

KMI/ALGO3 — Algoritmy 3

3. Hladové algoritmy na grafech

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

Táž otázka na grafech

Minule jsme vybírali intervaly, řadili úlohy a stavěli kódový strom; teď budeme pracovat s vrcholy a hranami. Ptáme se ale pořád na totéž: kterou lokální volbu můžeme provést nevratně, aniž ztratíme optimální řešení?

- u **minimální kostry** bude bezpečnou volbou nejlehčí hrana přes vhodný řez;
- u **nejkratších cest** budeme definitivně uzavírat vrchol s nejmenší dosud známou vzdáleností.

V obou případech rozhodujeme na *hranici* mezi částí grafu, kterou už považujeme za vyřešenou, a částí, která vyřešena ještě není.

Nové budou především důkazové techniky: řez, bezpečná hrana, relaxace, množina uzavřených vrcholů.

Dva problémy pro dnešní přednášku

Graf $G = (V, E)$, každé hraně váha $w(e)$ (cena kabelu, délka silnice, doba průchodu. . .).

- **Minimální kostra:** levně propojit *všechny* vrcholy.
- **Nejkratší cesty:** dostat se z jednoho zdroje co nejlevněji do jednotlivých vrcholů.

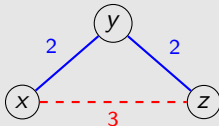
Dva problémy pro dnešní přednášku

Graf $G = (V, E)$, každé hraně váha $w(e)$ (cena kabelu, délka silnice, doba průchodu. . .).

- **Minimální kostra:** levně propojit *všechny* vrcholy.
- **Nejkratší cesty:** dostat se z jednoho zdroje co nejlevněji do jednotlivých vrcholů.

Příklad 1 (Trojúhelník stačí)

$w(xy) = w(yz) = 2$, $w(xz) = 3$.



Minimální kostra vezme obě hrany s vahou 2; cesta z x do z po kostře měří 4. Nejkratší cesta ale vede přímo po xz a měří 3, tato hrana však v kostře *chybí*.

Minimální kostra

Nechť $G = (V, E)$ je souvislý neorientovaný graf. **Kostra** je souvislý podgraf obsahující všechny vrcholy a neobsahující kružnici; má právě $|V| - 1$ hran.

Definice 2 (Minimální kostra)

Váha množiny hran $F \subseteq E$ je $w(F) = \sum_{e \in F} w(e)$. **Minimální kostra** grafu G je kostra T s nejmenší možnou vahou $w(T)$.

Minimální kostra

Nechť $G = (V, E)$ je souvislý neorientovaný graf. **Kostra** je souvislý podgraf obsahující všechny vrcholy a neobsahující kružnici; má právě $|V| - 1$ hran.

Definice 2 (Minimální kostra)

Váha množiny hran $F \subseteq E$ je $w(F) = \sum_{e \in F} w(e)$. **Minimální kostra** grafu G je kostra T s nejmenší možnou vahou $w(T)$.

Proč nestačí vzít $|V| - 1$ nejlevnějších hran?

Obce a, b, c, d ; hrany $ab:1$, $bc:2$, $ac:3$ a jediná další $ad:5$.

Tři nejlevnější hrany ab, bc, ac tvoří **trojúhelník** a obec d nechávají bez připojení.

Musíme tedy vybírat levně, ale *bez kružnic a s ohledem na souvislost*.

Řez a bezpečná hrana

Definice 3 (Řez grafu)

Řezem grafu $G = (V, E)$ je rozdělení $(S, V - S)$ vrcholů na dvě neprázdné části. Hrana $\{u, v\}$ řez *překračuje*, leží-li jeden konec v S a druhý ve $V - S$.

Každá kostra musí každý řez překročit alespoň jednou — jinak by mezi částmi nevedla cesta. Nabízí se hladová myšlenka: *přes řez vezměme nejlehčí možnou hranu*.

Definice 4 (Bezpečná hrana)

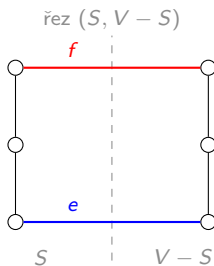
Nechť $F \subseteq E$ je podmnožinou nějaké minimální kostry. Hrana $e \notin F$ je **bezpečná** pro F , jestliže také $F \cup \{e\}$ je podmnožinou nějaké minimální kostry.

