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. Málokdy 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 orientovaný graf $G = (V, E)$. Každé hraně $e \in E$ přiřadíme její *ohodnocení* neboli *délku,* což bude nějaké reálné číslo $\ell(e)$. Tím vznikne *ohodnocený graf* nebo též *síť.* 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 hran, které na něm leží.

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: Pro každý uv-sled existuje uv-cesta stejné nebo menší délky.

Důkaz: Pokud sled není cestou, znamená to, že se v něm opakují vrcholy. Označme tedy t první vrchol (v pořadí od u), který se zopakoval. Část sledu mezi prvním a druhým výskytem vrcholu t tvoří kružnici. Tu vystřihneme a získáme uv-sled stejné nebo menší délky. Přitom ubyla alespoň jedna hrana, takže opakováním tohoto postupu časem dostaneme cestu. □

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. □

Důsledek: Pro vzdálenosti platí *trojúhelníková nerovnost:*

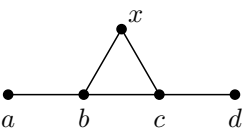
$$d(u, v) \leq d(u, w) + d(w, v).$$

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. □

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

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 stále platilo. Proto budeme v této kapitole grafy se zápornými hranami, ale bez záporných cyklů připouštět. Ostatně, jak uvidíme ve cvičeních, občas se to hodí.

Cvičení

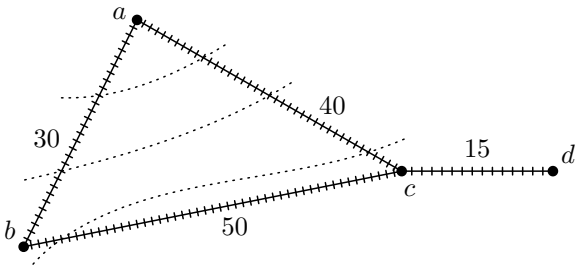
1. Ukažte, jak pro libovolné n sestrojit graf na nejvýše n vrcholech, v němž mezi nějakými dvěma vrcholy existuje $2^{\Omega(n)}$ nejkratších cest.
2. Připomeňte si definici metrického prostoru v matematické analýze. Kdy tvoří množina všech vrcholů spolu s funkcí $d(u,v)$ metrický prostor?
3. Už víme, že mají-li všechny hrany stejnou délku, nejkratší cestu najdeme prohledáváním do šířky. Uměli byste si poradit, kdyby délky hran byly malá přirozená čísla, řekněme 1 až 10?
4. Navrhněte lineární algoritmus pro výpočet vzdálenosti dvou vrcholů v acyklickém orientovaném grafu.
5. 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.

1.2. Dijkstrův algoritmus

Dnes asi nejpoužívanější algoritmus pro hledání nejkratších cest vymyslel v roce

1959 pan Edsger Dijkstra.[1] Funguje efektivně, kdykoliv jsou všechny hrany ohodnocené nezápornými čísly. Nás k němu dovede následující myšlenkový pokus.

Mějme orientovaný graf, jehož hrany jsou ohodnocené celými kladnými čísly. Každou hranu nyní „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. 1.1: 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. Vlna prvních 30 kroků prochází vnitřkem hran ab a ac. Pak dorazí do vrcholu b, načež se šíří dál zbytkem 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 prokousáváme 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 přeskočíme 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.

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

[1] Je to holandské jméno, takže ho čteme „dajkstra“.

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
Vstup: Graf G a počáteční vrchol v_0

1. Pro všechny vrcholy v:
2. $\quad stav(v) \leftarrow$ *nenalezený*
3. $\quad h(v) \leftarrow +\infty$
4. $stav(v_0) \leftarrow$ *otevřený*
5. $h(v_0) \leftarrow 0$
6. Dokud existují nějaké otevřené vrcholy:
7. $\quad$ Vybereme otevřený vrchol v, jehož $h(v)$ je nejmenší.
8. $\quad$ Pro všechny následníky w vrcholu v:
9. $\quad\quad$ Pokud $h(w) < h(v) + \ell(v,w)$:
10. $\quad\quad\quad h(w) \leftarrow h(v) + \ell(v,w)$
11. $\quad\quad\quad stav(w) \leftarrow$ *otevřený*
12. $\quad\quad\quad P(w) \leftarrow v$
13. $\quad stav(v) \leftarrow$ *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 následující sekci, teď se zaměříme především na časovou složitost.

Věta: Dijkstrův algoritmus spočte v grafu s nezáporně ohodnocenými hranami vzdálenosti od vrcholu v_0 a zkonstruuje strom nejkratších cest 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 sekce), takže vnějším cyklem projdeme nejvýše n-krát. Pokaždé hledáme minimum z n ohodnocení vrcholů a procházíme až n následníků. □

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 $\mathcal{O}(m)$. Brzdí nás ale hledání minima v kroku 7.

Proto si pořídíme haldu a uložíme do ní všechny otevřené vrcholy (uspořádané podle ohodnocení). Nalézt a odstranit minimum potrvá $\mathcal{O}(\log n)$, vložit novou hodnotu nebo změnit stávající potrvá taktéž $\mathcal{O}(\log n)$. Přitom vkládání prvků a hledání minima zopakujeme nejvýše n-krát a změnu hodnoty nanejvýš m-krát, takže pro celkovou časovou složitost platí: Fix!

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

Fix: Je někde vysvětleno, jak mění prvek v haldě? Potřebujeme rozebrat, jak ho tam najít.

S chytřejšími datovými strukturami lze tuto složitost ještě zlepšit. Fix!

