Korespondenční Seminář z Programování

35. ročník | **KSP** | **Prosinec 2022**

Vzorová řešení druhé série třicátého pátého ročníku KSP

35-2-1 Kde je Dan?

Ještě než se pustíme do popisu řešení, pojďme si ujasnit několik věcí. Během popisu řešení budeme často mluvit o nějaké skupině lidí, co stojí vedle sebe (např. těch K lidí, které Dan vidí). Tuto skupinu lidí popíšeme pomocí intervalu pozic, na které stojí. Jelikož ale stojí v kroužku, tak rozlišujeme interval 3–6, což jsou pozice $3, 4, 5$ a 6, a interval 6–3, což jsou pozice $6, 7, 8, \ldots, N-1, N, 1, 2, 3$. Délkou takového intervalu rozumíme počet pozic, které pokrývá.

Přímočaré řešení

Jistě bychom mohli úlohu vyřešit tak, že vyzkoušíme všechny Danovy pozice. Ke každé si spočítáme, v jakém intervalu musí být pozice člověka, aby ho Dan viděl. Potom se podíváme, kolik organizátorů je v tomto intervalu, a pokud jich je X, tak jsme našli další možnou pozici Dana.

Všimněme si ale, že každé Danově pozici odpovídá jiný interval pozic délky K a taky že každému intervalu délky K odpovídá jedna Danova pozice. Nemusíme tedy procházet Danovy pozice, ale jenom možné intervaly délky K.

Spočítání počtu organizátorů v daném intervalu zvládneme jedním průchodem seznamu organizátorů, který má velikost M. Možných umístění intervalu je N. Takže výsledná složitost je $\mathcal{O}(NM)$.

To je hodně pomalé

Nešlo by to rychleji? Šlo. Vytvoříme si pole N jedniček a nul. Číslo na pozici N je jedna, pokud na pozici N stojí organizátor, jinak je nula. Pomocí tohoto pole můžeme počet organizátorů v určitém intervalu vyjádřit pomocí součtu čísel na odpovídajích pozicích. Takovéto pole si vytvoříme v čase $\mathcal{O}(N)$, všude dáme nuly, a pak projdeme seznam organizátorů a na odpovídající místa v poli zapíšeme jedničku, což trvá čas $\mathcal{O}(M)$. Celkový čas je ale $\mathcal{O}(N)$, jelikož $M \leq N$.

Zatím se zdá, že jsme si moc nepomohli. Když si vezmeme intervaly začínající na pozicích i a $i+1$, jejich velká část se překrývá (resp. překrývají se na $K-1$ pozicích). Tohoto překrývání můžeme využít pro rychlý přepočet součtu – odečteme číslo, které bylo ve starém intervalu, ale ne v novém, a přičteme to, které je v novém, ale nebylo ve starém.

Bereme-li tedy postupně intervaly 1–K, 2–$K+1$ atd., můžeme vždy v konstantním čase přepočítat počet organizátorů v aktuálním intervalu. To vede k času $\mathcal{O}(N)$.

Dodáváme, že počítat součet daného intervalu v nějakém poli jde v konstantním čase i pomocí prefixových součtů.[1]

Ale N může být velké

V pozdější vstupech bylo $N \approx 10^{15}$, což je pro předchozí řešení příliš velké. Nedalo by se s tím něco dělat?

Jelikož M je relativně malé, všimneme si, že organizátoři musí stát daleko od sebe. Respektive existují dlouhé intervaly, kde nestojí žádný organizátor. Tyto intervaly bychom mohli šikovně přeskakovat a tím si ušetřit čas. Organizátory si nejprve setřídíme podle jejich pozice. Na začátku umístíme zkoumaný interval tak, aby začínal těsně za pozicí prvního organizátora, a spočítáme, kolik je v něm organizátorů. Budeme si pamatovat, který organizátor je první v intervalu a který je první těsně za aktuálnim intervalem. Nyní se podíváme, co se stane, když intervalem budeme posouvat – přibyde dřív nový organizátor do intervalu, nebo z něj dřív nějaký vypadne? Na tuto pozici přeskočíme a upravíme aktuální počet organizátorů. Pokud během tohoto skoku byl počet organizátorů X, tak každá přeskočená pozice (i ta, ze které jsme skákali) je pozice, která splňuje zadání a musíme je tedy všechny započítat. Když se pozice intervalu vrátí na startovní pozici, tak skončíme.

Jak jsme zlepšili čas? No, na setřídění pozic organizátorů potřebujeme čas $\mathcal{O}(M \log M)$ a na následné posouvání intervalu potřebujeme čas $\mathcal{O}(M)$, jelikož každý organizátor právě jednou do intervalu vstoupí a právě jednou ho opustí. Výsledný čas je tedy $\mathcal{O}(M \log M)$.