Dokud přidáváme jen bezpečné hrany, nemůžeme o optimum přijít.

Vlastnost řezu

Lemma 5 (Vlastnost řezu)

Nechť $F \subseteq E$ je podmnožinou nějaké minimální kostry a nechť $(S, V - S)$ je řez, který žádná hrana z F nepřekračuje. Je-li e nejlehčí hrana překračující tento řez, existuje minimální kostra obsahující $F \cup \{e\}$ — hrana e je pro F bezpečná.



Důkaz vlastnosti řezu

Důkaz.

Nechť T je minimální kostra obsahující F . Je-li $e \in T$, jsme hotovi. Nechť tedy $e \notin T$.

Přidáním e do T vznikne právě jedna kružnice. Protože e přechází z S do $V - S$, musí se kružnice někudy *vrátit* — obsahuje tedy další hranu f překračující tentýž řez. Žádná hrana z F řez nepřekračuje, takže $f \notin F$.

Položme $T' = T - \{f\} + \{e\}$. Je to opět kostra, obsahuje celé F i hranu e . Protože e je nejlehčí přes řez, je $w(e) \leq w(f)$, tedy

$$w(T') = w(T) - w(f) + w(e) \leq w(T).$$

T byla minimální, takže T' má stejnou (optimální) váhu.

Je to **výměnný argument** z minulé přednášky, použitý na kostry.

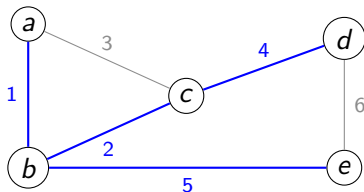
Kruskalův algoritmus

Dívá se na hrany od nejlehčí k nejtěžší a přijme tu, která nevytvoří kružnici.

```
1 def Kruskal( $G = (V, E), w$ ):  
2     seřaď hrany v neklesajícím pořadí podle váhy  
3      $F \leftarrow \emptyset$ ; foreach  $v \in V$  do  
4         MakeSet( $v$ )  
5     foreach hrana  $e = \{u, v\}$  v seřazeném pořadí do  
6         if Find( $u$ )  $\neq$  Find( $v$ ) then  
7              $F \leftarrow F \cup \{e\}$ ; Union( $u, v$ )  
8             if  $|F| = |V| - 1$  then return ( $V, F$ )  
9     return ( $V, F$ )
```

J. B. Kruskal (1956). Hladové kritérium je *globální*: vždy nejlehčí dosud nezpracovaná hrana v celém grafu.

Příklad běhu



1. $ab(1)$: různé komponenty — **přijmeme**.
2. $bc(2)$: $\{a, b\}$ vs. $\{c\}$ — **přijmeme**.
3. $ac(3)$: a, c už v téže komponentě — **odmítneme**.
4. $cd(4)$: připojí d — **přijmeme**.
5. $be(5)$: připojí e — **přijmeme**. Máme $|V| - 1 = 4$ hrany, končíme; $de(6)$ už nezkoumáme.

Váha kostry = $1 + 2 + 4 + 5 = 12$. Mimo kostru zůstala právě *nejtěžší hrana každé kružnice*, což odpovídá vlastnosti řezu.

Správnost Kruskalova algoritmu

Věta 6

Kruskalův algoritmus vrací minimální kostru souvislého neorientovaného grafu.

Invariant.

Existuje minimální kostra, která obsahuje všechny hrany v F .

Zachování invariantu.

Přidává-li algoritmus $e = \{u, v\}$, leží u, v v různých komponentách lesa (V, F) . Za S vezměme vrcholy komponenty obsahující u ; řez $(S, V - S)$ nepřekračuje žádná hrana z F .

Hrana e je nejlehčí přes tento řez: lehčí hrana přes řez by spojovala dvě různé komponenty, takže by ji algoritmus zpracoval dříve a přijal. Předpoklady lemmatu jsou splněny.

Správnost (dokončení)

Algoritmus nasbírá $|V| - 1$ hran.

Kdyby les (V, F) měl na konci méně hran, měl by alespoň dvě komponenty. Za S vezmeme vrcholy jedné z nich. Souvislý graf obsahuje hranu překračující $(S, V - S)$.

Její konce leží v různých komponentách výsledného lesa — a protože se komponenty během běhu jen slučují, ležely v různých komponentách i v okamžiku, kdy algoritmus hranu zkoumal. Tehdy by ji přijal. **Spor.**

Závěr.