Cvičení

1. 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. Fix!
2. Lze se v algoritmech na hledání nejkratší cesty zbavit záporných hran tím, že ke všem ohodnocením hran přičteme nějaké velké číslo K?
3. Mějme mapu města, která má časem potřebným na průjezd ohodnocené nejen hrany (silnice), ale také vrcholy (křižovatky). Upravte Dijkstrův algoritmus, aby našel nejrychlejší cestu i v tomto případě.
4. 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?
5. 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, pro níž projede co nejvyšší náklad?
6. 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.
7. Pokračujeme v předchozím cvičení: Mezi všemi nejrychlejšími spojeními chceme najít takové, v němž je nejméně přestupů.

1.3. Relaxační algoritmy

Na Dijkstrův algoritmus se také můžeme dívat trochu obecněji. Získáme tak nejen důkaz jeho správnosti pro neceločíselné délky hran, ale také několik dalších zajímavých algoritmů.

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

Na počátku výpočtu ohodnotíme vrchol v_0 nulou a všechny ostatní vrcholy nekonečnem. Pak opakujeme následující krok: vybereme nějaký vrchol v s konečným

Fix: Rozepsat, zejména odkázat na kapitolu o Fibonacciho haldách, až bude.
Fix: Tady by se hodil odkaz na geometrické algoritmy, kde se používá podobná myšlenka.

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 8 až 12 Dijkstrova algoritmu.

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

Algoritmus tedy pokaždé vybere jeden otevřený vrchol, ten uzavře 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: $stav(v) \leftarrow$ *nenalezený*, $h(v) \leftarrow +\infty$.
2. $stav(v_0) \leftarrow$ *otevřený*, $h(v_0) \leftarrow 0$
3. Dokud existují otevřené vrcholy:
4. $\quad v \leftarrow$ nějaký otevřený vrchol
5. $\quad$ Pro všechny následníky w vrcholu v: *(Relaxujeme vrchol v)*
6. $\quad\quad$ Pokud $h(w) > h(v) + \ell(u, v)$:
7. $\quad\quad\quad h(w) \leftarrow h(v) + \ell(u, v)$
8. $\quad\quad\quad stav(w) \leftarrow$ *otevřený*
9. $\quad\quad\quad P(w) \leftarrow v$
10. $\quad stav(v) \leftarrow$ *uzavřený*

Výstup: Ohodnocení vrcholů h a pole předchůdců P

Dijkstrův algoritmus je tedy speciálním případem relaxačního algoritmu, kde vždy vybíráme otevřený vrchol s nejmenším ohodnocením. Některá tvrzení ovšem platí i o obecném algoritmu.

Invariant H: Ohodnocení $h(v)$ 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 níž snižujeme $h(w)$. □

Lemma D: 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. □

Lemma V: 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.

Vybereme špatný vrchol v takový, že nejkratší v_0v-cesta je tvořena nejmenším možným počtem hran. Buď p předchůdce vrcholu v na této cestě (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. □

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: V každém kroku výpočtu platí:

(1) Kdykoliv je z uzavřený vrchol a o otevřený, platí $h(z) \leq h(o)$.

(2) Jakmile nějaký vrchol uzavřeme, jeho ohodnocení se nemění.

Důkaz: Obě vlastnosti dokážeme dohromady indukcí podle délky výpočtu. Na počátku výpočtu 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. □

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 lemmat **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 tedy skutečně odpovídá vzdálenostem. □

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 (ale bez záporných cyklů). Snadno nahlédneme, že Dijkstrův algoritmus na takových grafech může vrcholy otevírat opakovaně, ba dokonce může běžet exponenciálně dlouho (cvičení 1.3.1).

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 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 B: 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: Tvrzení dokážeme indukcí 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 $h(v) \leq h(u)+\ell(u,v) \leq \ell(S')+\ell(u,v) = \ell(S)$. □

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. □

Věta: V grafu bez záporných cyklů nalezne 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 $\mathcal{O}(m)$. □

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. Spočte tento algoritmus správné vzdálenosti? Jak si stojí v porovnání s Bellmanovým-Fordovým algoritmem?
5.* Dokažte, že v grafu bez záporných cyklů se obecný relaxační algoritmus zastaví, ať už vrchol k uzavření vybíráme libovolně.
6. Směnárna obchoduje s n měnami (měna číslo 1 je koruna) a vyhlašuje 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. Navrněte 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?

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. Pochází od pánů Floyda a Warshalla a matici vzdálenosti spočítá v čase $\Theta(n^3)$. Dokonce mu ani nevadí záporné hrany.

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

Pozorování: Hodnoty D^k_{ij} mají následující vlastnosti:

- D^0_{ij} nedovoluje používat žádné vnitřní vrcholy, takže je to buďto délka hrany ij, nebo $+\infty$, pokud taková hrana neexistuje.
- D^n_{ij} už vnitřní vrcholy neomezuje, takže je to vzdálenost z i do j.

Algoritmus tedy dostane na vstupu matici D^0 a postupně bude počítat matice D^1, D_2, …, až se dopočítá k D^n a vydá ji jako výstup.

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

(1) Buďto neobsahuje vrchol $k+1$, v tom případě je stejná, jako nejkratší cesta z i do j přes $\{1,\ldots,k\}$. 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 $\{1,\ldots,k\}$ 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 $\{1,\ldots,k\}$ 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 $k = 0,\ldots,n-1$:
2. Pro $i = 1,\ldots,n$:
3. Pro $j = 1,\ldots,n$:
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 využít 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}. To ale nevadí:

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. □

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 našel nejkratší kružnici (v grafu bez záporných cyklů).
3. Upravte Floydův-Warhsallův algoritmus, aby zjistil, zda v grafu existuje záporný cyklus.