Kapitola 1

Nejkratší cesta v grafu

Motivace: Silniční síť si můžeme znázornit grafem. Křižovatky odpovídají vrcholům grafu a silnice mezi nimi hranám. Každý úsek silnice odpovídající hraně má svojí délku. Ta odpovídá ohodnocení hrany. Můžeme se ptát jaká je nejkratší cesta z Prahy do Ostravy? Kolik je to kilometrů a kudy vede? Maximální povolená rychlost na každém úseku silnice nám říká, jak rychle po ní můžeme jet. Z rychlosti a délky silnice se dá dopočítat, za jak dlouhou úsek silnice projedeme. Proto se můžeme ptát i na nejrychlejší cestu z Prahy do Ostravy. Nejrychlejší cesta nemusí být zrovna nejkratší.

Problém hledaní nejkratší cesty v grafu a jemu podobné problémy se vyskytují téměř všude: při hledání autobusového nebo vlakového spojení v online jízdních řádech, při hledání optimální cesty v navigacích do aut, při plánování pohybu robotů a nebo při routování paketů v internetu.

Definice: Co to je vzdálenost? Vzdálenost je matematicky popsána pojmem metrika.[1] My si v této kapitole vystačíme se vzdáleností v grafech, tj. s grafovou metrikou. Nechť G je graf s ohodnocením hran $c : E(G) \to \mathbb{R}$. Číslo $c(e)$ se nazývá cena hrany e, ale v kontextu hledání nejkratší cesty mu budeme říkat délka hrany e. *Délka cesty* $x = v_0e_1v_1e_2 \dots v_{k-1}e_kv_k = y$ se počítá jako $\sum_{i=1}^{k} c(e_i)$. *Vzdálenost dvou vrcholů* x a y v grafové metrice je délka nejkratší cesty mezi x a y.

Úkol 1: (Shortest path) Dostaneme graf G s ohodnocením hran $c : E \to \mathbb{R}$, počáteční vrchol s a cílový vrchol t. Chceme nalézt nejkratší cestu z s do t.

Všechna prakticky používaná řešení místo výše uvedeného problému řeší problém obecnější. Místo jedné cesty z s do t hledáme nejkratší cestu z s do všech ostatních vrcholů.

Úkol 2: (Single source shortest path) Dostaneme graf G s ohodnocením hran

[1] Nechť M je neprázdná množina. *Metrika* je zobrazení $\rho : M \times M \to \mathbb{R}_+$, které pro libovolné $x, y \in M$ splňuje axiomy:

1. (totožnost): $\rho(x, y) = 0$ právě tehdy když $x = y$
2. (symetrie): $\rho(x, y) = \rho(y, x)$
3. (trojúhelníková nerovnost): $\rho(x, z) \leq \rho(x, y) + \rho(y, z)$.

Z prvního a třetího axiomu vyplývá nezápornost metriky $\rho(x, y) \geq 0$. Číslu $\rho(x, y)$ říkáme vzdálenost bodů x a y. Nechť $p_1 = (x_1, y_1)$ a $p_2 = (x_2, y_2)$ jsou dva body v rovině. Mezi nejpoužívanější metriky v rovině patří:

- eukleidovská metrika – vzdálenost p_1 a p_2 je délka úsečky p_1p_2, neboli vzdálenost vzdušnou čarou.
- maximová metrika – vzdálenost p_1 a p_2 je maximum z $|x_1 - x_2|$ a $|y_1 - y_2|$.
- manhattanská metrika – po Manhattanu v New Yorku se chodí v síti pravoúhlých ulic, které jsou rovnoběžné s osami souřadného systému. Vzdálenost p_1 a p_2 je $|x_1 - x_2| + |y_1 - y_2|$.

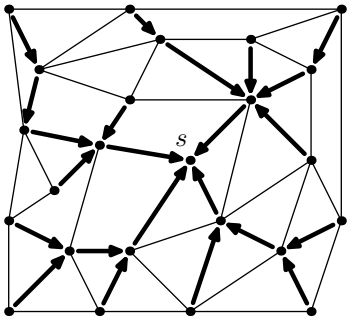
$c : E \to \mathbb{R}$, počáteční vrchol s. Chceme nalézt nejkratší cestu z s do všech ostatních vrcholů $v \in V(G)$.

Předchozí úkol můžeme ještě více zobecnit.

Úkol 3: (All pairs shortest path) Dostaneme graf G s ohodnocením hran $c : E \to \mathbb{R}$ a chceme nalézt nejkratší cestu mezi každou dvojicí vrcholů.

Nejkratší cesta splňuje princip optimality. Ten říká, že každá podcesta nejkratší cesty je také nejkratší cestou mezi svými koncovými vrcholy. Přesněji řečeno, je-li xPy nejkratší cesta z x do y a u,v jsou dva vrcholy na P, tak je i úsek uPv nejkratší cestou mezi u a v. Pokud mezi u a v existuje kratší cesta uQv, tak v původní cestě xPy nahradíme úsek uPv cestou uQv a dostaneme kratší cestu. To je spor.

Podívejme se na to, jak by měl vypadat výstup řešení druhého úkolu. V každém vrcholu v si budeme pamatovat předchůdce $\pi(v)$ na nejkratší cestě do s. Jednoduše řečeno, $\pi(v)$ je směrovka říkající kudy máme vrchol v opustit, abychom se dostali zpět do vrcholu s po nejkratší cestě. Je-li je graf G souvislý, tak z každého vrcholu v existuje cesta do s a každý vrchol kromě s má právě jednoho předchůdce $\pi(v)$ na nejkratší cestě.[2] Hrany $\{v, \pi(v)\}$ tvoří strom orientovaný do kořene s, který nazveme *strom nejkratší cesty.* Cílem hledání nejkratší cesty v grafu je najít strom nejkratší cesty.[3]



Protože ve městech kromě běžných silnic existují i jednosměrky, tak je zcela přirozené hledat nejkratší cestu i v orientovaných grafech. Vzdálenost dvou vrcholů s a t v orientovaném grafu je opět délka nejkratší orientované cesty z s do t.[4] Všechny algoritmy, které si v této kapitole uvedeme, fungují i pro orientované grafy. Není to nic překvapivého, vždyť i neorientovaný graf, ve kterém hledáme cestu, máme reprezentovaný jako orientovaný graf. Každou hranu si pamatujeme jako dvě orientované hrany/šipky vedoucí proti sobě.

1.1 Realizace grafu pomocí provázků a kuliček

Abychom lépe porozuměli tomu, jak vypadá nejkratší cesta z počátečního vrcholu s do ostatních vrcholů, tak si představme následující realizaci grafu. Místo vrcholů vezmeme kuličky a pokud mezi dvěma vrcholy vede hrana, spojíme odpovídající kuličky provázkem takové délky, kolik je ohodnocení hrany.

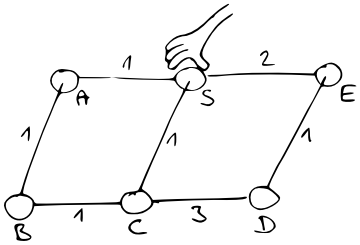
[2] Kdyby existovali dvě stejně dlouhé cesty, tak zvolíme libovolnou z nich.

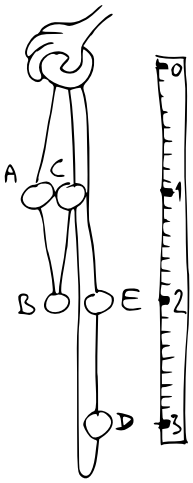
[3] Strom nejkratší cesty je spíše kostra, protože pokrývá všechny vrcholy. Ale pozor, tato kostra nemusí být minimální kostrou grafu. Zkuste najít protipříklad.

[4] Poznamenejme ale, že grafová vzdálenost v orientovaných grafech už nemusí být metrikou, protože nemusí splňovat symetrii.

Tento graf chytneme za kuličku, která odpovídá počátečnímu vrcholu s, a zvedneme ji do výšky. Ostatní kuličky zůstanou viset směrem dolů a to v takových vzdálenostech od s, které odpovídají délce nejkratší cesty.

Ke zjištění nejkratší cesty stačí vzít metr a změřit si vzdálenost s, t.





1.2 Neohodnocený graf

Délka cesty v neohodnoceném grafu je počet hran na cestě. Vzdálenost mezi vrcholy x a y je délka nejkratší cesty mezi x a y. Pokud neexistuje cesta mezi x a y, tak je vzdálenost nekonečná. Takto definovaná vzdálenost odpovídá vzdálenosti v ohodnoceném grafu, kde je každá hrana ohodnocená jedničkou.

Nejkratší cestu můžeme hledat pomocí průchodu do šířky, kterému se někdy říká *algoritmus vlny* (viz sekce **??**). Nazýváme ho tak proto, že vrcholy procházíme ve vlnách, které se šíří z počátečního vrcholu jako vlny na vodní hladině. Jednotlivým vlnám se říká *vrstvy* průchodu do šířky. V *i-té vrstvě* jsou obsaženy právě vrcholy ve vzdálenosti i od počátečního vrcholu s.

V realizaci grafu pomocí kuliček a provázků jednotkové délky budou po zvednutí počátečního vrcholu ostatní kuličky viset přesně po vrstvách, které odpovídají průchodu do šířky.

1.3 Nezáporné ohodnocení hran

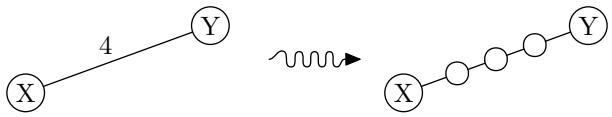
Ve většině aplikací mají hrany jinou než jednotkovou délku, protože délka hrany odpovídá například délce úseku silnice. Předpokládejme, že délky hran jsou nezáporná celá čísla. Jak najít nejkratší cestu teď?

Mohli byste navrhnout, abychom použili BFS úplně stejně jako v neohodnocených grafech. To by ale nefungovalo, protože přímá hrana z s do t může být mnohem delší než cesta vedoucí po dvou krátkých hranách.

