Korespondenční Seminář z Programování

34. ročník **KSP** **Únor 2022**

Vzorová řešení třetí série třicátého čtvrtého ročníku KSP

34-3-1 Ideální žádost

Naším úkolem je simulovat práci dvou úředníků. Kdybychom věděli, v jakém pořadí úředníci berou žádosti, úloha by byla snadná. U obou úředníků bychom si udržovali, kdy dodělají úkol, na kterém právě pracují, a další úkol bychom přiřadili úředníkovi, který svůj předchozí úkol dokončil jako první. Jakmile už by nezbývaly úkoly, vrátíme čas, kdy pomalejší úředník vyřeší svůj poslední úkol.

Zbývá rozmyslet, jak v každém kroku hledat ideální žádost. Můžeme si všimnout, že ideální žádost je přesně ta, která je poslední v prvním klesajícím úseku od začátku posloupnosti žádostí. Všechny předchozí totiž úředník přeskočí, jelikož se za nimi nachází snazší, jenže tuto už nepřeskočí, neboť je za ní složitější (popř. žádná) žádost.

Kdybychom v každém kroku takovouto úlohu hledali znovu od začátku, stálo by nás čas $\mathcal{O}(n)$ pro každou žádost, což je příliš pomalé.

Ke zrychlení můžeme využít informaci o předchozí ideální žádosti. Budeme-li si pamatovat počáteční klesající úsek v zásobníku, není složité tento úsek po každém kroku aktualizovat. Poté, co odebereme žádost z vrchu zásobníku (tu jsme právě zpracovali), mohou nastat dva případy. Počáteční klesající úsek se nezmění (až na odebrání právě zpracované žádosti), anebo se odebráním žádosti stalo, že se neklesající úsek prodloužil (např. pokud bychom v posloupnosti 3 1 2 odebrali 1). V takovém případě musíme přidávat zatím nezpracované žádosti, dokud tvoří klesající úsek. Ideální žádost je pak vždy na vrchu zásobníku.

Ačkoliv jsme v každém kroku mohli stále provést mnoho operací, každý vrchol jsme přidali a odebrali ze zásobníku právě jednou. Časová složitost na režii zásobníku a stejně tak i celková složitost je tak $\mathcal{O}(n)$. Paměťová složitost je také $\mathcal{O}(n)$.

Program (Python):
`http://ksp.mff.cuni.cz/viz/34-3-1.py`

Úlohu připravili: Honza Černý, Martin „Medvěd“ Mareš, Ondra Sladký

34-3-2 Faktoriál

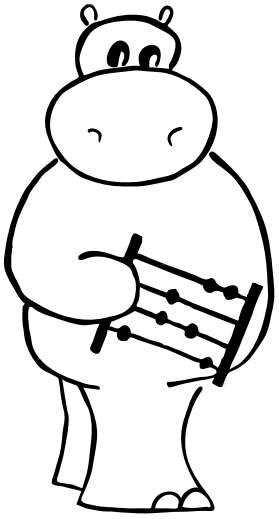
Je jasné, proč se Filip snažil vyhnout počítání celého faktoriálu: již pro $N = 20$ se výsledek nevejde do 64bitového celého čísla. I kdybychom uměli pracovat s čísly většími, brzy vzrostou výsledky tak, že by se ani nevešly do operační paměti. Budeme tedy potřebovat chytřejší přístup.

Prozatím si zjednodušme situaci a omezme se na dvojkovou soustavu. Kdy má číslo na konci nulu? Je to právě tehdy, když je dělitelné 2. Poté, co dělení provedeme, bude mít o nulu méně. Zajímá nás tedy, kolikrát můžeme vydělit $N!$ dvěma beze zbytku.

Může se zdát, že jsme si nepomohli. Vždyť nevíme, kolik je $N!$. Naštěstí to vědět nepotřebujeme, místo toho nám stačí spočítat, kolikrát můžeme vydělit 2 každý z činitelů v definici $N! = 1 \cdot 2 \cdot \ldots \cdot N$. Následně tyto hodnoty sečteme, čímž získáme hledaný výsledek. Proč to platí? Můžeme si představit, že dvojky z každého z činitelů vytkneme na začátek celého součinu. Pak je vidět, že alespoň tolik dvojek lze z $N!$ vydělit. Zbude nám součin čísel, z nichž žádné není dělitelné 2. A protože 2 je prvočíslo, ani tento součin nebude dělitelný 2.

Musíme zpracovat každé číslo, tento postup by tedy měl časovou složitost $\Omega(N)$. Horní odhad je stejný, nicméně úvahy vedoucí k jeho odvození by nás samy o sobě zavedly k efektivnějšímu přístupu, podívejme se tedy rovnou na něj.

