# KMI/ALGO3 — Algoritmy 3

## 6. Dynamické programování I: hledání stavů

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

# Co když bezpečnou volbu neznáme

- Hladové algoritmy: provedli jsme bezpečnou lokální volbu a už se k ní nevraceli.
- Rozděl a panuj: vytvářeli jsme menší, převážně *nezávislé* instance.

A co když bezpečnou první volbu neznáme a různé možnosti vedou ke *stejným* podproblémům? Pak se ptáme, jak uchovat výsledky menších rozhodnutí, abychom každý podproblém vyřešili právě jednou.

Dynamické programování rozloží problém na stavy s přechody.
Nejdůležitější částí návrhu je volba toho, co stav znamená.

# Pět součástí každého DP

1. význam stavu,
2. přechodová rovnice,
3. základní případy,
4. pořadí výpočtu,
5. rekonstrukce řešení.

## Složitost

$$\approx \textit{počet stavů} \times \textit{počet možností zkoušených v jednom stavu}$$

Pozor: druhý činitel se často zapomíná — tabulka $n \times n$ s lineárním přechodem stojí $\Theta(n^3)$, ne $\Theta(n^2)$.

# Kdy se podproblémy překrývají

$$F_0 = 0, \qquad F_1 = 1, \qquad F_n = F_{n-1} + F_{n-2}$$

Přímočará rekurze pro $F_n$ volá $F_{n-1}$ a $F_{n-2}$ — ale první větev při výpočtu $F_{n-1}$ znovu potřebuje $F_{n-2}$.

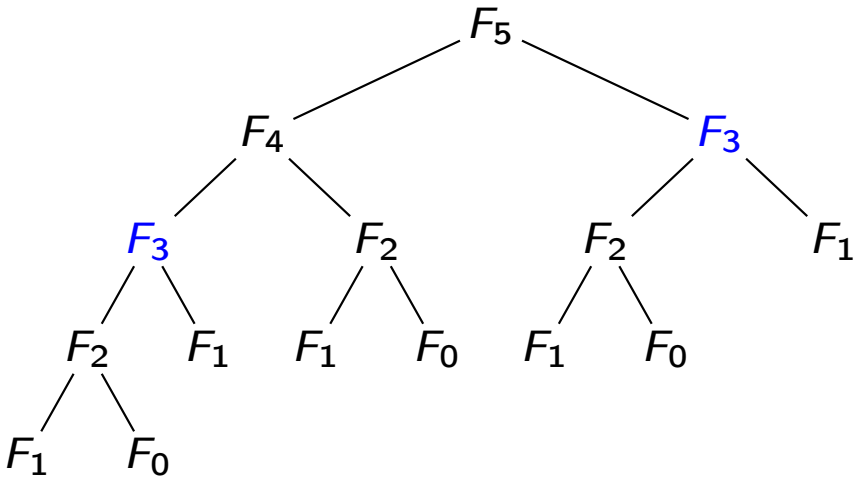
## Příklad 1 (Opakovaný podproblém)

$$F_5 \longrightarrow F_4, F_3, \qquad F_4 \longrightarrow F_3, F_2$$

Podproblém $F_3$ se objeví nejméně *dvakrát*. U větších indexů se opakování rychle násobí a rekurzivní strom roste exponenciálně.

Stačí si přitom hodnotu po prvním výpočtu zapamatovat.

# Rekurze bez pamatování




Modré podstromy jsou identické a počítají se dvakrát — a uvnitř nich se totéž opakuje znovu.

Celkový počet volání je $2F_{n+1} - 1$, tedy $\Theta(\varphi^n)$:

| $n$ | 10 | 30 | 50 |
|---|---|---|---|
| volání naivní rekurze | 177 | 2 692 537 | 40 730 022 147 |
| stavů s pamatováním | 11 | 31 | 51 |

Rozdíl je dán jen tím, že totéž nepočítáme dvakrát.

# Odkud se vzalo jméno metody



R. Bellman.
*Eye of the Hurricane: An Autobiography*.
World Scientific (1984).

Richard Bellman pracoval v 50. letech v RAND Corporation pod ministrem obrany, o němž píše, že měl „chorobný odpor ke slovu *výzkum*“. Potřeboval tedy pro svou metodu jméno, které nikoho nepohorší:

> *Slovo „dynamické“ má naprosto přesný význam… a navíc je nemožné použít je v hanlivém smyslu.*

Slovo „programování“ zde neznamená psaní programů, ale *plánování* či *sestavování rozvrhu* — stejně jako v pojmu „lineární programování“.

