Korespondenční Seminář z Programování

33. ročník **KSP** **Únor 2021**

Dostává se k vám řešení třetí série tohoto ročníku KSP. Zvláštní pozornosti doporučujeme věnovat kompetitivní úloze, **v komentářích** k řešení brzy totiž vydáme i **řešení od několika řešitelů**. Zároveň si můžete ve webové verzi řešení prohlédnout vizualizace.

Připomínáme, že **seriál můžete za menší bodové ohodnocení stále odevzdávat** a to až do konce ročníku, takže jeho vzorové řešení vám vydáme až po skončení celého ročníku.

Vzorová řešení třetí série třicátého třetího ročníku KSP

33-3-1 Bezpečný tunel

Chceme najít nejkratší cestu mezi políčky, na která se umí legálně dostat Kevin, a políčky, na která se legálně umí dostat Zuzka. Nejdříve tedy najdeme, jaká políčka to vůbec jsou: spustíme prohledávání do šířky se startem v Kevinově domě, navštívená políčka si označíme, a pokud se dostaneme do vzdálenosti více než K metrů, skončíme. To samé provedeme se startem v Zuzčině domě.

Jak nyní najít dvě různě označená políčka, která jsou k sobě nejblíž? Použijeme prohledávání do šířky ještě jednou. Tentokrát však ne se startem v jediném bodě, ale za startovní políčka prohlásíme všechna políčka, na která se umí dostat Kevin. Tato políčka vložíme do fronty, nastavíme jim nulovou vzdálenost a pokračujeme standardním prohledáváním do šířky, ignorujícím zdi, dokud nenarazíme na první políčko, na které se umí dostat Zuzka. Toto bude druhý konec tunelu, ten první můžeme najít, když si během prohledávání budeme ukládat zpětné odkazy a pak po nich projdeme do jednoho z políček, odkud jsme prohledávání začali.

Jaká bude časová složitost tohoto algoritmu? Používáme třikrát za sebou prohledávání do šířky a pak ještě jednou projdeme některá políčka. Prohledávání do šířky na čtvercové mapě s rozměry $M \times N$ nám zabere $\mathcal{O}(MN)$ času, stejně jako následné procházení políček. To bude tedy výsledná časová složitost. Samotné načítání mapy do paměti zabere $\mathcal{O}(MN)$ času, lepší algoritmus tedy nenajdeme.

Co kdybychom však mapu v paměti mít nemuseli a načítali jen políčka, která potřebujeme? Začneme stejně, najdeme si políčka, na která se Kevin a Zuzka umí dostat. Těchto políček je $\mathcal{O}(K^2)$, prohledáváním do šířky to zvládneme v čase $\mathcal{O}(K^2)$. Nyní předpokládejme, že Kevinův dům je víc vlevo než Zuzčin dům. Nechť souřadnice nejpravějšího políčka, na které se Kevin umí dostat, je x_K, a souřadnice nejlevějšího políčka, na které se umí dostat Zuzka, je $x_Z = x_K + c$. V takovém případě libovolný tunel, který Zuzka s Kevinem vykopou, musí jít alespoň c políček vodorovným směrem. Můžeme si tedy Zuzčin dům a všechna políčka, na která se umí dostat, pomyslně posunout o c políček doleva. To samé uděláme i se svislým směrem a ypsilonovou souřadnicí. Takto si můžeme mapu zmenšit na velikost $\mathcal{O}(K) \times \mathcal{O}(K)$ a opět stejným způsobem najít oba konce tunelu. Časová i paměťová složitost bude v takovém případě $\mathcal{O}(K^2)$.

Úlohu připravili: Zuzka Urbanová, Lucka Vomelová

33-3-2 Ospalý student a diktát

Pohlédněme na úlohu grafově. Délka úkolu n odpovídá počtu vrcholů a každý útržek popisuje orientovanou cestu na těchto vrcholech. Když tedy rozdělíme útržky na hrany, dostaneme nějaký orientovaný graf. V něm pak chceme najít *jednoznačné* pořadí jeho vrcholů tak, aby vedly orientované hrany vždy z vrcholu s nižším pořadím.

Jinými slovy, graf chceme *topologicky setřídit*. Aby toto setřídění existovalo, v grafu nesmí existovat žádný orientovaný cyklus. Nahlédněme, že pokud by takový cyklus existoval, tak ať už zvolíme na tomto cyklu libovolný vrchol jako první v pořadí, bude do něj vést hrana nesprávným směrem. Takové grafy nazýváme *orientované acyklické grafy* nebo též DAGy (zkratka z anglického názvu).

Je potřeba, aby hran v grafu bylo alespoň $n - 1$. Kdyby ne, graf bude určitě nesouvislý. Pořadí vrcholů z různých komponent lze libovolně prohazovat, tím pádem nemůže být jednoznačné.

