Korespondenční Seminář z Programování

33. ročník **KSP** **Červen 2021**

Komentáře k páté sérii třicátého třetího ročníku KSP

33-5-4 Nevěřící Bob

Ukážeme ještě jednu konstrukci důkazu k nejkratší cestě. Za inspiraci děkujeme Honzovi Kotovskému, Jirkovi Kvapilovi a Kristýně Petrlíkové.

Jelikož délky hran jsou nezáporné, můžeme vzdálenosti počítat Dijkstrovým algoritmem. V něm nás ale brzdí výběr minima, takže i s tou nejlepší Fibonacciho haldou poběží v čase $\Theta(m + n \log n)$.

My si proto v důkazu necháme napovědět, v jakém pořadí Dijkstrův algoritmus zavírá vrcholy. Pak stačí kontrolovat, že jsme každý vrchol zavřeli právě jednou, že spočítané vzdálenosti vrcholů v pořadí zavírání tvoří neklesající posloupnost a že zavřeným vrcholům se vzdálenost už nemění.

Jelikož jediné, co na Dijkstrově algoritmu není lineární, je práce s haldou, dokážeme pomocí důkazu Dijkstru odsimulovat v lineárním čase.

Nakonec dodejme, že v zadáni zmíněné ohodnocení hran *přirozenými čísly* mělo dovolovat i nulové hrany. Leč moc snadno se zapomíná na to, že středoškolská „definice“ čísel nepovažuje nulu za přirozené číslo – na rozdíl od značné části matematiků a informatiků (pozor na to!). Proto jsme vám odpouštěli, když vaše důkazové systémy selhávaly na nulových hranách, a zejména cyklech ze samých nulových hran.

33-5-X1

Co s tím?

Jako u každé úlohy začneme tím, že se podíváme na zadání. Ouha, ale tady žádné zadání není! Je tu jen odkaz na odevzdávátko, tak nemáme jinou možnost než se tam podívat. Stáhneme si zadání prvního vstupu a získáme:

```
A:BCF
B:AGC
C:ABD
D:FCIE
E:FDIH
F:ADE
G:B
H:E
I:DE
```

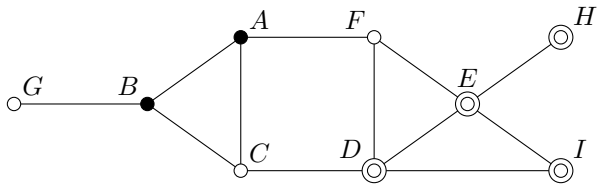
V případě, že stále nevíme, co máme dělat, tak se můžeme pokusit odevzdat třeba `42`. Ale ne, to nefunguje! Odevzdávátko nám ovšem poskytnulo hlášku `Označení '4' není validní identifikátor vrcholu.` Víme tedy, že řešíme grafovou úlohu.

Co ale jsou tedy validní označení vrcholů, když čísla nefungují? Po chvíli testování zjistíme, že validní označení jsou `A` až `I`. Při tom také dostáváme hlášky typu `Vrcholy D E G H I nejsou pokryté.`

Jak ale pokrýt všechny vrcholy? Když napíšeme `A B C D E F G H I`, tak dostaneme pořád stejnou hlášku. Evidentně se tedy část vstupu za mezerou vůbec nevyhodnocuje. Vypíšeme vrcholy bez mezery. Tentokrát už dostaneme o trošku více bodů doprovázených hláškou: `Vybral jsi příliš mnoho vrcholů.` Víme tedy, že řešíme grafovou úlohu na vrcholech `A` až `I`. Máme vybrat co nejméně vrcholů tak aby bylo něco nějak pokryté.

Pojďme se podrobněji podívat na vstup. Pro každý vrchol je tu jeden řádek. Na něm je seznam dalších vrcholů. Z toho tedy můžeme usoudit, že se jedná o sousedy daného vrcholu. Pojďme si tedy tento graf nakreslit. Při tom si všimneme, že se jedná o neorientovaný graf.

Když vybereme vrcholy `AB`, tak nám odevzdávátko odpoví, že nepokryté vrcholy jsou `DEHI`:



Z obrázku si snadno všimneme, že pokryté vrcholy jsou ty, co mají od vybraných vrcholů vzdálenost ≤ 1. Tím už tedy víme, co po nás zadání chce. Snadno tedy najdeme řešení `BE`.