Název tedy nepopisuje metodu, nýbrž okolnosti, za kterých vznikala.

# Dvě podoby téže myšlenky

## Definice 2 (Memoizace a tabulace)

- Memoizace — rekurzivní výpočet *shora dolů*: hodnotu stavu po prvním vyhodnocení uložíme a každý další dotaz ji jen přečte.
- Tabulace — výpočet *zdola nahoru*: začneme základními případy a stavy vyplňujeme v pořadí, aby všechny potřebné menší stavy byly hotové dříve.

U Fibonacciho čísel obě varianty počítají $n + 1$ stavů s konstantní prací na stav, tedy $O(n)$.

Na Fibonacciho číslech je dobře vidět, co je překrývání podproblémů, ale jako ukázka návrhu nestačí: stav je už napsán v zadání.

# Memoizace, nebo tabulace?

| | **memoizace (shora)** | **tabulace (zdola)** |
|---|---|---|
| zápis | blízko rekurenci, snadno se ověřuje | smyčky, nutné znát pořadí |
| počítá | jen dosažitelné stavy | *všechny* stavy |
| pořadí | samo se najde | musíme je dokázat |
| režie | zásobník rekurze | nejmenší možná |
| paměť | celé úložiště | lze zúžit na pár řádků |
| riziko | přetečení zásobníku | čtení prázdné buňky |

## Doporučený postup

Napište nejdřív memoizaci; je to doslovný přepis rekurence, takže se v ní špatně chybuje. Na tabulaci přepisujte, až když potřebujete rychlost nebo úsporu paměti.

# Vážené intervalové plánování

V přednášce 2 jsme maximalizovali *počet* kompatibilních intervalů a bezpečné bylo přijmout ten končící nejdříve. Teď má interval $j$ nezápornou hodnotu $v_j$ a maximalizujeme *součet hodnot*.

## Definice 3 (Vážené intervalové plánování)

Intervaly $[s_j, f_j)$ s hodnotami $v_j \geq 0$. Přípustné řešení je množina po dvou kompatibilních intervalů; maximalizujeme součet jejich hodnot.

# Vážené intervalové plánování

V přednášce 2 jsme maximalizovali *počet* kompatibilních intervalů a bezpečné bylo přijmout ten končící nejdříve. Teď má interval $j$ nezápornou hodnotu $v_j$ a maximalizujeme *součet hodnot*.

## Definice 3 (Vážené intervalové plánování)

Intervaly $[s_j, f_j)$ s hodnotami $v_j \geq 0$. Přípustné řešení je množina po dvou kompatibilních intervalů; maximalizujeme součet jejich hodnot.

Hladové pravidlo nejdřívějšího konce už nefunguje: $a = [0, 2)$ s hodnotou 1 končí dříve než $b = [0, 3)$ s hodnotou 100, ale vybrat lze jen jeden — hladově dostaneme 1 místo 100.

Selhávají i další pravidla: *nejvyšší hodnota* může zablokovat mnoho výhodných intervalů, *nejvyšší poměr hodnoty k délce* zase ignoruje polohu.

# Uspořádání a předchůdce

Intervaly očíslujeme tak, aby $f_1 \leq f_2 \leq \cdots \leq f_n$. Pro interval $j$ definujeme

$$p(j) = \max\{i < j \mid f_i \leq s_j\},$$

tj. index *nejpozdějšího* intervalu, který s $j$ nekoliduje ($p(j) = 0$, neexistuje-li takový).

