

1. Nejkratší cesty

Často potřebujeme hledat mezi dvěma vrcholy grafu cestu, která je v nějakém smyslu optimální – typicky nejkratší možná. Už víme, že prohledávání do šířky najde cestu s nejmenším počtem hran. Málodky jsou ale všechny hrany rovnocenné: silnice mezi městy mají různé délky, po různých vedeních lze přenášet elektřinu s různými ztrátami a podobně.

V této kapitole proto zavedeme grafy s ohodnocenými hranami a odvodíme několik algoritmů pro hledání cesty s nejmenším součtem ohodnocení hran.

1.1. Ohodnocené grafy a vzdálenost

Mějme graf $G = (V, E)$. Pro větší obecnost budeme předpokládat, že je orientovaný. Algoritmy se tím nezkomplikují a neorientované grafy můžeme vždy převést na orientované zdvojením hran.

Každé hraně $e \in E$ přiřadíme její *ohodnocení* neboli *délku*, což bude nějaké reálné číslo $\ell(e)$. Tím vznikne *ohodnocený orientovaný graf* nebo též *sít*. Zatím budeme uvažovat pouze nezáporná ohodnocení, později se zamyslíme nad tím, co by způsobila záporná.

Délku můžeme přirozeně rozšířit i na cesty, či obecněji sledy. Délka $\ell(S)$ sledu S bude prostě součet ohodnocení jeho hran.

Pro libovolné dva vrcholy u, v definujeme jejich *vzdálenost* $d(u, v)$ jako minimum z délek všech uv -cest (cest z u do v), případně $+\infty$, pokud žádná uv -cesta neexistuje. Toto minimum je vždy dobře definované, protože cest v grafu existuje pouze konečně mnoho. Pozor na to, že v orientovaném grafu se $d(u, v)$ a $d(v, u)$ mohou lišit.

Libovolné uv -cestě, jejíž délka je rovná vzdálenosti $d(u, v)$ budeme říkat *nejkratší cesta*. Nejkratších cest může být obecně více.

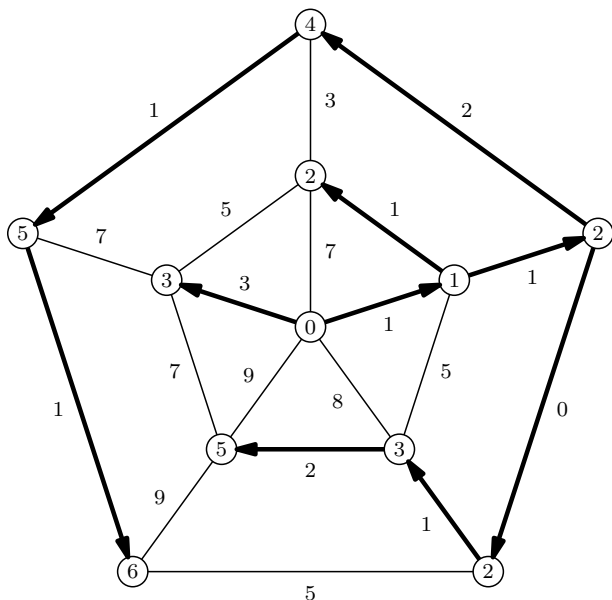
Vzdálenost bychom také mohli zavést pomocí nejkratšího sledu. Podobně jako u dosažitelnosti by se nic podstatného nezměnilo, protože sled lze zjednodušit na cestu:

Lemma (o zjednodušování sledů): Pro každý uv -sled existuje uv -cesta stejné nebo menší délky.

Důkaz: Pokud sled není cestou, znamená to, že se v něm opakují vrcholy. Uvažme tedy nějaký vrchol, který se zopakoval. Část sledu mezi prvním a posledním výskytem tohoto vrcholu můžeme „vystříhnout“, čímž získáme uv -sled stejné nebo menší délky. Jelikož ubyla alespoň jedna hrana, opakováním tohoto postupu časem získáme cestu. \square

Důsledek: Nejkratší sled existuje a má stejnou délku jako nejkratší cesta.

Důkaz: Nejkratší cesta je jedním ze sledů. Pokud by tvrzení neplatilo, musel by existovat nějaký kratší sled. Ten by ovšem šlo zjednodušit na ještě kratší cestu. \square



Obr. 1.1: Vzdálenosti od středového vrcholu (čísla uvnitř vrcholů) v ohodnoceném neorientovaném grafu. Zvýrazněné hrany tvoří strom nejkratších cest.

Důsledek: Pro vzdálenosti platí *trojúhelníková nerovnost*:

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

Důkaz: Pokud je $d(u, w)$ nebo $d(w, v)$ nekonečná, nerovnost triviálně platí. V opačném případě uvažme spojení nejkratší uw -cesty s nejkratší wv -cestou. To je nějaký uv -sled a ten nemůže být kratší než nejkratší uv -cesta. \square

Naše grafová vzdálenost se tedy chová tak, jak jsme u vzdáleností zvyklí.

