq \frac{8}{7}$.

> Hint. Use the fact that $E(Y) = \frac{7}{8}m = \frac{1}{2} \cdot E(Y|x_1 = 0) + \frac{1}{2} \cdot E(Y|x_1 = 1)$, and thus necessarily at least one of the values $E(Y|x_1 = 0)$, $E(Y|x_1 = 1)$ is $\geq \frac{7}{8}m$. Let the (deterministic) algorithm put $x_1 = b_1$ so that $E(Y|x_1 = b_1) \geq E(Y|x_1 = 1-b_1)$. Denote by $\bar{Y}_1$ the random variable counting the number of satisfied clauses on condition that $x_1 = b_1$, and observe that $E(\bar{Y}_1) = \frac{1}{2} \cdot E(\bar{Y}_1|x_2 = 0) + \frac{1}{2} \cdot E(\bar{Y}_1|x_2 = 1)$. Since we have shown $E(\bar{Y}_1) \geq \frac{7}{8}m$, we must have that at least one of the values $E(\bar{Y}_1|x_2 = 0)$, $E(\bar{Y}_1|x_2 = 1)$ is $\geq \frac{7}{8}m$. ....

Prvočíselnost

Prakticky používaný pravděpodobnostní algoritmus lze ilustrovat na problému prvočíselnosti:

> NÁZEV: *Prvočíselnost*
>
> VSTUP: Přirozené číslo n (v dekadickém zápise).
>
> VÝSTUP: ANO, když n je prvočíslo, NE, když n je číslo složené.

Všimněme si, že přímočaré testování prvočíselnosti podle definice vede k (deterministickému) algoritmu s exponenciální složitostí, protože velikost vstupu n je v našem případě $\log n$. Žádný polynomiální *deterministický* algoritmus nebyl až do léta 2002 znám. (Teď už je znám, ale přesto má smysl kvůli větší rychlosti nadále uvažovat pravděpodobnostní algoritmus.)

> Všimněme si, že je zřejmé, že doplňkový problém – tj. problém složenosti čísla – je v NP (proč?). Využitím jistých výsledků teorie čísel se dá ukázat, že i problém prvočíselnosti je v NP. Tento problém byl dlouho jedním z mála „přirozených" problémů v NP (resp. v průniku NP ∩ coNP), u kterých není znám polynomiální algoritmus a ani se neumí prokázat NP-úplnost. (Měli bychom upřesnit, že již dříve byla známa existence polynomiálního algoritmu, který by prvočíselnost rozhodoval za předpokladu, že platí tzv. Riemannova hypotéza – to se ovšem neví.)

S využitím poznatků teorie čísel lze sestavit rychlý pravděpodobnostní algoritmus A

> (časová složitost algoritmu je tedy polynomiální; to znamená, že existuje polynom p tak, že délka každé větve stromu výpočtu pro vstup x je omezena hodnotou $p(size(x))$; délka každého výpočtu je tedy omezena polynomiální funkcí velikosti vstupu),

který má tyto vlastnosti

> - jestliže n je prvočíslo, vydá A odpověď Ano s pravděpodobností 1 (Ne vůbec nemůže vydat);
> - jestliže n není prvočíslo, vydá A odpověď Ne s pravděpodobností alespoň $\frac{1}{2}$.

Představme si nyní, že výpočet A zopakujeme pro dané n např. 50-krát. Když (alespoň) jednou vydá Ne, víme, že n jistě není prvočíslo (a dále už A opakovat nemusíme). Když ve všech případech vydá Ano, tak n je prvočíslo s pravděpodobností větší než $1 - \frac{1}{2^{50}}$, tedy n je prvočíslo „prakticky stoprocentně" (proč ?).

Než se podíváme na detaily konkrétního algoritmu pro prvočíselnost, všimneme si, že obecně nám stačí rychlý algoritmus pro Ano/Ne problém, který vydá chybnou odpověď nanejvýš s pravděpodobností ε, kde $0 \leq \varepsilon < \frac{1}{2}$.