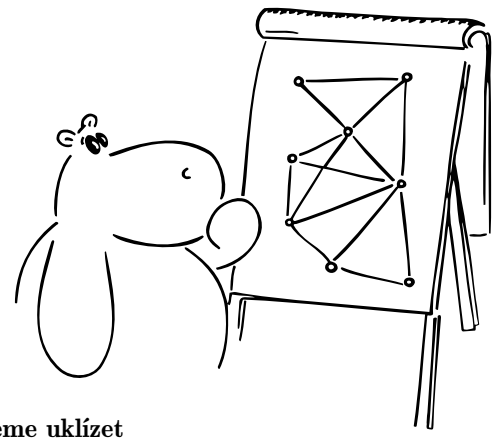
Program (Python):
`http://ksp.mff.cuni.cz/viz/35-2-1.py`

Program (C++):
`http://ksp.mff.cuni.cz/viz/35-2-1.cpp`

Úlohu připravili: Robert Jaworski, Ondra Sladký

35-2-2 Zápisky z přednášky

Když už máme v ruce haldu, tak proč ji rovnou nepoužít? Provedeme v této haldě K-krát operaci *extract_min* a vypíšeme K-tý odebraný prvek. Operace *extract_min* má časovou složitost $\mathcal{O}(\log N)$, tedy celkem $\mathcal{O}(K \log N)$



Nebudeme uklízet

Vzhledem k tomu, že je K výrazně menší než N, rádi bychom se zbavili závislosti na N. Na to si všimněme, že K-tý nejmenší prvek určitě bude nejhlouběji v K-té hladině. (Jinak by bylo nad ním alespoň K menších prvků, a potom by nebyl K-tý nejmenší.)

Proto si můžeme představit, že halda obsahuje pouze těchto K vrstev. Při operaci *extract_min* zastavíme probublávání ve chvíli, kdy se dostaneme do K té hladiny. Hlouběji náš hledaný prvek již nebude, tudíž tam už můžeme nechat naši haldu rozbitou.

[1] `http://ksp.mff.cuni.cz/viz/kucharky/zakladni-algoritmy`

A jakou toto má časovou složitost? Uděláme K operací *extract_min* a každá z nich projde K hladin, tedy celková složitost bude $\mathcal{O}(K^2)$.

Když jedna halda nestačí...

Podívejeme se na naši situaci jinak. Uvažujme, že máme množinu L nejmenších prvků v naší haldě. Pokud bychom chtěli tuto množinu rozšířit, tak nám do množiny stačí přidat nejmenší prvek mimo ní. Ovšem prvek nad přidávaným vrcholem musí být menší než on, tedy musí být menší než nejmenší prvek mimo L. Přidaný vrchol tedy musí být synem nějakého vrcholu z L.

Opakovaným procházením všech synů a postupným přidáváním se dostaneme na složitost $\mathcal{O}(K^2)$, ale všimněme si, že množina synů se mění jen málo – konkrétně jeden prvek odebereme a dva prvky přidáme. Tedy můžeme si naše prvky udržovat v nějaké šikovné datové struktuře – třeba haldě.

Pojďme si tedy pořídit (mini)haldu, ve které si budeme udržovat aktuální syny. Na začátku do minihaldy dáme kořen původní haldy. Poté $K-1$-krát budeme opakovat:

– Odstraníme nejmenší prvek z minihaldy.
– Přidáme jeho dva syny v původní haldě do minihaldy.

A na závěr vypíšeme nejmenší prvek z minihaldy.

A jakou toto bude mít složitost? Odstranění z minihaldy bude trvat $\mathcal{O}(\log M)$, kde M je velikost minihaldy. Přidání synů bude též trvat $\mathcal{O}(\log M)$. Ale jaká je velikost minihaldy? V každé z $K-1$ iterací přidáme 2 prvky, tedy v minihaldě bude nejvýše $\mathcal{O}(K)$ prvků. A odstranění a přidání provedeme $\mathcal{O}(K)$-krát, tedy výsledná složitost bude $\mathcal{O}(K \log K)$

Úlohu připravili: Jirka Kalvoda, Kristýna Petrlíková, Dan Skýpala

35-2-3 Hroší šanta

Grafy, grafy, všude grafy

Jak nám nadpis napovídá, můžeme problém převést na graf. Vrcholy budou dvojice objem vody v první nádobě a objem vody v druhé nádobě. Ty budeme značit (a, b). Hrany budou odpovídat jednotlivým akcím (nalití, vylití, přelití).

Nyní nám stačí najít nejkratší cestu z vrcholu $(0, 0)$ do kteréhokoliv z vrcholů (K, b) nebo (a, K), kde a, b můžou být libovolné. A hledání můžeme provést prohledáváním do šířky, pro jehož oživení můžete nahlédnout do kuchařky o grafech.[2]

Program (Python):
`http://ksp.mff.cuni.cz/viz/35-2-3a.py`

Pár důkazů

Předchozí algortitmus má časovou složitost $\mathcal{O}(AB)$. Nešlo by použít nějaký menší graf?