Všimněme si, že každý DAG musí obsahovat vrchol s, do kterého nevede žádná hrana. Kdyby tomu tak nebylo, najdeme orientovaný cyklus: začněme v libovolném vrcholu s_0. Do vrcholu s_i vede hrana, vydejme se tedy „pozpátku“ po ní do s_{i+1}. Tohle můžeme opakovat, dokud z s_ℓ nenarazíme na již navštívený vrchol s_k. Orientovaný cyklus je tedy $s_k \to s_\ell \to \ldots \to s_{k+1} \to s_k$.

Pokud vrchol s bez vstupních hran existuje jen jeden, můžeme ho bezpečně prohlásit za první v pořadí. Pokud najdeme alespoň dva různé vrcholy bez vstupních hran, rovnou prohlásíme, že tyto vrcholy nemají jasné pořadí. Pokud žádný vrchol bez vstupních hran nenajdeme, vyskytuje se v grafu orientovaný cyklus, který můžeme třeba pomocí DFS najít a vypsat.

Pokud z DAGu odstraníme libovolný vrchol v, dostaneme znovu DAG, a v něm bude jistě existovat jiný vrchol s' bez vstupních hran. Obdobně, pokud graf obsahuje orientovaný cyklus C, vrchol s jistě nebude jeho součástí a jeho odstranění C nepřeruší.

Takto můžeme topologicky třídit. Začínáme s prázdným pořadím na původním grafu. Pokud najdeme $s_1 \neq s_2$ bez vstupních hran, rovnou prohlásíme, že s_1 a s_2 nemají jasné pořadí. Pokud žádné s nenajdeme, můžeme třeba pomocí DFS najít orientovaný cyklus a ten vypíšeme. Jinak přidáme jednoznačné s do rozpracovaného pořadí, odstraníme jej z grafu a pokračujeme dál stejným způsobem, dokud neurčíme pořadí všech vrcholů.

Tohle nám jistě dává algoritmus, který najde správné pořadí. Pokud by graf nebyl DAG, tak na orientovaný cyklus dříve či později narazíme tím, že nenajdeme s. Stejně tak pokud by bylo pořadí nejednoznačné, tak někdy narazíme na více různých $s_1 \neq s_2$ v nějakém podgrafu.

Pokud tento algoritmus však naprogramujeme takto přímočaře, bude mít časovou složitost $\Omega(n^2)$. Nejvíce nás totiž zdržuje nalezení s, které prochází všechny vrcholy n-krát. Naštěstí každé odstranění vrcholu sníží počet vstupních

hran jeho sousedů o 1.

Stačí si tudíž udržovat u každého vrcholu počet aktivních vstupních hran a seznam výstupních hran. Kromě toho chceme udržovat seznam vrcholů s nula aktivními hranami, označme jej S. Zpočátku pro každý vrchol nastavíme počet aktivních hran na počet vstupních hran. Vrcholy, které nemají žádné vstupní hrany, přidáme do S rovnou. To uděláme jednoduchým průchodem všech vrcholů.

Poté n-krát zopakujeme určení i-tého vrcholu v pořadí: Podíváme se na délku seznamu $|S|$. Pokud $|S| \neq 1$, prohlásíme patřičný problém. Jinak vrcholu $s \in S$ určíme pořadí i, všem jeho výstupním sousedům snížíme počet aktivních hran o 1 a smažeme jej z S. Sousedy, kterým klesl počet aktivních hran na 0, přidáme do S.

Každý krok trvá $\mathcal{O}(1+d)$, kde d je počet hran vycházejících z aktuálního s. Všech n kroků se pak sečte na čas $\mathcal{O}(n + m) = \mathcal{O}(m)$, kde m je počet hran.

Zbývá dořešit, jak převést vstup na graf. Každý útržek rozdělíme na hrany a přidáme je příslušným vrcholům. Ještě musíme ověřit, že ve všech útržcích dohromady bylo opravdu n různých čísel. Pokud ne, nahlásíme chybu. Celý algoritmus pak bude mít složitost $\mathcal{O}(N)$, kde N je délka vstupu – součet délek všech útržků.

Zatím jsme však stále neřešili, jak indexovat vrcholy. Pokud jsou vrcholy dostatečně malá přirozená čísla v rozsahu $\mathcal{O}(n)$, můžeme je třeba indexovat v poli. To ale nemusí platit – čísla mohou být třeba obrovská, záporná nebo desetinná, žádnou bližší vlastnost jsme neslíbili. Není však problém v čase $\mathcal{O}(n \log n)$ vstup přečíslovat na čísla $1, \ldots, n$.

Úlohu připravili: Jirka Kalvoda, Vašek Končický, Tom Sláma

33-3-3 Stříbrných stříkaček

Budeme využívat názvosloví treap. Tedy dostřik bude klíč a náhodná čísla budou priority.

Vždy budeme chtít odpovídat maximálním možným počtem čísel, co jde, tedy N.

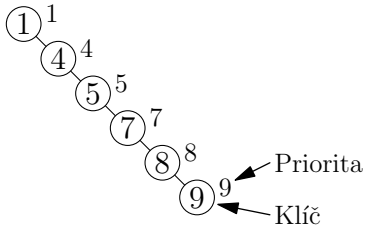
Na první vstup stačí odpovědět `3 1 2`.