Není těžké si všimnout, že je zbytečné zpracovávat každé číslo zvlášť. Vždyť víme, že každé druhé číslo je dělitelné 2. A z nich je každé druhé dělitelné 2 znovu, a tak dále. Výsledek tedy získáme tak, že sečteme $N/2^m$ (celočíselné dělení) pro všechny mocniny dvojky $2^m \leq N$. Takových mocnin je $\mathcal{O}(\log N)$, stejná bude i celá časová složitost tohoto přístupu.



Obecná soustava

Postup uvedený výše funguje pro soustavy o základu K, kde K je prvočíslo. Tuto vlastnost jsme ostatně potřebovali při argumentaci o korektnosti. Co si počneme s obecným základem?

Uvažme například soustavu o základu 40. Opět chceme zjistit, kolikrát můžeme $N!$ vydělit 40. Uvědomíme si, že číslo je dělitelné 40 právě tehdy, když je dělitelné 8 a 5 zároveň. Už umíme zjistit, kolikrát je $N!$ dělitelné 5, protože se jedná o prvočíslo. Stejně tak umíme zjistit, kolikrát je $N!$ dělitelné 2. Na každou osmičku padnou tři dvojky (protože $8 = 2^3$), pokud tedy počet dvojek vydělíme 3, získáme počet osmiček. Jako výsledný počet nul na konci zápisu $N!$ prohlásíme menší z počtu pětek a počtu osmiček.

Obecně budeme potřebovat prvočíselný rozklad čísla K. Pro každé z prvočísel p v rozkladu spočítáme, kolikrát lze $N!$ vydělit p, a tuto hodnotu podělíme mocninou p v rozkladu. Za výsledek prohlásíme minimum z takto získaných hodnot.

Zbývá rozmyslet, jak prvočíselný rozklad čísla K spočítat. Označme Z zbytek, který ještě budeme rozkládat, na začátku $Z = K$. Projdeme všechna čísla p od 2 a každým vydělíme Z, kolikrát to jen půjde. Počet úspěšných vydě-

lení bude mocnina u p v rozkladu. Zastavíme se, jakmile $p > \sqrt{Z}$. Buď nám zbude $Z = 1$ (už jsme našli všechny dělitele), nebo zbude $Z > 1$. Zbytek Z není dělitelný žádným z čísel menších než $\sqrt{Z}$ (kromě jedničky), tedy všechny dělitele jsou větší než $\sqrt{Z}$, a proto jediným dalším dělitelem musí být Z. Jinými slovy Z je prvočíslo.

Prvočíselný rozklad má časovou složitost $\mathcal{O}(\sqrt{K})$. Prvočísel bude nejvýše $\log_2 K$, na každém strávíme čas $\mathcal{O}(\log_p N)$, dohromady se spokojíme s odhadem $\mathcal{O}(\log K \log N)$, skutečnost bude ještě trošku lepší. Celý algoritmus pak běží v čase $\mathcal{O}(\sqrt{K} + \log K \log N)$. Prostoru potřebujeme konstantně pro každé prvočíslo, tedy $\mathcal{O}(\log K)$.

Program (Python):
`http://ksp.mff.cuni.cz/viz/34-3-2.py`

Úlohu připravili: Jirka Kalvoda, David Klement, Kristýna Petrlíková, Lucka Vomelová

34-3-3 Jízda tramvají

Nejprve fyzikálně rozeberme situaci, kdy na Jirku působí nějaká síla F pod úhlem φ vzhledem ke spojnici jeho ramen (tedy úhel $\varphi = 0$ odpovídá situaci, kdy síla působí z boku, a $\varphi = \frac{\pi}{2}$, kdy působí zepředu či zezadu).

Pokud bychom dostali sílu působící v předozadním či bočním směru, je snadné pro ni říct, zda Jirku povalí či nikoliv. Pro obecný případ pak umíme využít faktu, že libovolnou sílu umíme rozložit na dvě složky, kde jedna složka o velikosti $|F| \cos \varphi$ působí z boku a druhá o velikosti $|F| \sin \varphi$ zepředu či zezadu. Jelikož víme, že z boku Jirka vydrží síly libovolné velikosti, určující bude, zda velikost složky v předozadním změru bude menší či rovna velikosti F_0, tedy zda:

$$|F| \sin \varphi \leq |F_0|$$
$$\sin \varphi \leq \frac{|F_0|}{|F|}$$

Pokud si tedy představíme, že máme sílu v nějakém fixním směru, možná natočení Jirky, tak aby nespadl, tvoří dvě protilehlé výseče.

Nyní se již můžeme zamyslet nad řešením samotné úlohy.

Postupně budeme zpracovávat síly, během čehož si budeme udržovat množinu všech stavů, ze kterých se lze dostat na minimální celkové otočení, a po každé nové síle tuto množinu aktualizujeme.

Začínáme ve stavu, kdy je Jirka natočen počátečním směrem a zatím se nemusel nijak otáčet.

Jakmile zpracováváme další sílu, pro každý stav mohou nastat 2 případy.