Podívejme se, jak se mění objemy vody ve džbánech. Pojďme si vzít například dva džbány s kapacitami 4 a 11:

$$(0, 0) \to (0, 11) \to (4, 7) \to (4, 0) \to \cdots$$

Zdá se, že vždy je alespoň jeden džbán plný nebo prázdný. Pojďme si to dokázat.

[2] `http://ksp.mff.cuni.cz/viz/kucharky/grafy`

Lemma o plnoprázdnosti: Vždy bude alespoň jeden ze džbánů plný nebo prázdný.

V začátečním stavu $(0, 0)$ to zřejmě platí. A poté pro jednotlivé operace:

- Po naplnění bude alespoň tento džbán plný.
- Po vylití bude alespoň tento džbán prázdný.
- Přelití skončí tak, že vyprázdníme jeden džbán nebo naplníme druhý. Tak či tak bude na konci jedna z podmínek splněna.

Tedy náš algoritmus může uvažovat pouze stavy, kde jedna z nádob je plná nebo prázdná. Takových stavů je $\mathcal{O}(A+B)$.

Pojďme naše řešení ještě vylepšit. Všimněme si, že celkový objem vody v obou nádobách můžeme napsat vyjádřit jako:

$$c_1 A + c_2 B$$

kde $c_1, c_2 \in \mathbb{Z}$. Proč to platí? Když přeléváme, zřejmě se stav vody nemění. Když plníme prázdný nebo vyléváme plný džbán, tak měníme příslušné c o 1. Pokud plníme nebo vyléváme z části naplněný džbán, tak druhý je plný nebo prázdný, a potom nový stav určitě půjde zapsat v tomto tvaru.

Můžeme si všimnout, že předchozí součet je určitě násobek $\gcd(A, B)$, takže nebude možné dosáhnout objemu, který není dělitelný největším společným dělitelem. A zase z lemmatu o plnoprázdnosti, jeden ze džbánů musí být plný, takže stavů bude nejvýše $\mathcal{O}\left((A + B)/\gcd(A, B)\right)$

Možností je málo

Ukážeme, že předchozí postup je zbytečně složitý. Stačí všimnout, že většinou je další akce jasně daná, protože ostatní akce by nám byly k ničemu.

Stav, kdy je první džbán plný nebo prázdný a druhý džbán je plný nebo prázdný nazvěme *triviální stav*. Do triviálních stavů se z počáteční pozice zvládneme dostat pomocí nejvýše dvou operací, nedává tedt smysl se tam vracet později.

Pojďme provést rozbor případů podle toho, jakou akci jsme právě provedli. Bez újmy na obecnosti první džbán je plný nebo prázdný, což můžeme předpokládat díky lemmatu o plnoprázdnosti a tomu, že na pořadí džbánů nezáleží.

- Poslední akce byla vylití džbánu (`a`). Potom stav je $(0, b)$ a akce `bB` nás dostanou do triviálního stavu, akce `>a` nic nedělají a na akci `A` jsme nemuseli džbán A vylívat. Jediná rozumná akce je tedy `<`.
- Poslední akce byla naplnění džbánu (`A`). Potom stav je (A, b) a akce `bB` nás dostanou do triviálního stavu, akce `<A` nic nedělají a na akci `a` jsme nemuseli A plnit. Jediná rozumná akce je tedy `>`.
- Poslední akce byla přelití džbánu (`>`). Stav může být jedna ze dvou možností:
  - (a, B): `aA` vedou do triviálního stavu, `>B` nic nedělají, pro `<` nebylo nutné předchozí `>`. Jediná rozumná akce je tedy `b`.
  - $(0, b)$: `bB` vedou do triviálního stavu, `>a` nic nedělají, pro `<` nebylo nutné předchozí `>`. Jediná rozumná akce je tedy `A`.

A na začátku máme dvě možnosti, co dělat: Nalít první nebo druhý džbán. Dále je zbytek akcí daný. Stačí nám procházet obě větve najednou, a neprojdeme určitě více než je velikost výstupu, takže dosáhneme optimální složitosti, totiž lineární s velikostí výstupu $O(V)$.

Program (Python):
`http://ksp.mff.cuni.cz/viz/35-2-3b.py`

Drobnost na závěr

Podívejme se ještě jednou na celkový objem vody ve džbánech a položme jej rovný K (za podmínky, že se K do jednoho džbánu vleze):

$$c_1 A + c_2 B = K$$

Tohle je ve skutečnosti Bézoutova identita, která je řešitelná Rozšířeným Eukleidovým algoritmem. (Co není v kuchařkách?[3])

Z Bézoutových koeficientů můžeme vyrobit přesný návod, jak dosáhnout výsledného postupu. Kladný koeficient říká, kolikrát máme daný džbán naplnit (z kohoutku, ne z druhého džbánu). Záporný koeficient říká, kolikrát máme daný džbán vylít (do dřezu).