Prvním přidáním se dvakrát zavolá `split` a dvakrát `merge` (bez žádné rekurze). Druhým pak čtyřikrát a třikrát (každý `split` zavolaný z `main` dostane neprázdný vstup, tedy alespoň jednou zavolá sám sebe a jeden `merge` zavolá také sám sebe, protože oba dva parametry budou neprázdné treapy). Posledním pak alespoň čtyřikrát a čtyřikrát (každé volání z `main` se alespoň jednou zrekurzí).

Celkem tedy funkce budou zavolány alespoň 21krát bez ohledu na náhodná čísla.

Toto byl jen cvičný vstup na uvědomění si fungování treapu.

Dále si povšimneme, že spoustu volání funkcí na jedno číslo dostaneme tehdy, když budeme pracovat se stromy s velkou hloubkou. Je tedy vhodné, aby se strom nijak nevětvil, tedy aby se jednalo o cestu. Cestu získáme třeba tak, že každý vrchol bude mít stejnou hodnotu jako váhu. Pro získání nějakých bodů tedy stačí odpovědět stejná čísla, jako byla na vstupu.

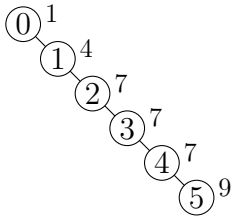


Program (C++):
`http://ksp.mff.cuni.cz/viz/33-3-3-1.cpp`

První problém tohoto řešení nastane, když generátor náhodných čísel bude rozbitý a bude vracet opakovaně stejná čísla. Naše řešení tedy nebude vytvářet nové vrcholy treapu. Stačí se ale podívat na přesnou implementaci treapu a povšimnout si, že při rovnosti vah se nahoru dostane levý vrchol, tedy při dalších výskytech stejného čísla stačí odpovědět o trošku větším číslem. Díky tomu vznikne nový vrchol a stále zůstane zachován tvar cesty.

Díky tomu ale bude nutné všechna čísla přečíslovat. Tedy každé prioritě na vstupu přiřadit unikátní klíč mezi 0 a $N-1$ tak, aby se nezměnilo uspořádání. Tedy když priorita na pozici a je menší než priorita na pozici b, tak musí být i klíč na pozici a menší než klíč na pozici b. Pro rovnající se priority musí být klíče setříděné podle výše popsaného pozorování.

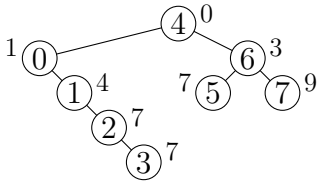
Setřídíme si dvojice (priorita, index) podle priority a v případě rovnosti podle indexu. Tuto posloupnost projdeme a postupně jí přiřadíme čísla klíčů od 0 do $N-1$. Poté, například pomocí setřídění podle indexu, přejdeme k původnímu pořadí.



Když bude náhodný generátor generovat klesající posloupnost, dosavadní řešení také nebude fungovat. Vždy totiž připojí nový vrchol nad původní kořen pomocí malého počtu volání funkcí.

Vzorové řešení tedy z poloviny čísel vytvoří předchozím algoritmem treap ve tvaru cesty. Zbylým prioritám určí hodnotu tak, aby byla těsně za posledním vrcholem této cesty. Díky tomu algoritmus musí rozdělit treap tak, že projde celou cestu.

Počet operací tedy musí být větší než $2 \cdot (N/2) \cdot (N/2)$. Přidáním každého z $N/2$ vrcholů v druhé skupině se provede alespoň $N/2$ operací `split` a následně `merge`. S tímto řešením již vyřešíme všechny vstupy.



Program (C++):
`http://ksp.mff.cuni.cz/viz/33-3-3-2.cpp`

Úlohu připravili: Jirka Kalvoda, Michal Kodad

33-3-4 Obsazování území

Úloha s nakupováním domů na mapě a obsazováním území byla netradiční, ale o to zajímavější. Stala se velmi populární i mezi organizátory, kteří se společně s účastníky předháněli v tom, komu se povede pokrýt všechny zastavěné plochy za co nejméně peněz.

Tato úloha patří mezi ty, na něž je těžké najít optimální řešení. Velikost stavového prostoru roste exponenciálně (i když se dá velmi dobře prořezávat) a mapa už je moc velká na to, aby šlo vyzkoušet všechny možnosti. Taková úloha se řeší *heuristicky* – nehledáme nejlepší možné řešení, ale pokoušíme se odhadnout nějaké, které alespoň nebude vyloženě špatné.