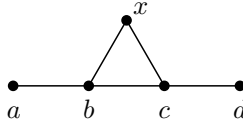
Při prohledávání grafů do šířky se osvědčilo popisovat nejkratší cesty z daného vrcholu v_0 do všech ostatních vrcholů pomocí stromu. Tento přístup funguje i v ohodnocených grafech a nyní ho zavedeme pořádně.

Definice: *Strom nejkratších cest* z vrcholu v_0 je podgraf zkoumaného grafu, který obsahuje všechny vrcholy dosažitelné z v_0 . Má tvar stromu orientovaného z vrcholu v_0 . Pro libovolný dosažitelný vrchol w platí, že cesta z v_0 do w ve stromu je jednou z nejkratších cest z v_0 do w ve zkoumaném grafu.

Stejně jako nejsou jednoznačně určeny nejkratší cesty, i stromů nejkratších cest může existovat více. Vždy ale existuje alespoň jeden (důkaz ponecháme jako cvičení 6).

Záporné hrany

Ještě si krátce rozmysleme, co by se stalo, kdybychom povolili hrany záporné délky. V následujícím grafu nastavíme všem hranám délku -1 . Nejkratší cesta z a do d vede přes b , x a c a má délku -4 . Sled $abxcbxcd$ je ovšem o 3 kratší, protože navíc obešel záporný cyklus $bxcb$. Pokud bychom záporný cyklus obkroužili vícekrát, získávali bychom kratší a kratší sledy. Tedy nejen že nejkratší sled neodpovídá nejkratší cestě, on ani neexistuje.



Podobně neplatí trojúhelníková nerovnost: $d(a, d) = -4$, ale $d(a, x) + d(x, d) = (-3) + (-3) = -6$.

Kdyby ovšem v grafu neexistoval záporný cyklus, sama existence záporných hran by problémy nepůsobila. Snadno ověříme, že lemma o zjednodušování sledů by nadále platilo, a tím pádem i trojúhelníková nerovnost. Grafy se zápornými hranami bez záporných cyklů jsou navíc užitečné (jak uvidíme ve cvičeních). Proto je budeme v některých částech této kapitoly připouštět, ale pokaždé na to výslovně upozorníme.

Cvičení

1. V neorientovaném grafu může roli záporného cyklu hrát dokonce i jediná hrana se záporným ohodnocením. Po ní totiž můžeme chodit střídavě tam a zpět. Naleznete příklad takového grafu.
2. Ukažte, jak pro libovolné n sestavit graf na nejvýše n vrcholech, v němž mezi nějakými dvěma vrcholy existuje $2^{\Omega(n)}$ nejkratších cest.
3. Připomeňte si definici metrického prostoru v matematické analýze. Kdy tvoří množina všech vrcholů spolu s funkcí $d(u, v)$ metrický prostor?
4. Navrhněte algoritmus pro výpočet vzdálenosti $d(u, v)$, který bude postupně počítat čísla $d_i(u, v)$ – nejmenší délka uv -sledu o nejvýše i hranách. Jaké časové složitosti jste dosáhli? Porovnejte ho s ostatními algoritmy z této kapitoly.
5. Dokažte, že pro nejkratší cesty platí takzvaná *prefixová vlastnost*: Nechť P je nějaká nejkratší cesta z v_0 do w a t nějaký vrchol na této cestě. Poté úsek (prefix) cesty P z v_0 do t je jednou z nejkratších cest z v_0 do t .
6. Dokažte, že pro libovolný ohodnocený graf a počáteční vrchol v_0 existuje strom nejkratších cest z v_0 . Může se hodit předchozí cvičení.

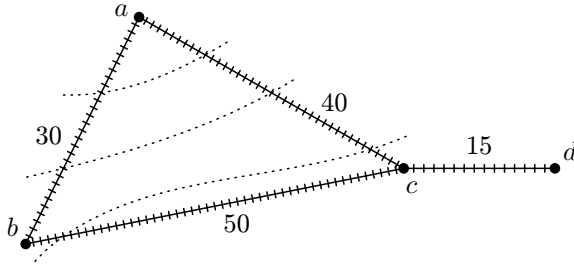
1.2. Dijkstrův algoritmus

Dnes asi nejpoužívanější algoritmus pro hledání nejkratších cest vymyslel v roce 1959 Edsger Dijkstra.⁽¹⁾ Funguje efektivně, kdykoliv jsou všechny hrany ohodnocené

⁽¹⁾ Je to holandské jméno, takže ho čteme „dajkstra“.

nezápornými čísly. Dovede nás k němu následující myšlenkový pokus.

Mějme orientovaný graf, jehož hrany jsou ohodnocené celými kladnými čísly. Každou hranu „podrozdělíme“ – nahradíme ji tolika jednotkovými hranami, kolik činila její délka. Tím vznikne neohodnocený graf, ve kterém můžeme nejkratší cestu nalézt prohledáváním do šířky. Jak podrozdělení funguje, vidíme na následujícím obrázku.