Adaptace průchodu do šířky

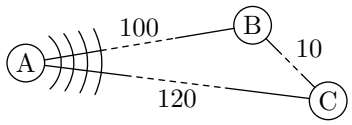
Nemohli bychom BFS upravit tak, aby už fungovalo? Ano to se dá. Zamysleme se nad podstatou BFS. Jednoduchá představa je, že vezmeme miliardu Číňanů, postavíme je do výchozího vrcholu s a necháme je procházet graf. Všichni Číňané chodí stejně rychle a pokud z vrcholu vede více cest, po kterých se mohou vydat, tak se rozdělí.

Průchod hrany délky dva jim bude trvat stejně dlouho jako průchod dvou hran jednotkové délky. Proto můžeme virtuálně každou hranu délky k rozdělit na k jednotkových úseků tím, že na ni přidáme $k-1$ pomocných vrcholů. Jinými slovy hranu délky k nahradíme cestou délky k. Touto úpravou dostaneme graf G'.



Graf G' už má jen hrany jednotkové délky a proto na nalezení nejkratší cesty můžeme použít BFS. Nejkratší cesta mezi původními vrcholy v G' odpovídá nejkratší cestě mezi stejnými vrcholy v G.

U grafů s vysokým ohodnocením hran je sledování průběhu algoritmu docela nudné, protože většinu času uvidíme jen to, jak Číňané postupují skrz hrany. Podívejme se například na graf K_3 s vrcholy A, B, C a s ohodnoceními hran $c(AB) = 100$, $c(AC) = 120$ a $c(BC) = 10$. Při zahájení průchodu z vrcholu A budeme čekat 100 časových jednotek, než Číňané dorazí do dalšího vrcholu. Jelikož dopředu známe délku hrany, po které se Číňané vydali, tak můžeme jít na chvíli spát a nastavit si budík na čas, kdy Číňané dorazí na druhý konec hrany.



Do každého vrcholu umístíme budík, který zazvoní v momentě, kdy do vrcholu dorazí Číňané. Budíky budou na začátku nastaveny na nekonečný čas. Pouze budík v počátečním vrcholu s bude nastaven tak, aby zazvonil okamžitě. Pokud v průběhu algoritmu zjistíme, že do vrcholu dorazíme dříve, než kolik je aktuálně nastavený čas na budíku, tak budík přeřídíme na dřívější čas.

Během algoritmu stačí být vzhůru jen v momentech krátce po té, co zazvoní budík u některého vrcholu. Zbytek času můžeme klidně prospat. Vždy, když nás probudí budík, tak se podíváme do kterých vrcholů dorazili Číňané, na které vstoupili hrany a pokud některá hrana bude zkratkou vedoucí do neprozkoumaného vrcholu, tak přeřídíme odpovídající budík na dřívější čas. Pak už zase můžeme jít spát.

Protože přeřizování budíků probíhá pouze když jsme vzhůru, tak můžeme těsně před usnutím přesně určit, který budík zazvoní jako první.

Zpracování vrcholů v pořadí, jak v nich zvoní budíky, můžeme snadno realizovat použitím prioritní fronty. Na začátku všechny budíky vložíme do prioritní fronty, kde priorita každého budíku je čas zvonění. Postupně budeme budíky z fronty odebírat (v pořadí jak mají zvonit) a zpracovávat události ve vrcholech. Tím jsme právě odvodili Dijkstrův algoritmus.

1.4 Dijkstrův algoritmus

Dijkstrův algoritmus[5] najde nejkratší cestu v orientovaném grafu s nezáporným ohodnocením hran. Už jsme si ho jednou odvodili v předchozí sekci. Nyní si ho ukážeme znova a nezávisle. Zdůrazněme raději ještě jednou, že ohodnocení hran grafu musí být nezáporné, jinak nebudeme moci zaručit správnost algoritmu.

Nejprve si vysvětlíme význam proměnných, které budeme používat. Množina $T \subseteq V$ je množina trvalých vrcholů, to je těch, do kterých známe nejkratší cestu (a do kterých už došli Číňané). Hodnota $\mathtt{d}[v]$ délka nejkratší cesty z s do v vedoucí jen přes trvalé vrcholy a proto je horním odhadem na vzdálenost z s do v (víceméně odpovídá času, na který je nastaven budík). Do pole $\mathtt{odkud}[v]$ si ukládáme předchůdce na nejkratší cestě z s do v (odkud jsme do v přišli po nejkratší cestě).

[5] Čte se „Dajkstrův“ algoritmus. Struktura algoritmu je téměř stejná jako v Jarníkově algoritmu pro hledání minimální kostry.

Začneme s prázdnou množinou T, vzdálenost do startu $\mathtt{d}[s]$ nastavíme na 0 a odhady vzdáleností do ostatních vrcholů na nekonečno. Postupně budeme odebírat netrvalé vrcholy s minimálním $\mathtt{d}[v]$ a prohlašovat je za trvalé. Během prohlašování musíme aktualizovat odhady $\mathtt{d}[w]$ u ostatních vrcholů w, protože jsme se do nich mohli dostat zkratkou z nově prohlášeného trvalého vrcholu v.

$T := \emptyset$
$\mathtt{d}[s] := 0$ a $\forall\, v \in V \setminus \{s\}$: $\mathtt{d}[v] := \infty$
$\forall\, v \in V$: $\mathtt{odkud}[v] := nil$
vytvoř prioritní frontu H z vrcholů V s prioritami $\mathtt{d}[v]$
`while` fronta H neprázdná `do`
  $v :=$ `DELETE_MIN`(H) {vyber netrvalý vrchol s nejmenším $\mathtt{d}[v]$}
  $T := T \cup \{v\}$
  `for each` $vw \in E$ `do`
    `if` $\mathtt{d}[w] > \mathtt{d}[v] + c(vw)$ `then`
      $\mathtt{d}[w] := \mathtt{d}[v] + c(vw)$
      $\mathtt{odkud}[w] := v$
      `DECREASE_KEY`(H, w) {aktualizuj haldu}

O realizaci operací `DELETE_MIN` a `DECREASE_KEY` se dočtete v kapitole **??** o haldě.[6] Algoritmus je konečný, protože v každé iteraci prohlásí jeden vrchol za trvalý. Vrcholů je jen n a z každého vede nejvýše n hran.

Pro každý vrchol x, do kterého neexistuje cesta ze startu s, zůstane po skončení algoritmu $\mathtt{d}[x] = \infty$. Proto můžeme už v průběhu algoritmu testovat, jestli je $\mathtt{d}[v]$ vybraného vrcholu v rovno nekonečnu a případně skončit (k žádným změnám už by stejně nedošlo). Podobně pokud hledáme pouze nejkratší cestu do vrcholu t a ne do všech vrcholů, tak můžeme skončit v momentě, kdy bude t prohlášen za trvalý.

Časová složitost Dijkstrova algoritmu odpovídá času potřebnému na provedení $n \times$ `DELETE_MIN` a $m \times$ `DECREASE_KEY`. Stačí tedy dosadit časové složitosti těchto haldových operací. Prioritní frontu můžeme implementovat v poli – pak dostaneme celkový čas $\mathcal{O}(n^2 + m)$, a nebo pomocí binární haldy – dostaneme celkový čas $\mathcal{O}((m + n) \log n)$. Kterou implementaci si máme vybrat? To záleží, jestli je graf hustý (má hodně hran) a nebo řídký (má málo hran). V každém grafu G je $m < n^2$. Pokud je počet hran $m = \Omega(n^2)$, tak je implementace v poli rychlejší. Když počet hran klesne pod $n^2/\log n$, tak už se vyplatí binární halda.

Můžete se ptát, jak to dopadne, pokud použijeme d-regulární haldu, která pro vhodná d zobecňuje obě předchozí řešení. Ale jak zvolit parametr d? Časová složitost Dijkstrova algoritmu s d-regulární haldou je $\mathcal{O}((nd + m)\frac{\log n}{\log d})$. Po chvilce počítání[7] se ukáže, že nejvýhodnější volba je $d \approx m/n$ (průměrný stupeň grafu). Pro velmi řídké grafy s $m = \mathcal{O}(n)$ dostaneme časovou složitost $\mathcal{O}(n \log n)$, stejně jako u řešení s binární haldou. Pro husté grafy s $m = \Omega(n^2)$ dostaneme lineární[8] časovou složitost $\mathcal{O}(n^2)$, stejně jako u řešení s polem. Pro grafy se střední hustotou $m = n^{1+\delta}$ dostaneme také lineární časovou složitost $\mathcal{O}(m/\delta) = \mathcal{O}(m)$.

Zbývá dokázat správnost Dijkstrova algoritmu.

Invariant: *Pro každý trvalý vrchol v je $\mathtt{d}[v]$ délka nejkratší cesty z s do v.*

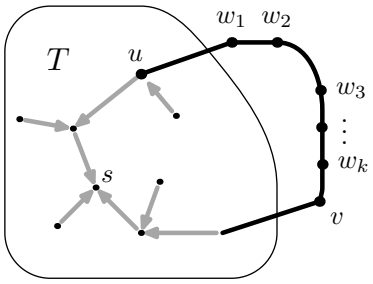
Invariant dokážeme indukcí podle prohlašování vrcholů za trvalé. Pro počáteční vrchol s to platí. Předpokládejme, že v momentě před prohlášením v za trvalý

[6] Pozorný čtenář si mohl všimnout, že jsme změnili parametry u operací pracujících s haldou. V Dijkstrově algoritmu navíc zdůrazňujeme, že pracujeme s haldou H, a přidáme H mezi parametry. Dále u `DECREASE_KEY`(H, w) neuvádíme *okolik* se zmenší klíč $\mathtt{d}[w]$, protože se zmenšení klíče provádí na předchozích řádcích algoritmu.

[7] Tady je vidět, k čemu se hodí matematická analýza – vyšetřování průběhu funkce a hledání minima. Profíci už to ale počítají z hlavy.

[8] Časová složitost je lineární ve velikosti vstupu, tj. velikosti grafu.

vrchol, invariant platí pro všechny vrcholy z množiny T. Ukažme indukční krok tj., že v ten moment je $\mathtt{d}[v]$ délka nejkratší cesty z s do v.