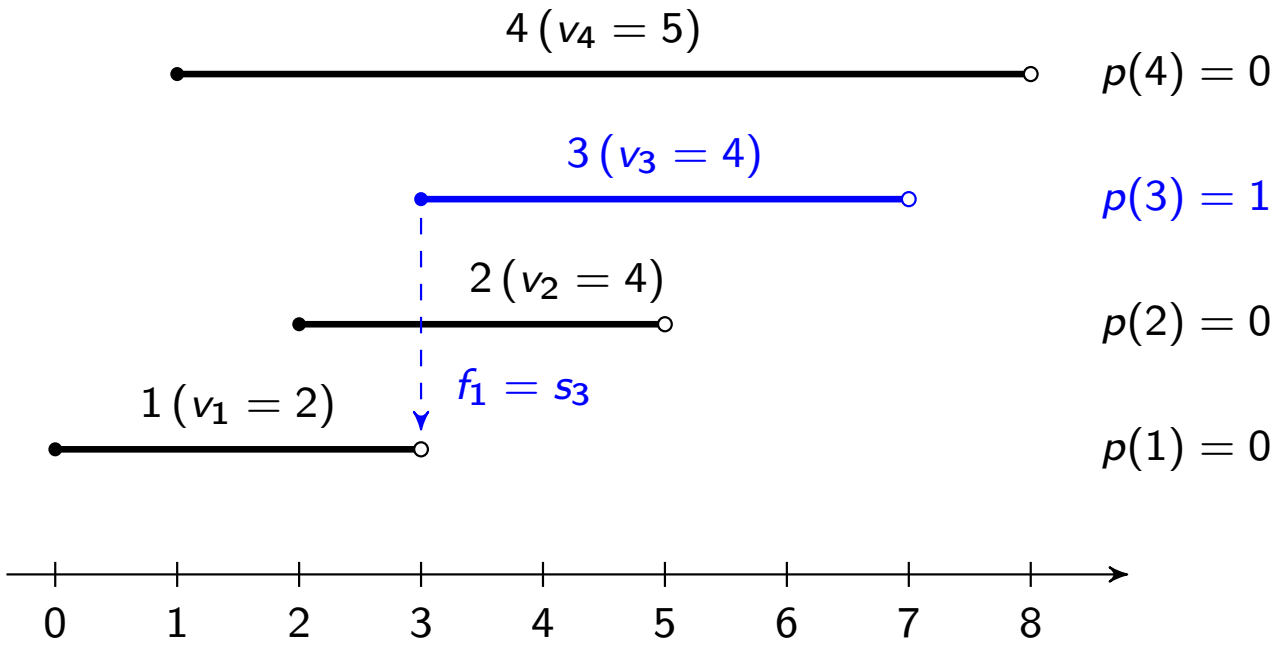
Dotyk je povolen: končí-li $i$ přesně tam, kde $j$ začíná ($f_i = s_j$), smějí být v řešení oba.

## Příklad 4 (Malá instance)

| $j$ | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| $[s_j, f_j)$ | $[0, 3)$ | $[2, 5)$ | $[3, 7)$ | $[1, 8)$ |
| $v_j$ | 2 | 4 | 4 | 5 |
| $p(j)$ | 0 | 0 | 1 | 0 |

Interval 3 začíná přesně tam, kde 1 končí — proto $p(3) = 1$. Dlouhý interval 4 zasahuje do všech ostatních, tedy $p(4) = 0$.

# Instance na časové ose




Plný kroužek = uzavřený začátek, prázdný = otevřený konec. Interval 3 se intervalu 1 jen *dotýká*, proto $p(3) = 1$.

# Volba stavu

## Definice 5 (Význam stavu)

Pro $j = 0, 1, \ldots, n$ označme OPT($j$) největší hodnotu přípustného řešení používajícího *pouze intervaly* $1, \ldots, j$. Platí OPT(0) $= 0$.

Pro tento stav platí:

- stav neříká, zda interval $j$ vybereme;
- shrnuje nejlepší řešení celého prefixu.

Díky řazení podle konců tvoří prefix uzavřený menší problém: po rozhodnutí o intervalu $j$ se vrátíme k některému *dřívějšímu prefixu*.

# Dvě možnosti posledního intervalu

- Interval $j$ *nepoužijeme*: hodnota je nejvýše $\mathrm{OPT}(j-1)$.
- Interval $j$ *použijeme*: pak nesmíme použít žádný z $p(j)+1, \ldots, j-1$, a k $v_j$ přidáme optimum prefixu $1, \ldots, p(j)$.

## Bellmanova rovnice

$$\mathrm{OPT}(j) = \max\{\ \mathrm{OPT}(j-1),\ \ v_j + \mathrm{OPT}(p(j))\ \}$$

## Věta 6

*Tato rekurence se základním případem* $\mathrm{OPT}(0) = 0$ *určuje optimální hodnotu váženého intervalového plánování.*

# Důkaz a optimální podstruktura

## Důkaz.

Indukcí podle $j$; základ indukce je zřejmý. Nechť $O_j$ je optimální řešení pro prvních $j$ intervalů.