Po přijetí $|V| - 1$ hran je (V, F) strom obsažený (podle invariantu) v nějaké minimální kostře. Oba stromy mají $|V| - 1$ hran, musí se tedy rovnat.

Union–find: jak rychle poznat kružnici

Procházet pokaždé celý les by bylo drahé.

Použijeme strukturu disjunktních množin (DSS):

- $\text{MakeSet}(v)$ — vytvoří $\{v\}$;
- $\text{Find}(v)$ — vrátí reprezentanta množiny s v ;
- $\text{Union}(u, v)$ — spojí obě množiny.

Hranu $\{u, v\}$ přijmeme právě když $\text{Find}(u) \neq \text{Find}(v)$.

S **kompresí cest** a **sjednocováním podle hodnoty** je amortizovaná cena operace téměř konstantní.

Složitost Kruskala

Dominuje třídění: $O(|E| \log |E|) = O(|E| \log |V|)$ času, $O(|V| + |E|)$ paměti.

Jarníkův–Primův algoritmus

Jiná organizace téže myšlenky: udržujeme *jediný rostoucí strom*.

Zvolíme počáteční vrchol r , $S = \{r\}$. Dokud $S \neq V$: vybereme nejlehčí hranu $\{u, v\}$ s $u \in S$, $v \notin S$, přidáme ji do kostry a vložíme v do S .

Správnost — týž invariant.

Všechny hrany budovaného stromu T mají oba konce v S , takže žádná nepřekračuje řez $(S, V - S)$. Vybraná hrana je přes tento řez nejlehčí, tedy podle vlastnosti řezu **bezpečná**. Jakmile $S = V$, má T všech $|V| - 1$ hran.

V. Jarník (1930); o čtvrt století později nezávisle R. C. Prim — odtud dvojité jméno.

Prim na témž grafu a jeho implementace

Z $r = a$: nejlehčí ven z $\{a\}$ je $ab(1)$; z $\{a, b\}$ pak $bc(2)$; z $\{a, b, c\}$ vedou ven $cd(4)$ a $be(5)$ (hrana ac má oba konce uvnitř) — vezmeme cd , nakonec be .

Stejná kostra, jiné pořadí přidávání.

Implementace prioritní frontou

Do fronty vložíme vrcholy mimo S ; klíčem vrcholu v je váha nejlehčí známé hrany spojující v s S .

V každém kroku odebereme vrchol s nejmenším klíčem a projdeme jeho sousedy: nabízí-li nová hrana levnější připojení, snížíme klíč (*decrease-key*).

Každý vrchol frontu opustí jednou, každá hrana způsobí nejvýše jedno snížení klíče. S binární haldou tedy $O((|V| + |E|) \log |V|) = O(|E| \log |V|)$.

Kruskal, nebo Jarník–Prim?

Oba používají **stejnou vlastnost řezu**, ale jinak organizují práci.

Kruskal

globálně nejlehčí bezpečná hrana

spojuje komponenty (les)

třídění + union–find

$O(|E| \log |V|)$

Jarník–Prim

rozšiřuje jednu komponentu

udržuje jediný strom

prioritní fronta s decrease-key

$O(|E| \log |V|)$

Volba implementace závisí také na reprezentaci grafu a jeho hustotě.

Kruskal krok za krokem: co dělá union-find

Náš graf, hrany v pořadí podle váhy. Sledujme komponenty:

hrana	váha	Find(u) vs. Find(v)	komponenty po kroku
—	—	—	$\{a\} \{b\} \{c\} \{d\} \{e\}$
ab	1	$a \neq b$ přijmi	$\{a, b\} \{c\} \{d\} \{e\}$
bc	2	$ab \neq c$ přijmi	$\{a, b, c\} \{d\} \{e\}$
ac	3	$abc = abc$ odmítni	$\{a, b, c\} \{d\} \{e\}$
cd	4	$abc \neq d$ přijmi	$\{a, b, c, d\} \{e\}$
be	5	$abcd \neq e$ přijmi	$\{a, b, c, d, e\}$

- Po pátém kroku má les $|V| - 1 = 4$ hrany — *končíme*.
- Hrana de s vahou 6 se už vůbec nezkoumá.
- Každá odmítnutá hrana je **nejtěžší hranou nějaké kružnice**.