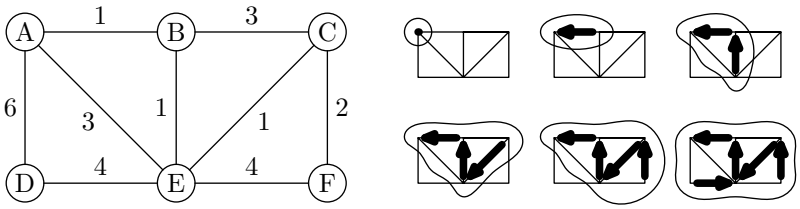


Předpokládejme pro spor, že existuje kratší cesta sPv. Nechť uw_1 je první hrana na cestě P vedoucí z trvalého do netrvalého vrcholu. Taková hrana určitě existuje, protože s je trvalý vrchol, ale v už trvalý není. Z volby vrcholu v jako netrvalého vrcholu s minimálním $\mathtt{d}[v]$ vyplývá $\mathtt{d}[w_1] \geq \mathtt{d}[v]$.

Z indukčního předpokladu je $\mathtt{d}[u]$ délka nejkratší cesty z s do u. Při prohlašování vrcholu u za trvalý jsme zkoušeli snížit hodnotu $\mathtt{d}[w_1]$ na $\mathtt{d}[u] + c(uw_1)$. To se buď povedlo a nebo už byla hodnota $\mathtt{d}[w_1]$ nižší. Proto je $\mathtt{d}[w_1] \leq \mathtt{d}[u] + c(uw_1)$.

Označme další vrcholy na cestě w_1Pv jako $w_2, w_3, \ldots, w_k$. Protože všechny hrany mají nezáporné ohodnocení, tak $c(sPv) = \mathtt{d}[u] + c(uw_1) + c(w_1w_2) + \cdots + c(w_kv) \geq \mathtt{d}[w_1] \geq \mathtt{d}[v]$. A to je spor s tím, že sPv je kratší cesta do v.

Příklad: V následujícím grafu pomocí Dijkstrova algoritmu najdeme nejkratší cestu z vrcholu A do všech ostatních vrcholů. Průběh Dijkstrova algoritmu je zachycen na obrázcích vpravo po řádcích. Množina trvalých vrcholů je zakroužkována. Šipky znázorňují strom nejkratší cesty na trvalých vrcholech.



1.5 Floyd-Warshallův algoritmus

Floyd-Warshallův algoritmus pracuje na orientovaném grafu, který neobsahuje záporné cykly[9] (to nám nezápornost hran bez problémů zaručí), a najde nejkratší orientované cesty mezi každou dvojicí vrcholů. Navíc ze všech cest stejné délky vybere tu s nejmenším počtem hran.

Pokud chceme spočítat vzdálenost každé dvojice vrcholů, tak můžeme n-krát použít Dijkstrův algoritmus (na každý vrchol). Lepší možností je použít Floyd-Warshallův algoritmus, který počítá všechny vzdálenosti přímo, proběhne rychleji než n-krát použitý Dijkstrův algoritmus a ještě se snadněji implementuje. Dokonce je tak jednoduchý, že pokud nám nebude záležet na časové složitosti, ale jen na rychlosti naprogramování, tak je lepší volbou než Dijkstrův algoritmus.

Vrcholy grafu očíslujeme čísly od jedničky do n. Vzdálenosti mezi každou dvojicí vrcholů si budeme ukládat do matice $n \times n$. Celý trik Floyd-Warshallova algoritmu spočívá v tom, že vzdálenosti nepočítáme přímo, ale v n iteracích.

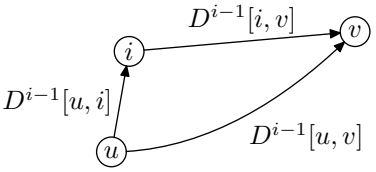
V i-té iteraci spočítáme matici D^i. Hodnota $D^i[u, v]$ je délka nejkratší cesty z u do v, která smí procházet pouze přes vrcholy $\{1, 2, \ldots, i\}$. Jinými slovy $D^i[u, v]$ je délka nejkratší cesty v podgrafu indukovaném vrcholy $\{1, 2, \ldots, i\}$.[10] V nulté iteraci začneme s maticí D^0. Hodnota $D^0[u, v]$ je délka hrany uv, pokud z u vede hrana do v, nula na diagonále a nekonečno jinak. Matice D^0 je tedy *matice vzdáleností* (upravená matice sousednosti, která místo jedniček obsahuje délky hran a místo

[9] Cyklus je záporný, pokud je součet ohodnocení hran podél cyklu záporný.

[10] Podgraf grafu G indukovaný množinou vrcholů $W \subseteq V$ dostaneme tak, že z grafu G vymažeme vrcholy $V \setminus W$ a všechny hrany z nich vedoucí (viz sekce **??** o grafových pojmech).

nul mimo diagonálu nekonečna). V poslední iteraci skončíme s maticí D^n, která už bude obsahovat hledané vzdálenosti, protože cesty mezi u a v smí procházet přes všechny vrcholy.

Pozorování 1 $D^i[u,v] = \min\{\, D^{i-1}[u,v]\,,\; D^{i-1}[u,i] + D^{i-1}[i,v]\,\}$



Nejkratší cesta mezi u a v, která smí procházet pouze přes vrcholy $\{1,2,\ldots,i\}$, buď projde přes vrchol i a nebo ne. Pokud nejkratší cesta neobsahuje i, tak je její délka $D^{i-1}[u,v]$. V opačném případě cestu rozložíme na dva úseky—před příchodem do i a po jeho opuštění. Ani jeden z úseků neobsahuje i a tak je délka této cesty $D^{i-1}[u,i] + D^{i-1}[i,v]$.

Mohli byste namítnout, že nemůžeme délky úseků jen tak sečíst, protože oba úseky mohou obsahovat stejný vrchol w. Složení úseků by nebyla cesta, ale jen tah. V tom případě můžeme část tahu mezi oběma výskyty w vypustit (cyklus z w do i a zpět) a dostaneme kratší tah, který už je cestou. Délka tahu se zkrátila, protože graf neobsahuje záporné cykly. Nově vzniklá cesta už neobsahuje i a byla uvažována v prvním případě.

Pozorování 2 *Hodnoty $D^{i-1}[*,i]$ a $D^{i-1}[i,*]$ se v i-té iteraci nezmění. Nebo-li $D^i[*,i] = D^{i-1}[*,i]$ a $D^i[i,*] = D^{i-1}[i,*]$ (hvězdička značí libovolný index).*

Pozorování plyne z předchozího pozorování a faktu, že $D^j[w,w]$ je rovno nule pro každé j a w. K výpočtu $D^i[u,v]$ potřebujeme znát jen hodnoty $D^{i-1}[u,v]$, $D^{i-1}[u,i]$, $D^{i-1}[i,v]$, ale poslední dvě hodnoty se během i-té iterace nezmění. Proto můžeme nové hodnoty $D^i[u,v]$ zapisovat do stejné matice jako předchozí iteraci. Přepsanou položku $D^{i-1}[u,v]$ už nebude během iterace potřebovat. V celém algoritmu si tedy vystačíme jen s jednou maticí $D[*,*]$, do které budeme zapisovat všechny iterace. Floyd-Warshallův algoritmus vypadá následovně:

```
D := D^0      { matice vzdáleností}
for i = 1 to n do
   for u = 1 to n do
   for v = 1 to n do
      if D[u,v] < D[u,i] + D[i,v] then
         D[u,v] := D[u,i] + D[i,v]
```

Časová složitost Floyd-Warshallova algoritmu je $\mathcal{O}(n^3)$.[11] Pokud si chceme zapamatovat kudy nejkratší cesty vedou, tak si v průběhu algoritmu budeme pro každou dvojici u,v pamatovat nejvyšší číslo vrcholu, přes který nejkratší cesta vede (poslední volbu k, která vedla ke zkrácení cesty). Z toho už se dá nejkratší cesta zrekonstruovat pomocí rekurze.

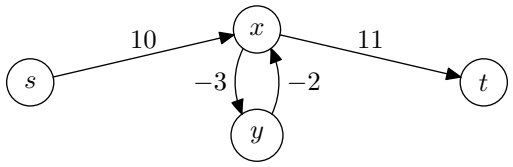
1.6 Obecné ohodnocení hran

V tomto případě už mohou být hrany grafu ohodnoceny čímkoliv, tedy i zápornými čísly. Záporná ohodnocení hran moc neodpovídají délkám hran. Ale ohodnocení si můžeme představit i jako cenu, kterou musíme zaplatit za průchod hranou. Záporná cena znamená, že naopak někdo zaplatí nám.

[11] Protože je struktura algoritmu hodně podobná algoritmu pro násobení matic, můžeme algoritmus urychlit stejným trikem kterým Strassenův algoritmus zrychluje násobení matic. Tím docílíme časové složitosti $\mathcal{O}(n^{\log_2 7}) = \mathcal{O}(n^{2.807})$.

Pokud jsou hrany ohodnoceny i zápornými čísly, tak nemůžeme použít Dijkstrův algoritmus. Je to z toho důvodu, že nejkratší cesta může vést nejdříve do vrcholu, který je dál, a pak se vrátit po záporné hraně.

Ba co hůř, graf může obsahovat záporné cykly. Průchodem po záporném cyklu vyděláme. Je výhodné po něm procházet pořád dokola a postupně snižovat aktuální cenu cesty. Po nekonečně mnoha průchodech ji snížíme až na mínus nekonečno (nekonečně vyděláme). Pokud graf obsahuje záporný cyklus, tak nejlevnější řešení neexistuje. Na druhou stranu nejkratší cesta zcela jistě existuje, protože v konečném grafu je jen konečně mnoho cest. Taková cesta ale nemusí být nejlevnějším řešením naší úlohy. V případě záporných cyklů nemůžeme použít ani Floyd-Warshallův algoritmus, protože by nám mohl vydat nesprávnou odpověď (nenašel by cestu, ale tah, který ani nebude optimální).



1.7 Bellman-Fordův algoritmus

Bellman-Fordův algoritmus najde nejkratší cestu v orientovaném grafu s libovolným ohodnocením hran.

Na chvíli předpokládejme, že graf neobsahuje záporný cyklus. Později si ukážeme, jak rozpoznat, zda ho graf obsahuje.

Budeme postupovat podobně jako v Dijkstrově algoritmu. Pro každý vrchol v si budeme udržovat hodnotu $\mathtt{d}[v]$, která odpovídá délce nějaké cesty vedoucí z s do v (ne nutně té nejkratší). Cena $\mathtt{d}[v]$ je vždy horním odhadem pro cenu nejkratší cesty do v. V průběhu algoritmu budeme tento odhad vylepšovat.