- Zůstane-li Jirka v tomto stavu, nespadne. Pak určitě nemá smysl se jakkoliv otáčet. Pokud by tak učinil nyní, může tak zcela jistě učinit při příští síle, aniž by se musel více otáčet. Tedy stav se nijak nezměnil.
- Jirka se musí otočit, aby nespadl. Pak se musí jedním, nebo druhým směrem otočit tak, aby se dostal na hranici výseče, kdy nespadne. Z podobné úvahy jako v předchozím případě se však více otáčet nikdy nemusí. Tedy původní stav vyhodíme a přidáme dva nové.

Přímočará implementace předchozích pozorování vede na exponenciální algoritmus. Ten však můžeme snadno zrychlit, uvědomíme-li si, že pokud se Jirka musel otáčet, nezávisle na původním stavu skončil na jedné ze dvou hranic výseče. Nové stavy tak můžeme spojit. Stačí vzít pro každý ze směrů pouze stav s nejmenším celkovým otočením.

Jelikož v každém kroku přibudou nejvýše dva stavy, celkem bude na konci nejvýše $\mathcal{O}(n)$ stavů. Z nich pak lze snadno najít nejmenší možné celkové otočení.

Jelikož každý stav umíme zpracovat v konstantním čase, dostáváme tak kvadratické řešení.

To však lze ještě zrychlit použitím vhodné datové struktury.

Stavy si budeme udržovat v binárním vyhledávacím stromu, kde klíčem bude aktuální natočení Jirky. Výseče, ve kterých Jirka nespadne, jsou intervaly v tomto stromu. Ze stromu chceme vždy odebrat stavy, které nejsou v ani jedné z výsečí. Tyto stavy tvoří dva souvislé úseky (ve skutečnosti mohou být až tři, jelikož úsek může jít přes úhel 2π, fyzicky tedy bude rozdělen mezi začátek a konec stromu). Najít konce těchto úseků můžeme v $\mathcal{O}(\log n)$, stejně tak odebrat každý prvek v těchto úsecích. Následně dva stavy přidáváme, což je také logaritmická složitost.

Celkem provádíme $\mathcal{O}(n)$ hledání konců úseků, přidáváme $\mathcal{O}(n)$ vrcholů a zřejmě každý vrchol, který mažeme, jsme museli přidat, tedy mažeme nejvýše $\mathcal{O}(n)$-krát. Proto je celková časová složitost řešení $\mathcal{O}(n \log n)$.

Prostorová je určena binárním vyhledávacím stromem, tedy $O(n)$.

Úlohu připravil: Ondra Sladký

34-3-4 Horňáci a Dolňáci

Situaci můžeme popsat grafem. Vrcholy budou obyvatelé městečka. Kdykoliv Hercule viděl nějaké dva lidi hádat se, nakreslíme mezi příslušnými vrcholy hranu. Rozdělení na Horňáky a Dolňáky pak odpovídá nějakému *obarvení grafu dvěma barvami* – každý vrchol chceme obarvit červeně (Horňák) nebo zeleně (Dolňák) tak, aby hrany vedly jenom mezi vrcholy různých barev. Grafům, které se dají takto obarvit, se říká *bipartitní* (česky bychom mohli říci „dvojčásťové“).

Testování bipartitnosti

Nejprve se zamyslíme nad tím, jak o grafu zjistit, jestli je bipartitní. Cesty určitě jsou bipartitní (stačí vrcholy barvit nastřídačku), sudé cykly také, liché cykly nikoliv (barvení nastřídačku se pokazí, když se vrátíme tam, kde jsme začali barvit). Pokud se tedy v nějakém složitém grafu vysktyne lichý cyklus, hned je jasné, že graf není bipartitní.

Teď ukážeme, že liché cykly jsou jedinou překážkou bipartitnosti. Popíšeme jednoduchý algoritmus, který pro libovolný graf bez lichých cyklů najde obarvení dvěma barvami.

Bez újmy na obecnosti předpokládejme, že graf je souvislý – kdyby nebyl, stačilo by ho obarvit po komponentách. Každý souvislý graf má *kostru* – to je podgraf bez cyklů (tedy strom) propojující všechny vrcholy.

Kostru (stejně jako všechny ostatní stromy) můžeme obarvit snadno. Zakořeníme si ji v nějakém vrcholu; na to se hodí hostinský, který má známou barvu. Pak z kořene spustíme prohledávání grafu do hloubky nebo do šířky a budeme si udržovat ujitou vzdálenost. Vrcholy v sudé vzdálenosti obarvíme stejně jako kořen, ty v liché opačnou barvou.

Ale pozor: v grafu mohla být spousta dalších hran, které jsme do kostry nezahrnuli. Co kdyby některá z nich spojovala nějaké vrcholy u a v stejné barvy? Tehdy se podíváme na cestu mezi u a v ve stromu (ta je jednoznačné určená).