Třída BPP (Bounded-error Probabilistic Polynomial time)

BPP je třída Ano/Ne problémů (jazyků), pro něž existují polynomiální pravděpodobnostní algoritmy s omezenou chybou. Přesněji: jazyk $L \subseteq \Sigma^*$ patří do BPP, jestliže existuje ε, $0 \leq \varepsilon < \frac{1}{2}$, a pravděpodobnostní algoritmus A s polynomiální (časovou) složitostí, který pro každé $w \in \Sigma^*$ vydává Ano nebo Ne, přičemž Ano vydá s pravděpodobností alespoň $1-\varepsilon$ pro $w \in L$ a s pravděpodobností nanejvýš ε pro $w \notin L$; v tom případě říkáme, že *algoritmus A rozhoduje jazyk L s pravděpodobností chyby (nanejvýš) ε* (např. $\varepsilon = \frac{1}{3}$).

Někdy se třída BPP definuje jen užitím fixního ε, např. $\varepsilon = \frac{1}{3}$. Ukážeme teď, že výsledná třída problémů je vždy stejná, a také zároveň ukážeme, proč problémy z BPP můžeme považovat za prakticky zvládnutelné. Je to tedy rozumné rozšíření třídy P, i když nevíme, zda neplatí P=BPP. (Vztah BPP k NP je nejasný z hlediska obou inkluzí.)

Robustnost třídy BPP a praktická řešitelnost problémů z BPP.
Mějme polynomiální algoritmus A, který rozhoduje jazyk $L \subseteq \Sigma^*$ s pravděpodobností chyby ε, $0 < \varepsilon < \frac{1}{2}$. Navrhněme polynomiální algoritmus A', který rozhoduje jazyk L s pravděpodobností chyby $2^{-t} = \frac{1}{2^t}$, např. pro $t = 100$.

Následující postup je přebrán z knihy Sipser: Introduction to the theory of computation.

Algoritmus A' pro vstup $w \in \Sigma^*$ spustí A vícekrát na w, konkrétně $2k$-krát, kde k závisí pouze na ε a t a bude specifikováno níže. Dostane sekvenci S sestávající z $2k$ prvků množiny $\{\text{Ano}, \text{Ne}\}$. Vydá Ano, pokud je odpovědí Ano v sekvenci S více než polovina (tedy více než k), jinak vydá Ne.

Zkoumejme pravděpodobnost toho, že odpověď algoritmu A' na w je špatná (vydá Ano, i když $w \notin L$, nebo Ne, i když $w \in L$). Pokud je výsledná odpověď nesprávná, tak alespoň k odpovědí v sekvenci S bylo špatných, tedy příslušná sekvence S má pravděpodobnost výskytu nanejvýš $\varepsilon^k(1-\varepsilon)^k$ (proč?). Počet takových špatných sekvencí nebudeme odhadovat, stačí prostě vzít počet všech možných sekvencí délky $2k$, tedy 2^{2k}, neboli 4^k. Pravděpodobnost, že A' vydá chybnou odpověď, je tedy $\leq (4\varepsilon(1-\varepsilon))^k$. (Uvědomme si, že máme $4\varepsilon(1-\varepsilon) < 1$ a tedy se zvětšujícím se k pravděpodobnost chyby exponenciálně klesá.)

Chtěli bychom mít $(4\varepsilon(1-\varepsilon))^k \leq 2^{-t}$; po zlogaritmování tedy

$$\log_2(4\varepsilon(1-\varepsilon))^k \leq \log_2 2^{-t}, \text{ tedy } k \cdot \log_2(4\varepsilon(1-\varepsilon)) \leq -t.$$

Stačí tedy vzít $k \geq \frac{t}{\alpha}$, kde $\alpha = |\log_2(4\varepsilon(1-\varepsilon))|$.

Např. pro $\varepsilon = \frac{2}{5}$ je $\alpha = |\log_2 \frac{24}{25}| = \log_2 25 - \log_2 24 \geq 4.64 - 4.59 = \frac{1}{20}$; tedy stačí $k \geq 20\,t$; v případě $t = 100$ tedy vezmeme $k = 2000$ (algoritmus A' spustí A 4000 krát).