Na začátku u všech vrcholů kromě počátečního vrcholu s nastavíme $\mathtt{d}[v]$ na nekonečno a hodnotu $\mathtt{d}[s]$ na nulu.

Teď si vysvětlíme, co je to update hrany. Pokud víme, že se do vrcholu u umíme dostat za cenu $\mathtt{d}[u]$ a že uv je hrana, tak se umíme dostat do vrcholu v za cenu $\mathtt{d}[u] + c(uv)$ (cestu vedoucí do u prodloužíme o hranu uv). Hrana uv je *korektní,* pokud $\mathtt{d}[v] \leq \mathtt{d}[u] + c(uv)$. V opačném případě je hrana uv *nekorektní* a můžeme zlepšit odhad $\mathtt{d}[v]$. Kontrolu korektnosti hrany uv a případnou opravu zajistí procedura update:

```
procedure update(u,v)
    d[v] := min{ d[v], d[u] + c(u,v) }
```

Důležité vlastnosti updatování hrany jsou:

- Nastaví správnou hodnotu $\mathtt{d}[v]$ v případě, že u je předposlední vrchol na nejkratší cestě do v a hodnota $\mathtt{d}[u]$ už je správně nastavena.
- Nikdy nemůže zmenšit hodnotu $\mathtt{d}[v]$ pod cenu nejkratší cesty do v. V tomto ohledu je použití updatování hrany bezpečné.

Nechť $se_1v_1e_2v_2 \ldots v_{k-1}e_kv_k$ je nejkratší cesta z s do v_k. Podle první vlastnosti updatování hrany postupné zavolání procedury update na hrany e_1, e_2, …, e_k zajistí, že $\mathtt{d}[v]$ bude obsahovat cenu nejkratší cesty z s do v_k. Mezi updatováním jednotlivých hran cesty jsme mohli updatovat ještě i jiné hrany grafu, ale ty výslednou hodnotu $\mathtt{d}[v]$ neovlivní.

Na Dijkstrův algoritmus se můžeme nyní dívat jako na řadu volání procedury update ve správném pořadí. Abychom dostali správný výsledek i v případě záporných hran, musíme používat updatování hran o něco opatrněji.

V jakém pořadí updatovat jednotlivé hrany, abychom updatovali hrany na nejkratší cestě ve správném pořadí? Každá cesta má nejvýše $n-1$ hran. Stačí tedy v $(n-1)$ iteracích updatovat všechny hrany grafu. Jinými slovy posloupnost

$$\underbrace{e_1\, e_2\, \ldots\, e_m\, e_1\, e_2\, \ldots\, e_m\, e_1\, \ldots \qquad\qquad \ldots\, e_m}_{(n-1)\times}$$

má $(n-1)m$ hran a obsahuje každou posloupnost $n-1$ hran jako podposloupnost. Tím pádem updatujeme každou posloupnost $n-1$ hran ve správném pořadí. Celkem se provede $\mathcal{O}(nm)$ updatů. Tomuto algoritmu se říká Bellman-Fordův algoritmus.

$\forall\, v \in V(G)$: $\mathtt{d}[v] := \infty$
$\mathtt{d}[s] := 0$
`for` $i := 1$ `to` $n-1$ `do`
  `for all` $e \in E(G)$ `do`
    update(e)

Poznámka k implementaci: v celé řadě případů obsahuje nejkratší cesta mnohem méně než $n-1$ hran. Proto můžeme skončit po méně jak $n-1$ iteracích. Z toho důvodu se vyplatí v každé iteraci zjišťovat, jestli byla některá hrana nekorektní a tedy jestli vůbec proběhl update nějaké hrany. Pokud ne, tak jsou všechny hrany grafu korektní, algoritmus by v další iteraci také neprovedl žádný update a proto můžeme skončit.

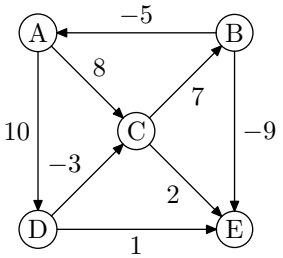
Detekce záporných cyklů

Na začátku jsme předpokládali, že graf neobsahuje záporný cyklus. Ale co když ano? Jak to poznáme? Ukázali jsme si, že když v grafu nebude záporný cyklus, tak Bellman-Fordův algoritmus najde nejkratší cestu a skončí v momentě, kdy jsou všechny hrany korektní.

Pokud graf obsahuje záporný cyklus, tak bude v grafu neustále existovat nekorektní hrana (podél cyklu můžeme odhady na cenu vylepšovat do nekonečna).

Proto stačí použít Bellman-Fordův algoritmus a po jeho skončení otestovat, jestli jsou všechny hrany grafu korektní. Pokud ano, tak algoritmus nalezl nejkratší cestu, pokud ne, tak nejkratší řešení neexistuje.

Příklad: V následujícím ohodnoceném grafu najdeme nejkratší cesty z vrcholu A do všech ostatních vrcholů. Hrany máme zadané v pořadí AC, AD, BA, BE, CB, CE, DC, DE. Průběh Bellman-Fordova algoritmu je částečně zachycen tabulkou, která pro každý vrchol v obsahuje hodnoty $\mathtt{d}[v]$ na konci každé iterace.



Vrchol	Iterace				
	0	1	2	3	4
A	0	0	0	0	0
B	∞	15	14	14	14
C	∞	7	7	7	7
D	∞	10	10	10	10
E	∞	10	6	5	5

1.8 Acyklické orientované grafy

Acyklické orientované grafy nám už z definice zaručují, že nebudou obsahovat záporný cyklus. Neobsahují totiž žádný cyklus. Proto v nich najdeme nejkratší cestu

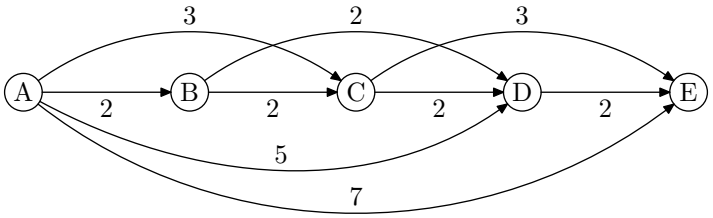
následujícím jednoduchým způsobem:

1. Nalezneme topologické uspořádání.
2. Procházíme vrcholy v nalezeném pořadí a updatujeme hrany z nich vedoucí.

První fázi vyřešíme pomocí průchodu do hloubky. Druhou jednoduchým průchodem vrcholů. Tím pádem zvládneme obě fáze v čase $\mathcal{O}(n+m)$ a celý algoritmus poběží v lineárním čase. Připomeňme, že hrany mohou být ohodnoceny libovolně. Tedy i zápornými čísly.

O Dijkstrově algoritmu se dá říci, že funguje podobně jako hledání nejkratší cesty v acyklických orientovaných grafech, akorát uspořádání vrcholů podle nejkratší cesty hledáme za chodu.

Poznamenejme ještě, že v acyklických orientovaných grafech můžeme podobným způsobem hledat i nejdelší cestu. Takovému algoritmu se říká *algoritmus kritické cesty*. Byl pojmenován kvůli následující aplikaci. Acyklický orientovaný graf popisuje závislosti mezi činnostmi projektu. Hrany odpovídají činnostem a ohodnocení hran době, jak dlouho bude daná činnost probíhat. Doba potřebná k dokončení projektu je délka nejdelší cesty. Nejdelší cestě se říká kritická, protože každé zpoždění činností na kritické cestě způsobí zpoždění celého projektu. **Příklad:** Na následujícím ohodnoceném grafu $G = (V, E)$ najděte nejkratší cestu z A do všech ostatních vrcholů.



Délky nejkratších cest z A do A, B, C, D, E jsou 0, 2, 3, 4, 6. Na papíře můžeme vzdálenosti počítat rovnou. Procházíme vrcholy zleva doprava (v topologickém pořadí) a pro každý vrchol v spočítáme podle hran do něj vedoucích jeho $\mathtt{d}[v] := \min\{\mathtt{d}[w] + c(wv) \mid wv \in E\}$.

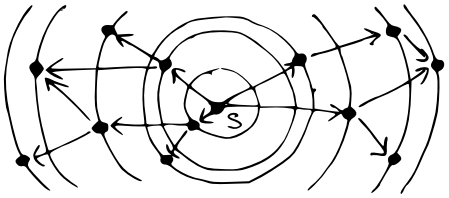
1.9 Potenciál

Ve fyzice je potenciál fyzikální veličina, která popisuje potencionální energii v poli. Například elektrický potenciál popisuje energii v elektrickém poli. Potenciálový rozdíl mezi místy A, B je množství práce, které musíme vykonat, abychom přenesli jednotkový náboj z místa A do místa B.[12] Nezáleží na cestě, kudy náboj přenášíme, ale jen na výchozím a cílovém bodě.

Potenciál a vzdálenost v grafu od počátečního vrcholu s mají hodně společného. Když jdeme po hraně směrem od s, tak vzdálenost roste. Když půjdeme na opačnou stranu, tak klesá. Když vyjdeme z vrcholu v na procházku, tak se vzdálenost bude měnit, ale po návratu do v bude stejná jako na začátku.

Označme délku nejkratší cesty z s do v pomocí y_v. Pro každou hranu vw je $y_v + c(vw) \geq y_w$, jinak můžeme cestu z s do w zkrátit (hrana vw musí být korektní). Tím se necháme inspirovat.

[12] Potenciálový rozdíl v elektrickém poli se nazývá napětí.



Definice: Je dán orientovaný graf $G = (V, E)$ s ohodnocením hran $c(e)$ a s počátečním vrcholem s. Vektor $y = (y_v : v \in V)$ je *přípustný potenciál* pro graf G, ohodnocení c a počátek s, pokud splňuje

$$\begin{aligned} &(i) \quad y_v + c(vw) \geq y_w \quad \text{pro každou hranu } vw \in E \\ &(ii) \quad y_s = 0. \end{aligned}$$

Hladinu nulového potenciálu si můžeme nastavit jak chceme, protože i po přičtením stejné konstanty ke každému y_v zůstane první podmínka platná. Druhá podmínka říká, že si hladinu nulového potenciálu nastavíme tak, aby byl v počátečním vrcholu nulový.[13]

Pro lepší představu o potenciálu si představíme následující gumičkovou realizaci. Místo vrcholů vezmeme kuličky a pokud jsou vrcholy spojeny hranou délky $c(vw)$, tak je spojíme gumičkou, která se natáhne nejvýše do délky $c(vw)$. Při roztažení do větší délky praskne. Vrcholy přišpendlíme na vysoký sloup do takové výšky, kolik je y_v. Potenciál je přípustný, pokud žádná gumička nepraskne ($y_w - y_v \leq c(vw)$) a vrchol s bude umístěn ve výšce 0.

