# KMI/ALGO3 — Algoritmy 3

## 7. Dynamické programování II: vícerozměrné stavy

Jan Konečný

Katedra informatiky, PřF UP Olomouc

# Kde jsme: sylabus 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

# Když jeden index nestačí

Minule byl stav určen jediným indexem: prefixem intervalů, vrcholem DAG. Mnoho úloh však potřebuje víc:

- u batohu musíme vědět, které předměty smíme použít *a* kolik kapacity zbývá;
- u porovnávání řetězců sledujeme současně *dva* prefixy.

Musíme se tedy ptát, které parametry minulosti mohou ovlivnit budoucí možnosti. Každý takový parametr přidá tabulce další rozměr, takže návrh je hledáním nejmenší informace, která budoucnost úplně vystihuje.

# Dnešní tři vzory

- **Batoh** — pseudopolynomiální čas a jednorozměrná optimalizace paměti.
- **Editační vzdálenost** — tabulka dvou prefixů a rekonstrukce zarovnání.
- **Optimální BST** — intervalový stav, v němž jeden přechod zkouší *mnoho* možných kořenů.

# Problém 0/1 batohu

## Definice 1 (0/1 batoh)

$n$ předmětů, předmět $i$ má celočíselnou hmotnost $w_i > 0$ a hodnotu $v_i \geq 0$; batoh unese $W$. Přípustné řešení je $S \subseteq \{1, \ldots, n\}$ s $\sum_{i \in S} w_i \leq W$; maximalizujeme $\sum_{i \in S} v_i$.

# Problém 0/1 batohu

## Definice 1 (0/1 batoh)

$n$ předmětů, předmět $i$ má celočíselnou hmotnost $w_i > 0$ a hodnotu $v_i \geq 0$; batoh unese $W$. Přípustné řešení je $S \subseteq \{1, \ldots, n\}$ s $\sum_{i \in S} w_i \leq W$; maximalizujeme $\sum_{i \in S} v_i$.

**Hladově?** Řazení podle poměru $v_i / w_i$ řeší *zlomkovou* variantu. Pro 0/1 batoh selže:

$W = 4$, předměty $(w, v) = (3, 5), (2, 3), (2, 3)$.

První má nejlepší poměr $5/3$, ale po jeho vložení se už nic nevejde — hodnota 5. Dva lehčí mají dohromady 6.

# Problém 0/1 batohu

## Definice 1 (0/1 batoh)

$n$ předmětů, předmět $i$ má celočíselnou hmotnost $w_i > 0$ a hodnotu $v_i \geq 0$; batoh unese $W$. Přípustné řešení je $S \subseteq \{1, \ldots, n\}$ s $\sum_{i \in S} w_i \leq W$; maximalizujeme $\sum_{i \in S} v_i$.

**Hladově?** Řazení podle poměru $v_i / w_i$ řeší *zlomkovou* variantu. Pro 0/1 batoh selže:

$W = 4$, předměty $(w, v) = (3, 5), (2, 3), (2, 3)$.

První má nejlepší poměr $5/3$, ale po jeho vložení se už nic nevejde — hodnota 5. Dva lehčí mají dohromady 6.

Rozhodnutí tedy nelze udělat jen podle lokálních parametrů předmětu. Musíme porovnat „přijmout“ s „vynechat“ — a přitom sledovat *zbývající kapacitu*.

# Volba stavu

## Definice 2 (Stav batohu)

Pro $i = 0, \ldots, n$ a $c = 0, \ldots, W$ je $K[i, c]$ největší hodnota, kterou lze získat z *prvních* $i$ předmětů při kapacitě nejvýše $c$.

- První index omezuje množinu dostupných předmětů.
- Druhý index si pamatuje zdroj, který rozhodnutí spotřebovávají.

Bez kapacity by dvě částečná řešení se stejnou hodnotou vypadala nerozlišitelně — přestože jedno ponechává víc prostoru pro budoucnost.