Třetí algoritmus: Borůvka (1926)



O. Borůvka.

O jistém problému minimálním.

Práce moravské přírodovědecké společnosti 3(3), 37–58 (1926).

Otakar Borůvka (1899–1995), brněnský matematik, dostal roku 1926 velmi praktické zadání od Západomoravských elektráren:

jak elektrifikovat jižní Moravu s co nejmenší spotřebou vodičů?

Jeho odpověď je **historicky první publikovaný algoritmus pro minimální kostru** — třicet let před Kruskalem a o několik let dříve než Jarník.

Jarníkova práce z roku 1930 je přímou reakcí na Borůvkovu úlohu; Prim ji nezávisle znovuobjevil v roce 1957, Dijkstra roku 1959. Algoritmy pro minimální kostru tedy mají původ v Brně.

Borůvkův algoritmus

Dokud není graf souvislý:

1. *každá* komponenta lesa nezávisle vybere svou **nejlehčí odchozí hranu**;
2. všechny vybrané hrany přidáme *najednou*.

- Každá vybraná hrana je nejlehčí přes řez „komponenta vs. zbytek“ — podle vlastnosti řezu je tedy **bezpečná**.
- Každá komponenta se v jedné fázi spojí alespoň s jednou další, takže počet komponent **klesne alespoň na polovinu**.
- Fází je proto nejvýše $\log_2 |V|$, jedna fáze stojí $O(|E|)$:

$$O(|E| \log |V|).$$

Pozor na remízy: při shodných vahách mohou vybrané hrany uzavřít kružnici. Řeší se pevným pravidlem (např. lexikografické pořadí hran), které zaručí jednoznačnost „nejlehčí“ hrany.

Borůvkův algoritmus dnes

- Fáze je **snadno paralelizovatelná**: každá komponenta pracuje nezávisle. Kruskal potřebuje globálně seřazené hrany, Prim jedinou frontu.
- Proto se objevuje v algoritmech pro GPU, distribuované grafy a externí paměť.
- Borůvkovy fáze jsou součástí asymptoticky rychlých algoritmů pro minimální kostru, např. randomizovaného algoritmu s očekávaným lineárním časem (Karger, Klein, Tarjan 1995).

Společný základ tří algoritmů

Kruskal	nejlehčí hrana <i>globálně</i>
Jarník–Prim	nejlehčí hrana z <i>jedné</i> komponenty
Borůvka	nejlehčí hrana z <i>každé</i> komponenty naráz

Všechny tři jsou instancemi téhož lemmatu o řezu.

Kostra jako nástroj shlukování

Chceme rozdělit body do k skupin tak, aby si body *uvnitř* skupiny byly blízko.

Jednoduchý recept

Sestroj minimální kostru na úplném grafu vzdáleností a **odstraň $k - 1$ nejtěžších hran**. Vzniklých k komponent jsou hledané shluky.



Toto je přesně shlukování metodou nejbližšího souseda (*single-linkage clustering*). Záruka: nejmenší vzdálenost mezi body různých shluků (*rozestup*) je největší možná ze všech rozdělení do k shluků.

Nevýhoda: kostra je citlivá na „řetízky“ bodů — dva vzdálené shluky spojené řádkou stopou splynou.

Nejkratší cesty z jednoho zdroje

Orientovaný graf $G = (V, E)$ s vahami. Délka cesty = součet vah hran. Značíme $\delta(s, v)$ délku nejkratší cesty z s do v ($= \infty$, neexistuje-li).

Definice 7 (Problém nejkratších cest z jednoho zdroje)

Vstupem je ohodnocený orientovaný graf a zdroj s . Výstupem jsou pro každý vrchol v hodnota $\delta(s, v)$ a předchůdci, z nichž lze nejkratší cesty zrekonstruovat.

Budeme předpokládat

$$w(e) \geq 0 \quad \text{pro každou hranu } e.$$

Tento předpoklad je podstatný: dovolí prohlásit nejbližší dosud neuzavřený vrchol za *definitivně vyřešený*.

Uvolnění hran

Pro každý vrchol udržujeme odhad $d[v]$; na začátku $d[s] = 0$ a $d[v] = \infty$ jinak.

Definice 8 (Uvolnění)

Uvolnění hrany (u, v) : jestliže $d[v] > d[u] + w(u, v)$, položíme

$$d[v] \leftarrow d[u] + w(u, v)$$

a jako předchůdce v uložíme u .