Po odevzdání ale stále nemáme celý bod. Odevzdávátko se nám nicméně snaží říct, že úloha ještě pokračuje. Pokusíme se tedy za `BE` připsat třeba `42`. Hláška se sice nezmění, ale už máme o 0.05 bodů více. Připíšeme tedy ještě třeba `13`. To se ale odevzdávátku moc nelíbí: `Řetězec '13' není validní označení vrcholu.` Napíšeme tedy místo toho třeba `A`. Ovšem `Vrchol A nesousedí se startem.` Pomocí dalších submitů zjistíme, že se startem sousedí pouze vrcholy `BCF`. Startem tedy musí být `A`.

Když odevzdávátku předhodíme vrchol sousedící se startem (např. `B`), tak nám řekne: `Skončil jsi ve vrcholu B. Možná tvůj vstup končí moc brzy.` Dopíšeme tedy nějaký vrchol sousedící s `B`. To ale nefunguje: `Řetězec 'C' obsahuje nepovolený znak. Povolené znaky jsou jen '+' a '-'.`

Zkusíme tedy `+`. Tím získáme: `Skončil jsi ve vrcholu G. Možná tvůj vstup končí moc brzy.` Při odevzdání `-` se dostaneme do vrcholu `C`. Všimneme si, že se jedná o sousední vrcholy námi označeného vrcholu `B`. Nejspíše tedy popisujeme nějakými instrukcemi pohyb po grafu. Jelikož původní hláska si stěžovala na nepovolený znak v *řetězci*, tak vyzkoušíme třeba i `+-`. Zde také ale moc nepochodíme: `Kombinace + a - v jednom řetězci není dovolená.` Ale i když odevzdáme třeba `++` nebo `--`, tak na tom nejsme o moc lépe: `Řetězec ++ ve vrcholu B není zapsaný nejkratší možností.`

Odevzdávátko nám ale tvrdí, že vstup končí moc brzy, co tedy třeba napsat `B + +`? `Instrukce + ve vrcholu G směřuje zpět. Otáčet se do protisměru je ale zakázané.` No jo, z vrcholu `G` ale přeci jiná hrana než zpět není. Tudy cesta tedy asi nevede.

Pojďme tedy přes vrchol `C` pomocí instrukcí `B - -`. Tím se dostaneme do vrcholu `A`. Při odevzdání `B - +` jsme tedy ve

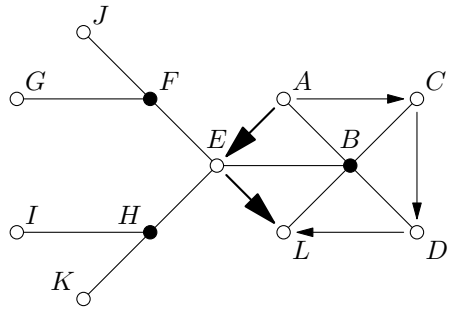
vrcholu `D`. Každým slovem tedy vybíráme jeden ze sousedních vrcholů aktuálního vrcholu. Ale podle jakého pravidla? Všimneme si, že vstup nám dává ještě jedu informaci, kterou jsme zatím nevyužili. U každého vrcholu vyjmenovává sousedy v určitém pořadí. Všimneme si tedy, že znak `+` nás posouvá v tomto seznamu směrem doprava od vrcholu, kudy jsme přišli a znak `-` nás posouvá směrem doleva. Seznam uvažujeme jako cyklický. Jak se ale dostat do vrcholu, co nesousedí s tím, kudy jsme přišli? K tomu už využijeme několik znamének za sebou. Například z vrcholu `D`, při příchodu z `C`, se dostaneme do vrcholu `E` pomocí `++`, a nebo dokonce i `--`.