Je-li $j \notin O_j$, používá $O_j$ jen prvních $j-1$ intervalů, tedy hodnotu nejvýše $\mathrm{OPT}(j-1)$. Je-li $j \in O_j$, patří všechny ostatní zvolené intervaly do prefixu $p(j)$. Optimum tedy nepřesahuje maximum v rekurenci.

Oba kandidáti jsou zároveň *dosažitelní*, takže maximum je přesně optimální hodnota.

## Optimální podstruktura

Obsahuje-li optimální řešení interval $j$, musí být jeho zbývající část optimální pro prefix $p(j)$ — jinak bychom ji nahradili lepší a zlepšili celek.

# Memoizovaná varianta

```
1 def Solve(j):
2     if j = 0 then return 0
3     if M[j] je známé then return M[j]
4     M[j] ← max{Solve(j − 1), v_j + Solve(p(j))}
5     return M[j]
```

Pole $M$ je *úložiště stavů*: buňka $M[j]$ drží OPT($j$). Krok s maximem je jen přepisem Bellmanovy rekurence.

Každý stav se plně vyhodnotí nejvýše jednou, práce mimo rekurzi je konstantní — při známém poli $p$ tedy $O(n)$ času i paměti.

Memoizace počítá jen stavy dosažitelné z dotazu `Solve(`$n$`)`. Zde budou potřeba téměř všechny, ale u řídkých stavových prostorů může být úspora podstatná.

# Tabulace zdola nahoru

Každý přechod z $j$ odkazuje jen na *menší* indexy, takže lze vyplňovat zleva doprava:

$$M[0] = 0, \qquad M[j] = \max\{M[j-1],\ v_j + M[p(j)]\}.$$

| $j$ | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| $p(j)$ | — | 0 | 0 | 1 | 0 |
| $v_j$ | — | 2 | 4 | 4 | 5 |
| $M[j]$ | 0 | 2 | 4 | 6 | 6 |

- $M[3] = \max\{M[2],\ v_3 + M[p(3)]\} = \max\{4,\ 4+2\} = 6$ — vyplatí se spojit interval 3 s intervalem 1.
- $M[4]$: porovnáme $M[3] = 6$ s $v_4 + M[0] = 5$ — dlouhý interval 4 *nepřijmeme*.

Optimální hodnota je $M[4] = 6$.

# Celý algoritmus a jeho složitost

1. Seřaď intervaly podle času konce.
2. Pro každý interval vypočítej index $p(j)$.
3. Vyplň pole $M$ zleva doprava podle rekurence.
4. Zpětným průchodem rekonstruuj zvolené intervaly.

- Časy konců jsou seřazené $\Rightarrow$ $p(j)$ najdeme binárním vyhledáváním v $O(\log n)$.
- Předzpracování všech předchůdců: $O(n \log n)$, stejně jako počáteční řazení.
- Vyplnění tabulky: lineární.

Celkem $O(n \log n)$ času a $O(n)$ paměti.

# Větší příklad: pět intervalů

| $j$ | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| $[s_j, f_j)$ | $[0,2)$ | $[1,4)$ | $[3,5)$ | $[4,7)$ | $[6,8)$ |
| $v_j$ | 4 | 7 | 5 | 8 | 4 |

# Větší příklad: pět intervalů

| $j$ | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| $[s_j, f_j)$ | $[0,2)$ | $[1,4)$ | $[3,5)$ | $[4,7)$ | $[6,8)$ |
| $v_j$ | 4 | 7 | 5 | 8 | 4 |

Nejprve předchůdci — poslední dřívější interval s koncem nejvýše $s_j$:

$$p(1) = 0, \quad p(2) = 0, \quad p(3) = 1, \quad p(4) = 2, \quad p(5) = 3$$

Interval 4 začíná v čase 4; interval 2 končí *přesně* ve 4, je tedy kompatibilní, kdežto interval 3 končí až v 5.