Nejjednodušší je případ, kdy $c = 0$, tedy K je násobkem objemu jednoho ze džbánů. Poté nám stačí daný džbán plnit a přelévat do druhého.

V opačném případě jeden z koeficientů bude záporný. (Jinak by rovnice zřejmě neplatila, protože jeden ze džbánů má větší objem než K.) Potom stačí plnit džbán s kladným koeficientem a vždy přelít do druhého džbánu, dokud nebude plný. Potom, co ho bude plný, ho vylejeme, nalejeme do něj zbytek z prvního džbánu a začneme znova, dokud nedosáhneme kýženého výsledku.

Potom nám stačí najít takové dva koeficienty, které nám dají nejmenší počet akcí. Jsou dvě možnosti, totiž nejmenší kladné a největší záporné c_1 a k nim jejich odpovídající c_2. Celkový počet akcí potom bude $2(c_1 + c_2)$. Stačí obě možnosti porovnat a výslednou posloupnost akcí vygenerovat. Mimochodem to znamená, že počet akcí umíme najít v čase $\mathcal{O}(\log(\min(A, B)))$.

Úlohu připravili: Jirka Kvapil, Martin „Medvěd“ Mareš, Kristýna Petrlíková, Dan Skýpala

35-2-4 Kamínky

Přímočarý přístup

Přímočaré řešení spočívá v tom, že budeme řadu kamínků opakovaně procházet a odstraňovat sousední stejnobarevné kamínky. Složitost tohoto přístupu je však $\mathcal{O}(N^2)$.

Pokud má úloha řešení, určitě má i řešení s maximálním počtem hladových odebrání, tedy takových, která vyřadí první stejnobarevnou dvojici. Představme si tedy, že existuje řešení, které přeskočilo nějakou stejnobarevnou dvojici. Zaměřme se na tu nejlevější dvojici takto neodebraných kamínků k_1 a k_2. Pokud se jedná o platné řešení, znamená to, že k_1 se v řešení odebírá společně s nějakým q_1 a k_2 společně s nějakým q_2. Můžeme nahlédnout, že lze tyto čtyři kamínky eliminovat v jiném uspořádání, to jest k_1 spolu s k_2 a q_1 spolu s q_2, čímž jsme dosáhli řešení, které dělá více hladových odebrání.

Tak jsme tedy dokázali, že řešení s maximálním počtem hladových odebrání je to řešení, které dělá hladové odebrání vždy.

Řešení s použitím zásobníku

Vytvoříme si zásobník a budeme do něj postupně ukládat navštívené barvy. V případě, že bychom měli do zásobníku přidat kamínek barvy b a v zásobníku byl současný vrchní kamínek také barvy b, prostě je oba odstraníme.

Takto projedeme celou posloupnost a pokud v zásobníku na konci něco zbylo, má pravdu Světlana a není možné vyhrát. Pokud je však zásobník na konci prázdný, znamená to, že Khebir vyhrát může.

Protože jsme pro každý prvek posloupnosti vykonali řádově $\mathcal{O}(1)$ operací, má celý algoritmus složitost $\mathcal{O}(N)$.

Proč algoritmus funguje?

V zásobníku máme vždy uloženy kamínky tak, že zde nejsou dva kamínky stejné barvy vedle sebe. Na začátku zásobník tuto podmínku splňuje a vždy, když přidáme nový kamínek, tak odstraníme poslední dvojici, pokud jsou to dva stejné vedle sebe (a na zbytku zásobníku už různobarevnost sousedů předpokládáme). Tudíž se tato vlastnost zachovává.

Když projdeme celou posloupnost a takto jí vložíme do zásobníku, máme na konci zajištěné, že v zásobníku je posloupnost, ve které nejsou dvě stejné barvy vedle sebe, a do zásobníku byly přidány všechny kamínky. Pokud je tedy na konci délka zásobníku nenulová, znamená to, že už žádnou další eliminaci provést nelze a tudíž nelze vyhrát, naopak pokud je zásobník na konci prázdný, znamená to, že všechny kamínky vyřadit lze.

