

KMI/ALGO3 — Algoritmy 3

2. Hladové algoritmy

Jan Konečný

Katedra informatiky, PřF UP Olomouc

Kde jsme: syllabus KMI/ALGO3

- Co znamená navrhnout algoritmus — specifikace, invariant, složitost
- **Hladové algoritmy — kdy je lokální volba bezpečná**
- Hladové algoritmy na grafech — kostry, nejkratší cesty
- Rozděl a panuj — rekurence, Master theorem
- Rozděl a panuj II — chytré rozklady
- Dynamické programování I — hledání stavů
- Dynamické programování II — vícerozměrné stavy
- Toky v sítích — max-flow/min-cut jako návrhový nástroj
- Systematické prohledávání — hrubá síla a backtracking
- Branch-and-bound — meze přesného řešení, P a NP
- Adversariální návrh — minimax, alfa-beta
- Když přesné optimum není realistické — aproximace, heuristiky
- Randomizace a syntéza — jak si vybrat metodu

Loot z dungeonu

Vracíme se z dungeonu, v poslední místnosti leží poklad. Unesete ale nejvýš 10 kg (weight limit). Co vzít?

	předmět	váha	cena	cena/kg
	Dračí koruna	6 kg	300 zl	50
	Elfí meč	5 kg	200 zl	40
	Runový štít	5 kg	200 zl	40

Přirozená strategie: *vezmi nejcennější věc, která se ještě vejde; opakuj.*

Loot z dungeonu

Vracíme se z dungeonu, v poslední místnosti leží poklad. Unesete ale nejvýš 10 kg (weight limit). Co vzít?

	předmět	váha	cena	cena/kg
	Dračí koruna	6 kg	300 zl	50
	Elfí meč	5 kg	200 zl	40
	Runový štít	5 kg	200 zl	40

Přirozená strategie: *vezmi nejcennější věc, která se ještě vejde; opakuj.*

- **Lákavé:** nezkoušíme všechny možnosti, nevracíme se k dřívějším rozhodnutím.

Loot z dungeonu

Vracíme se z dungeonu, v poslední místnosti leží poklad. Unesete ale nejvýš **10 kg** (weight limit). Co vzít?

	předmět	váha	cena	cena/kg
	Dračí koruna	6 kg	300 zl	50
	Elfí meč	5 kg	200 zl	40
	Runový štít	5 kg	200 zl	40

Přirozená strategie: *vezmi nejcennější věc, která se ještě vejde; opakuj.*

- **Lákavé:** nezkoušíme všechny možnosti, nevracíme se k dřívějším rozhodnutím.
- **Problém:** koruna zabere 6 kg a meč ani štít se už nevejdou: **300 zl**.
Meč a štít dohromady: **400 zl**. Stejně dopadne i výběr podle ceny za kilogram.

Lokálně nejlepší rozhodnutí tedy nemusí být součástí globálně nejlepšího řešení. (Úloha se jmenuje **problém batohu**, *knapsack*.)

Tři otázky této přednášky

Zajímá nás, kdy je lokální volba bezpečná, konkrétně:

1. Jak navrhnout hladové pravidlo?
2. Jak rychle najít **protipříklad** ke špatnému pravidlu?
3. Jak dokázat, že správné pravidlo je opravdu bezpečné?

Hladový způsob konstrukce sám o sobě správnost algoritmu nezaručuje. Rychlý algoritmus tedy získáme snadno; obtížná část je dokázat, že jsme si ukvapenou volbou nezavřeli cestu k optimu.

Co je hladová volba

Algoritmus udržuje částečné řešení A a opakovaně do něj přidává prvek. V každém kroku odpovídá na dvě otázky:

- Které prvky lze přidat, aby řešení zůstalo **přípustné**?
- Který z nich je podle zvoleného pravidla **nejlepší**?

Definice 1 (Hladový algoritmus)

Hladový algoritmus buduje řešení posloupností lokálních voleb. V každém kroku vybere přípustnou možnost, která je podle pevně daného **hladového kritéria** aktuálně nejlepší, a provedenou volbu už později **nemění**.

Definice **neříká**, že hladový algoritmus vrací optimum. Hladových pravidel je pro jeden problém mnoho a většina selže.