Obr. 1.2: Podrozdělený graf a poloha vlny v časech 10, 20 a 35

Tento algoritmus je funkční, ale sotva efektivní. Označíme-li L maximální délku hrany, podrozdělením vznikne $\mathcal{O}(Lm)$ nových vrcholů a hran.

Sledujme na obrázku, jak výpočet probíhá. Začneme ve vrcholu a . Vlna prvních 30 kroků prochází vnitřkem hran ab a ac . Pak dorazí do vrcholu b , načez se šíří dál zbytek hrany ac a novou hranou bc . Za dalších 10 kroků dorazí do c hranou ac , takže pokračuje hranou cd a nyní již zbytečnou hranou bc .

Většinu času tedy trávíme dlouhými úseky výpočtu, uvnitř kterých se prokoušáváme stále stejnými podrozdělenými hranami. Co kdybychom každý takový úsek zpracovali najednou?

Pro každý původní vrchol si pořídíme „budík“. Jakmile k vrcholu zamíří vlna, nastavíme jeho budík na čas, kdy do něj vlna má dorazit (pokud míří po více hranách najednou, zajímá nás, kdy dorazí poprvé).

Místo toho, abychom krok po kroku simulovali průchod vlny hranami, můžeme zjistit, kdy poprvé zazvoní budík. Tím přeskochíme nudnou část výpočtu a hned se dozvíme, že se vlna zarazila o vrchol. Podíváme se, jaké hrany z tohoto vrcholu vedou, a spočítáme si, kdy po nich vlna dorazí do dalších vrcholů. Podle toho případně přenastavíme další budíky. Opět počkáme, až zazvoní nějaký budík, a pokračujeme stejně.

Ještě by se mohlo stát, že vlna vstoupí do podrozdělené hrany z obou konců a obě části se „srazí“ uvnitř hrany. Tehdy můžeme nechat budíky zazvonit, jako by ke srážce nedošlo, a části zastavit až na konci hrany.

Zkusme tuto myšlenku zapsat v pseudokódu. Pro každý vrchol v si budeme pamatovat jeho *ohodnocení* $h(v)$, což bude buďto čas nastavený na příslušném budíku, nebo $+\infty$, pokud budík neběží. Podobně jako při prohledávání grafů do šířky

budeme rozlišovat tři druhy vrcholů: *nenalezené*, *otevřené* (to jsou ty, které mají nastavené budíky) a *uzavřené* (jejich budíky už zazvonily). Také si budeme pamatovat předchůdce vrcholů $P(v)$, abychom uměli rekonstruovat nejkratší cesty.

Algoritmus DIJKSTRA (nejkratší cesty v ohodnoceném grafu)

Vstup: Graf G a počáteční vrchol v_0

1. Pro všechny vrcholy v :
2. $stav(v) \leftarrow \text{nenalezený}$
3. $h(v) \leftarrow +\infty$
4. $P(v) \leftarrow \text{nedefinováno}$
5. $stav(v_0) \leftarrow \text{otevřený}$
6. $h(v_0) \leftarrow 0$
7. Dokud existují nějaké otevřené vrcholy:
8. Vybereme otevřený vrchol v , jehož $h(v)$ je nejmenší.
9. Pro všechny následníky w vrcholu v :
10. Pokud $h(w) > h(v) + \ell(v, w)$:
11. $h(w) \leftarrow h(v) + \ell(v, w)$
12. $stav(w) \leftarrow \text{otevřený}$
13. $P(w) \leftarrow v$
14. $stav(v) \leftarrow \text{uzavřený}$

Výstup: Pole vzdáleností h , pole předchůdců P

Z úvah o podrozdělování hran plyne, že Dijkstrův algoritmus dává správné výsledky, kdykoliv jsou délky hran celé kladné. Důkaz správnosti pro obecné délky najdete v následujícím oddílu, teď se zaměříme především na časovou složitost.

Věta: Dijkstrův algoritmus spočte v grafu s nezáporně ohodnocenými hranami vzdálenosti od vrcholu v_0 v čase $\mathcal{O}(n^2)$.

Důkaz: Inicializace trvá $\mathcal{O}(n)$. Každý vrchol uzavřeme nejvýše jednou (to jsme zatím nahlédli z analogie s BFS, více viz příští oddíl), takže vnějším cyklem projdeme nejvýše n -krát. Pokaždé hledáme minimum z n ohodnocení vrcholů a procházíme až n následníků. \square

Pokud bychom kromě vzdáleností chtěli vypsát i nejkratší cesty, můžeme je získat ze zapamatovaných předchůdců. Chceme-li získat nejkratší cestu z vrcholu v_0 do nějakého w , stačí se podívat na $P(w)$, $P(P(w))$ a tak dále až do v_0 . Tak projdeme pozpátku nějakou cestu z v_0 do w , která má délku $h(w)$, a tedy je nejkratší. Dokonce platí, že předchůdci kódují hrany stromu nejkratších cest: kdykoliv $P(v) = u$, leží hrana uv ve stromu nejkratších cest. Důkaz těchto tvrzení ponecháváme jako cvičení 1.3.7.