Pozorování 3 *Přípustný potenciál je dolním odhadem na délku nejkratší cesty.*

Pro každou hranu vw platí $c(vw) \geq y_w - y_v$. Posčítáním těchto odhadů podél libovolné cesty $sPv = v_0 e_1 v_1 \ldots e_k v_k$, kde $s = v_0$ a $v = v_k$, dostaneme

$$c(sPv) = \sum_{i=1}^{k} c(e_i) \geq \sum_{i=1}^{k} y_{v_i} - y_{v_{i-1}} = y_{v_k} - y_{v_0} = y_v.$$

Dokonce jsme ukázali víc. Ukázali jsme, že potenciálový rozdíl $y_w - y_v$ je dolním odhadem na délku libovolné cesty z v do w.

Pozorování 4 *Graf má přípustný potenciál právě tehdy, když neobsahuje záporný cyklus.*

První implikace je jednoduchá. Předpokládejme, že graf má přípustný potenciál y a obsahuje záporný cyklus C. Podle předchozího pozorování je potenciálový rozdíl dolním odhadem na délku cesty. Z toho dostaneme odhad na délku cyklu $c(C) = \sum_{e \in C} c(e) \geq 0$. To je spor s tím, že má cyklus zápornou délku.

K důkazu druhé implikace můžeme použít Bellman-Fordův algoritmus. Ten nám najde hodnoty y_v (tj. délky nejkratších cest do všech vrcholů) právě tehdy, když graf neobsahuje záporný cyklus.

Úprava ohodnocení grafu pomocí potenciálu

Máme graf G, u kterého známe přípustný potenciál. Vytvoříme si nové ohodnocení grafu $c' : E \to \mathbb{R}$, které definujeme jako $c'(vw) = c(vw) + y_v - y_w$. Nové ohodnocení c' nezmění nejkratší cesty, protože pro každou cestu uPv platí $c'(uPv) = c(uPv) +$

[13] Zavedení potenciálu je zcela přirozené, protože hledání maximálního přípustného potenciálu je v lineárním programování duálním problémem k hledání nejkratší cesty.

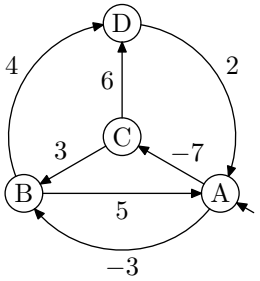
$y_u - y_v$ a potenciálový rozdíl mezi u a v je pro všechny cesty stejný. Pouze se změní ceny nejkratších cest.

Proto můžeme nejkratší cestu hledat i v grafu G s ohodnocením c' a nalezená nejkratší cesta bude stejná jako v grafu s původním ohodnocením.

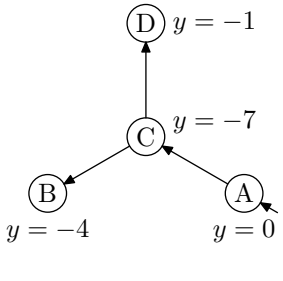
Důležitá poznámka: Pro tuto úvahu jsme používali pouze vlastnost (i) z definice přípustného potenciálu. Hladina nulového potenciálu mohla být nastavena libovolně.

Když známe přípustný potenciál, tak se můžeme zbavit záporných hran tím, že ke hranám přičteme vhodný násobek potenciálových rozdílů konců hran. Na graf s novým ohodnocením, tj. bez záporných hran, už můžeme použít Dijkstrův algoritmus.[14]

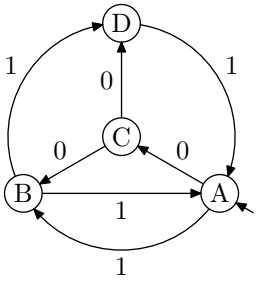
Příklad: V následujícím grafu chceme spočítat nejkratší cestu z A do všech ostatních vrcholů. V grafu se zápornými hranami (levý obrázek) spočítáme přípustný potenciál pomocí Bellman-Fordova algoritmu (vzdálenost vrcholu od A je přípustným potenciálem). Jako výstup dostaneme strom nejkratší cesty a přípustný potenciál (prostřední obrázek). Výše popsaným způsobem upravíme ohodnocení hran a dostaneme graf na pravém obrázku. Graf s novým ohodnocením má stejný strom nejkratší cesty jako graf s původním ohodnocením.



Původní graf se záporným ohodnocením.



Strom nejkratší cesty a přípustný potenciál.



Graf s upraveným ohodnocením bez záporných hran.

Jak najít přípustný potenciál? Můžeme si ho spočítat Bellman-Fordovým algoritmem. To už ale samo obnáší nalezení nejkratší cesty. Zkusme to ještě jinak.

Pokud náš graf odpovídá silniční síti, tak můžeme za přípustný potenciál ve vrcholech zvolit jejich vzdálenost od startu vzdušnou čarou.

A teď otázka pro vás. Můžeme za přípustný potenciál zvolit vzdálenost vzdušnou čarou od libovolného pevného vrcholu? Ano, můžeme. Bod (i) z definice přípustného potenciálu je splněn. Abychom splnili bod (ii), tak musíme nastavit hladinu nulového potenciálu tak, aby byla ve vrcholu s nulová. Toho docílíme přičtením vhodné konstanty (jaké?). Jak jsme si ale ukázali při úpravě ohodnocení grafu pomocí potenciálu, pro nalezení nejkratších cest není hladina nulového potenciálu podstatná. Můžeme si jí zvolit libovolně, jen už pak nemůžeme mluvit o přípustném potenciálu.

V následujících odstavcích si vysvětlíme, proč je nejlepší zvolit za potenciál vzdálenost od cíle t.

Heuristika pro Dijkstrův algoritmus

Dijkstrův algoritmus hledá nejkratší cestu rovnoměrně na všechny strany – prochází všechny vrcholy v pořadí určeném vzdáleností od počátku. Tedy i když hledáme cestu z Prahy do Brna, tak ve „vlně“ kolem 120 kilometrů spočítáme nejkratší

[14] Využití viz Johnsonův algoritmus na straně 17.

cestu do Jihlavy i do Karlových Varů. Přitom nám bude pouhým pohledem do mapy jasné, že je výpočet nejkratší cesty do Karlových Varů zbytečný, protože od Prahy leží na opačné straně než Brno. I kdybychom z Karlových pokračovali do Brna vzdušnou čarou, tak to bude mnohem více kilometrů než po nejkratší cestě z Prahy do Brna. Proto chceme Dijkstrovu algoritmu pomoci tak, aby nejdříve hledal nejkratší cestu do vrcholů správným směrem.

Co kdybychom v Dijkstrově algoritmu počítali místo $\mathtt{d}[v]$ se vzdáleností $\mathtt{d}'[v] := \mathtt{d}[v]$+vzdálenost z v do cíle vzdušnou čarou. Pak bychom nejkratší cestu do Jihlavy spočítali dříve než do Karlových Varů. Vrcholy poblíž cíle by byly zvýhodněny. Ale bude to fungovat? Ano, k $\mathtt{d}[v]$ jsme přičetli něco podobného přípustnému potenciálu, až na nastavení hladiny nulového potenciálu, ale to není pro hledání nejkratších cest potřeba.

Nyní celý algoritmus zopakujeme. Nechť G je graf, který odpovídá silniční síti. Vzdálenost vrcholů od cíle vzdušnou čarou je téměř přípustným potenciálem (až na nastavení hladiny nulového potenciálu). Nejkratší cestu budeme hledat v grafu G s ohodnocením c', které je upraveno podle potenciálu. To nám zajistí, že vlny, ve kterých Dijkstra prohledává vrcholy, budou trochu zdeformovány a protaženy správným směrem.

1.10 Dálniční hierarchie

Představme si, že graf G odpovídá silniční síti celé Evropy a má skoro milión vrcholů. Pokud chceme najít nejkratší cestu z Londýna do Budapešti, tak můžeme použít například Dijkstrův algoritmus. Ale co když se budeme na ptát na nejkratší cestu hodně často? Nemůžeme si něco předpočítat nebo nemůžeme použít nějakou fintu, abychom odpověď nalezli rychleji než Dijkstrovým algoritmem?

Co když chceme realizovat plánovač tras? To je internetový server, kterému zadáte odkud a kam jedete a on vám na mapě najde nejkratší cestu. Dokonce vám i vypíše instrukce, kam máte na které křižovatce odbočit. Takový server dostane během jediné vteřiny stovky dotazů a musí na ně stihnout odpovědět.

Prakticky se ukazuje, že když chcete jen někam hodně daleko, tak na začátku pojedete po lokálních silnicích. Pak najedete na dálnici a pojedete po ní skoro až do cíle.[15] Poblíž cíle sjedete z dálnice a do cíle dojedete po lokálních silnicích.

Nyní využijeme předchozího pozorování o dálnicích a vysvětlíme si hlavní myšlenku dálničních hierarchií. V České republice je mnoho silnic, ale jen málo hraničních přechodů. Pokud budeme přes Českou republiku jen projíždět, tak nemusíme znát všechny silnice, ale stačí znát nejkratší cesty mezi libovolnými hraničními přechody. Ty můžeme mít předpočítané.

Když budeme hledat nejkratší cestu z Londýna do Budapešti, tak ji nebudeme hledat v celé silniční síti Evropy (to je příliš velký graf), ale ve 3 fázích a v mnohem menších grafech. V první fázi najdeme nejkratší cesty z Londýna na hraniční přechody Velké Británie. Ve druhé fázi najdeme nejkratší cesty z hraničních přechodů Velké Británie do hraničních přechodů Maďarska. To hledáme ve zjednodušeném grafu evropské silniční sítě, který jako vrcholy obsahuje pouze hraniční přechody. Ve třetí fázi najdeme nejkratší cesty z hraničních přechodů Maďarska do Budapešti. Z výsledků všech 3 fázích poskládáme acyklický orientovaný graf, který obsahuje pouze Londýn, Budapešť a hraniční přechody Velké Británie a Maďarska. Ten už má jen pár vrcholů a navíc je acyklický, proto v něm nejkratší cestu můžeme najít v lineárním čase. Grafy ve všech fázích obsahují mnohem méně vrcholů, než silniční síť celé Evropy. Díky tomu dosáhneme zrychlení.