Návrh probíhá ve dvou krocích

1. snažíme se formulovat co **nejjednodušší** lokální pravidlo;
2. snažíme se toto pravidlo co **nejrychleji rozbít** malým protipříkladem.

Obecný důkaz má smysl hledat až pro pravidlo, které se nám nepodařilo vyvrátit.

Co je třeba u hladového algoritmu dokázat

1. **Bezpečnost volby:** existuje optimální řešení, které obsahuje právě provedenou hladovou volbu.
2. **Zachování struktury:** po provedení volby zbývá menší problém stejného typu.

Hlavní částí důkazu bývá první vrstva; druhá nám dovolí argument opakovat.

Intervalové plánování

Jedna přednášková místnost, množina žádostí o její použití. Žádost i má začátek s_i a konec f_i , $s_i < f_i$; chápeme ji jako polootevřený interval $[s_i, f_i)$ — akce smí začít přesně tehdy, kdy předchozí skončí.

Definice 2 (Intervalové plánování)

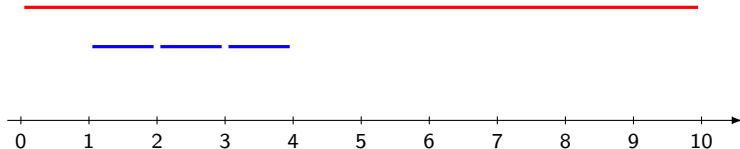
Instanci tvoří množina intervalů $I = \{[s_i, f_i) \mid i = 1, \dots, n\}$. Přípustným řešením je podmnožina $A \subseteq I$ **po dvou kompatibilních** (nepřekrývajících se) intervalů. Cílem je maximalizovat $|A|$.

Podle čeho vybrat první interval?

- ten, který **začíná nejdříve**;
- **nejkratší** interval;
- ten, který se překrývá s **nejmenším počtem** ostatních;
- ten, který **končí nejdříve**.

Protipříklad 1: nejdřívejší začátek

Jeden interval může začít jako první a zabrat celý den, zatímco uvnitř něj leží mnoho krátkých kompatibilních intervalů.



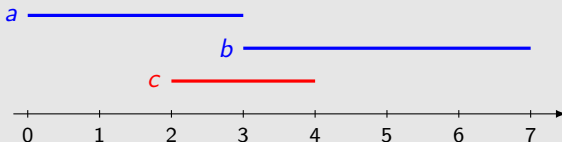
$[0, 10)$, $[1, 2)$, $[2, 3)$, $[3, 4)$

Pravidlo vybere $[0, 10)$ a skončí s jedním intervalem. Optimum jsou tři krátké intervaly.

Protipříklad 2: nejkratší interval

Příklad 3 (Nejkratší selhává)

Uvažujme $a = [0, 3)$, $b = [3, 7)$, $c = [2, 4)$.

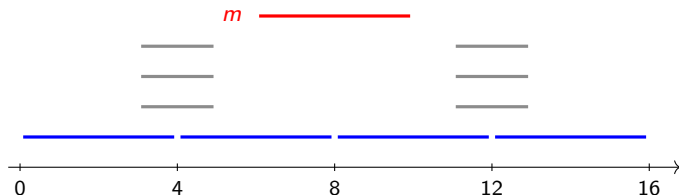


Interval c je nejkratší, ale překrývá se s a i b . Pravidlo vrátí $\{c\}$; optimum je $\{a, b\}$.

Délka intervalu ignoruje, *kde* interval leží.

Protipříklad 3: nejmenší počet překryvů

Vypadá nejrozumněji, protipříklad je proto složitější.



- krajní modré: 3 překryvy prostřední modré: 4
- každá šedá kopie: 4 *m*: jen 2

Pravidlo začne intervalem *m*, zablokuje oba prostřední modré a vlevo i vpravo přidá nejvýše jeden interval — *tři* místo *čtyř*.

Co ukazují protipříklady

- **Délka** ignoruje, *kde* interval leží.
- **Čas začátku** ignoruje, kolik prostoru interval po sobě ponechá.
- **Počet překryvů** se sice dívá na okolí, ale jen staticky a nerozlišuje, že překážet dvěma *důležitým* intervalům je horší než překážet čtyřem postradatelným.