# Větší příklad: pět intervalů

| $j$ | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| $[s_j, f_j)$ | $[0,2)$ | $[1,4)$ | $[3,5)$ | $[4,7)$ | $[6,8)$ |
| $v_j$ | 4 | 7 | 5 | 8 | 4 |

Nejprve předchůdci — poslední dřívější interval s koncem nejvýše $s_j$:

$$p(1) = 0, \quad p(2) = 0, \quad p(3) = 1, \quad p(4) = 2, \quad p(5) = 3$$

Interval 4 začíná v čase 4; interval 2 končí *přesně* ve 4, je tedy kompatibilní, kdežto interval 3 končí až v 5.

| $j$ | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| $p(j)$ | — | 0 | 0 | 1 | 2 | 3 |
| $v_j$ | — | 4 | 7 | 5 | 8 | 4 |
| $M[j]$ | 0 | 4 | 7 | 9 | 15 | 15 |

$M[4] = \max\{M[3],\, 8 + M[2]\} = \max\{9, 15\} = 15;$
$M[5] = \max\{15,\, 4 + M[3]\} = \max\{15, 13\} = 15.$

# Proč tu hladovost selhává a DP ne

| **nevážená úloha (přednáška 2)** | **vážená úloha (dnes)** |
|---|---|
| každý interval „stojí“ totéž | intervaly mají různou hodnotu |
| výměnný argument: první interval optima lze nahradit nejdříve končícím | výměna zachová *počet*, ale může zničit hodnotu |
| bezpečná lokální volba existuje | neexistuje: musíme porovnat obě možnosti |
| $O(n \log n)$, žádná tabulka | $O(n \log n)$, tabulka $O(n)$ |

Selže-li tedy výměnný argument kvůli hodnotě, je namístě zkusit dynamické programování.

# Rekonstrukce řešení

Tabulka $M$ obsahuje optimální *hodnotu*, ne samotnou množinu intervalů. Rozhodnutí ale přečteme pozpátku.

Ve stavu $j$ porovnáme tytéž dva kandidáty jako v rekurenci:

- $M[j-1] \geq v_j + M[p(j)] \Rightarrow$ interval $j$ nevybereme, přejdeme do $j-1$;
- jinak interval $j$ *vybereme* a přejdeme přímo do $p(j)$.

Proces končí ve stavu 0; vybrané intervaly vznikají v opačném pořadí, takže je obrátíme.

**Věta 7 (Korektnost rekonstrukce)**

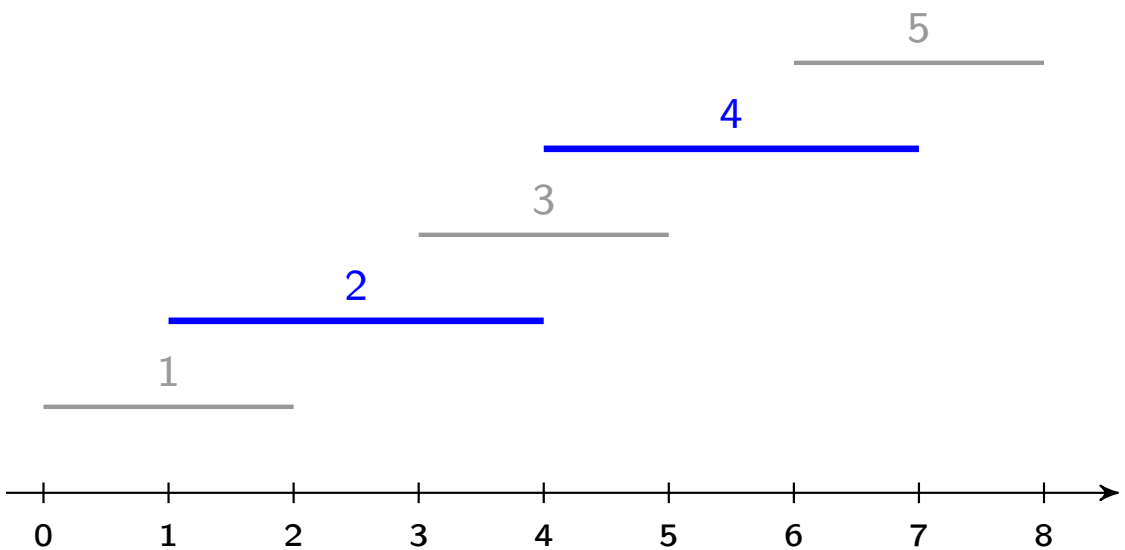
*Zpětný průchod vrátí kompatibilní množinu intervalů s hodnotou $M[n]$.*

*Důkaz.* V každém stavu volíme větev dosahující uložené hodnoty. Při přijetí $j$ přejdeme na $p(j)$, takže všechny další rekonstruované intervaly končí nejpozději v $s_j$ — kompatibilita se zachová.