Dijkstrův algoritmus s haldou

Složitost $\mathcal{O}(n^2)$ je příznivá, pokud máme co do činění s hustým grafem. V grafech s malým počtem hran je náš odhad zbytečně hrubý: v cyklu přes následníky

trávíme čas lineární se stupněm vrcholu, za celou dobu běhu algoritmu tedy pouze $\mathcal{O}(m)$. Stále nás ale brzdí hledání minima v kroku 8.

Proto si pořídíme nějakou datovou strukturu, která umí rychle vyhledávat minimum. Nabízí se například binární halda z oddílu ?? . Uložíme do ní všechny otevřené vrcholy, jako klíče budou sloužit ohodnocení $h(v)$. V každé iteraci Dijkstrova algoritmu nalezneme minimum pomocí operace EXTRACTMIN. Když v kroku 11 měníme ohodnocení vrcholu, buďto tento vrchol nově otevíráme (takže provedeme INSERT do haldy), nebo již je otevřený (což znamená DECREASE prvku, který již v haldě je). Nesmíme zapomenout, že DECREASE neumí prvek vyhledat, takže si u každého vrcholu budeme průběžně udržovat jeho pozici v haldě.

Věta: Dijkstrův algoritmus s haldou běží v čase $\mathcal{O}(n \cdot T_i + n \cdot T_x + m \cdot T_m)$, kde T_i , T_x a T_d jsou složitosti operací INSERT, EXTRACTMIN a DECREASE.

Důkaz: Operaci INSERT provádíme při otevírání vrcholu, EXTRACTMIN při jeho zavírání. Obojí pro každý vrchol nastane nejvýše jednou. DECREASE voláme, když při zavírání vrcholu v snižujeme hodnotu jeho souseda w . To se pro každou hranu vw stane za celou dobu běhu algoritmu nejvýše jednou. Zbývající operace algoritmu trvají $\mathcal{O}(n + m)$. \square

Důsledek: Dijkstrův algoritmus s binární haldou běží v čase $\mathcal{O}((n + m) \cdot \log n)$.

Důkaz: V haldě není nikdy více než n prvků, takže všechny tři operace pracují v čase $\mathcal{O}(\log n)$. \square

I tento čas lze ještě vylepšit. Některé z možných přístupů prozkoumáme v cvičeních 3 a 4. V kapitole ?? posléze vybudujeme Fibonacciho haldu, která umí DECREASE v konstantním čase, aniž by se asymptoticky zpomalily ostatní operace.

Důsledek: Dijkstrův algoritmus s Fibonacciho haldou běží v čase $\mathcal{O}(m + n \log n)$.

Cvičení

1. Odsimulujte chod Dijkstrova algoritmu na grafu z obrázku 1.1.
2. Uvažujte „spojité BFS“, které bude plynule procházet vnitřkem hran. Pokuste se o formální definici takového algoritmu. Nahlédněte, že diskrétní simulací tohoto spojitého procesu (která se bude zastavovat ve významných událostech, totiž ve vrcholech) získáme Dijkstrův algoritmus. Podobnou myšlenku potkáme v kapitole ?? o geometrických algoritmech.
- 3.* Dijkstrův algoritmus s haldou provede m operací DECREASE, ale pouze n operací INSERT a EXTRACTMIN. Hodila by se tedy halda, která má DECREASE rychlejší, byť za cenu zpomalení ostatních operací. Tuto vlastnost mají například d -regulární haldy z cvičení ?? . Rozmyslete, jakou hodnotu d zvolit, aby se minimalizovala složitost celého algoritmu.
4. Nechť délky hran leží v množině $\{0, \dots, L\}$. Navrhněte datovou strukturu založenou na příhrádkách, s níž Dijkstrův algoritmus poběží v čase $\mathcal{O}(nL + m)$. Pokuste se vystačit s pamětí $\mathcal{O}(n + m + L)$.
5. Na mapě města jsme přiřadili každé silnici čas na průjezd a každé křižovatce průměrnou dobu čekání na semaforech. Jak hledat nejrychlejší cestu?

1.3. Relaxační algoritmy

Na Dijkstrův algoritmus se můžeme dívat trochu obecněji. Získáme tak nejen důkaz jeho správnosti pro neceločíselné délky hran, ale též několik dalších zajímavých algoritmů. Ty budou fungovat i pro grafy se zápornými hranami (stále bez záporných cyklů), které v tomto oddílu dovolíme.

Esencí Dijkstrova algoritmu je, že přiřazuje vrcholům nějaká *ohodnocení* $h(v)$. To jsou nějaká reálná čísla, která popisují, jakým nejkratším sledem se zatím umíme do vrcholu dostat. Tato čísla postupně upravujeme, až se z nich stanou skutečné vzdálenosti od v_0 . Pokusme se tento princip popsat obecně.