Jelikož u a v mají stejnou barvu, musí na cestě mezi nimi být sudý počet hran. K této cestě přidáme hranu uv a hned je na světě lichá kružnice.

Pokud v grafu žádná lichá kružnice není, i hrany mimo kostru spojují vrcholy různých barev, takže obarvení je korektní.

Zbývá rozebrat časovou složitost. Označíme-li n počet vrcholů a m počet hran, algoritmus doběhne v čase $\mathcal{O}(n+m)$, tedy lineárním. Kostru totiž najdeme prohledáním grafu do hloubky nebo do šířky, které je lineární. A stejným prohledáním rovnou můžeme barvit její vrcholy a kontrolovat hrany, které se do kostry nedostaly.

Pomalé řešení

Hercule má ovšem graf, který není bipartitní, a ptá se, jakou hranu může smazat, aby se bipartitním stal. Potřebujeme se zbavit všech lichých kružnic, tedy najít hranu, která leží na všech lichých kružnicích.

Nabízí se jedno přímočaré řešení: Najdeme jednu lichou kružnici a pro každou její hranu zkusíme, zda smazáním nedostaneme bipartitní graf. To má složitost $\mathcal{O}(nm)$, protože kružnice obsahuje nejvýše n hran. Nic moc, ale je to lepší než zkoušet mazat úplně všechny hrany grafu.

Sudé a liché hrany

Rychlejší řešení bude založeno na prohledávání grafu do hloubky (DFS). Zopakujme si, jaké role v něm mohou hrát jednotlivé hrany. V neorientovaném grafu každou hranu potkáme dvakrát. Podívejme se, v jakých rolích:

- *stromová hrana* – když na ni narazíme poprvé, vede do nově objeveného vrcholu. Po takové hraně se zavoláme rekurzivně. Všechny stromové hrany dohromady tvoří kostru grafu. Pokud stromovou hranu objevíme podruhé, vede do vrcholu „nad námi“ (na cestě do kořene), takže ji přeskočíme.
- *zpětná hrana* – když ji potkáme poprvé, vede do už označeného vrcholu „nad námi“. Z tohoto vrcholu vede cesta ze stromových hran (v kostře) do aktuálního vrcholu, která společně se zpětnou hranou tvoří kružnici. Až zpětnou hranu potkáme podruhé, povede do už navštíveného vrcholu „někde pod námi“.

Předchozí algoritmus na testování bipartitnosti by se tedy dal formulovat jako DFS, které každému vrcholu při první návštěvě nastaví vzdálenost od kořene po stromových hranách a podle její parity pak barvu. Pak u každé zpětné hrany uv zkontroluje, zda se barvy vrcholů u a v liší. Pakliže ano, hrana uv uzavírá sudou kružnici, takže ji budeme říkat *sudá hrana.* Pokud se neliší, prohlásíme ji za *lichou hranu* a víme, že graf není bipartitní. Stromové hrany nepovažujeme za sudé ani liché.

Kdyby v grafu existovala jen jediná lichá hrana, stačilo by ji smazat a obarvení by se stalo korektním.

Co když lichých hran existuje víc? Smazat kteroukoliv jednu nepomůže. Takže jediná další naděje je smazat nějakou stromovou hranu, která leží na všech lichých kružnicích.

Pokryté cesty

Pro každou zpětnou hranu uv uvážíme cestu mezi u a v ve stromu – té budeme říkat cesta *pokrytá* hranou uv. Najdeme průnik všech cest pokrytých lichými hranami. (To je mimochodem cesta, ale pozor, tohle platí jenom ve stromech). Smazáním kterékoliv hrany v průniku rozbijeme všechny kružnice uzavřené lichými hranami. To sice ještě nestačí (viz další kapitola), ale nejdřív ukážeme, jak průnik cest najít.

Každému vrcholu v přiřadíme nějaké celočíselné ohodnocení $h(v)$. Při návratu z vrcholu x budeme po hraně do jeho otce předávat součet $h(x)$ a součtů vrácených ze synů vrcholu x. Indukcí dostaneme, že tím jsme spočítali součet všech $h(v)$ přes vrcholy v v podstromu pod x.

Teď si představíme, co se stane, když pro nějakou zpětnou hranu z u do jeho předka v nastavíme $h(u) = 1$, $h(v) = -1$ a $h(\cdot) = 0$ všude jinde. Spočítané součty budou rovny 1 přesně na těch stromových hranách, které jsou pokryté zpětnou hranou. Všude jinde budou nulové.

Tím pádem pokud začneme s nulovými ohodnoceními, za každou lichou hranu uv zvýšíme $h(u)$ o 1 a snížíme $h(v)$ o 1, budou součty říkat, kolikrát je která stromová hrana pokryta lichými hranami. V hledaném průniku tedy leží ty stromové hrany, jejichž součet je roven počtu všech lichých hran.