# Rekonstrukce: které intervaly to byly

| stav | porovnání | rozhodnutí |
|---|---|---|
| $j = 5$ | $M[4] = 15 \geq v_5 + M[3] = 4 + 9 = 13$ | vynech 5, jdi na 4 |
| $j = 4$ | $M[3] = 9 < v_4 + M[2] = 8 + 7 = 15$ | vezmi 4, jdi na $p(4) = 2$ |
| $j = 2$ | $M[1] = 4 < v_2 + M[0] = 7 + 0 = 7$ | vezmi 2, jdi na $p(2) = 0$ |
| $j = 0$ | — | konec |

Řešení je $\{2, 4\}$ s hodnotou $7 + 8 = 15 = M[5]$. ✓




Modré intervaly na sebe přesně navazují v čase 4.

# Hodnota není řešení

## Častá chyba

Skončit po vyplnění tabulky. Zadání se však může ptát na konkrétní rozvrh, cestu nebo množinu předmětů.

Rekonstrukci plánujeme *současně s přechodem*:

- buď ukládáme zvoleného předchůdce,
- nebo rozhodnutí znovu odvodíme z hotových hodnot.

První varianta stojí paměť navíc, druhá čas navíc — a vyžaduje, aby šlo rozhodnutí z hodnot jednoznačně přečíst.

# Nejkratší cesty v DAG

DP není omezeno na pole — přirozeným stavovým prostorem může být *graf*.

Orientovaný acyklický graf $G = (V, E)$ s vahami $w$ a zdrojem $s$. Protože graf nemá orientovanou kružnici, lze vrcholy uspořádat topologicky: každá hrana vede z dřívějšího vrcholu do pozdějšího.

To je právě pořadí, v němž můžeme vyhodnocovat stavy.

## Definice 8 (Stav)

$d[v]$ je délka nejkratší orientované cesty ze $s$ do $v$; $d[s] = 0$ a $d[v] = \infty$ pro nedosažitelné.

Každá neprázdná cesta do $v$ končí nějakou hranou $(u, v)$, tedy

$$d[v] = \min_{(u,v)\in E} \{d[u] + w(u, v)\}.$$

# Algoritmus a jeho korektnost

1. Spočítej topologické pořadí vrcholů.
2. $d[s] = 0$, ostatní $\infty$.
3. Procházej vrcholy v topologickém pořadí a relaxuj jejich odchozí hrany.
4. Při zlepšení $d[v]$ ulož předchůdce $\pi[v] = u$.

## Důkaz.

Indukcí podle topologického pořadí. Zdroj má hodnotu 0; nedosažitelný vrchol nemá vstupní hranu z dosažitelného, takže zůstane $\infty$.

Pro dosažitelný $v \neq s$: každá cesta do $v$ končí hranou $(u, v)$ a vrchol $u$ leží *před* $v$, takže indukční předpoklad už zná nejlepší cestu do $u$. Relaxace všech vstupních hran tedy porovná všechny možné poslední kroky.

Čas $O(|V| + |E|)$ včetně topologického řazení.

# Proč tu záporné váhy nevadí

| **Dijkstra (přednáška 3)** | **DAG** |
|---|---|
| potřebuje nezápornost, aby uzavřený vrchol už nešlo zlepšit | k vrcholu se nevracíme, protože topologické pořadí zaručuje zpracování všech předchůdců předem |
| libovolný orientovaný graf | bez orientovaných kružnic |

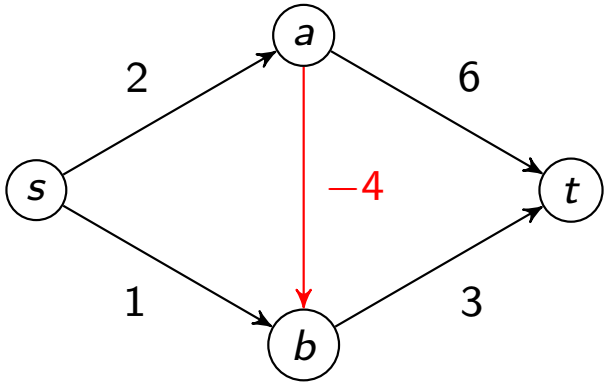
Záporná kružnice nemůže existovat — graf nemá *žádnou* orientovanou kružnici.