Řešení této úlohy tedy pojmeme tak, že se pokusíme ukázat několik různých heuristických přístupů, které používali organizátoři. Je poučné se podívat na různé přístupy k tomu samému. Zároveň jsme požádali i **několik nejlepších řešitelů**, aby také sepsali své postupy řešení, můžete si je přečíst v komentářích k řešení.[1]

Nejdříve trocha statistiky. Mapa Fiktivní Prahy sestávala celkem ze 268 435 456 (2^{28}) políček. Domů (zastavěných políček) bylo 21 295 829 (necelých 8 procent všech políček), a kdyby se někdo rozhodl koupit všechny, tak by ho to stálo neuvěřitelnou částku 114 338 661 881.

Nejlepší nalezené pokrytí Fiktivní Prahy bylo pokrytí 332 domy za celkovou cenu 532 293, za orgy těsně uhájil první místo Jirka Sejkora. Nejlepší účastnické pokrytí od Kristýny Petrlíkové skončilo těsně v závěsu s 329 domy za celkovou cenu 532 824.

Protože získat řešení s cenou blížící se jednomu milionu bylo celkem jednoduché a obtížný byl přesun od jednoho milionu k půl milionu, nastavili jsme bodovací funkci tak, že zhruba do 1 200 000 body přibývají pomalu a od této chvíle se počet bodů zvedá. Počet získaných bodů pro celkovou cenu C je:

$$body = \max\left(29 - \frac{C}{55000}, 9 - \frac{C}{625000}\right)$$

Maximální počet bodů jsme zastropovali na 15 a tímto gratulujeme Kristýně k jejich získání. Dosažené ceny a body ostatních (včetně organizátorů) si můžete prohlédnout v tabulce výsledků.[2]

Nyní již následují jednotlivá řešení. Na webu můžete najít jejich vizualizace.

Řešení Jirky Kalvody

Svůj plánek Prahy jsem si nejprve rozdělil na 8 svislých sloupců a každý jsem řešil samostatně. Doufal jsem, že překryvy na hranicích řešení moc nezdraží a díky rozříznutí stačilo vyřešit mnohem menší vstupy.

Každý vstup jsem pak procházel z vrchu dolů. V každý moment jsem si pamatoval několik možností, jak pokrýt již navštívené budovy. Pro každou budovu, kterou jsem potkal, jsem prošel všechny tyto možnosti pokrytí. U těch, které aktuální budovu nepokrývaly, jsem vyzkoušel různé možnosti, jak ji pokrýt, tedy koupit budovu v okruhu 500 metrů. Tím jsem jedno možné pokrytí nahradil několika novými, které již danou budovu pokryjí.

Takto by stačilo projít celý plánek a na konci vybrat pokrytí, které je nejlevnější. Ovšem je tu problém. Počet pokrytí při tomto algoritmu roste moc rychle, takže by nám brzy došla paměť, a navíc by algoritmus asi nestihl doběhnout v dosažitelném čase. Proto je nutné možnosti průběžně omezovat a vybírat ty nejnadějnější. Jenže jak na to? Každou možnost jsem se pokusil nějak ohodnotit a pak vybrat jen určitý počet těch nejlepších. Zde již nejspíše neexistuje žádný exaktní algoritmus ohodnocení, a proto přichází na řadu heuristika.

Levnější možnosti by mohly být lepší než dražší. Ale také záleží na tom, kolik domů (a taky které) krom již navštívených pokryje. Já jsem se proto rozhodl od ceny domu odečíst jednu k-tinu ceny pokrytých domů. Za k jsem zkoušel dosazovat různá čísla a pro každé jsem vybral několik nejlepších možností. Další možnosti jsem pak vybíral náhodně.

Každou z osmi oblastí jsem se tímto algoritmem pokusil vyřešit několikrát (s různým generátorem náhodných čísel). Pro každou oblast jsem pak vybral nejlepší řešení a ta jsem pak spojil.

Algoritmus po chvíli našel řešení s cenou 740 391.

Řešení Kuby Pelce

Nejdříve jsem zkusil přímočaré hladové řešení: postupně procházím všechna políčka, zleva doprava, shora dolů, a každý nepokrytý dům koupím. Výsledná cena je 3 174 465.

Tento hladový algoritmus jsem trochu vylepšil: když narazím na nepokrytý dům, najdu ten nejlevnější dům, který ho dokáže pokrýt, a ten koupím (bez ohledu na to, jestli už sám je nebo není pokrytý). Tímto postupem se lze dostat pod million, tento postup našel pokrytí za cenu 908 980.

Poté jsem vyzkoušel spoustu různých věcí, které ale příliš nepomohly. Například jsem domy kupoval podle jejich „cenavýkonu“, tedy poměr mezi jejich cenou a cenou všeho, co by koupě tohoto domu pokryla.

Nejlépe fungovalo randomizované řešení s poměrně jednoduchou myšlenkou: koupím mnoho náhodných domů (mnohem více, než jich je na pokrytí celého města potřeba), poté všechny zbytečné domy zahodím.