Dále také odhadneme, že se snažíme dostat do posledního vrcholu na vstupu, tedy `I`. Možná cesta tedy je `F - ++`. Bodů už sice máme již o další trošku více, takže jsme cíl nejspíš našli, ale i tak nemáme vyhráno: `Prošel jsi zakázaným vrcholem E.` Nesmíme tedy chodit přes vrcholy, které jsme vybrali v první části úkolu. Můžeme využít cestu `F + -`. Nyní si ale odevzdávátko začne stěžovat na číslo 42, které ve výstupu stále odevzdáváme i když nevíme, co vlastně znamená. Nahradíme ho tedy odevzdávátkem napovězenou 4. Dále můžeme pracovat s hypotézou, že se jedná o počet vrcholů na námi popsané cestě.

Podíváme se na druhý vstup a pokusíme se ho vyřešit stejným způsobem.

```
A:EBC
B:AELDC
C:ABD
D:CBL
E:FHLBA
F:JGE
G:F
H:KIE
I:H
J:F
K:H
L:EDB
```

Najdeme tedy pokrytí pomocí tří vrcholů `BFH` a pak i nejkratší trasu z `A` do `L` přes vrchol `E`. Odevzdáme tedy `BHF 3 E --`. Odevzdávátko nám ale tvrdí, že nebezpečnost 3 není dobře a má být 5. Po nahrazení však stále nemáme vyhráno: `Tvoje cesta není nejbezpečnější.`



Co teď s tím? Pojďme zvolit druhou z možných cest, tedy `4 C - -`. Zde již odevzdávátko nebude mít další stížnosti.

Pomocí dalších vstupů pak vykoukáme, že nebezpečnost značí počet vybraných vrcholů, které sousedí s vrcholy po cestě (v případě, že jeden vybraný vrchol bude sousední vícekrát, tak ho počítáme také několikrát).

Algoritmus

Na prvních pár vstupech jsme zjistili, co po nás vlastně zadání chce. Na další vstupy již ovšem nemáme neomezený čas a navíc narůstá počet vrcholů, z nichž se graf skládá. Proto dřív nebo později bude potřeba celý úkol přenechat počítači.

Nejprve můžeme sepsat nějaký „hloupý“ algoritmus. Projdeme všechny podmnožiny vrcholů a pro každou zjistíme, jestli se jedná o validní výběr. Jako výsledek pak vypíšeme tu nejlepší validní možnost.

Jelikož vrcholů je ve vstupu poměrně málo, můžeme pro reprezentaci množin vrcholů použít bitové masky. Tedy každému vrcholu přiřadíme jednu cifru v dvojkovém zápisu čísla. Když je na dané pozici 1, tak je daný vrchol součástí množiny.

Procházet přes všechny podmnožiny lze pak jednoduše – stačí projít všechna čísla od 0 do $2^N - 1$. Pojďme ještě elegantně vyřešit zjišťování, zda se jedná o validní výběr vrcholů. Pro každý vrchol si předpřipravíme bitmasku jeho sousedů včetně vrcholu samotného. Pak stačí probrat všechny vybrané vrcholy a spočítat *bitový or* jejich bitmasek. Jestliže výsledek obsahuje všechny vrcholy (tedy se rovná $2^N - 1$), máme validní výběr.

Najít trasu pak již není problém. Pro každý vrchol si nejprve spočítáme počet vybraných sousedů. A pak na grafu nevybraných vrcholů spustíme Dijkstrův algoritmus, přičemž délka orientované hrany bude odpovídat počtu sousedů cílového vrcholu. Tím získáme i výslednou trasu. Zbývá tedy jen vypisovat výstup ve správném formátu.

Tímto algoritmem vyřešíme několik vstupů, ovšem na dalších narazíme na dva problémy:

- Vstup začne obsahovat kromě velkých i malá písmena. To stačí vyřešit menší úpravou vstupu a výstupu algoritmu.
- Náš algoritmus bude moc pomalý. V časovém limitu se ani zdaleka výsledku nepřiblíží.

Největší brzdou našeho algoritmu je hledání nejmenšího pokrytí – to je exponenciálně pomalé, zatímco nalezení cesty stihneme v čase $\mathcal{O}((M+N)\cdot\log N)$. Jenže nejmenší pokrytí patří mezi známé NP-úplné problémy, takže spíš než polynomiální algoritmus chceme hledat rychlejší exponenciální. O to se pokusíme ve zbytku řešení.

Množiny podle velikosti

Nejmenší pokrytí zatím hledáme velmi primitivně: zkoušíme všechny podmnožiny vrcholů, pro každou z nich otestujeme, jestli pokrývá všechny ostatní vrcholy, a udržujeme si průběžné minimum.