**Základní případy:** $K[0, c] = 0$ pro všechna $c$, $K[i, 0] = 0$ pro všechna $i$.

# Přechod

- $w_i > c$: předmět se nevejde, tedy $K[i, c] = K[i-1, c]$.
- $w_i \leq c$: buď jej vynecháme, nebo vezmeme — po přijetí zbývá kapacita $c - w_i$ a smíme používat jen předměty $1, \ldots, i-1$.

$$K[i, c] = \begin{cases} K[i-1, c], & w_i > c, \\ \max\{K[i-1, c],\ v_i + K[i-1, c-w_i]\}, & w_i \leq c. \end{cases}$$

**Věta 3**

*Tabulka s uvedenými základními případy a přechodem obsahuje v $K[n, W]$ optimální hodnotu 0/1 batohu.*

*Důkaz.* Indukcí podle $i$. Optimum stavu $(i, c)$ buď předmět $i$ neobsahuje (pak je řešením stavu $(i-1, c)$), nebo obsahuje (pak zbytek řeší $(i-1, c-w_i)$). Oba kandidáti jsou zároveň dosažitelní.

# Pořadí a složitost

Každá buňka řádku $i$ závisí *jen* na řádku $i-1$; vyplňujeme tedy po řádcích. Stavů je $(n+1)(W+1)$, každý za konstantní čas:

$$\text{čas } O(nW), \qquad \text{paměť } O(nW).$$

## Pseudopolynomiální čas

Číslo $W$ je v binárním vstupu zapsáno pomocí $\Theta(\log W)$ bitů. Algoritmus je polynomiální v *číselné hodnotě* kapacity, ale může být exponenciální v délce jejího zápisu.

Tabulka o $W$ sloupcích je přiměřená pro kapacitu v tisících. Pro kapacitu $10^{12}$ ji nevytvoříme, ačkoli číslo má jen asi čtyřicet bitů.

Asymptotiku tedy vždy vztahujeme k reprezentaci vstupu, ne k hodnotám čísel.

# Nedá se to udělat lépe?

Pravděpodobně ne:

- Rozhodovací verze 0/1 batohu je NP-úplná, úloha sama je tedy NP-těžká (přesný význam vysvětlíme v přednášce 10).
- Polynomiální algoritmus proto nečekáme.

Čas $O(nW)$ je rozumný kompromis: pro kapacity zapsatelné rozumně velkým číslem řeší úlohu *přesně* a rychle.

Pro příliš velké kapacity lze použít aproximační algoritmy (přednáška 12).

# Rekonstrukce předmětů

Začneme v $(n, W)$: je-li $K[i, c] = K[i-1, c]$, předmět $i$ vynecháme; jinak jej vybereme a přejdeme do $(i-1, c-w_i)$.

| $K[i,c]$ | $c=0$ | $c=1$ | $c=2$ | $c=3$ | $c=4$ |
|---|---|---|---|---|---|
| $i=0$ | 0 | 0 | 0 | 0 | 0 |
| $i=1$ | 0 | 0 | 0 | 5 | 5 |
| $i=2$ | 0 | 0 | 3 | 5 | 5 |
| $i=3$ | 0 | 0 | 3 | 5 | 6 |

$W = 4$, předměty $(3,5), (2,3), (2,3)$. $K[3,4] = 6 \neq K[2,4] = 5 \Rightarrow$ bereme předmět 3, jdeme do $(2,2)$. $K[2,2] = 3 \neq K[1,2] = 0 \Rightarrow$ bereme předmět 2, jdeme do $(1,0)$. $K[1,0] = K[0,0] \Rightarrow$ předmět 1 vynecháme. Optimum $\{2,3\}$ s hodnotou 6.

Při shodě obou kandidátů existuje více optimálních řešení. Chceme-li navíc např. nejmenší počet předmětů, musí být toto sekundární kritérium součástí stavu nebo pravidla pro řešení shod.

# Větší tabulka: $W = 7$, čtyři předměty

| $i$ | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| $w_i$ | 1 | 3 | 4 | 5 |
| $v_i$ | 1 | 4 | 5 | 7 |

| $K[i, c]$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| $i = 0$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| $i = 1$ | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| $i = 2$ | 0 | 1 | 1 | 4 | 5 | 5 | 5 | 5 |
| $i = 3$ | 0 | 1 | 1 | 4 | 5 | 6 | 6 | 9 |
| $i = 4$ | 0 | 1 | 1 | 4 | 5 | 7 | 8 | 9 |

Několik kontrolních míst:

- $K[2, 3] = \max\{K[1, 3]{=}1,\ 4 + K[1, 0]{=}4\} = 4$ — vyplatí se vyhodit lehký předmět 1 a vzít předmět 2.
- $K[3, 7] = \max\{K[2, 7]{=}5,\ 5 + K[2, 3]{=}9\} = 9$.
- $K[4, 7] = \max\{K[3, 7]{=}9,\ 7 + K[3, 2]{=}8\} = 9$ — těžký předmět 4 se nevyplatí, ačkoli má nejvyšší hodnotu.