Moje generování náhodných domů ale není úplně přímočaré. Nejdříve vyberu náhodný bod na mapě. Tento bod nemusí odpovídat žádnému domu. Poté najdu dům co možná nejbližší k tomuto bodu. K tomu jsem použil předpočítanou mapu (pomocí BFS), která mi pro každé pole říká, kterým směrem se posunout, abych se přiblížil k nějakému domu. Nakonec v nějakém čtverci okolo takto nalezeného domu vyberu ten nejlevnější dům a ten koupím.

Abych měl zaručeno, že tato množina náhodně koupených domů skutečně pokryje celé město, všechny nepokryté domy pokryji pomocí předchozího hladového algoritmu.

Nyní vím, že tato množina pokrývá celé město, ale také obsahuje spoustu zbytečných domů, které je třeba nějak zahodit.

Pro každé políčko mapy si spočítám, kolikrát je množinou pokryto. Poté projdu všechny domy množiny od nejdražšího po nejlevnější. Pokud pro nějaký koupený dům platí, že všechny domy, které pokrývá, jsou pokryty více než jednou, tak daný dům zahodím, protože i potom bude stále pokryto

[1] `http://ksp.mff.cuni.cz/viz/33-3-4/komentare`

[2] `https://ksp.mff.cuni.cz/h/ulohy/33/33-3-4/vysledky`

celé město.

Tímto algoritmem jsem se dostal na cenu 665 550.

Řešení Jirky Sejkory

Můj přístup je založený na vylepšování už existujících řešení. Kombinací různých postupů jsem došel až k řešení s cenou 532 293.

Celou dobu jsem udržoval databázi řešení, která na konci obsahovala 21 730 řešení s celkovým počtem 7 601 772 domů.

Detailní popis včetně zajímavých optimalizací a zdrojových kódů je dostupný v Gitovém repozitáři.[3] Následuje jen velmi zběžné shrnutí popisu z předchozího odkazu.

Prvotní generování řešení bylo plně náhodné a později se při generování začal brát ohled na již existující řešení. Domy z dobrých řešení měly větší pravděpodobnost na vybrání do řešení.

Náhodně vygenerovaná řešení jsem zlepšoval pomocí přesunů domů na levnější místa a spojováním dvojic domů v jeden. Tyto přesuny vždy zachovaly validitu řešení. Jak to dělat efektivně (v lineárním čase s počtem pokrytých políček) je popsáno v detailní verzi.

Taktéž jsem kombinoval 1500 až 3500 nejlepších řešení pomocí vertikálních a horizontálních řezů, pokud zachovaly validitu řešení. Toto vyžadovalo netriviální množství optimalizací, aby to doběhlo v rozumném čase. Finální verze to zvládala za deset až dvacet minut.

A jako poslední hlavní část jsem vyřezával z nejlepšího řešení podměsto, které jsem optimalizoval samostatně a poté sloučil opět zpátky.

Tyto přístupy jsem různě střídal a ručně vybíral, které části má smysl vyřezávat.

Celkově řešení spotřebovalo přibližně týden čistého času plného vytížení jednoho procesoru s 12 vlákny a v některých momentech jsem měl problém vejít se do 32 GB paměti — což by šlo výrazně vylepšit za cenu menší rychlosti.

Po všech optimalizacích ale věřím, že k podobně dobrému řešení lze dojít znovu za jednotky hodin.

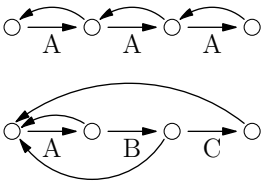
Úlohu připravili: Jirka Kalvoda, Kuba Pelc, Jirka Sejkora, Jirka Setnička

33-3-X1 Z(a)tracené kouzlo

Nejprve si připomeneme, jak v nějakém řetězci kouzlo najít. Použijeme algoritmus Knuth-Morris-Pratt neboli KMP.[4] Ten vytvoří *automat.* To je graf, jehož *stavy* (vrcholy) odpovídají prefixům kouzla. Mezi stavy vedou dva druhy hran: *dopředné,* které prodlužují prefix o jeden znak, a *zpětné.* Ty pro každý prefix kouzla určují, jaký nejdelší jeho neúplný sufix je také prefixem kouzla.

Celým řešením nás bude provázet příklad ze zadání: Máme abecedu $\{\mathtt{A}, \mathtt{B}, \mathtt{C}\}$, spočítejme všechny řetězce délky $N = 5$ se zaklínadlem `AAA` respektive `ABC`.