Invariant odhadů

Každá konečná hodnota $d[v]$ je délkou nějakého sledu z s do v .

Indukcí podle počtu uvolnění: na začátku je $d[s] = 0$ délka prázdného sledu; uvolnění prodlouží sled do u o hranu (u, v) . Sled zkrátíme na cestu vypuštěním kružnic (při nezáporných vahách se délka jen zmenší). Proto vždy $d[v] \geq \delta(s, v)$.

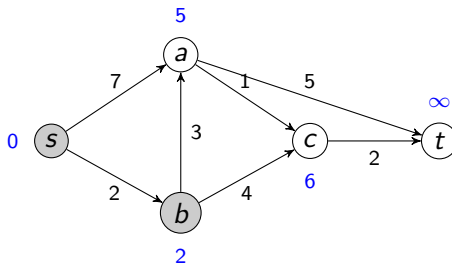
Dijkstrův algoritmus

Vrcholy dělí na **uzavřené** (hodnota definitivní) a neuzavřené.

```
1 def Dijkstra( $G, w, s$ ):  
2    $d[s] \leftarrow 0$ ;  $d[v] \leftarrow \infty$  pro  $v \neq s$ ;  $S \leftarrow \emptyset$   
3   while existuje dosažitelný vrchol mimo  $S$  do  
4      $u \leftarrow \arg \min \{ d[v] : v \notin S \}$  // hladová volba  
5      $S \leftarrow S \cup \{u\}$   
6     foreach hrana  $(u, v)$  s  $v \notin S$  do  
7        $\quad$  uvolni  $(u, v)$   
8   return  $d$ , předchůdci
```

E. W. Dijkstra (1959). Hladová volba je řádek s minimem: algoritmus tvrdí, že nejmenší dočasný odhad už pozdější cesta nezlepší.

Běh: okamžik po uzavření s a b



Vyplněné vrcholy jsou uzavřené, modré hodnoty jsou aktuální odhady. Relaxace hrany (b, a) už zlepšila $d[a]$ ze 7 na 5.

Celý běh v tabulce

kolo	uzavřen, uvolnění	$d[s]$	$d[a]$	$d[b]$	$d[c]$	$d[t]$
0	inicializace	0	∞	∞	∞	∞
1	s ; (s, a) , (s, b)	0	7	2	∞	∞
2	b ; (b, a) , (b, c)	0	5	2	6	∞
3	a ; (a, c) , (a, t)	0	5	2	6	10
4	c ; (c, t)	0	5	2	6	8
5	t ; —	0	5	2	6	8

- V kole 3 nabídla hrana (a, c) hodnotu $5 + 1 = 6$ — odhad $d[c] = 6$ to nezlepšilo.
- Nejkratší cesta do t měří 8; z předchůdců přečteme $s \rightarrow b \rightarrow c \rightarrow t$.
- **Vrchol a byl uzavřen s hodnotou 5**, ačkoli první nabídka přes (s, a) zněla 7: dokud vrchol není uzavřen, odhad se může zlepšovat.

Správnost Dijkstrova algoritmu

Věta 9

Má-li každá hrana nezápornou váhu, platí při uzavření každého vrcholu u rovnost $d[u] = \delta(s, u)$.

Důkaz.

Pro spor necht' u je *první* vrchol uzavřený s nesprávnou hodnotou. Podle invariantu $d[u] \geq \delta(s, u)$, takže $d[u] > \delta(s, u)$.

Vezmeme nejkratší cestu P ze s do u . V okamžiku výběru u byl na P alespoň jeden neuzavřený vrchol (aspoň u). Necht' y je *první* neuzavřený vrchol na P a x jeho předchůdce; x už byl uzavřen, a protože u je první chybný, platilo $d[x] = \delta(s, x)$.

Algoritmus tehdy uvolnil (x, y) , takže

$$d[y] \leq d[x] + w(x, y) = \delta(s, x) + w(x, y) = \delta(s, y),$$

a s invariantem dohromady $d[y] = \delta(s, y)$.

Správnost (dokončení)

Dva případy.

$y = u$: pak $d[u] = \delta(s, u)$ — uvolnění hrany (x, u) dala u správnou hodnotu.
Přímý spor s $d[u] > \delta(s, u)$.