Každý protipříklad ukazuje, *kteřá informace nestačí*.

Co ukazují protipříklady

- **Délka** ignoruje, *kde* interval leží.
- **Čas začátku** ignoruje, kolik prostoru interval po sobě ponechá.
- **Počet překryvů** se sice dívá na okolí, ale jen staticky a nerozlišuje, že překážet dvěma *důležitým* intervalům je horší než překážet čtyřem postradatelným.

Každý protipříklad ukazuje, *která informace nestačí*.

To vede k pravidlu, které se dívá na **pravý konec**: vybereme interval končící nejdříve. Tím zanecháme co nejvíce času všem dalším žádostem.

Algoritmus Earliest-Finish-Time

```
1 def EarliestFinishTime(I):  
2     seřaď intervaly v neklesajícím pořadí podle času konce  $f_i$   
3      $A \leftarrow \emptyset$ ;  $t \leftarrow -\infty$   
4     foreach interval  $[s_i, f_i)$  v seřazeném pořadí do  
5         if  $s_i \geq t$  then  
6              $A \leftarrow A \cup \{[s_i, f_i)\}$   
7              $t \leftarrow f_i$   
8     return A
```

Algoritmus je zjevně hladový: interval po přijetí nikdy neodebere.

Proč je nejdřívější konec *správné* kritérium?

Výměnný argument

Vezmeme libovolné optimální řešení a ukážeme, že jeho první volbu lze **bez zhoršení vyměnit** za naši hladovou volbu.

Lemma 4 (Bezpečnost nejdřívějšího konce)

Nechť a je interval s nejmenším časem konce. Existuje optimální řešení intervalového plánování, které obsahuje a .

Důkaz.

Nechť O je optimální řešení a o jeho nejdříve končící interval. Je-li $o = a$, jsme hotovi. Jinak z volby a plyne $f_a \leq f_o$. Položme $O' = (O - \{o\}) \cup \{a\}$.

Velikost se nezměnila: $|O'| = |O|$.

Všechny ostatní intervaly z O začínají nejdříve v čase f_o (s o se nepřekrývají, a kdyby některý skončil před začátkem o , končil by dříve než o). Protože $f_a \leq f_o$, nepřekrývá se a s žádným z nich. Množina O' je přípustná, a tedy optimální.

Pozor, co přesně jsme dokázali

Netvrdíme, že *každé* optimální řešení obsahuje nejdříve končící interval.

Tvrdíme, že *jedno* takové optimální řešení existuje. Pro bezpečnost první volby to stačí.

Věta 5 (Správnost Earliest-Finish-Time)

Algoritmus vrací množinu s největším možným počtem navzájem kompatibilních intervalů.

Důkaz.

První zvolený interval je podle lemmatu součástí nějakého optimálního řešení. Po jeho zvolení odstraníme nekompatibilní intervaly — zbytek je *nová instance téhož problému*. Indukcí podle počtu zbývajících intervalů dostaneme optimální řešení obsahující každou volbu algoritmu. Když algoritmus skončí, žádný kompatibilní interval už přidat nelze.

Tentýž algoritmus jinak: „zůstává napřed“

Nechť $a_1, \dots, a_k$ jsou intervaly zvolené hladově (v pořadí) a $o_1, \dots, o_m$ intervaly libovolného optimálního řešení.

Lemma 6 (Hladové řešení zůstává napřed)

Pro každé $r \leq \min\{k, m\}$ platí $f_{a_r} \leq f_{o_r}$.

Důkaz.

Indukcí podle r . Pro $r = 1$ tvrzení platí, protože a_1 končí nejdříve ze všech intervalů.

Nechť $f_{a_{r-1}} \leq f_{o_{r-1}}$. Interval o_r je kompatibilní s o_{r-1} , takže $s_{o_r} \geq f_{o_{r-1}} \geq f_{a_{r-1}}$. Když algoritmus vybíral a_r , byl tedy o_r jedním z kandidátů; algoritmus z nich vybral ten s nejmenším koncem.