Na počátku výpočtu ohodnotíme vrchol v_0 nulou a všechny ostatní vrcholy nekonečnem. Pak opakujeme následující postup: vybereme nějaký vrchol v s konečným ohodnocením a pro všechny jeho následníky w otestujeme, zda se do nich přes v nedovedeme dostat lépe, než jsme zatím dovedli. Této operaci se obvykle říká *relaxace* a odpovídá krokům 9 až 13 Dijkstrova algoritmu.

Jeden vrchol můžeme obecně relaxovat vícekrát, ale nemá smysl dělat to znovu, pokud se jeho ohodnocení mezitím nezměnilo. To ohlídá stav vrcholu: *otevřené* vrcholy je potřeba znovu relaxovat, *uzavřené* jsou relaxované a zatím to znovu nepotřebují.

Z toho vychází obecný relaxační algoritmus. Ten pokaždé vybere jeden otevřený vrchol, uzavře ho a relaxuje a pokud se tím změní ohodnocení jiných vrcholů, tak je otevře. Všimněte si, že na rozdíl od prohledávání do šířky můžeme jeden vrchol otevřít a uzavřít vícekrát.

Algoritmus RELAXACE

Vstup: Graf G a počáteční vrchol v_0

1. Pro všechny vrcholy v :
2. $stav(v) \leftarrow \text{nenalezený}$, $h(v) \leftarrow +\infty$, $P(v) \leftarrow \text{nedefinováno}$
3. $stav(v_0) \leftarrow \text{otevřený}$, $h(v_0) \leftarrow 0$
4. Dokud existují otevřené vrcholy:
5. $v \leftarrow$ nějaký otevřený vrchol
6. Pro všechny následníky w vrcholu v : \triangleleft *relaxujeme vrchol* w
7. Pokud $h(w) > h(v) + \ell(u, v)$:
8. $h(w) \leftarrow h(v) + \ell(u, v)$
9. $stav(w) \leftarrow \text{otevřený}$
10. $P(w) \leftarrow v$
11. $stav(v) \leftarrow \text{uzavřený}$

Výstup: Ohodnocení vrcholů h a pole předchůdců P

Dijkstrův algoritmus je tedy speciálním případem relaxačního algoritmu, v němž vždy vybíráme otevřený vrchol s nejmenším ohodnocením. Prozkoumejme, co o relaxačním algoritmu platí obecně.

Invariant O (ohodnocení): Hodnota $h(v)$ nikdy neroste. Je-li konečné, rovná se délce nějakého sledu z v_0 do v .

Důkaz: Indukcí podle doby běhu algoritmu. Na počátku výpočtu tvrzení určitě platí, protože jediné konečné je $h(v_0) = 0$. Kdykoliv pak algoritmus změní $h(w)$, stane se tak relaxací nějakého vrcholu v , jehož $h(v)$ je konečné. Podle indukčního předpokladu tedy existuje v_0v -sled délky $h(v)$. Jeho rozšířením o hranu vw vznikne v_0w -sled délky $h(v) + \ell(v, w)$, což je přesně hodnota, na niž snižujeme $h(w)$. \square

Lemma D (dosažitelnost): Pokud se výpočet zastaví, uzavřené jsou právě vrcholy dosažitelné z v_0 .

Důkaz: Dokážeme stejně jako obdobnou vlastnost BFS. Jediné, v čem se situace liší, je, že uzavřený vrchol je možné znovu otevřít. To se ovšem, pokud se výpočet zastavil, stane pouze konečně-krát, takže stačí uvážit situaci při posledním uzavření. \square

Lemma V (vzdálenost): Pokud se výpočet zastaví, konečná ohodnocení vrcholů jsou rovna vzdálenostem od v_0 .

Důkaz: Inspirujeme se důkazem obdobné vlastnosti BFS. Vrchol v označíme za špatný, pokud $h(v)$ není rovno $d(v_0, v)$. Jelikož $h(v)$ odpovídá délce nějakého v_0v -sledu, musí být $h(v) > d(v_0, v)$. Pro spor předpokládejme, že existují nějaké špatné vrcholy.

Mezi všemi nejkratšími cestami z v_0 do špatných vrcholů vybereme tu s nejmenším počtem hran. Označíme v poslední vrchol této cesty a p její předposlední vrchol (ten jistě existuje, neboť v_0 je dobrý). Vrchol p musí být dobrý, takže ohodnocení $h(p)$ je rovno $d(v_0, p)$. Podívejme se, kdy p získal tuto hodnotu. Tehdy musel p být otevřený. Později byl tudíž zavřen a relaxován, načež muselo platit $h(v) \leq d(v_0, p) + \ell(p, v) = d(v_0, v)$. Jelikož ohodnocení vrcholů nikdy nerostou, došlo ke sporu. \square

Rozbor Dijkstrova algoritmu

Nyní se vraťme k Dijkstrovu algoritmu. Ukážeme, že nejsou-li v grafu záporné hrany, průběh výpočtu má následující zajímavé vlastnosti.

Invariant M (monotonie): V každém kroku výpočtu platí:

- (1) Kdykoliv je z uzavřený vrchol a o otevřený, platí $h(z) \leq h(o)$.
- (2) Ohodnocení uzavřených vrcholů se nemění.