Automaty algoritmu KMP vypadají takto:

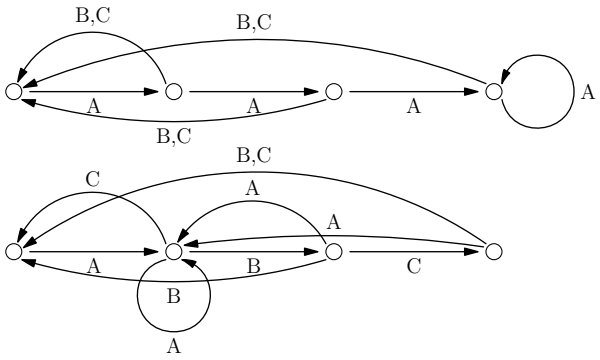


Pomocí hran se pak při čtení řetězce pohybujeme mezi stavy. Aktuální stav vždy určuje, jaký nejdelší sufix již prošlého řetězce je prefixem hesla. Když další písmeno řetězce odpovídá dopředné hraně, tak se po ní posuneme. V opačném případě se pohybujeme po zpětných hranách do té doby, než bude předchozí podmínka splněná, anebo již žádná zpětná hrana nebude existovat. Pro nás je ovšem nepraktické uvažovat dva typy hran, tak se jich pojďme zbavit.

Vytvoříme jiné hrany, které budou přímo reprezentovat přechody mezi stavy KMP. Pro každý stav a každé písmenko je totiž jednoznačně určeno, do jakého stavu se dostaneme. V původním KMP se tam dostaneme nějakou posloupností zpětných hran a nejvýše jedné dopředné. Ale my celou tuto posloupnost nahradíme jedinou hranou označenou příslušným písmenem abecedy. Tím vznikne multigraf – může mít paralelní hrany a může obsahovat smyčky. (Mimochodem, pokud znáte konečné automaty, přesně takový jsme teď popsali.)

Nové hrany můžeme vytvořit indukcí podle stavu (od prázdného prefixu k celému kouzlu). Z prázdného prefixu povede dopředná hrana označená prvním písmenem kouzla. Pro ostatní písmena kouzla zřídíme smyčky vedoucí zpět do počátečního stavu. Pro každý další stav ponecháme dopřednou hranu a všechny ostatní hrany zkopírujeme z toho stavu, do kterého v KMP vedla zpětná hrana.

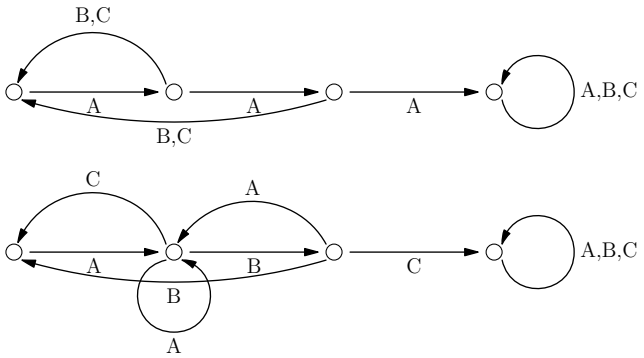
Pro naše příklady to dopadne takto:



Ještě si ale tento graf mírně upravíme. Zajímá nás totiž jen, jestli řetězec kouzlo obsahuje či nikoliv. Je nám jedno, kde přesně výskyty jsou, ani kolik jich je. Proto stav „právě jsme na konci kouzla“ nahradíme stavem „už jsme celé kouzlo alespoň jednou našli“. Z tohoto stavu se v průchodu již nikdy nedostaneme. Stačí tedy vzít původní koncový stav a hrany nahradit smyčkami, které se vrací zase do tohoto stavu:

[3] `https://gitea.ks.matfyz.cz/Sejsel/33-3-4`

[4] `http://ksp.mff.cuni.cz/viz/kucharky/hledani-v-textu`



Teď se konečně dostaneme k zadané úloze. Ta vůbec nechce zjistit, zdali nějaký řetězec obsahuje kouzlo – místo toho nás zajímá, kolik takových řetězců existuje. Pojďme tedy prohledávání spustit na všech řetězcích současně.

Uvědomíme si, že výpočet našeho vyhledávacího algoritmu pro nějaký řetězec odpovídá právě jedné procházce (sledu) po předchozím grafu. Ta vede z počátečního stavu (prázdného prefixu) přes hrany označené postupně písmeny řetězce. A pokud jsme našli kouzlo, procházka skončí v koncovém stavu.

Řetězce délky N obsahující kouzlo tedy přesně odpovídají procházkám délky N v tomto grafu, které začínají v počátečním stavu a končí v koncovém. Za různé procházky přitom považujeme ty, které se liší v alespoň jedné hraně po cestě (nestačí rozdílnost ve vrcholech, protože více hran může spojovat stejné vrcholy).

Toto vyřešíme pomocí dynamického programování. Pro každou délku procházky budeme počítat, kolik existuje různých procházek končících v každém stavu. Při zvyšování délky procházky pak jen projdeme všechny hrany grafu. K cílovým vrcholům pro procházku délky $n+1$ přičteme hodnoty počátečních vrcholů procházky délky n. Odpověď pak bude počet procházek délky N hran končících ve stavu „obsahuje kouzlo“.

Opět příklad:

Délka	0	1	2	3	4	5
`""`	1	2	6	18	52	152
`"A"`	0	1	2	9	18	52
`"AA"`	0	0	1	3	6	18
Obsahuje	0	0	0	1	5	21

Délka	0	1	2	3	4	5
`""`	1	2	5	14	40	115
`"A"`	0	1	3	9	26	75
`"AB"`	0	0	1	3	9	26
Obsahuje	0	0	0	1	6	27

Abychom nenaráželi na velikost datových typů, celý výpočet budeme provádět modulo velké prvočíslo P.