[15] Předpokládáme, že v Evropě existuje kvalitní dálniční síť. Pokud ne, tak za dálnici prohlásíme i významnou a hodně frekventovanou silnici.

Hierarchie může mít i více úrovní. Celou Evropu se můžeme rozdělit na oblasti podle států, státy na oblasti podle krajů... Oblasti vůbec nemusí odpovídat právnímu rozdělení. Jde jen o to, aby každá oblast měla málo vstupních míst, tak zvaných portů (obdoba hraničních přechodů). Čím méně jich bude mít, tím více se zjednoduší graf obsahující pouze porty a hrany mezi nimi.

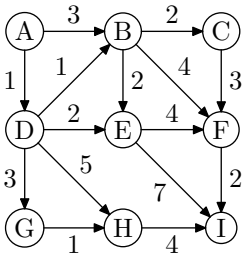
Problém dálničních hierarchií spočívá v tom, jak nalézt ty správné dálnice. Jak najít ty správné oblasti s malým počtem vstupních míst. To už je malinko složitější a je velké množství různých heuristik, které to řeší.[16]

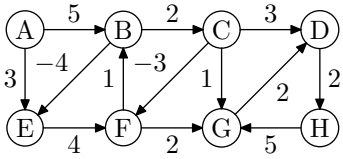
Pro ilustraci časových úspor uvedeme příklad. Na grafu evropské silniční sítě trvá Dijkstrův algoritmus řádově desítky minut. Stejný dotaz řešený pomocí dálničních hierarchií trvá méně jak milisekundu.

1.11 Příklady

1.11.1 Přímé procvičení vyložených algoritmů

1. V následujících grafech najděte nejkratší cestu z vrcholu A do všech ostatních vrcholů. Pokud byste během algoritmu měli na výběr z několika vrcholů, tak si vyberte abecedně menší vrchol. V každém kroku algoritmu (v každé iteraci) vypište hodnoty $\mathtt{d}[v]$ pro všechny vrcholy v. Nakreslete strom nejkratší cesty.





   (a) Použijte Dijkstrův algoritmus.

   (b) Použijte Bellman-Fordův algoritmus.

   (c) Využijte toho, že první graf jde topologicky uspořádat. V prvním grafu nalezněte nejkratší cestu pomocí algoritmu pro acyklické orientované grafy.

2. Máme orientovaný graf, jehož jediné záporně ohodnocené hrany jsou ty vedoucí z počátku. Ostatní hrany mají kladné ohodnocení. Bude v takovém grafu fungovat Dijkstrův algoritmus? Dokažte.

3. Profesor Všezlepšil navrhuje následující algoritmus na hledání nejkratší cesty z s do všech ostatních vrcholů v orientovaných grafech s obecným ohodnocením hran (tedy i záporným): Vezmeme dostatečně velkou konstantu a přičteme ji k ohodnocení každé hrany. Tím získáme nezáporné ohodnocení hran. Potom už můžeme použít Dijkstrův algoritmus. Nalezená nejkratší cesta bude stejná jako nejkratší cesta v grafu s původním ohodnocením.

   Dokažte, že navrhovaná metoda funguje, nebo nalezněte protipříklad.

[16] O silničních sítích je známo, že mají poměrně malou *treewidth (stromovou šířku)*. To si můžeme představit následovně. Pomocí jedné hranice s malým počtem portů rozdělíme graf na dvě poměrně velké části. Každou část můžeme rekurzivně dělit stejným způsobem, dokud nezbudou jen malinkaté části. Nepřesně řečeno, stromová šířka grafu je číslo omezující počet portů na každé hranici takové, aby šel graf ještě rekurzivně rozdělit.

4. (Floyd-Warshall a záporné cykly) Ukázali jsme si, že Floyd-Warshallův algoritmus nalezne nejkratší cestu mezi každou dvojicí vrcholů, pokud graf neobsahuje záporný cyklus. Co by se stalo, kdyby graf obsahoval záporný cyklus?

   Když dostaneme graf, o kterém nevíme, jestli obsahuje záporný cyklus. Nemůžeme pomocí Floyd-Warshallova algoritmu spočítat nejkratší cesty a nebo odpovědět, že graf určitě obsahuje záporný cyklus? Vymyslete, podle čeho to jednoduše poznat.

1.11.2 Varianty problému nejkratší cesty

1. Dostanete graf a ohodnocení jeho hran. Vaším úkolem je nalézt cestu z vrcholu s do vrcholu t takovou, aby největší ohodnocení hrany na cestě bylo co nejmenší. Tj. vzdálenost t od s po cestě sPt počítáme jako $\max_{e \in P}\{c(e)\}$.

2. Dostanete graf jehož hrany jsou ohodnoceny kladnými reálnými čísly. Délka cesty se počítá jako součin ohodnocení hran ležících na cestě, to je $\prod_{e \in sPt} c(e)$.

   (a) Najděte nejkratší cestu z vrcholu s do vrcholu t.
   Nápověda: Zkuste převést násobení na sčítání.

   (b) Profesor Všezlepšil povídá, že hledání nejkratší cesty není nic složitého. Navrhuje použít Dijkstrův algoritmus, kde nahradíme sčítání násobením (výpočet $\mathtt{d}[u] + c(uv)$ nahradíme výpočtem $\mathtt{d}[u] \cdot c(uv)$). Najděte panu profesorovi protipříklad demonstrující, že to tak jednoduše nejde.[17]

3. Hledání nejkratší cesty podle několika kritérií je zcela přirozené. Když jedeme někam autem, tak se tam chceme dostat co nejrychleji, ale také chceme najezdit co nejméně kilometrů, abychom zaplatili co nejméně za benzín.

   (a) Dostanete graf jehož hrany jsou ohodnoceny uspořádanou dvojicí (a, b). Najděte cestu z vrcholu s do vrcholu t takovou, aby byla nejkratší podle ohodnocení a. Pokud by takových cest existovalo více, tak z nich vyberte tu, která je kratší podle ohodnocení b. Délku cesty počítáme jako součet ohodnocení hran na cestě po složkách.

   (b) Co kdybychom neměli hrany ohodnocené jen dvojicí, ale trojicí hodnot? Hledáme cestu, která je nejkratší nejprve podle ohodnocení a, u cest se stejnou vzdáleností podle a vybereme tu s nejmenší vzdáleností podle b a ze všech cest se stejnou nejmenší vzdáleností podle a i b vybereme tu s nejmenší vzdáleností podle c.

4. Celé Norsko je podkopáno ohromným množstvím tunelů. Před každým tunelem je dopravní značka říkající, jak vysoké auto tunelem projede. Firma sídlící ve městě X rozváží celkem velké, ale hlavně vysoké zakázky, a má s tím pěkný problém. Dobře znají silniční síť a pro každou silnici vědí, jak vysoký náklad tudy provezou a jak je silniční úsek dlouhý.

   (a) Firma zná výšku nákladu naloženého na náklaďáku. Nejprve by je zajímalo, jestli vůbec mohou takto vysoký náklad dopravit do města Y. Pokud ano, tak by je zajímala nejkratší cesta ze skladu X do cílového místa Y.

   Samozřejmě mohou jezdit jen tak, aby projeli všemi tunely na nalezené cestě a žádný neponičili.

[17] Student se přihlásí a povídá: „Pane profesore, to lemma, co dokazujete, neplatí. Mám protipříklad.“ Profesor se tím nenechá rozhodit a odpoví: „To nevadí. Mám 2 důkazy.“

(b) Jaký nejvyšší náklad lze dopravit po silnicích z města X do města Y? Pochopte, že čím vyšší zakázku firma vyrobí, tím více vydělá. Na druhou stranu firma musí být schopna dopravit zakázku na místo určení.

5. Jedete na dovolenou do Norska, protože za vyřešení předchozí úlohy vám norská firma zaplatila dovolenou. Víte odkud vyjíždíte, víte kam jedete a znáte silniční síť. O každé silnici víte, jak je dlouhá a jak rychle po ní lze jet. Můžete si tedy spočítat, za jak dlouho ji projedete. Najděte takovou cestu do cílového místa, abyste tam dorazili co nejrychleji plus mínus 15min a za druhé, abyste najeli co nejkratší vzdálenost.

6. Mezi N městy označenými čísly 1 až N jezdí autobusové linky. Pro každou autobusovou linku dostanete údaje odkud a kam jezdí, a cenu jízdného.

   (a) Vypište všechna města, do kterých se lze dostat z města 1 na nejvýše 2 přestupy.

   (b) Zjistěte, jak se co nejlevněji dostat z města 1 do města N. Vypište cenu a návod jak přestupovat. Pokud by existovalo více nejlevnějších cest, tak vypište tu s nejmenším počtem přestupů.

   (c) Navrhněte program pro autobusovou informační kancelář. Do kanceláře volají jednotliví cestující a ptají se, jak se nejlevněji dostanou z města X do města Y. Chtěli bychom jim co nejrychleji odpovědět. Protože jsme slušná informační kancelář, tak v případě, kdy existuje více nejlevnějších cest, chceme zákazníkovi předložit tu s nejmenším počtem přestupů.

   (Jiná formulace problému je, že dostane d dotazů a máte na ně co nejrychleji odpovědět.)

7. (Nejspolehlivější cesta) Máme rozlehlou počítačovou síť, která je realizována rádiovým spojením. Rádiové spojení může být rušeno jiným vysíláním a tudíž není moc spolehlivé. Síť si reprezentujeme jako orientovaný graf $G = (V, E)$. Ke každé hraně $e \in E$ dostanete ještě číslo $0 \leq p(e) \leq 1$, které můžeme interpretovat jako spolehlivost hrany. Číslo $p(e)$ je pravděpodobnost, že informace poslaná z vrcholu x pro hraně $e = xy$ dorazí do y v pořádku (nedojde k chybě). Pokud budeme informace posílat po dvou zřetězených hranách $e, f \in E$, kde $e = xy$ a $f = yz$, tak je pravděpodobnost, že informace vyslané z x dorazí do z v pořádku, rovna $p(e) \cdot p(f)$. Pravděpodobnosti na cestě se násobí. Najděte v zadané síti nejspolehlivější cestu z vrcholu s do vrcholu t. Nejspolehlivější cesta je ta s nejmenší pravděpodobností chyby.