Tentýž algoritmus jinak: „zůstává napřed“

Nechť $a_1, \dots, a_k$ jsou intervaly zvolené hladově (v pořadí) a $o_1, \dots, o_m$ intervaly libovolného optimálního řešení.

Lemma 6 (Hladové řešení zůstává napřed)

Pro každé $r \leq \min\{k, m\}$ platí $f_{a_r} \leq f_{o_r}$.

Důkaz.

Indukcí podle r . Pro $r = 1$ tvrzení platí, protože a_1 končí nejdříve ze všech intervalů.

Nechť $f_{a_{r-1}} \leq f_{o_{r-1}}$. Interval o_r je kompatibilní s o_{r-1} , takže $s_{o_r} \geq f_{o_{r-1}} \geq f_{a_{r-1}}$. Když algoritmus vybíral a_r , byl tedy o_r jedním z kandidátů; algoritmus z nich vybral ten s nejmenším koncem.

Slovy: hladový rozvrh nikdy nekončí později než odpovídající část optima. Pro další intervaly tedy nikdy neponechá méně času.

Dokončení argumentu a složitost

Kdyby $k < m$, interval o_{k+1} by začínal nejdříve v čase $f_{o_k} \geq f_{a_k}$. Algoritmus by jej mohl přidat — nemohl by skončit po k intervalech. **Spor.** Tedy $k \geq m$, a protože m je optimum, $k = m$.

Složitost

- seřazení podle času konce: $O(n \log n)$;
- průchod: $O(n)$ (jedno porovnání na interval).

Celkem $O(n \log n)$ — dominuje třídění.

Přicházejí-li intervaly už seřazené, běží celý algoritmus v $O(n)$.

Předpoklad o uspořádání vstupu tedy opět ovlivňuje výslednou složitost.

Minimalizace maximálního zpoždění

Jeden procesor, úlohy $1, \dots, n$. Úloha i potřebuje čas $p_i > 0$ a má termín d_i . Procesor zpracovává nejvýše jednu úlohu a nepřerušuje ji.

Je-li úloha i dokončena v čase C_i , její **zpoždění** je

$$L_i = C_i - d_i \quad (\text{nekladné} = \text{stihli jsme to}).$$

Cílem je minimalizovat

$$L_{\max} = \max_i L_i.$$

Minimalizace maximálního zpoždění

Jeden procesor, úlohy $1, \dots, n$. Úloha i potřebuje čas $p_i > 0$ a má termín d_i . Procesor zpracovává nejvýše jednu úlohu a nepřerušuje ji.

Je-li úloha i dokončena v čase C_i , její **zpoždění** je

$$L_i = C_i - d_i \quad (\text{nekladné} = \text{stihli jsme to}).$$

Cílem je minimalizovat

$$L_{\max} = \max_i L_i.$$

Bez prostojů

Kdyby v rozvrhu byl prostoj, posuneme všechny následující úlohy doleva o jeho délku. Žádný C_i se nezvětší, tedy ani žádné L_i . Existuje proto optimální rozvrh **bez prostojů**.

Pravidlo nejkratší úlohy napřed selhává

Příklad 7 (Dvě úlohy stačí)

$$A: p_A = 1, d_A = 100$$

$$B: p_B = 10, d_B = 10$$

- **Nejkratší napřed** (A, B): $C_B = 11$, tedy $L_{\max} = 1$.
- **Opačné pořadí** (B, A): $C_B = 10$, $C_A = 11$, tedy $L_{\max} = 0$.

Délka úlohy sama o sobě nic neříká o tom, jak moc úloha *spěchá*. Pro náš cíl je přirozenější řídit se přímo termíny.

Algoritmus Earliest-Deadline-First

Úlohy seřadíme v neklesajícím pořadí podle termínů d_i a v tomto pořadí je bez mezer provedeme.

Jiný typ důkazu: opravujeme lokální chyby

Zde nevyměníme první prvek. Budeme opravovat inverze v libovolném rozvrhu.

Definice 8 (Inverze termínů)

V rozvrhu tvoří dvojice úloh (i, j) **inverzi termínů**, jestliže je i naplánována před j , ale $d_i > d_j$.