Co způsobí smazání hrany

Máme tedy nějaký průnik cest pokrytých lichými hranami. Chceme smazat nějakou hranu xy z tohoto průniku (řekněme, že x je blíž ke kořeni než y). V kostře tuto hranu nahradíme některou z lichých hran, které pokrývají xy. Tím získáme zase kostru, ale musíme v celém podstromu pod y prohodit barvy.

Nyní se všechny původní liché hrany staly sudými (protože každá z nich pokrývala xy). Ale co když byla hrana xy pokrytá i nějakou sudou hranou? Z té by se stala lichá a ani nové obarvaní by nebylo korektní.

Smazat tedy můžeme jenom takovou stromovou hranu, která je pokrytá všemi lichými hranami a současně není pokryta žádnou sudou hranou. Pokrytí sudými hranami přitom spočítáme stejným algoritmem jako pro liché. (Dokonce bychom mohli součty počítat jen jednou a pro sudé hrany nastavit znaménka ohodnocení opačně než pro liché. Rozmyslete si, že to funguje.)

Voilà, algoritmus hotov. Zbývá rozebrat jeho složitost: DFS je samo o sobě lineární. Naše úpravy přidají konstantní čas ke zpracování každého vrcholu a hrany, čímž DFS pouze konstanta-krát zpomalí. Paměť postačí rovněž lineární: ke každému vrcholu a hraně si pamatujeme konstantní množství informací.

Úlohu připravili: Jirka Kalvoda, Martin „Medvěd“ Mareš

34-3-X1 Dráteníci

Nápověda zveřejněná po prvním termínu odevzdání:

Při řešení této úlohy se vám můžou hodit algoritmy řešící problém toků v síti.[1] *Graf, který bude vstupem algoritmu na toky, nemusí nutně být ten, který je na vstupu úlohy. Například může být i mnohem větší.*

Naším úkolem bylo nalézt nejmenší možný počet barev kabelů, které musíme použít k propojení centrálního routeru a domů v grafu tak, aby ke každému domu vedl jeden nevětvící se kabel a aby skrz žádnou hranu nevedly dva kabely stejné barvy. Navíc jsme měli nejen odpovědět nejmenším možným počtem barev K, ale zároveň jsme měli i takové

[1] `http://ksp.mff.cuni.cz/viz/kucharky/toky-v-sitich`

rozložení kabelů nalézt. Pojďme si úlohu postupně vyřešit přes několik zjednodušených verzí.

Stačí K barev?

Vezměme si nejprve zjednodušenou verzi úlohy: Rozhodnout, zda K různých barev kabelů stačí. Protřelý řešitel už v problému jistě začíná vidět toky v sítích (a kdo toky v sítích nezná, může si je připomenout v naší tokové kuchařce).[2] Ale přímočará aplikace toků nám nepomůže. Musíme si nejprve postavit vhodnou síť, která bude správně reprezentovat naši úlohu.

V typické tokové úloze hledáme maximální tok z jednoho zdroje do jednoho stoku skrz graf s hranami ohodnocenými jejich kapacitou. Ford-Fulkersonovo algoritmem zmíněným v kuchařce umíme takovou úlohu vyřešit v čase $\mathcal{O}(NM^2)$, kde N je počet vrcholů a M je počet hran. O něco lepším Dinicovo algoritmem, zmíněným například v Průvodci labyrintem algoritmů,[3] se lze dostat na čas $\mathcal{O}(N^2M)$.

První budeme muset vymyslet reprezentaci *stoku*. Do každého domu chceme natáhnout právě jeden kabel, takže nám bude stačit vyrobit si jeden virtuální stok a ten napojit na každý z domů hranou s kapacitou jedna. Jako *zdroj* použijeme router.

Pokud bychom hledali odpověď jen pro $K = 1$, tak už bychom byli skoro hotovi. Stačilo by nám jen zajistit, aby žádnou hranou neprošlo více kabelů, což bychom zajistili nastavením kapacit všech hran na jedna. Pak bychom v takovéto síti našli maximální tok a jestli by měl hodnotu odpovídající počtu domů, tak bychom věděli, že řešení existuje (do každého domu může vést jen jeden kabel, každou hranou také, a pokud se ze zdroje do stoku dostane tok velikosti F, víme, že existuje F vrcholově disjunktních cest do F domů).

Jak ale najít odpověď pro větší K? Nemůžeme jen zvednout kapacitu hran na K, to nám nijak nezajistí to, že každý z K kabelů na hraně bude mít jinou barvu. Musíme na to chytřeji. Zduplikujeme si síť jednou pro každou z K barev a vhodně je spojíme dohromady.

Nejdříve pojďme vyřešit domy. Je nám jedno, jak barevný kabel nám do domu povede, potřebujeme jen, aby byl právě jeden. Proto si pro každý dům vyrobíme virtuální vrchol, který spojíme hranami s kapacitou jedna se všemi K kopiemi domu v K sítích, a pak hranou s kapacitou jedna se stokem. Tak nám skrz dům do stoku povede nejvýše jeden kabel jedné barvy a tato podmínka je splněna.