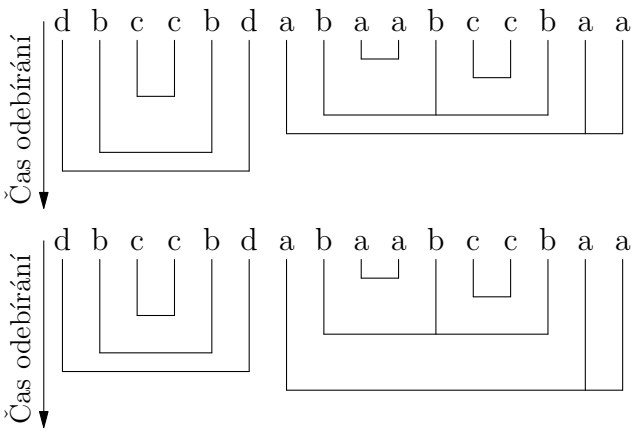
Úlohu připravili: Martin Koreček, Dan Skýpala, Lukáš Veškrna

35-2-X1 Oblázky

Začneme jednoduchým pozorováním: Když máme posloupnost sousedních kamínků stejné barvy delší než jedna, tak se jí umíme zbavit. Existuje spousta možností jak na to. Například sudé posloupnosti budeme odebírat po dvou a u lichých začneme odebráním trojice, pokud něco zbude, tak můžeme opakovat odebírání dvojic. Můžeme tedy povolit odebírat i více než dva nebo tři kamínky a hra se tím nijak nezmění.

Dále nahlédneme, že můžeme předpokládat o posledním kamínku, že bude odebrán jako poslední. Podívejme se na nějakou validní posloupnost tahů jak odebrat všechny kamínky. Poslední kamínek bude odebraný spolu s nějakými dalšími stejné barvy (těm budeme říkat kamarádi). Všechny kamínky mezi kamarády a posledním musely být odebrány ještě před posledním (jinak by nešlo provést uvažované odebrání). Po odebrání už tedy zbudou jen kamínky nalevo od prvního kamaráda. Libovolná následující operace tedy už může odstraňovat pouze kamínky z této části. Vůbec tedy nezáleží na tom, jestli tyto operace proběhnou před nebo po odebrání nejpravějšího kamínku. Můžeme tedy operaci odebírající poslední kamínek provést až úplně nakonec a stále máme validní způsob odebírání.

[3] `http://ksp.mff.cuni.cz/viz/kucharky/teorie-cisel`

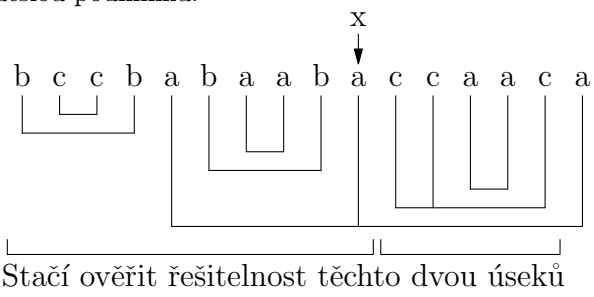


Nyní již k samotnému algoritmu: Využijeme dynamického programování. Pro každý souvislý úsek kamínků určíme, jestli jde nebo nejde odstranit. Prázdné úseky jsou vždy řešitelné, protože již není co odstraňovat. Při počítání nějakého úseku již budeme mít spočteny všechny jeho souvislé podúseky. To můžeme snadno zařídit tím, že budeme počítat od nejkratších úseků k delším. Zkusíme všechny možnosti, jak mohl vypadat poslední tah při odstraňování daného úseku. Tedy vyzkoušíme všechny dvojice nebo trojice kamínků stejné barvy obsahující poslední kamínek. Když alespoň jedna bude fungovat, tak máme vyhráno. Všechny ostatní kamínky úseku již musely být odstraněny. Ovšem kamínky odstraněné v posledním tahu tvoří jakousi bariéru: části úseku, které oddělují, spolu nemohly nijak interagovat, a tedy každá z částí (což je zase úsek) musela být odstraněna samostatně. Jestli bylo možné odstranit každou z částí zjistíme z již spočítaných hodnot dynamického programování.

Pokud je celá posloupnost řešitelná, tak snadno zvládneme zjistit výherní posloupnost tahů. U každého podúseku si zapamatujeme, jakým posledním tahem jsem naši řešení, a pak využijeme jednoduché rekurzivní funkce, která se nejprve zarekurzí na všechny úseky, které odděluje poslední tah, a pak vypíšeme i onen tah.

Jak to bude rychlé? Počítáme $\Theta(N^2)$ souvislých podúseků. V každém ovšem zkoušíme spoustu možností posledního tahu. Pokud budeme počítat s původní hrou bez žádných pozorování, tak máme $\mathcal{O}(N^3 + N^2)$ možností posledního tahu a každou z nich je nutné ověřit. Celkem tedy algoritmus běží v $\mathcal{O}(N^5)$. Když využijeme druhé pozorování, tak víme, že stačí zkoušet pouze poslední kamínek a jeden nebo dva z předešlých. Celkem tedy $\mathcal{O}(N^4)$.