Rozvrh bez inverzí je přesně rozvrh v neklesajícím pořadí termínů — až na libovolné pořadí úloh se *stejným* termínem.

Lemma 9 (Výměna sousední inverze)

Jsou-li v rozvrhu bez prostojů dvě sousední úlohy i, j v tomto pořadí a platí $d_i > d_j$, jejich výměna maximální zpoždění nezvětší.

Důkaz lemmatu o výměně

Důkaz.

Nechť t je čas, kdy začíná úloha i . Původně

$$C_i = t + p_i, \quad C_j = t + p_i + p_j.$$

Po výměně: $C'_j = t + p_j \leq C_j$, tedy $L'_j \leq L_j$.

Úloha i po výměně končí tam, kde původně končila j , tj. $C'_i = C_j$. Protože $d_i > d_j$,

$$L'_i = C_j - d_i < C_j - d_j = L_j.$$

Zpoždění obou vyměněných úloh je tedy nejvýše původní L_j ; časy ostatních úloh se nezměnily, takže se nezvětšilo ani maximum.

Správnost EDF

Věta 10 (Správnost Earliest-Deadline-First)

Rozvrh v neklesajícím pořadí termínů minimalizuje maximální zpoždění.

Důkaz.

Vezměme optimální rozvrh bez prostojů. Obsahuje-li inverzi, obsahuje také *sousední* inverzi; tu podle lemmatu vyměníme, aniž zvětšíme $L_{\max}$. Každá výměna sníží počet inverzí alespoň o jednu, takže po konečně mnoha krocích získáme rozvrh **bez inverzí**, který je stále optimální.

Správnost EDF

Věta 10 (Správnost Earliest-Deadline-First)

Rozvrh v neklesajícím pořadí termínů minimalizuje maximální zpoždění.

Důkaz.

Vezměme optimální rozvrh bez prostojů. Obsahuje-li inverzi, obsahuje také *sousední* inverzi; tu podle lemmatu vyměníme, aniž zvětšíme $L_{\max}$. Každá výměna sníží počet inverzí alespoň o jednu, takže po konečně mnoha krocích získáme rozvrh **bez inverzí**, který je stále optimální.

Pozor: rozvrh bez inverzí není určen jednoznačně — úlohy se stejným termínem mohou jít v libovolném pořadí. Ale *všechny* rozvrhy bez prostojů a bez inverzí mají *stejně* $L_{\max}$ (viz další slajd), takže i výstup algoritmu je optimální.

Proč na pořadí uvnitř bloku nezáleží

V rozvrhu bez inverzí jdou úlohy po **blocích** se shodným termínem, seřazených podle termínů.

- Blok úloh s termínem d zabírá vždy tentýž časový úsek: jeho začátek i délka jsou dány součty dob p_i předchozích bloků a bloku samotného.
- Ty na vnitřním pořadí nezávisí.
- Největší zpoždění v bloku má úloha končící poslední — a ta končí *na konci bloku* bez ohledu na vnitřní pořadí.
- Její zpoždění je „konec bloku — d “.

Maximum přes všechny bloky je tedy pro každý rozvrh bez inverzí totéž.

Složitost

Seřazení n úloh podle termínů: $O(n \log n)$. Naplánování: jediný průchod, $O(n)$ — pro každou úlohu přičteme p_i k běžícímu času. Celkem $O(n \log n)$; opět dominuje třídění.

Obecnější použití tohoto důkazu

Chceme-li ukázat správnost *uspořádání*, začneme libovolným optimálním řešením a lokálními výměnami odstraňujeme jeho „chyby“.

Pokud výměny nezhorší hodnotu a dovedou nás k řešení vytvořenému algoritmem, máme důkaz optimality.

Huffmanovo kódování

Chceme zapisovat znaky binárními řetězci: časté znaky krátce, vzácné delší.

Nestačí přidělit libovolné řetězce. Při kódech

$$a \mapsto 0, \quad b \mapsto 01$$

nevíme při čtení nuly, zda už skončil a , nebo teprve začíná b .

Definice 11 (Prefixový kód)

Binární kód je **prefixový**, jestliže žádné kódové slovo není prefixem jiného kódového slova.