# Rekonstrukce z té tabulky

| buňka | porovnání | rozhodnutí |
|---|---|---|
| $(4,7)$ | $K[4,7] = 9 = K[3,7]$ | vynech 4, jdi na $(3,7)$ |
| $(3,7)$ | $K[3,7] = 9 \neq K[2,7] = 5$ | vezmi 3, jdi na $(2,7-4)$ |
| $(2,3)$ | $K[2,3] = 4 \neq K[1,3] = 1$ | vezmi 2, jdi na $(1,3-3)$ |
| $(1,0)$ | $K[1,0] = 0 = K[0,0]$ | vynech 1, konec |

Řešení $\{2,3\}$: hmotnost $3+4=7$ (batoh plný), hodnota $4+5=9$. ✓

## Co by udělal hladový algoritmus

Hustoty: $v_i/w_i = 1{,}00 \;\; 1{,}33 \;\; 1{,}25 \;\; 1{,}40$. Hladový algoritmus vezme nejprve předmět 4 ($w=5$), pak už se vejde jen předmět 1 — hodnota 8 místo 9.

# Jednorozměrná tabulka na téže instanci

Jediné pole $A[0..7]$, kapacity sestupně:

| po předmětu | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| — (start) | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 $(1, 1)$ | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 2 $(3, 4)$ | 0 | 1 | 1 | 4 | 5 | 5 | 5 | 5 |
| 3 $(4, 5)$ | 0 | 1 | 1 | 4 | 5 | 6 | 6 | 9 |
| 4 $(5, 7)$ | 0 | 1 | 1 | 4 | 5 | 7 | 8 | 9 |

Řádky se shodují s plnou tabulkou; paměť klesla z $O(nW)$ na $O(W)$.

# Jednorozměrná tabulka na téže instanci

Jediné pole $A[0..7]$, kapacity sestupně:

| po předmětu | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| — (start) | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 $(1,1)$ | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 2 $(3,4)$ | 0 | 1 | 1 | 4 | 5 | 5 | 5 | 5 |
| 3 $(4,5)$ | 0 | 1 | 1 | 4 | 5 | 6 | 6 | 9 |
| 4 $(5,7)$ | 0 | 1 | 1 | 4 | 5 | 7 | 8 | 9 |

Řádky se shodují s plnou tabulkou; paměť klesla z $O(nW)$ na $O(W)$.

Vzestupně, jen pro předmět 2 ($w$=3, $v$=4):

$$A[3] \leftarrow 4, \quad A[4] \leftarrow 5, \quad A[5] \leftarrow 5, \quad A[6] \leftarrow \max\{1,\, 4 + \underbrace{A[3]}_{\text{už } 4}\} = 8$$

Předmět 2 byl použit *dvakrát* — to je odpověď na *jinou* úlohu.

# Vzestupný průchod řeší jinou úlohu

Vzestupný průchod řeší neomezený batoh: každý typ předmětu smí být použit libovolně mnohokrát.

## Neomezený batoh

Stav je *jednorozměrný* — na prefixu předmětů nezáleží, protože každý typ je stále k dispozici:

$$A[c] = \max\Big\{0,\ \max_{i\,:\,w_i \le c} \big\{v_i + A[c - w_i]\big\}\Big\}$$

Čas $O(nW)$, paměť $O(W)$.

| | **0/1 batoh** | **neomezený** |
|---|---|---|
| stav | $K[i, c]$ | $A[c]$ |
| kapacity ve smyčce | sestupně | vzestupně |
| odkaz vpravo | na řádek $i - 1$ | na *týž* řádek |

Oba algoritmy se tedy liší jediným slovem v hlavičce cyklu.