## DAG jako tabulka závislostí

Pole indexované $0, \ldots, n$ je jen zvláštní případ grafu stavů; hrana $u \to v$ znamená „$v$ závisí na $u$“. Tabulaci lze provést právě tehdy, když máme pořadí, v němž jsou všechny závislosti vyhodnoceny dříve.

# Nejkratší cesty v DAG: celý běh

Topologické pořadí $s, a, b, t$; hrany $(s, a)$:2, $(s, b)$:1, $(a, b)$:$-4$, $(a, t)$:6, $(b, t)$:3.




| krok | zpracován, relaxace | $d[s]$ | $d[a]$ | $d[b]$ | $d[t]$ |
|---|---|---|---|---|---|
| 0 | inicializace | 0 | $\infty$ | $\infty$ | $\infty$ |
| 1 | $s$; $(s, a), (s, b)$ | 0 | 2 | 1 | $\infty$ |
| 2 | $a$; $(a, b), (a, t)$ | 0 | 2 | $-2$ | 8 |
| 3 | $b$; $(b, t)$ | 0 | 2 | $-2$ | 1 |
| 4 | $t$; — | 0 | 2 | $-2$ | 1 |

Ve druhém kroku klesne $d[b]$ z 1 na $-2$ — záporná hrana nevadí, protože $a$ je v topologickém pořadí *před* $b$. Dijkstra by $b$ uzavřel už s hodnotou 1 (dříve než $a$) a vrátil by $d[t] = 4$.

# Táž tabulka, jiný cíl: nejdelší cesta

V obecném grafu je nejdelší cesta NP-těžká úloha. V DAG je triviální — stačí obrátit optimalizaci.

Pro každý vrchol $v$ nechť $L[v]$ je délka nejdelší cesty, která ve $v$ *končí* (délka = počet hran).

Základní případy: $L[v] = 0$ pro vrcholy bez vstupních hran. V topologickém pořadí

$$L[v] = \max\big(\{0\} \cup \{L[u] + 1 : (u, v) \in E\}\big).$$

Uložením předchůdce dosahujícího maxima cestu zrekonstruujeme; čas $O(|V| + |E|)$.

## Proč to v obecném grafu nejde

Bez acykličnosti nemá stav „nejdelší cesta končící ve $v$“ *optimální podstrukturu*: prodloužení by mohlo vrchol zopakovat a cesta by přestala být cestou.

# Další použití tohoto DP

- **Plánování projektů (CPM).** Úlohy s předchůdci tvoří DAG; nejdelší cesta je *kritická cesta* — nejkratší možná doba celého projektu.
- **Sazba odstavce (TEX).** Vrcholy = místa zlomu řádku, hrany = možné řádky s cenou podle „ošklivosti“. Nejkratší cesta v DAG dá optimální zalomení celého odstavce.
- **Tabulky v tabulkovém procesoru.** Buňky tvoří DAG závislostí; přepočet jde v topologickém pořadí. Cyklus = chyba „kruhový odkaz“.
- **Sestavovací systémy** (`make`, CI pipeline) — totéž, jen s cílem „spusť každý krok až po jeho závislostech“.

Ve všech případech je návrhová práce stejná: najít stavy, ověřit acykličnost, určit pořadí.

# Jak navrhovat stav

Stav musí obsahovat právě tolik informace, aby budoucí rozhodnutí nezáviselo na podrobnostech minulosti, které jsme zahodili.

- Příliš chudý stav vede k chybné rekurenci.
- Příliš bohatý stav vytváří zbytečně velkou tabulku.

## Návrhový postup

1. Formuluj, co přesně hodnota stavu optimalizuje.
2. Zvol jedno rozhodnutí, které rozděluje všechny možnosti na *úplné* případy.
3. Pro každý případ urči menší stav shrnující zbytek problému.
4. Dokaž optimální podstrukturu.
5. Nakresli závislosti, stanov základní případy a zvol pořadí výpočtu.
6. Přidej informaci nutnou pro rekonstrukci.