Výsledky dynamického programování můžeme průběžně zapomínat. Stačí si jen pamatovat aktuální a předešlou délku řetězce.

Časová složitost algoritmu je $\mathcal{O}(AZN)$, kde A je velikost abecedy, Z délka kouzla a N délka řetězce. Paměťová složitost je $\mathcal{O}(AZ)$.

Zrychlujeme

Dále si všimneme, že každá počítaná hodnota pro nějak dlouhý řetězec je jen lineární kombinací počítaných hodnot pro o jedna kratší řetězec. To můžeme popsat pomocí násobení matic. Uvážíme sloupcový vektor $\mathbf{x}_n$, který odpovídá počtům procházek délky n do jednotlivých stavů. Pokud tento vektor vynásobíme zleva vhodnou maticí, dostaneme vektor $\mathbf{x}_{n+1}$ počtů procházek délky $n+1$ do jednotlivých stavů. Hodnoty v této matici budou odpovídat koeficientům lineárních kombinací z předchozího algoritmu – rozmyslete si, že nám vlastně budou říkat, kolik hran vede z i-tého stavu do j-tého. Je to tedy multigrafová obdoba matice sousednosti.

Máme tedy matici $\mathbf{A}$ takovou, že $\mathbf{A}\mathbf{x}_n = \mathbf{x}_{n+1}$. Proto $\mathbf{x}_N = \mathbf{A}^N\mathbf{x}_0$, kde $\mathbf{x}_0$ je vektor počtů procházek o 0 hranách (má tedy 1 v počátečním stavu a všude jinde 0.

Pro naše příklady:

$$\begin{pmatrix} 2 & 2 & 2 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 3 \end{pmatrix}^5 \cdot \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 152 \\ 58 \\ 18 \\ 21 \end{pmatrix}$$

Řetězců se zaklínadlem `AAA` je 21.

$$\begin{pmatrix} 2 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 3 \end{pmatrix}^5 \cdot \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 115 \\ 75 \\ 26 \\ 27 \end{pmatrix}$$

Řetězců se zaklínadlem `ABC` je ale 27.

Teď stačí využít standardní algoritmus pro výpočet N-té mocniny pomocí $\mathcal{O}(\log N)$ násobení. Ten funguje pro matice stejně jako pro čísla, takže $\mathbf{A}^N$ spočítá pomocí $\mathcal{O}(\log N)$ maticových násobení. Jelikož násobíme matice $(Z+1) \times (Z+1)$, jedno násobení trvá $\mathcal{O}(Z^3)$. Celý algoritmus proto seběhne v čase $\mathcal{O}(Z^3 \log N)$ a paměti $\mathcal{O}(Z^2)$.

(Můžete se podivovat, kam se poděla závislost na velikosti abecedy. Té jsme se dovedli zbavit proto, že všechny znaky, které se nevyskytují v zaklínadle, můžeme považovat za „ostatní“ a vždy si pořídit příslušný počet paralelních hran v grafu.)