8. Vrcholy grafu $G = (V, E)$ reprezentují města a hrany reprezentují silnice jednoho podivného království. Silnice vedou vždy z města do města a jinde se nekříží. Pro každou silnici znáte její délku v kilometrech. Máte auto, které má nádrž na L litrů benzínu. Všechna auta v tomto království mají stejnou spotřebu σ litrů na 100km (okamžitá spotřeba je konstantní, ať jezdíte jak jezdíte). Takže s plnou nádrží ujedete nejvýše K kilometrů. Pak vám dojde benzín. Dopravu v tomto království komplikuje to, že jsou benzínové stanice pouze ve městech. Na silnicích žádné nejsou. Proto se při svých cestách musíte omezit na silnice kratší než Kkm.

   (a) Navrhněte lineární algoritmus, který zjistí, jestli se svým autem můžete dojet z města s do města t.

   (b) Bohužel jste zjistili, že se svým starým autem moc daleko nedojedete. Chcete si proto koupit nové auto. Jaká musí být jeho minimální velikost nádrže, abyste byli schopni dojet z s do t? Zkuste vymyslet algoritmus pracující v čase $\mathcal{O}((n + m) \log n)$.

1.11.3 Další algoritmy a speciální případy

1. Dostanete graf jehož hrany jsou ohodnoceny čísly 1, 2, ..., k. Najděte co nejrychleji nejkratší cestu v tomto grafu.

   (a) Dokážete to v čase $\mathcal{O}(n + km)$? Pro začátek můžete zkusit přemýšlet o případu, kde $k = 2$.

   (b) A zvládnete najít řešení pracující v čase $\mathcal{O}((n + m) \log k)$?
   Nápověda: Kolik různých odhadů vzdálenosti $\mathtt{d}[v]$ může být v Dijkstrově algoritmu mezi netrvalými vrcholy?

2. Dostanete orientovaný graf s obecným ohodnocením hran (tedy i se záporným). Máte zaručeno, že nejkratší cesta mezi libovolnou dvojicí vrcholů obsahuje nejvýše k hran. Navrhněte algoritmus, který pro dva vrcholy u, v najde nejkratší cestu z u do v v čase $\mathcal{O}(km)$.

   Nápověda: Bellman-Ford.

3. (Yenovo vylepšení Bellman-Fordova algoritmu) Nechť v_1, ..., v_n je nějaké uspořádání vrcholů V takové, že $v_1 = s$. Hrany E rozdělíme do dvou skupin E_1 a E_2, kde $E_1 = \{v_i v_j \,|\, i < j\}$ a $E_2 = \{v_i v_j \,|\, i > j\}$ (hrany po směru a proti směru uspořádání). Ani jedna skupina z E_1, E_2 nemůže obsahovat orientovaný cyklus (rozmyslete si proč). Dá se tedy říci, že rozdělení hran do skupin odpovídá rozdělení grafu na dva acyklické podgrafy G_1, G_2.

   Nyní uspořádáme E_1 do posloupnosti $\mathcal{S}_1$ tak, aby hrana $v_i v_j$ předcházela hraně $v_k v_\ell$ pokud $i < k$. Podobně uspořádáme E_2 do posloupnosti $\mathcal{S}_2$ tak, aby hrana $v_i v_j$ předcházela hraně $v_k v_\ell$ pokud $i > k$. Hrany E_1 jsou v posloupnosti $\mathcal{S}_1$ v takovém pořadí, abychom postupným voláním procedury `update()` na hrany z $\mathcal{S}_1$ našli v podgrafu G_1 nejkratší cestu vedoucí z s do všech dostupných vrcholů.

   Nejkratší cesta v G ale může využívat zkratek přes hrany z E_2. Proto pro nalezení nejkratší cesty v celém grafu G použijeme Bellman-Fordův algoritmus s posloupností hran $\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_1, \mathcal{S}_2, \ldots$ Co můžeme říci o potřebném počtu iterací? Jaká bude časová složitost vylepšeného algoritmu?

4. (Johnsonův algoritmus; využití přípustného potenciálu) Dostaneme orientovaný graf, jehož hrany jsou ohodnoceny reálnými čísly (i zápornými). Chtěli bychom pro každou dvojici vrcholů (x, y) najít nejkratší cestu z x do y. Dopředu nevíme, jestli graf obsahuje záporný cyklus.

   Mohli bychom n krát použít Bellman-Fordův algoritmus (celkový čas $\mathcal{O}(n^2 m)$). Nebo bychom mohli pomocí Bellman-Fordova algoritmu zjistit, jestli graf obsahuje záporný cyklus a pak použít Floyd-Warshallův algoritmus (celkový čas $\mathcal{O}(n^3)$). Třetí možností je následující řešení:

   (a) Pomocí Bellman-Fordova algoritmu najdeme nejkratší cestu z libovolného vrcholu do všech ostatních. Při tom spočítáme přípustný potenciál (nebo odpovíme, že neexistuje a skončíme).

   (b) Pomocí přípustného potenciálu upravíme ohodnocení hran tak, aby bylo všude nezáporné. (Podívejte se na příklad na straně 12.)

   (c) Nakonec použijeme $(n-1)$ krát Dijkstrův algoritmus na graf s upraveným ohodnocením.

   Rozmyslete si detaily, zdůvodněte správnost algoritmu a určete jeho celkovou časovou složitost.

5. (Gabow's scaling algorithm) Scaling algorithm se do češtiny překládá jako algoritmus měnící měřítko. Algoritmus se dá použít pouze pro grafy, jejichž hrany jsou ohodnoceny celým číslem (integerem). V první iteraci algoritmus zváží pouze nejvyšší bit na každé hraně a vyřeší tento zjednodušený problém (ohodnocení hran jsou jednobitové). Ve druhé iteraci přidá další bit. Za pomoci výsledků z předchozí fáze vyřeší zjednodušený problém, kde u každé hrany zvažuje pouze 2 nejvyšší bity. V každé další iteraci přidá další bit z ohodnocení na hranách, až se nakonec dostane ke všem bitům a tím spočítá řešení původní úlohy.

   Dostaneme orientovaný graf $G = (V, E)$ s nezáporným ohodnocením hran $w(e)$. Chceme najít délky nejkratších cest z počátečního vrcholu s do všech ostatních vrcholů v. Nechť W označuje největší hodnotu, kterou je ohodnocena nějaká hrana. Chceme vymyslet algoritmus, který bude pracovat v čase $\mathcal{O}(|E| \cdot \log W)$.

   Algoritmus začne s grafem G, který má hrany ohodnoceny pouze nejvyšším bitem z původních ohodnocení. Postupně v dalších iteracích přidává další bity. Konkrétně, nechť $k := \lceil \log(W+1) \rceil$ je počet bitů v binární reprezentaci každé hodnoty na hraně. V každé iteraci pro $i = 1, 2, \ldots, k$ algoritmus uvažuje ohodnocení hran $w_i(e) := \lfloor w(e)/2^{k-i} \rfloor$ pro každou $e \in E$. Hodnota $w_i(e)$ je původní hodnota $w(e)$, která má snížené měřítko a obsahuje pouze nejvyšších i bitů. Proto je $w_k(e) = w(e)$ pro každou hranu e. Například pokud $k = 5$ a $w(e) = 25$, což je v binární reprezentaci $[11001]_2$, tak $w_3(e) = [110]_2 = 6$.

   Definujme $\delta_i(u, v)$ jako délku nejkratší cesty z u do v v grafu G s ohodnocením w_i. Délka nejkratší cesty v G s ohodnocením w $\delta(u, v) = \delta_k(u, v)$ pro všechny $u, v \in V$. Pro zadaný počáteční vrchol s algoritmus nejprve spočítá $\delta_1(s, v)$ pro všechny $v \in V$. V dalších iteracích spočítá $\delta_2(s, v)$ pro všechny $v \in V$, dále $\delta_3(s, v)$ pro všechny $v \in V$,..., až se v poslední iteraci spočte $\delta_k(s, v)$ pro všechny $v \in V$. Celou dobu budeme předpokládat, že $|E| \geq |V| - 1$. Pokud bude výpočet δ_i z δ_{i-1} trvat čas $\mathcal{O}(|E|)$, tak celý algoritmus poběží v čase $\mathcal{O}(|E| \cdot \log W)$.

   Nyní si odvodíte, proč a jak algoritmus funguje. Pomohou vám úkoly a nápovědy v následujích bodech.

   (a) Předpokládejte, že $\delta(s, v) \leq |E|$ pro všechny $v \in V$. Ukažte, jak se dá spočítat $\delta(s, v)$ pro všechny $v \in V$ v čase $\mathcal{O}(|E|)$.

   (b) Ukažte, jak spočítat $\delta_1(s, v)$ pro všechny $v \in V$ v čase $\mathcal{O}(|E|)$. Tato iterace odpovídá hledání nejkratších cest v grafu bez ohodnocení hran.

   Nyní se zaměříme na výpočet δ_i z δ_{i-1}.

   (c) Dokažte, že pro $i = 1, 2, \ldots, k$, buď $w_i(e) = 2w_{i-1}(e)$ nebo $w_i(e) = 2w_{i-1}(e) + 1$. Potom dokažte, že pro všechna $v \in V$

   $$2\delta_{i-1}(s, v) \leq \delta_i(s, v) \leq 2\delta_{i-1}(s, v) + |V| - 1.$$

   (d) Pro všechna $i = 2, 3, \ldots, k$ a všechny hrany $uv \in E$ definujme

   $$\widehat{w}_i(uv) = w_i(uv) + 2\delta_{i-1}(s, u) - 2\delta_{i-1}(s, v).$$

   Všimněte si, jak je nové ohodnocení hran podobné úpravě ohodnocení podle přípustného potenciálu (viz sekce 1.9).

   Pro všechna $i = 2, 3, \ldots, k$ a všechny hrany $uv \in E$ dokažte, že upravená hodnota hrany $\widehat{w}_i(uv)$ je celočíselná a nezáporná.