Dále vyřešíme router. Pořídíme si virtuální zdroj, který spojíme hranou o kapacitě nekonečno (pro praktické účely stačí i kapacita rovná počtu domů) se všemi K kopiemi routeru ve všech K sítích. Tím v extrému umožníme, aby všechny kabely mohly být stejné barvy, pokud to půjde.

V každé kopii sítě nastavíme všem zbylým hranám kapacitu na jedna a opět v takto vzniklé síti najdeme maximální tok. Každá z K kopií sítě nám reprezentuje kabely jedné barvy a uvnitř sítě je kapacitami hran vynucena podmínka, že skrz stejnou hranu nepovede více kabelů stejné barvy. Pokud je v takové síti maximální tok roven počtu domů, řešení existuje.

Jak dlouho jedno takovéto otestování trvá? Pokud si počet domů odhadneme počtem všech vrcholů N, tak bude mít duplikovaný graf nejvýše $KN + N + 2$ vrcholů a $KM + 2KN + K$ hran, což můžeme odhadovat jako $\mathcal{O}(KN)$ hran a $\mathcal{O}(KM)$ vrcholů. Konstrukce grafu nám zabere lineární čas vzhledem k jeho velikosti, takže největší položka bude samotné hledání maximálního toku. S použitím Dinicova algoritmu se dostaneme na časovou složitost $\mathcal{O}((KN)^2KM)$.

Nalezení nejmenšího K

Pro nalezení nejmenšího K, pro které ještě má úloha řešení, použijeme klasický trik s binárním vyhledáváním.

Začneme s $K_0 = 1$ a budeme ho postupně zvětšovat na dvojnásobek, dokud nám bude algoritmus z předchozí části vracet negativní výsledek. Poté, co poprvé zjistíme, že pro nějaké K_i již řešení existuje, dostaneme rozsah K_{i-1} až K_i, ve kterém leží námi hledané K. V tomto rozsahu pak zahájíme klasické půlení a budeme postupovat tak dlouho, dokud se nedostaneme na krok velikosti jedna a nenajdeme K takové, že pro něj řešení existuje, ale pro $K - 1$ už ne.

Jak dlouho bude takové vyhledání trvat? Uděláme odhad podle výsledné hodnoty K. V první části hledání postupně zvětšujeme K_i, dokud poprvé nepřekročíme K, které „přestřelíme“ i v nejhorším případě nejvýše na dvojnásobek. Každým krokem K_i zdvojnásobíme, takže uděláme nejvýše $\mathcal{O}(\log_K)$ kroků. Ve druhé části už provádíme klasické binární vyhledávání, které také zabere $\mathcal{O}(\log K)$ kroků.

V každém kroku provedeme výše popsaný algoritmus. Pokud nám tedy K vyjadřuje výsledný počet barev drátů, tak můžeme časovou složitost celého algoritmu odhadnout jako $\mathcal{O}((KN)^2KM \log K)$.

Vypsání jednotlivých cest kabelů

Poslední drobnost, která nám zbývá, je vypsat trasy jednotlivých kabelů. K tomu si vezmeme síť z běhu výše popsaného algoritmu pro číslo K, ve které vždy jednoduchým průchodem vypíšeme trasu jednoho kabelu. Začneme vždy od routeru v jedné z kopií původní sítě (což nám určí barvu kabelu) a vydáme se postupně po libovolných nenulových hranách, cestou přitom odečítáme všude jedničku. Takto pokračujeme, dokud nějaké cesty existují.

Nalezneme takto nejvýše N kabelů a každý bude dlouhý nejvýše M hran, takže se časová složitost vypsání bez problému vejde do časové složitosti celého algoritmu. A tím máme celou úlohu vyřešenou.

Úlohu připravili: Jirka Kalvoda, Martin „Medvěd“ Mareš, Pali Rohár, Jirka Setnička