Úlohu připravili: Jirka Kalvoda, Martin „Medvěd“ Mareš

Výsledková listina třetí série třicátého třetího ročníku KSP

	řešitel	*škola*	*ročník*	*sérií*	*3-1*	*3-2*	*3-3*	*3-4*	*3-S*	*3-X1*	*série*	*KSP-X*	*celkem*
0.					9	10	13	14	14	10	60,0	30,0	180,0
1.	Kristýna Petrlíková	SPŠJičín	3	13	9	9	13	15	14		60,0	0,0	181,0
2.	Jiří Kvapil	GTomkovaOL	3	17	9	8	13	8	14		52,0	3,0	168,0
3.	Daniel Skýpala	GTomkovaOL	3	18	9	9	13	11,5	14	0	56,5	5,0	167,5
4.	Jan Adámek	GKepleraPH	4	8	9	10	2	7	14	5	42,0	18,0	161,0
5.	Filip Hejsek	GPísnickáPH	4	5		10	13	6,5			29,5	0,0	151,5
6.	Ondřej Skácel	GTomkovaOL	2	3	2	4	13	14	14		47,0	0,0	150,0
7.	Robert Jaworski	GÚstavníPH	3	5	5	5	2	14	14		40,0	2,0	144,0
8.	Eliška Macáková	CENADA BA	1	3	8	9	0			10	17,0	20,0	133,0
9.	Jan Kotovský	GPísnickáPH	2	4	6	4	7	6	13		36,0	0,0	128,0
10.	Viktor Fukala	GKepleraPH	4	7				8,5			8,5	19,0	126,5
11.	Vít Skalický	GPísnickáPH	3	17	7	7	2		14		30,0	0,0	124,0
12.	Lukáš Veškrna	GKepleraPH	3	3	7	9	10	5	14		45,0	4,0	123,0
13.	Jakub Surga	ParkLane	3	3	9	6	7	0	13		35,0	0,0	122,0
14.	Vladimír Chudý	G Chrudim	4	18			7				7,0	4,5	110,0
15.	Jakub Ondroušek	GTomkovaOL	1	3	5		0	6,5	14		25,5	0,0	104,5
16.	Janek Hlavatý	GJirsíkaČB	2	7	8	9	7	6	12		42,0	0,0	102,0
17.	Patrik Herman	GTomkovaOL	2	4	1	0	7		10		18,0	0,0	87,0
18.	Dominik Farhan	GMikulášPL	4	7	6	8	7	10			31,0	0,0	85,0
19.	Ondřej Sladký	GMikulášPL	4	10						10	0,0	19,0	81,0
20.	Václav Janáček	GJarošeBO	4	6	9	9	10			8	28,0	16,0	80,0
21.	Matej Štencel	GPošKošice	4	5							0,0	0,0	77,0
22.	Pavel Jordán	GPOA Znojmo	2	2							0,0	0,0	66,0
23.	Adam Kolník	SSŠVTPraha	2	3					14		14,0	0,0	58,0
24.	Prokop Randáček	GFXŠaldyLI	2	4							0,0	0,0	53,0
25.	Šimon Genčur	GBBr	1	5	7			0			7,0	0,0	41,0
26.	David Holas	SPŠEMasLI	1	1	4	5	13	10,5		1	32,5	1,0	32,5
27.	Martin Havelka	Gym Třeboň	3	3							0,0	0,0	28,0
28.	Klára Grinerová	GZborovPH	4	3			2	6,5			8,5	0,0	27,5
29.	Jiří Bartošík	SUHr	3	2							0,0	0,0	22,0
30.	Albert Kučera	GNadŠtolPH	4	4							0,0	0,0	21,0
31.	Kristýna Umlaufová	SPŠOstrov	4	2							0,0	0,0	20,0
32.	Jáchym Tuma	G FrýdlNOs	0	2							0,0	0,0	19,0
33.	Andrej Thomas Dobrev	GJHroncaBA	4	1							0,0	0,0	18,0
34.	Michal Žáček	MensaG	4	1							0,0	0,0	15,0
35.	Tomáš Kašpárek	G FrýdlNOs	3	1							0,0	0,0	12,0
36.–42.	Vojtěch Březina	GCoubTábor	4	4							0,0	0,0	10,0
	Petr Filip	GLovosice	2	1							0,0	0,0	10,0
	Vojtěch Gaďurek	PORGPha	4	1							0,0	0,0	10,0
	Štěpán Kovář	GNadKavaPH	4	1							0,0	0,0	10,0
	Michal Pavlíček	MendelGOP	3	1							0,0	0,0	10,0
	Daniel Šoltýs	GTřeKošice	3	2							0,0	0,0	10,0
	Filip Úradník	GyMimoň	4	1							0,0	0,0	10,0
43.	Jan Ráček	SPŠEMasLI	1	1							0,0	0,0	7,0
44.–45.	Bohumil Kulvejt	G Sokolov	3	1							0,0	0,0	6,0
	Josef Malý	GPísnickáPH	2	1							0,0	0,0	6,0
46.–47.	Veronika Jůzková	MensaG	3	1							0,0	0,0	2,0
	Matěj Strnad	ZŠRiegraSM	0	1							0,0	0,0	2,0

Bonusové úlohy označené „X“ mají svou vlastní výsledkovou listinu a nepočítají se do normálního bodování ročníku.

Výsledková listina KSP-X po třetí sérii třicátého třetího ročníku

	řešitel	*škola*	*ročník*	*sérií*	*1-X1*	*2-X1*	*3-X1*	*celkem*
0.					10	10	10	30,0
1.	Eliška Macáková	CENADA BA	1	3	9	1	10	20,0
2.–3.	Viktor Fukala	GKepleraPH	4	7	9	10		19,0
	Ondřej Sladký	GMikulášPL	4	10	9		10	19,0
4.	Jan Adámek	GKepleraPH	4	8	7	6	5	18,0
5.	Václav Janáček	GJarošeBO	4	6	8		8	16,0
6.	Daniel Skýpala	GTomkovaOL	3	18	4	1	0	5,0
7.	Vladimír Chudý	G Chrudim	4	18	4,5			4,5
8.	Lukáš Veškrna	GKepleraPH	3	3	4			4,0
9.	Jiří Kvapil	GTomkovaOL	3	17	2	1		3,0
10.	Robert Jaworski	GÚstavníPH	3	5	1	1		2,0
11.	David Holas	SPŠEMasLI	1	1			1	1,0

Bonusové úlohy z jednotlivých sérií se nepočítají do bodování ročníku. Mají svou vlastní výsledkovou listinu a za jejich úspěšné vyřešení (alespoň polovina bodů za úlohu) udělujeme speciální odměny.