(e) Teď definujme $\widehat{\delta}_i(s,v)$ jako délku nejkratší cesty z s do v v grafu G s ohodnocením $\widehat{w}_i$. Pro všechna $i = 2, 3, \ldots, k$ a všechny hrany $uv \in E$ dokažte, že

$$\delta_i(s,v) = \widehat{\delta}_i(s,v) + 2\delta_{i-1}(s,v).$$

Na základě toho dokažte, že $\widehat{\delta}_i(s,v) \leq |V| - 1 \leq |E|$.

(f) Nyní konečně ukažte, jak spočítat $\delta_i(s,v)$ z hodnoty $\delta_{i-1}(s,v)$ pro všechny $v \in V$ v čase $\mathcal{O}(|E|)$. Z toho vyvoďte, jak spočítat $\delta(s,v)$ pro všechny $v \in V$ v čase $\mathcal{O}(|E| \log W)$.

1.11.4 Úlohy na úpravu grafu

Následující úlohy můžeme snadno převést na obyčejný problém hledání nejkratší cesty. Stačí vhodně upravit graf a ohodnocení hran.

1. V Absurdistánu měli N měst spojených navzájem leteckými linkami S různých dopravních společností. Ale jak již to v tomto státě bývá, byrokracie se rozmohla a tak každá dopravní společnost měla různé tarify za překlad nákladu z letadel různých jiných společností. Přinesli Vám následující tabulky:

   - $p(s,i,j)$ - cena účtovaná společností s za přepravu vašeho nákladu z města i do města j. Nekonečno, pokud taková letecká linka neexistuje.
   - $m(i,r,s)$ - cena účtovaná za přeložení nákladu z letadla společnosti r do letadla společnosti s v městě i. Nekonečno, pokud tomu úřední předpisy brání.

   Máte vyřešit, jak svůj náklad co nejlevněji přepravit z města 1 do města N.

2. (Nejkratší cesta mezi množinami A a B) Dostaneme orientovaný graf $G = (V,E)$, ohodnocení $c \in \mathbb{R}^m$ a dvě disjunktní množiny $A, B \subseteq V$. Najděte nejkratší cestu, která začíná ve vrcholu množiny A a končí ve vrcholu množiny B.

   Nápověda: Zkuste problém převést na běžný problém nejkratší cesty.

3. (Nejkratší cesta s lichým počtem hran) Dostanete orientovaný graf s nezáporným ohodnocením hran. Chceme najít nejkratší cestu z vrcholu s do t, která obsahuje lichý počet hran. Až to vyřešíte, tak si rozmyslete, co by se změnilo, kdybychom chtěli najít nejkratší cestu obsahující sudý počet hran?

   Nápověda: nahraďte každý vrchol kromě s a t dvojicí vrcholů.

4. (Zobecněný problém nejkratší cesty) Při internetovém routování dochází k prodlevám na jednotlivých linkách, ale také v samotných routrech. To nás motivuje k zobecnění problému nejkratší cesty.

   Kromě cen na hranách (ohodnocení hran) máme i ceny ve vrcholech (ohodnocení vrcholů). Délka cesty v tomto novém modelu bude součet cen na hranách cesty plus součet cen ve vrcholech cesty (včetně koncových vrcholů).

   Dostanete orientovaný graf $G = (V,E)$ s kladným ohodnocením hran $c(e)$ a kladným ohodnocením vrcholů $c(v)$. Dále dostanete počáteční vrchol $s \in V$. Navrhněte efektivní algoritmus, který pro všechny vrcholy $v \in V$ vypíše cenu nejlevnější cesty z s do v.

   Poznámka: Cena nejlevnější cesty z s do s je $c(s)$.

1.11.5 Ostatní úlohy

1. (Počet nejkratších cest) Dostaneme neorientovaný neohodnocený graf a dva jeho vrcholy u, v. Mezi vrcholy u a v může existovat i více nejkratších cest. Navrhněte lineární algoritmus, který vypíše počet nejkratších cest mezi u a v.

2. (Jednoznačnost nejkratší cesty) Dostanete orientovaný graf s kladným ohodnocením hran a počáteční vrchol s. Potřebovali bychom vyplnit boolovské pole `unique`$[\,\cdot\,]$, kde `unique`$[w]$ = `true` právě tehdy když je nejkratší cesta z s do w jednoznačná (unikátní).

3. (Ověření korektnosti řešení) Dostanete orientovaný graf $G = (V, E)$, ohodnocení jeho hran c (může být i záporné), počáteční vrchol s a strom T, o kterém Vám někdo tvrdí, že je stromem nejkratší cesty. Navrhněte algoritmus, který ověří, že strom T je opravdu stromem nejkratší cesty pro graf G s ohodnocením c a počátečním vrcholem s. Váš algoritmus by měl běžet v lineárním čase.

4. (Hledání čtverců) Navrhněte algoritmus, který dostane neorientovaný graf a rozhodne, jestli graf obsahuje kružnici délky čtyři (čtverec). Časová složitost Vašeho algoritmu by neměla překročit $\mathcal{O}(n^3)$.

5. (Algoritmus na délku nejkratší kružnice) Podívejte se na návrh algoritmu, který najde délku nejkratší kružnice v grafu bez ohodnocení hran.

   Graf budeme procházet pomocí průchodu do hloubky (DFS). Každá zpětná hrana uv, na kterou narazíme, musí spolu se stromovými hranami vedoucími z v do u tvořit kružnici. Délka této kružnice je $level(u) - level(v) + 1$, kde $level(w)$ je vzdálenost vrcholu w od kořene ve stromě průchodu do hloubky. Tohle pozorování nás přivádí k následujícímu algoritmu. Budeme graf procházet do hloubky a u každého vrcholu si budeme pamatovat jeho $level$. Vždy když narazíme na zpětnou hranu, tak si spočítáme délku kružnice a porovnáme ji s dosud nejmenší nalezenou délkou kružnice.

   Ukažte, že tento algoritmus nemusí vždy fungovat. Nalezněte protipříklad a vysvětlete, proč je protipříkladem.

6. (Délka nejkratšího cyklu)

   (a) Dostanete obyčejný graf bez ohodnocení hran. Nalezněte algoritmus, který najde délku nejkratší kružnice v grafu nebo odpoví, že graf žádnou kružnici neobsahuje. Vaše řešení by mělo běžet v čase nejvýše $\mathcal{O}(nm)$.

   (b) Dostanete orientovaný graf s kladným ohodnocením hran. Nalezněte algoritmus, který spočítá délku nejkratšího cyklu. Pokud je graf acyklický, tak by to měl algoritmus zjistit. Vaše řešení by mělo běžet v čase nejvýše $\mathcal{O}(n^3)$.

7. (Délka nejkratšího cyklu obsahujícího hranu e) Dostanete neorientovaný graf $G = (V, E)$ s délkami hran ℓ_e pro každou hranu $e \in E$. Dále dostanete jednu konkrétní hranu $f \in E$. Najděte v čase $\mathcal{O}(n^2)$ nejkratší kružnici obsahující hranu f.

8. (Hledání nejzápornějšího cyklu) Dostanete graf jehož hrany jsou ohodnoceny reálnými čísly, a to i zápornými. Najděte v tomto grafu záporný cyklus s co nejmenší hodnotou.

9. (Arbitráž) Proč hledat záporné cykly? Odpověď je jednoduchá. Abychom vydělali. Na světě funguje spousta směnáren. Kurz jedné měny vůči měně druhé

nám říká, kolik jednotek jedné měny dostaneme za měnu druhou. Například kurz $k_{Eur/Kc}$ udává, kolik Euro dostaneme za 1 korunu (kurzy jsou znormované). Pokud chceme vyměnit koruny za Eura a pak na výletě v Německu Eura za Dolary, tak se kurzy násobí. Počet dolarů, které dostaneme výměnou za 100 Kč, spočítáme jako $k_{\$/Eur} \cdot k_{Eur/Kc} \cdot 100$. Předpokládáme, že výměna ve směnárnách probíhá bez poplatků. Nakonec si můžeme v Americe vyměnit dolary zpátky na české koruny. Množství korun, které tak získáme, záleží na kurzech měn.

Jak provádět výměny v jednotlivých směnárnách, abychom vydělali? Pokud je součin kurzů podél cyklické výměny číslo větší než jedna, tak vyděláme. Pokud je to číslo menší než jedna, tak naopak proděláme.

Různé světové měny si můžete představit jako vrcholy grafu a kurzy měn jako ohodnocení hran. Navrhněte co nejrychlejší algoritmus, který v tomto grafu najde cykly, podél kterých se vyplatí vyměňovat peníze tak, abychom vydělali.

Nápověda: Stačí převést násobení hodnot na hranách cesty na jejich sčítání. Toho docílíme tak, že si vytvoříme kopii grafu, ve které hrany ohodnotíme mínus logaritmem původního ohodnocení. V novém grafu budeme hledat záporné cykly. Proč při záporném cyklu vyděláme a při kladném ne?

10. (Nejkratší cesty vedoucí přes vrchol v_0) Dostanete silně souvislý orientovaný graf $G = (V, E)$ s kladným ohodnocením hran. Graf je *silně souvislý* právě tehdy když z každého vrcholu existuje orientovaná cesta do libovolného jiného vrcholu. Dále dostanete jeden vrchol $v_0 \in V$. Navrhněte efektivní algoritmus, který najde nejkratší cesty mezi každou dvojicí vrcholů, ale s tou podmínkou, že každá cesta musí procházet přes vrchol v_0.

11. (Přidání hrany, která nejvíce zkrátí nejkratší cesty) Je dána silniční síť $G = (V, E)$ propojující města V. Pro každou silnici $e \in E$ známe její délku ℓ_e. Ministerstvo dopravy chce rozšířit silniční síť o jednu novou silnici. Zatím má dlouhý seznam dvojic měst E', které jsou kandidáty na propojení. Každá potenciální silnice $e' \in E'$ už je vyprojektovaná a víme, že bude mít délku $\ell_{e'}$. Vás poprosili, abyste jim pomohli vybrat tu nejvhodnější z uvažovaných silnic. Ministři nejčastěji jezdí z města s do města t a proto Vám položili následují otázku: Stavba které silnice povede k největšímu snížení vzdálenosti mezi městy s a t? Navrhněte efektivní algoritmus pro řešení této otázky.