Nyní pojďme uvažovat nad hrou, kde je možno odebrat i větší celky než trojice (v úvodním pozorování jsme si uvědomili, že je to ekvivalentní). To na první pohled vypadá, že jsme si vůbec nepomohli, protože teď je třeba uvažovat až exponenciálně množností posledního tahu. Pojďme to napravit. Zvlášť uvážíme možnost, že poslední tah odebral dvojici. Takové tahy prostě vyzkoušíme v lineárním čase. Zbývají tedy už jen odstranění alespoň trojiček. Podívejme se na předposlední z odstraněných kamínků v posledním tahu (označme x). Všimneme si, že úsek od začátku po x (včetně) musí být také řešitelný. Můžeme odstranit všechny úseky oddělené posledním tahem stejným způsobem a pak jen použijeme stejný poslední tah bez posledního kamínku. Jelikož uvažujeme jen poslední tahy odstraňující alespoň tři kamínky, tak nám zůstane stále validní tah. Ovšem toto platí i opačně. Když máme nějaké x stejné barvy jako poslední kamínek a úseky od začátku po x (včetně) a od x (mimo) po poslední kamínek (mimo) jsou řešitelné, tak i to celé je řešitelné. Důkaz ponecháváme čtenáři jako cvičení. Stačí tedy jen vyzkoušet všechny stejně obarvené x a ověřit předešlou podmínku.



Celková časová složitost tohoto řešení je $\mathcal{O}(N^3)$. Paměťová je $\mathcal{O}(N^2)$.

Úlohu připravili: Jirka Kalvoda, Dan Skýpala

35-2-S Lineární zobrazení

Úkol 1 – **Nelineární zobrazení** [2b]:

Těchto zobrazení existuje mnoho. My si ukážeme dvě jednoduchá.

Pro první zobrazení si představme, že máme obrázek nakreslený na průhledné fólii. Tuto fólii přehneme podle vodorovné osy, čímž se spodní část obrázku zrcadlově převrátí. Takové zobrazení lze zapsat pomocí absolutní hodnoty:

$$f((x_1, x_2)^\top) = (x_1, |x_2|)^\top$$

Absolutní hodnota očividně není lineární funkce, ale přesto ověříme, že naše zobrazení není lineární. Dokážeme najít protipříklad jak pro součet:

$$f((-3,3)^\top + (3,-3)^\top) = f((0,0)^\top) = (0,0)^\top \neq$$

$$f((-3,3)^\top) + f((3,-3)^\top) = (-3,3)^\top + (3,3)^\top = (0,6)^\top$$

Tak i pro násobek skalárem:

$$f(-1 \cdot (1,1)^\top) = (-1,1)^\top \neq -1 \cdot f((1,1)^\top) = (-1,-1)^\top$$

Druhé zobrazení je podobně přímočaré, obrázek o kus posuneme, konkrétně ve směru nějakého vektoru $\mathbf{a}$.

$$f((x_1, x_2)^\top) = (x_1 + a_1, x_2 + a_2)^\top$$

Ač se posunutí může zdát jako přirozená transformace, definici linearity nesplňuje. Ověřme to pro $\mathbf{a} = (1,1)^\top$:

$$f((-3,3)^\top + (3,-3)^\top) = f((0,0)^\top) = (1,1)^\top \neq$$

$$f((-3,3)^\top) + f((3,-3)^\top) = (-2,4)^\top + (4,-2)^\top = (2,2)^\top$$

$$f(-1 \cdot (1,1)^\top) = (0,0)^\top \neq -1 \cdot f((1,1)^\top) = (-2,-2)^\top$$

Můžeme jednoduše nahlédnout, že dané zobrazení není lineární, kdykoli vektor $\mathbf{a}$ není nulový.

Úkol 2 – **Fibonacciho čísla** [5b]:

Nejdříve potřebujeme vytvořit matici $\mathbf{Q}$, která bude generovat postupně další Fibonacciho čísla. Tedy když uděláme součin matice $\mathbf{Q}$ a vektoru $(F_n, F_{n+1})^\top$, chceme získat vektor $(F_{n+1}, F_{n+2})^\top$ neboli

$$\mathbf{Q} \cdot \begin{pmatrix} F_n \\ F_{n+1} \end{pmatrix} = \begin{pmatrix} F_{n+1} \\ F_n + F_{n+1} \end{pmatrix}.$$

První řádek matice je jednoduchý, chceme okopírovat druhou složku vektoru do první složky. Hledáme tedy koeficienty, kterými vynásobit složky prvního vektoru:

$$F_{n+1} = 0 \cdot F_n + 1 \cdot F_{n+1}$$

Druhý řádek není o moc těžší, potřebujeme členy F_n, F_{n+1} sečíst.

$$F_n + F_{n+1} = 1 \cdot F_n + 1 \cdot F_{n+1}$$

Výsledná matice $\mathbf{Q}$ bude vypadat následovně:

$$\mathbf{Q} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}$$

Můžeme provést rychou kontrolu, že máme vše správně:

$$\begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} F_n \\ F_{n+1} \end{pmatrix} = \begin{pmatrix} F_{n+1} \\ F_n + F_{n+1} \end{pmatrix} = \begin{pmatrix} F_{n+1} \\ F_{n+2} \end{pmatrix}$$