[2] `http://ksp.mff.cuni.cz/viz/kucharky/toky-v-sitich`

[3] `http://pruvodce.ucw.cz/`

Výsledková listina třetí série třicátého čtvrtého ročníku KSP

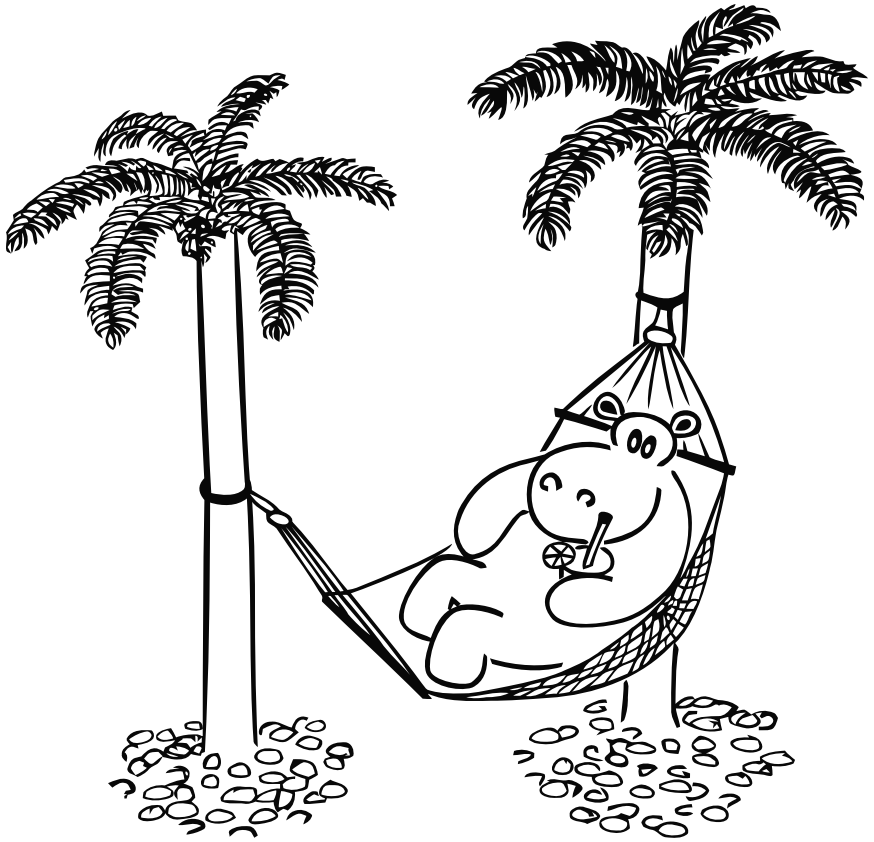
	řešitel	*škola*	*ročník*	*sérií*	*3-1*	*3-2*	*3-3*	*3-4*	*3-S*	*3-X1*	*série*	*KSP-X*	*celkem*
0.					10	11	12	12	15	10	60,0	36,0	180,0
1.	Benjamin Swart	MensaG	3	3	10	11	12	11	15,5	10	59,5	34,5	174,5
2.	Daniel Skýpala	GTomkovaOL	4	23	10	11	12	12	15	4	60,0	12,0	172,5
3.	Robert Jaworski	GÚstavníPH	4	10	10	11	12	0	16,5	10	49,5	16,0	157,5
4.	Lukáš Tomoszek	GTři	4	4	10	11			14		35,0	4,0	136,4
5.	Jakub Ondroušek	GTomkovaOL	2	8	10	4	3		15		32,0	0,0	126,0
6.	Jáchym Kouba	GJŠkodyPŘ	2	3	10	11	5	1	15		42,0	4,0	124,5
7.	Jan Kotovský	GPísnickáPH	3	9	10	4	3	4	16		37,0	0,0	120,5
8.	Adam Kolník	SSŠVTPraha	3	8	10	11			10,5		31,5	8,0	115,5
9.	Lukáš Veškrna	GKepleraPH	4	8	10	11					21,0	0,0	96,5
10.	David Kolář	GJírovcČB	3	3	10	11	12	4			37,0	0,0	92,5
11.	Jan Slíva	MensaG	1	3	10	11	9	4			34,0	0,5	85,5
12.	Prokop Randáček	GFXŠaldyLI	3	9	10						10,0	0,0	80,0
13.	Vít Skalický	GPísnickáPH	4	22	10	11					21,0	0,0	77,0
14.	Vladimír Sklenár	GTerVans	2	3	10	2		1	10		23,0	4,0	74,0
15.	Viktor Číhal	SPŠSmíchov	2	3	10	11	3				24,0	0,0	73,5
16.	Alex Olivier Michaud	GJírovcČB	3	3	10	11	7	4			32,0	0,5	70,0
17.	Richard Tichý	SG Kladno	0	3	10	11		2			23,0	0,5	69,0
18.	Ján Plachý	G VBN Prie	4	3	10						10,0	4,0	68,5
19.	Kryštof Maxera	GJírovcČB	1	5	10	6		2			18,0	0,0	60,0
20.	Jan Černohorský	G Brandýs	4	4					15,5		15,5	0,0	59,5
21.	Petr Šícho	GKepleraPH	4	4							0,0	0,0	55,0
22.–23.	Patrik Číhal	SŠKKamPard	2	2	10	11	8	2	15		46,0	0,0	54,5
	Jakub Mikeš	GJŠkodyPŘ	4	3					8		8,0	0,0	54,5
24.	Vojtěch Lančarič	SPŠG Třebešín	3	2	10	7	9				26,0	0,0	50,0
25.–26.	Patrik Herman	GTomkovaOL	3	8							0,0	0,0	44,5
	Jiří Kvapil	GTomkovaOL	4	20							0,0	0,0	44,5
27.	Eliška Macáková	CENADA BA	2	4							0,0	0,0	43,5
28.	Adam Kuča	PORG Krč	4	3	10	11	3	6			30,0	1,0	42,0
29.	Matúš Duchyňa	GGrössBA	3	3	10						10,0	0,0	36,0
30.	Daniel Šoltýs	GTřeKošice	4	6	7	1					8,0	0,0	32,0
31.	Nikolay Fomichev	SSŠVTPraha	3	2							0,0	0,0	26,0
32.	Ivan Trenčanský	GLSáru	3	2	10	5					15,0	0,0	24,0
33.	Veronika Jůzková	MensaG	4	6	10	2	1	1			14,0	0,0	23,0
34.–35.	Martin Belluš	GGrössBA	3	1							0,0	0,0	22,0
	Filip Siviček	GTimLučen	3	1	7	11	3	1		0	22,0	0,0	22,0
36.–37.	Petr Hladík	GMikulášPL	4	4	10	11					21,0	0,0	21,0
	Bobur Toshtemirov	GMikulášPL	3	2	10	11					21,0	0,0	21,0
38.–40.	Jonáš Dej	G Wicht	3	1							0,0	0,0	18,0
	Honza Kocourek	ParkLane	2	1	10	7	1				18,0	0,0	18,0
	Matúš Púll	GZborovPH	2	2	2	4	0				6,0	0,0	18,0
41.	Jakub Švojgr	GČeskáČB	3	1							0,0	0,0	14,0
42.–43.	Michal Pavlíček	MendelGOP	4	2							0,0	0,0	12,0
	Ondřej Skácel	GTomkovaOL	3	6							0,0	0,0	12,0
44.–47.	Michal Bernat	GZborovPH	2	1		11					11,0	0,0	11,0
	Vojtěch Franc	GArabskáPH	2	1							0,0	0,0	11,0
	Matouš Mišta	GTomkovaOL	1	1		11					11,0	0,0	11,0
	Filip Neubauer	AkademGPH	2	1	10	1					11,0	0,0	11,0
48.–49.	Zuzana Aubrechtová	GHeyrovPH	3	1	10						10,0	0,0	10,0
	Pavel Jordán	GPOA Znojmo	3	3							0,0	0,0	10,0
50.–51.	Martin Fof	MendelGOP	4	1							0,0	0,0	8,0
	Jakub Kopčil	GMikulášPL	3	1							0,0	0,0	8,0
52.	Michal Martínek	GÚstavníPH	1	1	7						7,0	0,0	7,0
53.	Jonáš Menšík	GJŠkodyPŘ	0	1	4	2					6,0	0,0	6,0
54.	Marek Maškarinec	SPŠEMasLI	1	1							0,0	0,0	4,0
55.	Jan Šuráň	GZborovPH	4	1							0,0	0,0	3,0