Pro $\varepsilon = \frac{1}{10}$ je $\alpha = |\log_2 \frac{36}{100}| = \log_2 100 - \log_2 36 \geq 6.64 - 5.17 = \frac{147}{100}$; tedy stačí $k \geq \frac{100}{147}t$; jelikož $\frac{100}{147} \leq 0.69$, stačí $k \geq 0.69\,t$, v případě $t = 100$ tedy vezmeme $k = 69$ (algoritmus proběhne 138 krát).

Týden 12

Problém prvočíselnosti

Připomínáme problém, který hraje velmi důležitou úlohu mj. v kryptografii.

NÁZEV: *Prvočíselnost*

VSTUP: Přirozené číslo n (v dekadickém, či binárním, zápise).

VÝSTUP: ANO, když n je prvočíslo, NE, když n je číslo složené.

Již jsme avizovali, že pro řešení tohoto problému se používá rychlý pravděpodobnostní algoritmus A, který má tyto vlastnosti:

- jestliže n je prvočíslo, vydá A odpověď ANO s pravděpodobností 1;
- jestliže n není prvočíslo, vydá A odpověď NE s pravděpodobností alespoň $\frac{1}{2}$.

Naznačíme, jak algoritmus A vypadá. Mj. se opírá o tzv. *malou Fermatovu větu*:

Věta 4 (Malá Fermatova věta) *Jestliže p je prvočíslo, tak pro každé $a, 0 < a < p$, platí*

$$a^{p-1} \equiv 1 \pmod{p}.$$

Důkaz. Rozvineme-li $(a+b)^p$ podle binomické věty, ověříme snadno, že výsledek modulo p je roven $a^p + b^p$ (modulo p), je-li p prvočíslo (ověřte !). Tedy, když $\equiv$ označuje kongruenci "modulo p", máme $1^p \equiv 1$, $2^p \equiv (1+1)^p \equiv 1^p + 1^p \equiv 1 + 1 \equiv 2$ a obecně $a^p \equiv a$. Je tedy $a(a^{p-1} - 1) \equiv 0$, neboli $a(a^{p-1} - 1) = k \cdot p$ pro nějaké k. Pokud největší společný dělitel a a p je 1 (což pro $0 < a < p$ je), musí být $(a^{p-1} - 1)$ násobkem p, tudíž $a^{p-1} \equiv 1$. □

Jako důkaz složenosti (tj. neprvočíselnosti) daného čísla m nejlépe slouží nějaký netriviální dělitel a čísla m (všichni známe rychlý algoritmus, kterým se přesvědčíme, že dané a skutečně dělí m).

> Z toho hned plyne, že problém složenosti (tj. neprvočíselnosti) patří do NP, a tedy problém prvočíselnosti patří do coNP.

Z Fermatovy věty ovšem plyne důležité pozorování: když nalezneme pro dané m číslo $a, 0 < a < m$ tak, že $a^{m-1} \not\equiv 1 \pmod{m}$, tak jsme našli *svědka složenosti* čísla m (tedy svědka toho, že m není prvočíslo), byť tím nezískáme netriviálního dělitele m (jen jsme tak dokázali jeho existenci).

> **Cvičení**. Ukázali jsme si, že 2 je svědek složenosti čísla 15, protože $2^{14} \not\equiv 1 (\mathrm{mod}\ 15)$.

Modulární umocňování. Je důležité si uvědomit, že počítání mocnin $(a^e \bmod m)$ umíme rovněž velmi rychle (i když a, e, m jsou velká čísla, zapsaná binárně či dekadicky), a sice tzv. metodou „opakovaného umocňování": počítáním $x_0 = a$, $x_1 = (x_0)^2 \bmod m$, $x_2 = (x_1)^2 \bmod m$, $x_3 = (x_2)^2 \bmod m$, ..., $x_i = (x_{i-1})^2 \bmod m$, ... dostáváme postupně a, $a^2 \bmod m$, $a^4 \bmod m$, $a^8 \bmod m$, ..., $a^{2^i} \bmod m$, ... Chceme-li pak například znát a^{560}, vynásobíme $a^{512} \cdot a^{32} \cdot a^{16}$ (tj. $x_9 \cdot x_4 \cdot x_5$; vše počítáno samozřejmě modulo m).