# Optimalizace paměti

Pro samotnou hodnotu stačí jediné pole $A[0..W]$:

$$A[c] \leftarrow \max\{A[c],\ v_i + A[c - w_i]\}.$$

## Lemma 4 (Kapacity musí klesat)

*Prochází-li se kapacita pro každý předmět sestupně od $W$ k $w_i$, čte pravá strana vždy hodnoty předchozího řádku $K[i-1, \cdot]$ a výpočet je správný.*

## Důkaz.

Při klesajícím průchodu je $c - w_i < c$ a tato buňka v aktuálním průchodu ještě nebyla změněna. Při rostoucím průchodu mohla být $A[c - w_i]$ už zlepšena *týmž* předmětem — ten by se přidal podruhé.

## Příklad 5

Jediný předmět $(w, v) = (2, 3)$, kapacita 4. Rostoucí průchod nastaví $A[2] = 3$ a pak $A[4] = 3 + A[2] = 6$ — předmět vložil dvakrát.

# Optimalizace má cenu

Závislost jen na předchozím řádku dovoluje staré hodnoty zahodit. Tabulka však současně sloužila jako *mapa pro zpětný průchod*.

Před optimalizací paměti se proto ptáme:

*Potřebujeme jen optimální hodnotu, nebo i konkrétní řešení?*

Chceme-li konkrétní předměty, můžeme

- ponechat celou tabulku,
- ukládat rozhodovací ukazatele,
- nebo použít techniku opakovaného výpočtu (viz Hirschberg dále).

# Editační (Levenshteinova) vzdálenost

$X = x_1 \dots x_m$, $Y = y_1 \dots y_n$. Operace: vložení, smazání, náhrada — každá za jednu. Hledáme nejmenší počet operací převádějících $X$ na $Y$.

## Definice 6 (Stav)

$D[i, j]$ je nejmenší cena převodu prefixu $x_1 \dots x_i$ na prefix $y_1 \dots y_j$, kde $0 \leq i \leq m$, $0 \leq j \leq n$.

**Základní případy:** $D[0, j] = j$ (samá vložení), $D[i, 0] = i$ (samá smazání).

Označme $\sigma(i, j) = 0$ při $x_i = y_j$ a 1 jinak. Pro $i, j > 0$:

$$D[i,j] = \min \begin{cases} D[i-1,j] + 1, & \text{smazání } x_i, \\ D[i,j-1] + 1, & \text{vložení } y_j, \\ D[i-1,j-1] + \sigma(i,j), & \text{shoda nebo náhrada.} \end{cases}$$

# Pozor na „poslední operaci“

Argument „podívejme se na poslední operaci skriptu“ nefunguje: editační operace smějí působit *kdekoli* v řetězci, takže optimální skript může končit třeba vložením znaku na úplný začátek.

## Zarovnání

Zápis obou řetězců pod sebe do sloupců, do nichž smíme vkládat mezery „−“: žádný sloupec nemá dvě mezery, horní řádek bez mezer dává $X$, dolní $Y$.

$(x_k, -)$ = smazání, $(-, y_\ell)$ = vložení, $(x_k, y_\ell)$ = shoda (cena 0) nebo náhrada (cena 1).

*Zarovnání a editační skripty popisují totéž.* Ze zarovnání přečteme skript téže ceny; naopak v libovolném skriptu sledujeme osud každého znaku $x_k$ (smazán / přežil jako nějaké $y_\ell$) a sloučením opakovaných náhrad dostaneme zarovnání, které nestojí víc.

# Korektnost podle posledního sloupce

## Věta 7

*$D[m, n]$ je nejmenší počet vložení, smazání a náhrad převádějících $X$ na $Y$.*

## Důkaz.

Indukcí podle $i + j$ ukážeme, že $D[i, j]$ je nejmenší cena *zarovnání* prefixů. Základ indukce je zřejmý.

Pro $i, j > 0$ má optimální zarovnání poslední sloupec právě jednoho ze tří tvarů:

- $(x_i, -)$: zbytek řeší $(i - 1, j)$, celkem $\geq D[i - 1, j] + 1$;
- $(-, y_j)$: zbytek řeší $(i, j - 1)$, celkem $\geq D[i, j - 1] + 1$;
- $(x_i, y_j)$: zbytek řeší $(i - 1, j - 1)$, celkem $\geq D[i - 1, j - 1] + \sigma(i, j)$.