Nyní můžeme začít řešit, jak vypočítat F_n. Budeme začínat s F_0 a F_1, tedy s vektorem $\mathbf{f}_0 = (0, 1)^\top$. Víme, že matice $\mathbf{Q}$ nám vypočítá následující člen. Když výsledný vektor vynásobíme znovu maticí $\mathbf{Q}$, tak dostaneme ještě další člen. Například F_3 bychom spočítali takto:

$$\mathbf{Q}\left(\mathbf{Q}\left(\mathbf{Q} \cdot \mathbf{f}_0\right)\right) = \mathbf{f}_3 = \begin{pmatrix} F_3 \\ F_4 \end{pmatrix}$$

Technicky by nám stačilo o jedno násobení maticí $\mathbf{Q}$ méně a výsledek bychom přečetli z druhé složky vypočteného vektoru. Poté bychom však museli ošetřit speciální případ pro výpočet F_0.

Jelikož je násobení matic asociativní, můžeme výpočet zapsat bez závorek:

$$\mathbf{Q} \cdot \mathbf{Q} \cdot \mathbf{Q} \cdot \mathbf{f}_0$$

Což lze ještě více zjednodušit na $\mathbf{Q}^3 \cdot \mathbf{f}_0$.

Náš obecný algoritmus tedy nejprve spočítá $\mathbf{Q}^n$, posléze touto maticí vynásobí vektor $\mathbf{f}_0$ a výsledek přečte z první složky.

V obecnosti násobení dvou čtvercových matic velikosti n podle definice trvá $\mathcal{O}(n^3)$, ale jelikož my násobíme jen matice 2×2 a velikost těchto matic není závislá na velikosti problému, který řešíme, tak jedno násobení budeme brát jako konstantní, tedy $\mathcal{O}(1)$. Pro výpočet $\mathbf{Q}^n$ potřebujeme n násobení, celkem tedy náš algoritmus běží v čase $\mathcal{O}(n)$.

To je však pomalé. Vždyť totéž umíme jednoduše bez lineární algebry! Zbývá poslední dílek skládanky – jak rychle vypočítat mocninu matice $\mathbf{Q}$.

Rychlé mocnění

Využijeme mocninový trik. Jistě ze střední školy znáte pravidlo sčítání mocnin:

$$3^a \cdot 3^b = 3^{a+b}$$

Tuto vlastnost využijeme obráceně. Dejte tomu, že chceme vypočítat 3^n, kde n je sudé. Pak stačí spočítat $3^{n/2}$, protože poté již dopočítáme $3^n = 3^{n/2} \cdot 3^{n/2}$.

Kdyby bylo n liché, spočítáme $3^{(n-1)/2}$. Díky tomu dopočítáme $3^n = 3^{(n-1)/2} \cdot 3^{(n-1)/2} \cdot 3$.

Spojením těchto dvou pravidel:

$$3^n = 3^{\lfloor n/2 \rfloor} \cdot 3^{\lfloor n/2 \rfloor} \cdot 3^{n \bmod 2}$$

Zápis $\lfloor n/2 \rfloor$ značí podíl zaokrouhlený dolů. Pro sudé n dostaneme $n/2$, pro liché n zase $(n-1)/2$, přesně jako výše. Dále zápis $n \bmod 2$ značí zbytek po dělení dvěma.

Pokud známe mocninu $3^{\lfloor n/2 \rfloor}$, výsledek spočítáme na konstantně mnoho operací. Pokud danou mocninu neznáme, tak ji můžeme rekurzivně vypočítat. V každém rekurzivním volání se mocnina sníží na půlku, celkem tedy budeme potřebovat $\log_2 n$ volání. Tedy celkový čas rychlého mocnění bude $\mathcal{O}(\log n)$.

Daný algoritmus jsme si ukazovali na číslu 3, pochopitelně však platí i pro matice, jen musíme provést maticové násobení. To v důsledku znamená, že n-té Fibonacciho číslo dokážeme vypočítat v čase $\mathcal{O}(\log n)$.

Úkol 3 – Dosažitelnost [5b]:

Umíme spočítat sledy délky ℓ z i do j. Pokud je tento počet pro nějaké ℓ nenulový, pak existuje cesta z i do j. První myšlenka tedy bude počítat $\mathbf{A}^\ell$ pro různá ℓ, hledat nenulové hodnoty a na příslušná místa v matici dosažitelnosti psát jedničky. Pro n vrcholů nám stačí projít $0 \leq \ell < n$, protože každá cesta bude mít nejvýše $n - 1$ hran.

Počet sledů však může růst exponenciálně, chceme se vyhnout počítání s takto velkými čísly. Naštěstí nepotřebujeme znát počet sledů přesně, zajímá nás pouze to, jestli je tento počet nenulový. Po každém kroku tedy všechny nenulové hodnoty v $\mathbf{A}^\ell$ změníme na jedničky.