Pro konkrétní e s binárním zápisem $b_k b_{k-1} \cdots b_0$ lze využít iterativního výpočtu počítajícího $(a^{b_k b_{k-1} \cdots b_i} \bmod m)$ pro $i = k{+}1, k, \ldots, 0$, což pro $i = k{+}1$ dává hodnotu 1:
Je-li $(a^{b_k b_{k-1} \cdots b_i} \bmod m) = d$, pro $i > 0$, pak $(a^{b_k b_{k-1} \cdots b_{i-1}} \bmod m) = (d^2 \cdot a^{b_{i-1}} \bmod m)$.

Multiplikativní grupy a svědkové složenosti. Vraťme se k algoritmu A, který využívá Fermatovu větu. Všimli jsme si, že každý prvek a, $0 < a < m$, pro nějž platí $a^{m-1} \not\equiv 1(\bmod m)$ je svědek složenosti m.

> **Cvičení**. Dokažte, že každé a, $0 < a < m$, které je soudělné s m, tedy $gcd(a, m) > 1$ (gcd označuje největšího společného dělitele [greatest common divisor]), je svědkem složenosti m v uvedeném smyslu.

Označme $\mathbb{Z}_m^* = \{a \mid 0 < a < m, gcd(a, m) = 1\}$. Např. $\mathbb{Z}_{15}^* = \{1, 2, 4, 7, 8, 11, 13, 14\}$.

> **Cvičení**. Dokažte, že $\mathbb{Z}_m^*$ je multiplikativní grupa (vzhledem k násobení modulo m). (Nápověda. Množina $\mathbb{Z}_m^*$ je očividně uzavřena na násobení (modulo m). Existenci inverzního prvku k $a \in \mathbb{Z}_m^*$ ukážeme takto: Uvažujeme sekvenci $a, a^2, a^3, \ldots$. Nutně nastane případ $a^i \equiv a^j$ pro $i < j$, tedy $a^i(a^{j-i} - 1) \equiv 0$. Jelikož $gcd(a, m) = 1$, dostáváme $(a^{j-i} - 1) \equiv 0$, a tedy $a \cdot a^{j-i-1} \equiv 1$, z čehož plyne, že a^{j-i-1} je inverzní prvek k a.)

Už jsme si ukázali, že každé a, $0 < a < m$, ležící mimo $\mathbb{Z}_m^*$ je svědkem složenosti m (jelikož $a^{m-1} \not\equiv 1(\bmod\ m)$). Dá se celkem snadno ukázat, že

nesvědkové v $\mathbb{Z}_m^*$ tvoří podgrupu grupy $\mathbb{Z}_m^*$

> (plyne to z uzavřenosti na násobení: když $a^{m-1} \equiv 1(\bmod\ m)$ a $b^{m-1} \equiv 1(\bmod\ m)$, tak $(ab)^{m-1} \equiv 1(\bmod\ m)$).

Pokud tedy existuje svědek složenosti v $\mathbb{Z}_m^*$, tak je těch svědků v $\mathbb{Z}_m^*$ nejméně polovina! To plyne z tzv. Lagrangeovy věty, která říká, že počet prvků podgrupy H konečné grupy G dělí počet prvků grupy G.

> Podrobnosti nalezne zájemce např. v první části materiálu [VP].

Algoritmus A hledající náhodně svědka složenosti čísla m (s rovnoměrným rozložením pravděpodobnosti na množině $\{a \mid 0 < a < m\}$) tedy uspěje s pravděpodobností minimálně $\frac{1}{2}$, pokud m je složené a v $\mathbb{Z}_m^*$ existuje alespoň jeden svědek složenosti.

Nejsme ovšem ještě úplně hotovi, protože existují (velmi řídká) tzv. Carmichaelova čísla, což jsou složená m u nichž platí $a^{m-1} \equiv 1(\bmod\ m)$ pro všechna $a \in \mathbb{Z}_m^*$. (Nejmenší Carmichaelovo číslo je $561 = 3 \cdot 11 \cdot 17$.)