# Typické chyby

- **Nejasný význam.** Symbol $M[i]$ bez věty „nejlepší řešení *čeho*?“ nelze spolehlivě použít ani dokázat.
- **Neúplné případy.** Přechod musí pokrýt *každé* přípustné řešení, včetně možnosti prvek nepoužít.
- **Cyklické závislosti.** Tabulace potřebuje acyklické pořadí stavů.
- **Pouhá rekurze.** Bez memoizace mohou překrývající se podproblémy způsobit exponenciální čas.
- **Chybějící rekonstrukce.** Optimální hodnota sama nemusí být požadovaným výstupem.
- **Podceněný přechod.** Složitost není jen počet buněk — násobíme jej počtem možností v jedné buňce.

# Nejasný stav: jak to vypadá v praxi

Student napíše pro vážené intervaly:

„$M[j]$ *je nejlepší řešení, když jsem u intervalu* $j$.“

Co to znamená? Máme aspoň tři různá čtení:

1. nejlepší řešení *používající pouze* intervaly $1, \ldots, j$ (naše definice — funguje);
2. nejlepší řešení, které interval $j$ *obsahuje* (taky funguje, ale přechod vypadá jinak!);
3. nejlepší řešení „někde kolem“ $j$ (není definováno).

# Nejasný stav: jak to vypadá v praxi

Student napíše pro vážené intervaly:

„$M[j]$ *je nejlepší řešení, když jsem u intervalu* $j$.“

Co to znamená? Máme aspoň tři různá čtení:

1. nejlepší řešení *používající pouze* intervaly $1, \ldots, j$ (naše definice — funguje);
2. nejlepší řešení, které interval $j$ *obsahuje* (taky funguje, ale přechod vypadá jinak!);
3. nejlepší řešení „někde kolem“ $j$ (není definováno).

Při čtení (2) by přechod zněl

$$M'[j] = v_j + \max\big(\{0\} \cup \{\, M'[i] : i \le p(j) \,\}\big),$$

a odpovědí by bylo $\max_j M'[j]$, ne $M'[n]$. Přímo počítané maximum stojí $O(n)$ na stav, celkem tedy $O(n^2)$.

Obě definice jsou samy o sobě správné; míchat je ale znamená chybný algoritmus.

Než tedy napíšete rekurenci, napište českou větu o tom, co stav znamená.

# Další použití Bellmanovy rovnice

Tvar „hodnota stavu = okamžitý zisk + hodnota následujícího stavu“ se objevuje daleko za hranicemi našeho kurzu:

- **Optimální řízení.** Spojitá verze rekurence je Hamiltonova–Jacobiho–*Bellmanova* rovnice.
- **Zpětnovazební učení.** Bellmanova rovnice pro hodnotovou funkci je základem algoritmů *value iteration* a *Q-learning*; stav je stav prostředí, přechod je akce.
- **Skryté Markovovy modely.** Viterbiho algoritmus je DP nad mřížkou (čas × skrytý stav) — rozpoznávání řeči, dekódování v telekomunikacích.
- **Ekonomie.** Modely spotřeby a úspor se formulují jako Bellmanova rovnice nad stavem „majetek“.

Ve všech případech je návrhová práce táž, jakou jsme dnes dělali: určit, co je stav, jaký je přechod a v jakém pořadí počítat.

# Shrnutí

Dynamické programování sdílí práci mezi překrývajícími se podproblémy. Rozhoduje dobře zvolený stav a dokázaná přechodová rovnice, nikoli samotná tabulka.

- Memoizace zachovává rekurzivní pohled a počítá stavy na vyžádání.
- Tabulace vyhodnocuje stavy v pořadí respektujícím závislosti.
- Vážené intervaly ukazují rozhodnutí „vezmi, nebo vynech“.
- Nejkratší cesty v DAG ukazují, že tabulka může mít podobu grafu.
- Rekonstrukce sleduje zpět větve, které dosáhly uložených hodnot.

Příště přidáme stavu další rozměr; ukážeme to na batohu a porovnávání posloupností.

# Literatura


R. Bellman.
*Dynamic Programming*.
Princeton University Press (1957).


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


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


S. Dasgupta, C. H. Papadimitriou, U. V. Vazirani.
*Algorithms*. McGraw–Hill (2006).