Huffmanovo kódování

Chceme zapisovat znaky binárními řetězci: časté znaky krátce, vzácné delší.

Nestačí přidělit libovolné řetězce. Při kódech

$$a \mapsto 0, \quad b \mapsto 01$$

nevíme při čtení nuly, zda už skončil a , nebo teprve začíná b .

Definice 11 (Prefixový kód)

Binární kód je **prefixový**, jestliže žádné kódové slovo není prefixem jiného kódového slova.

Prefixový kód znázorníme binárním stromem: levá hrana 0, pravá 1, znaky v listech. Žádný list neleží na cestě k jinému listu, takže kód je automaticky prefixový.

Účelová funkce

V optimálním stromu nemá žádný vnitřní vrchol jediného potomka — ten bychom stáhli a listy pod ním posunuli o úroveň výš. Stačí tedy uvažovat **plné** binární stromy.

Přichází-li znak c s četností $w(c) > 0$ a má ve stromu hloubku $d_T(c)$, je délka zakódovaného textu

$$B(T) = \sum_c w(c) d_T(c).$$

Hledáme prefixový strom s nejmenší hodnotou $B(T)$.

Huffmanův algoritmus — staví strom zdola

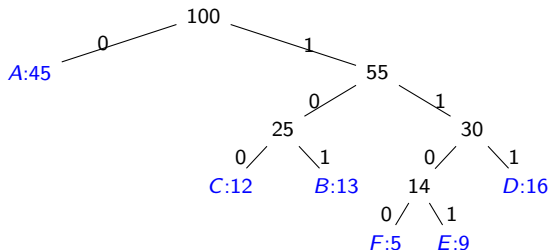
```
1 def Huffman(znaky s četnostmi w):  
2     pro každý znak vytvoř jednovrcholový strom s vahou  $w(c)$   
3     vlož všechny stromy do prioritní fronty  $Q$   
4     while  $|Q| > 1$  do  
5          $T_1 \leftarrow \text{extract-min}(Q)$ ;  $T_2 \leftarrow \text{extract-min}(Q)$   
6         vytvoř nový vrchol s potomky  $T_1, T_2$  a vahou  $w(T_1) + w(T_2)$   
7         vlož jej do  $Q$   
8     return jediný strom v  $Q$ 
```

V každém kroku spojíme dva nejlehčí stromy pod nového rodiče a jejich váhy sečteme.

Příklad: šest znaků

znak	A	B	C	D	E	F
četnost	45	13	12	16	9	5

Slučování: $5+9=14$, $12+13=25$, $14+16=30$, $25+30=55$, $45+55=100$.



$45 \cdot 1 + (12+13+16) \cdot 3 + (5+9) \cdot 4 = 224$ bitů na 100 znaků; pevný tříbitový kód by potřeboval 300.

Proč slučujeme dvě nejmenší četnosti

Dá se očekávat, že nejvzácnější znaky budou ležet nejhlouběji. Potřebujeme však silnější tvrzení.

Lemma 12 (Sourozenci v optimálním stromu)

Nechť x, y jsou dva znaky s nejmenšími četnostmi. Existuje optimální plný binární kódový strom, v němž jsou x, y sourozenci a leží v maximální hloubce.

Důkaz.

BÚNO $w(x) \leq w(y)$. V optimálním plném stromu vyberme dva sourozenecké listy a, b v maximální hloubce (leží-li v jednom z nich x , nazvěme jej a).

Neleží-li x v listu a , prohodme jej se znakem a' , který tam leží. Protože $w(x) \leq w(a')$ a $d(a) \geq d(x)$, je změna ceny

$$\Delta = (w(a') - w(x))(d(x) - d(a)) \leq 0.$$

Důkaz lemmatu (dokončení)

Důkaz.

Tedy list b . Leží-li v něm y , jsme hotovi. Jinak v něm leží znak c ; ten není x (ten už je v a) ani y , a protože x, y jsou dva nejméně časté znaky, platí $w(y) \leq w(c)$. List b je v maximální hloubce, takže prohození y a c podle téhož výpočtu cenu nezvětší.