Důkaz: Obě vlastnosti dokážeme dohromady indukcí podle délky výpočtu. Na počátku výpočtu obě triviálně platí, neboť neexistují žádné uzavřené vrcholy.

V každém dalším kroku vybereme otevřený vrchol v s nejmenším $h(v)$. Tehdy musí platit $h(z) \leq h(v) \leq h(o)$ pro libovolný z uzavřený a o otevřený. Nyní vrchol v relaxujeme: pro každou hranu vw se pokusíme snížit $h(w)$ na hodnotu $h(v) + \ell(v, w) \geq h(v)$.

- Pokud w byl uzavřený, nemůže se jeho hodnota změnit, neboť již před relaxací byla menší nebo rovna $h(v)$. Proto platí (2).
- Pokud w byl otevřený, jeho hodnota se sice může snížit, ale nikdy ne pod $h(v)$, takže ani pod $h(z)$ žádného uzavřeného z .

Nerovnost (1) v obou případech zůstává zachována a neporuší se ani přesunem vrcholu v mezi uzavřené. \square

Věta: Dijkstrův algoritmus na grafu bez záporných hran uzavírá všechny dosažitelné vrcholy v pořadí podle rostoucí vzdálenosti od počátku (každý právě jednou). V okamžiku uzavření je ohodnocení rovno této vzdálenosti a dále se nezmění.

Důkaz: Především z invariantu **M** víme, že žádný vrchol neotevřeme vícekrát, takže ho ani nemůžeme vícekrát uzavřít. Algoritmus proto skončí a podle lemat **D** a **V** jsou na konci uzavřeny všechny dosažitelné vrcholy a jejich ohodnocení odpovídají vzdálenostem.

Ohodnocení se přitom od okamžiku uzavření vrcholu nezměnilo (opět viz invariant **M**) a tehdy bylo větší nebo rovno ohodnocením předchozích uzavřených vrcholů. Pořadí uzavírání proto skutečně odpovídá vzdálenostem. \square

Vidíme tedy, že naše analogie mezi BFS a Dijkstrovým algoritmem funguje i pro neceločíselné délky hran.

Bellmanův-Fordův algoritmus

Zkusme se ještě zamyslet nad výpočtem vzdáleností v grafech se zápornými hranami. Snadno nahlédneme, že Dijkstrův algoritmus na takových grafech může vrcholy otevírat opakovaně, ba dokonce může běžet exponenciálně dlouho (cvičení 1).

Relaxace se ovšem není potřeba vzdávat, postačí změnit podmínku pro výběr vrcholu: namísto haldy si pořídíme obyčejnou frontu. Jinými slovy, budeme uzavírat nejstarší z otevřených vrcholů. Tomuto algoritmu se podle jeho objevitelů Richarda Ernesta Bellmana a Lestera Forda Jr. říká Bellmanův-Fordův. V následujících odstavcích prozkoumáme, jak efektivní je.

Definice: Definujeme fáze výpočtu následovně: ve fázi F_0 otevřeme počáteční vrchol v_0 , fáze F_{i+1} uzavírá vrcholy otevřené během fáze F_i .

Invariant F (fáze): Pro vrchol v na konci i -té fáze platí, že jeho ohodnocení je shora omezeno délkou nejkratšího v_0v -sledu o nejvýše i hranách.

Důkaz: Tvzení dokážeme indukci podle i . Pro $i = 0$ tvrzení platí – jediný vrchol dosažitelný z v_0 sledem nulové délky je v_0 sám; jeho ohodnocení je nulové, ohodnocení ostatních vrcholů nekonečné.

Nyní provedeme indukční krok. Podívejme se na nějaký vrchol v na konci i -té fáze ($i > 0$). Označme S nejkratší v_0v -sled o nejvýše i hranách.

Pokud sled S obsahuje méně než i hran, požadovaná nerovnost platila už na konci předchozí fáze a jelikož ohodnocení vrcholů nerostou, platí nadále.

Obsahuje-li S právě i hran, označme uv jeho poslední hranu a S' podsled z v_0 do u . Podle indukčního předpokladu je na začátku i -té fáze $h(u) \leq \ell(S')$. Na tuto hodnotu muselo být $h(u)$ nastaveno nejpozději v $(i - 1)$ -ní fázi, čímž byl vrchol u otevřen. Nejpozději v i -té fázi proto musel být uzavřen a relaxován. Na začátku relaxace muselo stále platit $h(u) \leq \ell(S')$ – hodnota $h(u)$ se sice mohla změnit, ale ne zvýšit. Po relaxaci tedy muselo platit $h(v) \leq h(u) + \ell(u, v) \leq \ell(S') + \ell(u, v) = \ell(S)$. \square

Důsledek: Pokud graf neobsahuje záporné cykly, po n -té fázi se algoritmus zastaví.