Optimum tedy není menší než minimum v rekurenci; a každého kandidáta lze dosáhnout připojením odpovídajícího sloupce k optimu menšího stavu.

# Vyplnění a rekonstrukce

Každá buňka závisí na levém, horním a levém horním sousedu — tabulku lze vyplnit po řádcích i sloupcích. Čas $O(mn)$, paměť $O(mn)$.

| $D[i,j]$ | $\varepsilon$ | h | a | d |
|---|---|---|---|---|
| $\varepsilon$ | 0 | 1 | 2 | 3 |
| h | 1 | 0 | 1 | 2 |
| r | 2 | 1 | 1 | 2 |
| a | 3 | 2 | 1 | 2 |
| d | 4 | 3 | 2 | 1 |

$X =$ `hrad`, $Y =$ `had`. Zpětně z $D[4,3] = 1$: diagonála (shoda `d`), diagonála (shoda `a`), *nahoru* (smazání `r`, cena 1), diagonála (shoda `h`).

Diagonální krok = shoda/náhrada, nahoru = smazání, doleva = vložení.

# Školní příklad: `kitten` → `sitting`

| $D$ | $\varepsilon$ | s | i | t | t | i | n | g |
|---|---|---|---|---|---|---|---|---|
| $\varepsilon$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| k | 1 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| i | 2 | 2 | 1 | 2 | 3 | 4 | 5 | 6 |
| t | 3 | 3 | 2 | 1 | 2 | 3 | 4 | 5 |
| t | 4 | 4 | 3 | 2 | 1 | 2 | 3 | 4 |
| e | 5 | 5 | 4 | 3 | 2 | 2 | 3 | 4 |
| n | 6 | 6 | 5 | 4 | 3 | 3 | 2 | 3 |

Vzdálenost je $D[6,7] = 3$. Zpětné čtení modré cesty (od pravého dolního rohu):

| krok | směr | operace |
|---|---|---|
| $(6,7) \to (6,6)$ | doleva | vlož g |
| $(6,6) \to (5,5)$ | diagonála | ponech `n` |
| $(5,5) \to (4,4)$ | diagonála | nahraď `e`→`i` |
| $(4,4) \to (3,3) \to (2,2) \to (1,1)$ | diagonály | ponech `t`, `t`, `i` |
| $(1,1) \to (0,0)$ | diagonála | nahraď `k`→`s` |

# Jak číst tabulku pouhým okem

- **Diagonála nul** v levém horním rohu znamená, že prefixy zatím souhlasí.
- **Hodnota vlevo nahoře** $+0$ = znaky se shodují — cesta pokračuje diagonálně a *zadarmo*.
- **Rostoucí okraje** ($D[i,0] = i$, $D[0,j] = j$) jsou triviální případy „vše smaž“ / „vše vlož“.
- **Sousední buňky se liší nejvýše o 1.** Plyne to z rekurence — a je to užitečná kontrola při ručním počítání.

## Rychlá kontrola výsledku

Vždy platí

$$\big|\, m - n \,\big| \;\le\; D[m,n] \;\le\; \max\{m,n\}.$$

Zde $|6-7| = 1 \le 3 \le 7$. ✓

# Blízká příbuzná: nejdelší společná podposloupnost

Zakážeme-li *náhradu* a necháme jen vložení a smazání za cenu 1, dostaneme jinou klasickou úlohu.

## LCS

$$L[i,j] = \begin{cases} 0, & i = 0 \text{ nebo } j = 0, \\ L[i-1,j-1] + 1, & x_i = y_j, \\ \max\{L[i-1,j],\, L[i,j-1]\}, & \text{jinak.} \end{cases}$$

Mezi oběma úlohami platí přímý vztah: znaky, které skript *nezmění*, tvoří společnou podposloupnost, takže $D_{\text{ins/del}}(X, Y) = m + n - 2L$.

Pro `kitten`/`sitting` je $L = 4$ (`ittn`), tedy $6 + 7 - 8 = 5$ operací — víc než 3, protože jsme si zakázali náhradu.

Je to táž tabulka s jinou účelovou funkcí — běžný způsob, jak z jednoho DP vyrobit celou rodinu úloh.