Po nejvýše dvou výměnách leží x, y v sourozeneckých listech v maximální hloubce a cena se nezvětšila.

Důkaz lemmatu (dokončení)

Důkaz.

Ted' list b . Leží-li v něm y , jsme hotovi. Jinak v něm leží znak c ; ten není x (ten už je v a) ani y , a protože x, y jsou dva nejméně časté znaky, platí $w(y) \leq w(c)$. List b je v maximální hloubce, takže prohození y a c podle téhož výpočtu cenu nezvětší.

Po nejvýše dvou výměnách leží x, y v sourozeneckých listech v maximální hloubce a cena se nezvětšila.

Zmenšení instance

Spojme x, y do pomocného znaku z s $w(z) = w(x) + w(y)$. Nahradíme-li sourozenecké listy x, y jejich rodičem z , sníží se hloubka obou znaků o jedna, takže

$$B(T) = B(T') + w(x) + w(y).$$

Přírůstek *nezávisí na zbytku stromu*.

Optimalita Huffmanova kódování

Věta 13

Huffmanův algoritmus vytváří prefixový kód s nejmenší váženou délkou $B(T)$.

Důkaz.

Indukcí podle počtu znaků. Pro jediný znak tvoří strom jen kořen a $B(T) = 0$.

Pro více znaků algoritmus spojí dva nejméně časté znaky x, y do z . Podle lemmatu existuje optimální strom, v němž jsou x, y sourozenci; jejich nahrazením znakem z dostaneme zmenšenou instanci. Podle indukčního předpokladu pro ni algoritmus vytvoří optimální strom a rozbalení z přidá ke všem řešením stejnou cenu $w(x) + w(y)$.

Složitost. $n - 1$ sloučení, každé dvě operace EXTRACT-MIN a jedno INSERT v $O(\log n) \Rightarrow O(n \log n)$ času a $O(n)$ paměti.

Jak poznat příležitost pro hladový algoritmus

Hladový algoritmus stojí za vyzkoušení, když řešení skládáme po částech a jedna volba zanechává *menší problém podobného tvaru*. Samotná podobnost však nestačí.

Dnes jsme viděli tři podoby obhajoby lokální volby:

- **intervaly:** první interval optima lze vyměnit za interval končící nejdříve;
- **termíny:** sousední inverzi lze opravit bez zhoršení;
- **prefixové kódy:** dva nejlehčí listy lze přesunout k sobě a problém zmenšit jejich sloučením.

Kdy hladovost naopak nečekat

Potřebujeme-li po lokální volbě znát mnoho podrobností o tom, jak jsme se k ní dostali, nebo mění-li se výhodnost volby podle pozdějších rozhodnutí, jednoduchý hladový algoritmus nebude stačit.

Kontrolní seznam důkazu hladového algoritmu

1. Je hladové kritérium přesně definované, včetně řešení shod?
2. Zůstává po každé volbě částečné řešení přípustné?
3. Dokážeme vytvořit malý protipříklad k jiným přirozeným pravidlům?
4. Lze optimální řešení upravit tak, aby obsahovalo hladovou volbu?
5. Nezvětší výměna nebo lokální oprava hodnotu cíle?
6. Je zbytek po volbě opět instancí stejného problému?
7. Zahrnuje analýza také třídění a použité datové struktury?

Kontrolní seznam důkazu hladového algoritmu

1. Je hladové kritérium přesně definované, včetně řešení shod?
2. Zůstává po každé volbě částečné řešení přípustné?
3. Dokážeme vytvořit malý protipříklad k jiným přirozeným pravidlům?
4. Lze optimální řešení upravit tak, aby obsahovalo hladovou volbu?
5. Nezvětší výměna nebo lokální oprava hodnotu cíle?
6. Je zbytek po volbě opět instancí stejného problému?
7. Zahrnuje analýza také třídění a použité datové struktury?

Na loot z úvodu (problém batohu) se hladový přístup nehodí: ať bereme věci podle ceny, váhy, nebo ceny za kilogram, vždy lze sestrojit instanci, kde nevratná volba zablokuje lepší kombinaci. Vrátime se k němu v přednáškách 7 (DP), 10 (meze) a 12 (aproximace).