$y \neq u$: pak y leží na nejkratší cestě do u ostře před u , a všechny váhy jsou **nezáporné**, tedy

$$\delta(s, y) \leq \delta(s, u) < d[u].$$

Když algoritmus vybíral u , existoval tedy neuzavřený vrchol y s *menším* odhadem — spor s hladovou volbou minima.

Správnost (dokončení)

Dva případy.

$y = u$: pak $d[u] = \delta(s, u)$ — uvolnění hrany (x, u) dala u správnou hodnotu.
Přímý spor s $d[u] > \delta(s, u)$.

$y \neq u$: pak y leží na nejkratší cestě do u ostře před u , a všechny váhy jsou **nezáporné**, tedy

$$\delta(s, y) \leq \delta(s, u) < d[u].$$

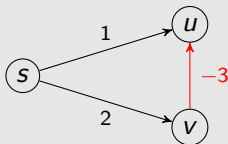
Když algoritmus vybíral u , existoval tedy neuzavřený vrchol y s *menším* odhadem — spor s hladovou volbou minima.

Poslední krok používá **nezápornost vah**, a to podstatně: kdyby po cestě mohla přijít záporná hrana, mohla by být vzdálenost k pozdějšímu vrcholu menší než k dřívějšímu.

Co se pokazí se zápornou hranou

Příklad 10 (Záporná hrana)

$(s, u) : 1, \quad (s, v) : 2, \quad (v, u) : -3.$



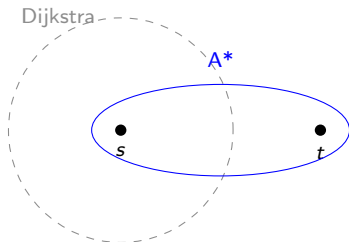
Dijkstra nejprve uzavře u s hodnotou 1. Skutečná nejkratší cesta vede přes v a má délku $2 - 3 = -1$.

Pro grafy se zápornými hranami potřebujeme jiný algoritmus (např. Bellmanův–Fordův).

Složitost Dijkstra. Prioritní fronta podle $d[v]$; každý vrchol jednou odebereme, každou hranu nejvýše jednou uvolňujeme — s binární haldou $O((|V| + |E|) \log |V|)$, paměť $O(|V| + |E|)$.

A*: informovaná hladová volba

Dijkstra zkoumá vrcholy podle vzdálenosti *od startu*. Při cestě z Olomouce do Prahy tak prozkoumává i směry vedoucí od cíle pryč.



Máme-li odhad zbývajících vzdáleností, můžeme hledání *zaměřit*. Nechť t je cíl a $h: V \rightarrow \mathbb{R}_{\geq 0}$ odhaduje cenu cesty z v do t . A* vybírá vrchol podle

$$f(v) = g(v) + h(v),$$

kde g je totéž průběžné d jako u Dijkstry.

Jaká heuristika je bezpečná

Definice 11 (Přípustná a konzistentní heuristika)

Heuristika je **přípustná**, jestliže nikdy nepřeceňuje skutečnou zbývající vzdálenost:

$$0 \leq h(v) \leq \delta(v, t).$$

Je **konzistentní**, jestliže $h(t) = 0$ a pro každou hranu (u, v) platí

$$h(u) \leq w(u, v) + h(v).$$

- Konzistence je **trojúhelníková nerovnost pro odhady**: odhad z u nesmí být větší než cena jednoho kroku plus odhad z v .
- Každá konzistentní heuristika je při nezáporných vahách také přípustná — stačí sečíst nerovnosti podél nejkratší cesty do cíle.

Algoritmus A* a přepočítání vah

Algoritmus A*

Jako Dijkstra, ale prioritou vrcholu v je $f(v) = g(v) + h(v)$. Odebereme vrchol s nejmenším f ; je-li to t , vrátíme cestu. Jinak jej definitivně uzavřeme a uvolňujeme výstupní hrany pomocí hodnot g .

Položme $w_h(u, v) = w(u, v) + h(v) - h(u)$. Konzistence říká právě to, že $w_h(u, v) \geq 0$.

Pro cestu $P = \langle s = v_0, \dots, v_k = v \rangle$ se členy vyruší teleskopicky:

$$w_h(P) = \sum_i (w(v_i, v_{i+1}) + h(v_{i+1}) - h(v_i)) = w(P) + h(v) - h(s).$$

Správnost A*

Pro pevný vrchol v se délky *všech* cest změní o stejnou konstantu — nejkratší cesty se tedy nezmění. Dijkstra na vahách w_h by řadil vrcholy podle $g(v) + h(v) - h(s)$, což je stejné pořadí jako podle $f(v) = g(v) + h(v)$.