Dá se ovšem tušit, že optimální pokrytí nebude moc velké – například nemůže obsahovat víc než polovinu vrcholů. Proto se vyplatí podmnožiny zkoušet od nejmenších. Pak se můžeme zastavit, jakmile najdeme první podmnožinu, která pokrývá všechny vrcholy.

Podmnožiny budeme opět kódovat posloupnostmi nul a jedniček. Pro každou velikost budeme procházet všechny podmnožiny dané velikosti v lexikografickém pořadí kódů. Například 3-prvkové podmnožiny 5-prvkové množiny uspořádáme takto:

```
00111
01011
01101
01110
10011
10101
10110
```

```
11001
11010
11100
```

Obecně pro k-prvkové podmnožiny n-prvkové množiny bude lexikograficky nejmenší kód $\mathtt{0}^{n-k}\mathtt{1}^k$ a největší $\mathtt{1}^k\mathtt{0}^{n-k}$.

Chceme-li najít lexikografického následníka nějaké podmnožiny, najdeme od konce kódu nejdelší souvislý úsek nul a před ním nejdelší souvislý úsek jedniček. Před jedničkami musí být zase nula (pokud by tam už nic nebylo, podmnožina byla poslední v pořadí). Kód tedy musí mít tvar $K = \alpha\mathtt{01}^i\mathtt{0}^j$ pro nějaký prefix α a čísla $i > 0$, $j \geq 0$.

Dokážeme, že hledaným následníkem je $K' = \alpha\mathtt{10}^{j+1}\mathtt{1}^{i-1}$. Jistě je $K' > K$, takže zbývá dokázat, že mezi K a K' neleží žádný další kód. Jak by mohl vypadat? V prefixu α se lišit nemůže (to by byl buďto menší než K nebo větší než K'). Z kódů, které za α mají `0`, je K největší, takže každý větší kód tam musí mít `1`. Ale z takových kódů je K' nejmenší.

K nalezení následníka tedy stačí odpočítat nuly a jedničky od konce, což jistě stihneme v čase $\mathcal{O}(n)$. (Dodejme, že na rozdíl od vyjmenování všech podmnožin přičítáním jedničky se zde nic nezamortizuje – hezky je to vidět na $k = 1$.)

Mimochodem, pokud podmnožiny kódujeme bitovými maskami, dá se následník masky `x` spočítat následující posloupností bitových operací. Důkaz necháváme na čtenáři ☺

```
a = x - 1
b = (x | a) + 1
c = (x ^ a) + 1
x = b | (((x ^ b) / c) >> 1)
```

Ořezávání rekurze

Probírání množin podle velikosti vyřeší většinu vstupů, ale ty největší ne. Zkoumané množiny proto omezíme ještě víc.

Množiny budeme prohledávat rekurzivně, v i-tém kroku se budeme rozhodovat, jestli do množiny přidáme i-tý vrchol. Rekurze je tedy popsaná binárním stromem hloubky N, který v i-tém patře rozhoduje o i-tém vrcholu.

Během rekurzivního průchodu si kromě *množiny vybraných vrcholů* budeme průběžně počítat i *množinu již vyřešených vrcholů* (vrcholů ve vzdálenosti nejvýše 1 od vybraných). Při přidání vrcholu do množiny vybraných vždy přidáme jak tento vrchol, tak jeho sousedy do množiny vyřešených (pomocí *bitového oru*). Jakmile jsou vyřešené všechny vrcholy, aktualizujeme průběžné minimum a tuto větev rekurze ukončíme.

Udržování obou množin nám umožní tyto optimalizace:

- Pokud by přidáním vrcholu do množiny vybraných nepřibyl žádný vrchol do množiny vyřešených, nemá smysl tuto větev prohledávat.
- Jakmile velikost vybrané množiny dosáhne velikosti zatím nejlepšího pokrytí, také můžeme tuto větev přeskočit, protože v ní menší pokrytí nebude. (Této technice se říká *branch and bound.*)

S tímto algoritmem je již možné vyřešit všechny vstupy.

Program (C):
`http://ksp.mff.cuni.cz/viz/33-5-X1.c`