Důkaz: Po $(n - 1)$ -ní fázi jsou všechna ohodnocení shora omezena délkami nejkratších cest, takže se v n -té fázi už nemohou změnit a algoritmus se zastaví. \square

Věta: V grafu bez záporných cyklů nalezneme Bellmanův-Fordův algoritmus všechny vzdálenosti z vrcholu v_0 v čase $\mathcal{O}(nm)$.

Důkaz: Podle předchozího důsledku se po n fázích algoritmus zastaví a podle lemmat **D** a **V** vydá správný výsledek. Během jedné fáze přitom relaxuje každý vrchol nejvýše jednou, takže celá fáze dohromady trvá $\mathcal{O}(m)$. \square

Cvičení

1. Ukažte příklad grafu s celočíselně ohodnocenými hranami, na kterém Dijkstrův algoritmus běží exponenciálně dlouho.
2. Upravte Bellmanův-Fordův algoritmus, aby uměl detekovat záporný cyklus dosažitelný z vrcholu v_0 . Uměli byste tento cyklus vypsat?
3. Papeho algoritmus funguje podobně jako Bellmanův-Fordův, pouze místo fronty používá zásobník. Ukažte, že tento algoritmus v nejhorším případě běží exponenciálně dlouho.
4. Uvažujme následující algoritmus: provedeme n fází, v každé z nich postupně relaxujeme všechny vrcholy. Spočíte tento algoritmus správné vzdálenosti? Jak si stojí v porovnání s Bellmanovým-Fordovým algoritmem?
5. První algoritmus vymyšlený Bellmanem relaxoval místo vrcholů hrany. Cyklicky procházel všechny hrany a pro každou hranu uv se pokusil snížit $h(v)$ na $h(u) + \ell(u, v)$. Dokažte, že tento algoritmus také spočítá správné vzdálenosti, a rozeberte jeho složitost.
- 6.* Dokažte, že v grafu bez záporných cyklů se obecný relaxační algoritmus zastaví, ať už vrchol k uzavření vybíráme libovolně.
7. Dokažte, že po zastavení obecného relaxačního algoritmu kódují předchůdci $P(v)$ hrany stromu nejkratších cest.

1.4. Matice vzdáleností a Floydův-Warshallův algoritmus

Někdy potřebujeme zjistit vzdálenosti mezi všemi dvojicemi vrcholů, tedy zkonstruovat *matici vzdáleností*. Pokud v grafu nejsou záporné hrany, mohli bychom spustit Dijkstrův algoritmus postupně ze všech vrcholů. To by trvalo $\mathcal{O}(n^3)$, nebo v implementaci s haldou $\mathcal{O}(n \cdot (n + m) \cdot \log n)$.

V této kapitole ukážeme jiný, daleko jednodušší algoritmus založený na dynamickém programování (tuto techniku později rozvineme v kapitole ??). Pochází z roku 1959 od Roberta Floyda a Stephena Warshalla. Matici vzdáleností spočítá v čase $\Theta(n^3)$, dokonce mu ani nevadí záporné hrany.

Definice: Označíme D_{ij}^k délku nejkratší cesty z vrcholu i do vrcholu j , jejíž vnitřní vrcholy leží v množině $\{1, \dots, k\}$. Pokud žádná taková cesta neexistuje, položíme $D_{ij}^k = +\infty$.

Pozorování: Hodnoty D_{ij}^k mají následující vlastnosti:

- D_{ij}^0 nedovoluje používat žádné vnitřní vrcholy, takže je to buďto délka hrany ij , nebo $+\infty$, pokud taková hrana neexistuje.

- D_{ij}^n už vnitřní vrcholy neomezuje, takže je to vzdálenost z i do j .

Algoritmus dostane na vstupu matici D^0 a postupně bude počítat matice D^1, D^2, \dots , až se dopočítá k D^n a vydá ji jako výstup.

Nechť tedy známe D_{ij}^k pro nějaké k a všechna i, j . Chceme spočítat D_{ij}^{k+1} , tedy délku nejkratší cesty z i do j přes $\{1, \dots, k+1\}$. Jak může tato cesta vypadat?

- (1) Buďto neobsahuje vrchol $k+1$, v tom případě je stejná, jako nejkratší cesta z i do j přes $\{1, \dots, k\}$. Tehdy $D_{ij}^{k+1} = D_{ij}^k$.
- (2) Anebo vrchol $k+1$ obsahuje. Tehdy se skládá z cesty z i do $k+1$ a cesty z $k+1$ do j . Obě dílčí cesty jdou přes vrcholy $\{1, \dots, k\}$ a obě musí být nejkratší takové (jinak bychom je mohli vyměnit za kratší). Proto $D_{ij}^{k+1} = D_{i,k+1}^k + D_{k+1,j}^k$.

Za D_{ij}^{k+1} si proto zvolíme minimum z těchto dvou variant.

Musíme ale ošetřit jeden potenciální problém: v případě (2) spojujeme dvě cesty, což ovšem nemusí dát cestu, nýbrž sled: některým vrcholem z $\{1, \dots, k\}$ bychom mohli projít vícekrát. Stačí si ale uvědomit, že kdykoliv by takový sled byl kratší než cesta z varianty (1), znamenalo by to, že se v grafu nachází záporný cyklus. V grafech bez záporných cyklů proto náš vzorec funguje.