	řešitel	*škola*	*ročník*	*sérií*	*3-1*	*3-2*	*3-3*	*3-4*	*3-S*	*3-X1*	*série*	*KSP-X*	*celkem*
56.	Albert Bakoč	GZborovPH	1	1		2					2,0	0,0	2,0
57.	Jakub Surga	ParkLane	4	6	1						1,0	0,0	1,0

Výsledková listina KSP-X po třetí sérii třicátého čtvrtého ročníku

	řešitel	*škola*	*ročník*	*sérií*	*1-X1*	*1-X2*	*2-X1*	*3-X1*	*celkem*
0.					8	8	10	10	36,0
1.	Benjamin Swart	MensaG	3	3	8,5	8	8	10	34,5
2.	Robert Jaworski	GÚstavníPH	4	10		4	2	10	16,0
3.	Daniel Skýpala	GTomkovaOL	4	23		8		4	12,0
4.	Adam Kolník	SSŠVTPraha	3	8		8			8,0
5.–8.	Jáchym Kouba	GJŠkodyPŘ	2	3		4			4,0
	Ján Plachý	G VBN Prie	4	3		4			4,0
	Vladimír Sklenár	GTerVans	2	3		4			4,0
	Lukáš Tomoszek	GTři	4	4		4			4,0
9.	Adam Kuča	PORG Krč	4	3			1		1,0
10.–12.	Alex Olivier Michaud	GJírovcČB	3	3	0,5				0,5
	Jan Slíva	MensaG	1	3	0,5	0			0,5
	Richard Tichý	SG Kladno	0	3	0,5				0,5

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.



KSP pro vás připravují studenti Matematicko-fyzikální fakulty Univerzity Karlovy.
Realizace projektu byla podpořena Ministerstvem školství, mládeže a tělovýchovy.

Webové stránky: *https://ksp.mff.cuni.cz/*
E-mail: *ksp@mff.cuni.cz*
Organizátoři a kontakty: *https://ksp.mff.cuni.cz/kontakty/*