Věta 12 (Správnost A*)

Je-li heuristika konzistentní, nalezne A při prvním odebrání cíle z prioritní fronty nejkratší cestu.*

Důkaz.

Váhy w_h jsou nezáporné a A* vybírá vrcholy ve stejném pořadí jako Dijkstra nad w_h . Podle věty o Dijkstrovi je nalezená cesta nejkratší v přepočítaných vahách; přepočet mění cenu každé cesty ze s do t o stejnou konstantu $h(t) - h(s) = -h(s)$.

Pro $h \equiv 0$ je A* *přesně* Dijkstra. Čím informativnější bezpečná heuristika, tím více vrcholů hledání přeskočí. (Hart, Nilsson, Raphael 1968.)

A* v praxi: odkud brát heuristiku

úloha	přípustná heuristika
plánování trasy v mapě	vzdušná vzdálenost do cíle (silnice nemůže být kratší než spojnice)
mřížka s překážkami	manhattanská vzdálenost (při pohybu po osách)
15-hlavalam	počet dlaždic mimo své místo; lépe součet manhattanských vzdáleností dlaždic
hledání cesty ve hře	vzdálenost v hrubší mapě vzniklé slučováním políček (sloučení vzdálenosti neztvrdí)

Jak heuristiku vyrobit

Odeberte z úlohy některá omezení a vyřešte tuto *volnější* verzi. Její optimum nikdy nepřecení skutečnou vzdálenost, takže je přípustnou heuristikou.

U 15-hlavalamu: dovolíme-li dlaždici skákat kamkoli, dostaneme „počet dlaždic mimo místo“; dovolíme-li jí procházet skrz ostatní, dostaneme součet manhattanských vzdáleností. *Druhé uvolnění je těsnější, a tedy informativnější.*

Když heuristiku „vylepšíme“ příliš

Lákavý nápad: vynásobme heuristiku konstantou $\lambda > 1$ a vybírejme podle

$$f(v) = g(v) + \lambda \cdot h(v).$$

Hledání se stočí k cíli mnohem ostřeji a prozkoumá méně vrcholů.

Cenou je ztráta záruky. Přenásobená heuristika už nemusí být přípustná, takže nalezená cesta nemusí být nejkratší.

Dá se ovšem dokázat, že není o moc horší: výsledek je nejvýše λ -krát delší než optimum. To je náš první **aproximační algoritmus** (když nepočítám cvičení) — vrátíme se k nim v přednášce 12.

Tři režimy jednoho algoritmu

$h \equiv 0$

Dijkstra — přesné, drahé

h konzistentní

A* — přesné, rychlejší

$\lambda \cdot h, \lambda > 1$

weighted A* — λ -aproximace, obvykle nejrychlejší

Společný důkazní vzor

Důkazy pro minimální kostru, Dijkstru a A* mají společnou strukturu:

1. Rozdělíme stav na **vyřešenou** a nevyřešenou část.
2. Na jejich **hranici** vybereme nejlevnější možnost.
3. Dokážeme, že existuje optimální řešení **slučitelné** s touto volbou.
4. Volbu prohlásíme za definitivní a hranici posuneme.

	hranice	proč je volba bezpečná
kostra	řez	výměna hrany na kružnici
Dijkstra	uzavřené vrcholy	nezápornost vah
A*	uzavřené vrcholy	přepoččet na nezáporné w_h

Literatura



V. Jarník. O jistém problému minimálním.
Práce mor. přírodovědecké společnosti 6 (1930), 57–63.



J. B. Kruskal. On the Shortest Spanning Subtree of a Graph...
Proceedings of the AMS 7 (1956), 48–50.



E. W. Dijkstra. A Note on Two Problems in Connexion with Graphs.
Numerische Mathematik 1 (1959), 269–271.



P. E. Hart, N. J. Nilsson, B. Raphael. A Formal Basis for the Heuristic
Determination of Minimum Cost Paths.
IEEE TSSC 4 (1968), 100–107.



J. Kleinberg, É. Tardos. *Algorithm Design*. Addison–Wesley (2006), kap. 4.