# Použití editační vzdálenosti

- `diff` **a verzovací systémy.** Řádky souboru jsou „znaky“; LCS řádků dává minimální sadu vložení a smazání. `git diff` hledá týž nejkratší skript, jen rychlejším (Myersovým) algoritmem.
- **Oprava překlepů a našeptávání.** Kandidáti se řadí podle editační vzdálenosti od napsaného slova. Efektivní implementace počítají tabulku *přírůstkově* nad slovníkovým trie.
- **Bioinformatika.** Needleman–Wunsch (globální zarovnání) je náš algoritmus s obecnými cenami; Smith–Waterman (lokální zarovnání) maximalizuje skóre podobnosti, přidá do maxima nulu a hledá nejlepší *úsek*.
- **Rozpoznávání řeči a gest.** *Dynamic time warping* je táž mřížka, jen se místo znaků porovnávají vzorky signálu a cesta smí čas „natahovat“.

# Obecné ceny a lineární paměť

- V biologickém zarovnání či opravě textu nemají operace stejnou cenu: zavedeme cenu mezery $\delta$ a cenu náhrady $\alpha(x_i, y_j)$ s $\alpha(a, a) = 0$. Přechod zůstává stejný — stav se nemění, protože typ potřebné informace je týž. (Needleman, Wunsch 1970.)
- Pro samotnou vzdálenost stačí *dva řádky*: paměť $O(\min\{m, n\})$, použijeme-li kratší řetězec jako rozměr řádku. Přímou rekonstrukci tím ale ztratíme.
- Hirschbergův algoritmus umí rekonstruovat zarovnání i v lineární paměti — spojení DP s metodou rozděl a panuj. (Hirschberg 1975.)

# Optimální binární vyhledávací strom

Seřazené klíče $1, \ldots, n$; klíč $i$ je vyhledáván s vahou $p_i \geq 0$. Minimalizujeme

$$\sum_{i=1}^{n} p_i \big(\mathrm{depth}(i) + 1\big).$$

Kořen má hloubku nula a stojí jedno porovnání.

Častěji hledané klíče chceme blízko kořene — ale lokální volba největší váhy nemusí dobře rozdělit zbytek.

## Definice 8 (Intervalový stav)

$C[i, j]$ je nejmenší vážená cena stromu obsahujícího *právě* klíče $i, i+1, \ldots, j$; pro prázdný interval $i > j$ je $C[i, j] = 0$.

Vlastnost BST určuje stav: je-li kořenem $r$, obsahuje levý podstrom souvislý interval klíčů *před* $r$ a pravý interval *za* $r$.

# Přechod zkouší všechny kořeny

$$C[i,j] = \min_{r=i,\ldots,j} \left\{ C[i,r-1] + C[r+1,j] + \sum_{k=i}^{j} p_k \right\}$$

## Důkaz.

Každý BST pro $[i,j]$ má právě jeden kořen $r$ a jeho podstromy obsahují kvůli vlastnosti BST přesně uvedené intervaly; kdyby některý nebyl optimální, nahradíme jej levnějším.

Cena obou podstromů počítá jejich kořeny od hloubky nula. Vložením pod $r$ se hloubka každého klíče zvýší o jedna — dodatečná cena je součet vah *všech* klíčů v intervalu, včetně porovnání v kořeni.

**Složitost.** Tabulku vyplňujeme podle rostoucí *délky intervalu*; součty vah z prefixových součtů v $O(1)$. Stavů $O(n^2)$, v každém $O(n)$ kořenů:

čas $O(n^3)$, paměť $O(n^2)$.

# Tři klíče, celý výpočet

Váhy $(p_1, p_2, p_3) = (5, 2, 1)$. Jednoprvkové intervaly: $C[1,1] = 5$, $C[2,2] = 2$, $C[3,3] = 1$.