Algoritmus A se dá rozšířit na tzv. "Miller-Rabin test", v němž se pojem svědka složenosti rozšíří tak, že pro každé složené číslo m platí, že množina nesvědků složenosti čísla m tvoří *vlastní* podgrupu grupy $\mathbb{Z}_m^*$.

> Podrobnosti nalezne zájemce rovněž v první části materiálu [VP].

> Poznámka. Ač problém testování prvočíselnosti je takto uspokojivě vyřešen, nejsou známy žádné rychlé algoritmy pro nalezení prvočíselného rozkladu složeného čísla. Tedy ač např. umíme rychle zjistit, že dané 500-místné číslo je složené, nezaručuje nám to rychlé nalezení netriviálního dělitele. Speciálně, když nám někdo dá 500-místné číslo, které vytvořil jako součin dvou zhruba 250-místných prvočísel, nemáme (zatím?) praktickou šanci tato prvočísla najít. Mj. i na tom je založena bezpečnost užívaných kryptografických protokolů v digitální komunikaci.

Třídy BPP, RP a co-RP

Jen poznamenejme, že jsme mj. ukázali, že problém prvočíselnosti patří do třídy BPP (při použití testu Miller-Rabin). Ve skutečnosti měl náš algoritmus jen jednostrannou chybu.

Třída RP je třída problémů pro nějž existují polynomiální pravděpodobnostní algoritmy, které

- pro pozitivní instanci vydají odpověď ANO s pravděpodobností alespoň $\frac{1}{2}$,
- pro negativní instanci vydají odpověď NE s pravděpodobností 1.

Je tedy vidět, že RP je podmnožinou NP (proč?).

My jsme ukázali, že problém prvočíselnosti patří do coRP (neboť jeho doplňkový problém patří do RP).

Poznamenejme, že vztah třídy BPP k NP je nejasný.

Týden 13

Zopakování látky, speciálně s ohledem na zkoušku.

Okruhy ke zkoušce

A) Otázky "důkazové":

1. Polynomiální algoritmus pro maximální vážené párování na bipartitních grafech.
2. Problém obchodního cestujícího. Algoritmy pro metrickou úlohu. Obtížnost aproximace obecné úlohy.
3. Problém minimálního Steinerova stromu.
4. Problém MinSetCover a hladový aproximační algoritmus.
5. Aproximační schéma pro úlohu batohu a pro úlohu Min-load-scheduling.
6. Lineární programování. Důkaz slabé duality.
7. MinVertexCover a MinSetCover pomocí (relaxovaného) lineárního programování (LP).
8. Celočíselnost řešení LP. Totálně unimodulární matice. Maximální párování a vrcholové pokrytí na bipartitních grafech.
9. Pravděpodobnostní algoritmy pro problémy Max-SAT a Max-Cut. Derandomizace metodou podmíněných očekávaných hodnot.
10. Testování prvočíselnosti. Fermatova věta. Modulární umocňování.

B) Otázky pojmové a přehledové (vždy s příklady)

1. Pojmy optimalizační problém, aproximační algoritmus a jeho aproximační záruky.
2. Obtížnost optimalizačních problémů. Třídy PO a NPO, vztah k třídám P a NP.
3. Třídy APX, PTAS, FPTAS.
4. Lineární programování a jeho varianty.
5. Simplexový algoritmus. Metoda řezajících rovin pro ILP.
6. Pravděpodobnostní algoritmy, třídy RP, co-RP, BPP.

Poznámky k průběhu zkoušky:
bez splněného zápočtu nelze jít na zkoušku;
přihlašování na termíny vypsané v IS STAG (pokud není domluveno jinak);
zkouška bude ústní;
student si náhodně vytáhne otázku z okruhu A a bude mu přidělena doplňující otázka z okruhu B, z jiné oblasti než bude zvolená otázka z okruhu A;
čas na písemnou přípravu: minimálně 20 minut (bez možnosti používání donesených materiálů);
čas na ústní zkoušení: zhruba 30 minut.

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] Bernhard Korte and Jens Vygen. *Combinatorial Optimization (Theory and Algorithms) 5th edition*. Springer, 2012.

[4] Jon Lee. *A First Course in Combinatorial Optimization*. Cambridge texts in applied mathematics, 2004 (second printing 2011).