Hotový algoritmus vypadá takto:

Algoritmus FLOYDWARSHALL

Vstup: Matice délek hran D^0

1. Pro $k = 0, \dots, n-1$:
2. Pro $i = 1, \dots, n$:
3. Pro $j = 1, \dots, n$:
4. $D_{ij}^{k+1} \leftarrow \min(D_{ij}^k, D_{i,k+1}^k + D_{k+1,j}^k)$

Výstup: Matice vzdáleností D^n

Časová složitost evidentně činí $\Theta(n^3)$, paměťová bohužel také. Nabízí se využití toho, že vždy potřebujeme pouze matice D^k a D^{k+1} , takže by nám místo trojrozměrného pole stačila dvě dvojrozměrná.

Existuje ovšem elegantnější, byť poněkud drzý trik: použijeme jedinou matici a budeme hodnoty přepisovat na místě. Pak ovšem nerozlišíme, zda právě čteme D_{pq}^k , nebo na jeho místě už je zapsáno D_{pq}^{k+1} . Zajímavé je, že na tom nezáleží:

Lemma: Pro všechna i, j, k platí $D_{k+1,j}^{k+1} = D_{k+1,j}^k$ a $D_{i,k+1}^{k+1} = D_{i,k+1}^k$.

Důkaz: Podle definice se levá a pravá strana každé rovnosti liší jenom tím, zda jsme jako vnitřní vrchol cesty povolili použít vrchol $k+1$. Ten je ale už jednou použit jako počáteční, resp. koncový vrchol cesty, takže se uvnitř tak jako tak neobjeví. \square

Věta: Floydův-Warshallův algoritmus s přepisováním na místě vypočte matici vzdálenosti grafu bez záporných cyklů v čase $\Theta(n^3)$ a prostoru $\Theta(n^2)$.

Cvičení

1. Jak z výsledku Floydova-Warshallova algoritmu zjistíme, kudy nejkratší cesta mezi nějakými dvěma vrcholy vede?
2. Upravte Floydův-Warshallův algoritmus, aby pro každý vrchol našel nejkratší kružnici, která jím prochází. Předpokládejte, že v grafu nejsou žádné záporné cykly.
3. Upravte Floydův-Warshallův algoritmus, aby zjistil, zda v grafu existuje záporný cyklus.

1.5. Další cvičení

1. Lze se v algoritmech na hledání nejkratší cesty zbavit záporných hran tím, že ke všem ohodnocením hran přičteme nějaké velké číslo k ?
2. Počítačovou síť popíšeme orientovaným grafem, jehož vrcholy odpovídají routerům a hrany linkám mezi nimi. Pro každou linku známe pravděpodobnost toho, že bude funkční. Pravděpodobnost, že bude funkční nějaká cesta, je dána součinem pravděpodobností jejích hran. Jak pro zadané dva routery najít nejpravděpodobnější cestu mezi nimi?
3. Mějme mapu města ve tvaru orientovaného grafu. Každou hranu ohodnotíme podle toho, jaký nejvyšší kamion po dané ulici může projet. Po cestě tedy projede maximálně tak vysoký náklad, kolik je minimum z ohodnocení jejích hran. Jak pro zadané dva vrcholy najít cestu, po níž projede co nejvyšší náklad?
4. V Tramtárii jezdí po železnici samé rychlíky, které nikde po cestě nestaví. V jízdním řádu je pro každý rychlík uvedeno počáteční a cílové nádraží, čas odjezdu a čas příjezdu. Nyní stojíme v čase t na nádraží a a chceme se co nejrychleji dostat na nádraží b . Navrhněte algoritmus, který najde takové spojení.
5. Pokračujeme v předchozím cvičení: Mezi všemi nejrychlejšími spojeními chceme najít takové, v němž je nejméně přestupů.
6. Směnárna obchoduje s n měnami (měna číslo 1 je koruna) a vyhláší matici kurzů K . Kurz K_{ij} říká, kolik za jednu jednotku i -té měny dostaneme jednotek j -té měny. Vymyslete algoritmus, který zjistí, zda existuje posloupnost směn, která začne s jednou korunou a skončí s více korunami.
7. Vymyslete algoritmus, který nalezne všechny hrany, jež leží na alespoň jedné nejkratší cestě.
8. Kritická hrana budiž taková, která leží na všech nejkratších cestách. Tedy ta, jejíž prodloužení by ovlivnilo vzdálenost. Navrhněte algoritmus, který najde všechny kritické hrany.
9. Silnice v mapě máme ohodnocené dvěma čísly: délkou a mýtem (poplatkem za projetí). Jak najít nejlevnější z nejkratších cest?
- 10.* Sestrojte algoritmus pro řešení soustavy lineárních nerovnic tvaru $x_i - x_j \leq c_{ij}$, kde c_{ij} jsou reálné, ne nutně kladné konstanty.