**Dvojice** (přičítáme součet vah intervalu):

$$C[1,2] = \min\{\underbrace{0+2}_{r=1}, \underbrace{5+0}_{r=2}\} + 7 = 2 + 7 = 9,$$

$$C[2,3] = \min\{\underbrace{0+1}_{r=2}, \underbrace{2+0}_{r=3}\} + 3 = 1 + 3 = 4.$$

**Celý interval**, součet vah $= 8$:

$$C[1,3] = \min\{\underbrace{0 + C[2,3]}_{r=1} = 4,\ \underbrace{C[1,1] + C[3,3]}_{r=2} = 6,\ \underbrace{C[1,2] + 0}_{r=3} = 9\} + 8 = 12$$

# Tři klíče, celý výpočet

Váhy $(p_1, p_2, p_3) = (5, 2, 1)$. Jednoprvkové intervaly: $C[1,1] = 5$, $C[2,2] = 2$, $C[3,3] = 1$.

**Dvojice** (přičítáme součet vah intervalu):

$$C[1,2] = \min\{\underbrace{0+2}_{r=1}, \underbrace{5+0}_{r=2}\} + 7 = 2 + 7 = 9,$$

$$C[2,3] = \min\{\underbrace{0+1}_{r=2}, \underbrace{2+0}_{r=3}\} + 3 = 1 + 3 = 4.$$

**Celý interval**, součet vah $= 8$:

$$C[1,3] = \min\{\underbrace{0 + C[2,3]}_{r=1} = 4,\ \underbrace{C[1,1] + C[3,3]}_{r=2} = 6,\ \underbrace{C[1,2] + 0}_{r=3} = 9\} + 8 = 12$$

Nejlepší kořen je $r = 1$ — nejtěžší klíč nahoře. Kontrola přímým výpočtem: hloubky $0, 1, 2$, tedy $5 \cdot 1 + 2 \cdot 2 + 1 \cdot 3 = 12$. ✓

Kořen $r = 2$ by dal $5 \cdot 2 + 2 \cdot 1 + 1 \cdot 2 = 14$ — vyváženější strom zde prohrává, protože nejčastější klíč zatlačil o patro níž.

# Jak Knuth ubral jeden řád

Náš přechod zkouší *všechny* kořeny $r \in \{i, \ldots, j\}$ — odtud faktor $O(n)$ navíc.

Knuth (1971) dokázal, že optimální kořeny jsou monotónní: označíme-li $r[i, j]$ nejlepší kořen intervalu $[i, j]$, platí

$$r[i, j-1] \;\le\; r[i, j] \;\le\; r[i+1, j].$$

Stačí tedy procházet jen kořeny v tomto rozmezí. Součet délek rozmezí přes celou diagonálu je $O(n)$, takže celkem $O(n^3) \longrightarrow O(n^2)$.

Stav i rekurence zůstaly stejné; o řád rychleji počítáme jen proto, že jsme dokázali strukturní vlastnost optima.

Monotonii optimálních voleb využívají i další techniky (*Knuth–Yao speedup*, *divide & conquer DP optimization*); každá má své předpoklady.

# Jak číst rozměry tabulky

Rozměr stavu odpovídá otázce, kterou musí znát budoucí rozhodnutí:

- $K[i, c]$: které předměty jsou dostupné a kolik kapacity smíme použít;
- $D[i, j]$: jak dlouhé prefixy obou řetězců už převádíme;
- $C[i, j]$: který souvislý interval klíčů má tvořit podstrom.

Ne každý údaj ale patří do stavu:

- aktuální hodnota batohu je *výsledek*, který optimalizujeme, ne omezení budoucnosti;
- u editace nepotřebujeme celý dosavadní skript — pro budoucí cenu stačí dvojice délek prefixů.

# Když je stavem celá podmnožina

Rozměry stavu nemusí být čísla. U TSP potřebujeme vědět, *které* vrcholy jsme už navštívili — a to je celá podmnožina.

## Held–Karp (1962)

Start ve vrcholu 1. Pro $S \subseteq \{2, \ldots, n\}$ a $j \in S$ je $C[S, j]$ = nejlevnější cesta z 1, která kromě 1 navštívila přesně $S$ a končí ve $j$; $C[\{j\}, j] = w(1, j)$.

$$C[S, j] = \min_{i \in S \setminus \{j\}} \left\{ C[S \setminus \{j\}, i] + w(i, j) \right\}$$

Stavů je $2^n \cdot n$, každý přechod stojí $O(n)$, tedy čas $O(2^n n^2)$ a paměť $O(2^n n)$.

# Když je stavem celá podmnožina

Rozměry stavu nemusí být čísla. U TSP potřebujeme vědět, *které* vrcholy jsme už navštívili — a to je celá podmnožina.

## Held–Karp (1962)

Start ve vrcholu 1. Pro $S \subseteq \{2, \ldots, n\}$ a $j \in S$ je $C[S, j]$ = nejlevnější cesta z 1, která kromě 1 navštívila přesně $S$ a končí ve $j$; $C[\{j\}, j] = w(1, j)$.

$$C[S, j] = \min_{i \in S \setminus \{j\}} \left\{ C[S \setminus \{j\}, i] + w(i, j) \right\}$$

Stavů je $2^n \cdot n$, každý přechod stojí $O(n)$, tedy čas $O(2^n n^2)$ a paměť $O(2^n n)$.

Proti hrubé síle $O(n!)$ je to výrazné zlepšení ($n = 20$: $4 \cdot 10^8$ místo $2 \cdot 10^{18}$) — a přesto stále exponenciální. Pro $n = 40$ by tabulka měla přes $4 \cdot 10^{13}$ položek.

Dodnes nejlepší známý *přesný* algoritmus pro obecné TSP. K tomu, proč lepší nejspíš neexistuje, se dostaneme v přednášce 10.

# Katalog: co bylo stavem

| úloha | stav | počet stavů | přechod |
|---|---|---|---|
| Fibonacci | index | $n$ | $O(1)$ |
| vážené intervaly | prefix | $n$ | $O(1)$ |
| nejkratší cesty v DAG | vrchol | $\|V\|$ | $O(\deg)$ |
| 0/1 batoh | prefix + kapacita | $nW$ | $O(1)$ |
| editační vzdálenost | dva prefixy | $mn$ | $O(1)$ |
| optimální BST | interval | $n^2$ | $O(n)$ |
| TSP (Held–Karp) | podmnožina + konec | $2^n n$ | $O(n)$ |

## Čtěte to jako stupnici

Čím bohatší stav, tím obecnější úloha — a tím dříve narazíme na paměť. Návrh DP je hledáním *nejchudšího* stavu, který ještě stačí.

Poslední řádek berte jako varování: jakmile musíme do stavu dát podmnožinu, jsme obvykle u NP-těžké úlohy.

# Odhad složitosti DP

1. Spočítej počet přípustných kombinací indexů, tedy počet stavů.
2. Urči počet kandidátů zkoušených v jednom přechodu.
3. Přidej cenu předzpracování a rekonstrukce.
4. Zkontroluj, zda číselné rozměry tabulky nejsou exponenciální v délce *binárního zápisu* vstupu.

Batoh: $O(nW) \cdot O(1)$. Editace: $O(mn) \cdot O(1)$. BST: $O(n^2) \cdot O(n)$.

# Shrnutí

Vícerozměrný stav je způsob, jak shrnout *přesně ty* podmínky, které přenášíme do budoucnosti.

- Batoh kombinuje prefix předmětů s dostupnou kapacitou.
- Editační vzdálenost kombinuje dva prefixy a klasifikuje poslední *sloupec zarovnání*.
- Optimální BST používá interval klíčů a zkouší všechny kořeny.
- Optimalizace paměti vychází ze směru závislostí, ale může ztížit rekonstrukci.
- Čas $O(nW)$ je *pseudopolynomiální* — $W$ je číselná hodnota, ne délka zápisu.

Příště přejdeme od tabulek k tokům v sítích: reziduální síť a certifikát optimálnosti ve formě řezu.

# Literatura



V. I. Levenshtein. Binary Codes Capable of Correcting Deletions, Insertions, and Reversals. *Soviet Physics Doklady* 10(8) (1966), 707–710.


S. B. Needleman, C. D. Wunsch. A General Method Applicable to the Search for Similarities. . . *J. Mol. Biol.* 48(3) (1970), 443–453.


D. S. Hirschberg. A Linear Space Algorithm for Computing Maximal Common Subsequences. *CACM* 18(6) (1975), 341–343.


J. Kleinberg, É. Tardos. *Algorithm Design*. Addison–Wesley (2006), kap. 6.


T. H. Cormen a kol. *Introduction to Algorithms*. MIT Press (2001).