Tento postup provede n násobení matic $n \times n$. Při použití násobení dle definice dostaneme časovou složitost $\mathcal{O}(n^4)$.

Nešlo by to lépe? Použijeme-li rychlé mocnění, zvládneme $\mathbf{A}^n$ spočítat jen na $\mathcal{O}(\log n)$ násobení. Jenže to nám moc nepomůže, potřebujeme i všechny mocniny mezi tím. Z matice $\mathbf{A}^n$ samotné totiž matici dosažitelnosti sestavit nedokážeme. Pokud například mezi vrcholy i, j existuje pouze jediný sled délky 1, bude $(\mathbf{A}^n)_{i,j} = 0$.

Jakmile je jednou pro nějaké ℓ počet sledů nenulový, chceme zařídit, aby už nenulový zůstal. Řešení je jednoduché, do grafu přidáme smyčky (hrany vedoucí zpět do téhož vrcholu). V matici sousednosti to znamená vyplnit diagonálu jedničkami.

S takto upravenou maticí $\mathbf{A}$ již můžeme použít rychlé mocnění. Přidáme nahrazování velkých čísel za jedničky a máme finální algoritmus. Jeho časová složitost bude $\mathcal{O}(n^3 \log n)$.

Jak je toto řešení vlastně rychlé? Srovnejme jej s přímočarým řešením, které z každého vrcholu spustí prohledávání do šířky. Jedno prohledání má složitost $\mathcal{O}(n + m)$, což by pro hustý graf s mnoho hranami bylo $\mathcal{O}(n^2)$. Prohledání ze všech vrcholů by zabralo $\mathcal{O}(n^3)$.

Zdá se, že naše sofistikované řešení je vlastně k ničemu. Stačí si však vzpomenout na Strassenův algoritmus, který naši složitost vylepší na $\mathcal{O}(n^{2.807} \log n)$, což už je lepší než přímočaré prohledávání. Nutno však podotknout, že pro praktické velikosti grafů je rozdíl minimální.

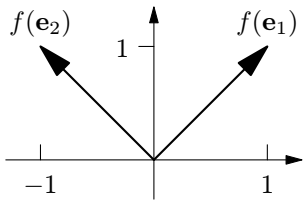
Úkol 4 – Obecná inverze [3b]:

Inverzi by jistě šlo spočítat řešením soustav rovnic. Zde však nabídneme geometrický náhled.

Konstanta c pouze obrázek zvětšuje, z matice ji můžeme vytknout.

$$\mathbf{A} = \begin{pmatrix} c & -c \\ c & c \end{pmatrix} = c \begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix}$$

Zjistíme, že matice otáčí o 45°. Navíc vektory ještě trochu protáhne, původní jednotkové vektory měly délku 1, nyní mají délku $\sqrt{1^2+1^2}=\sqrt{2}$.



Sestavme tedy obdobnou matici pro otočení v opačném směru:

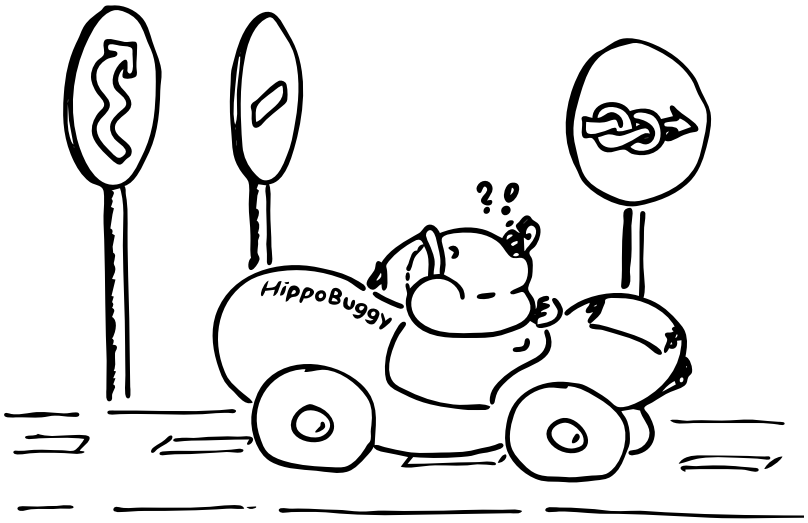
$$\mathbf{B}=\begin{pmatrix}1 & 1\\ -1 & 1\end{pmatrix}$$

To už je skoro hledaná inverzní matice. Ještě musíme zajistit, aby měly vektory správnou velikost. Matice $\mathbf{A}$ zvětšovala $c\sqrt{2}$-krát, matice $\mathbf{B}$ zvětšuje $\sqrt{2}$-krát. Podělíme tedy členem $2c$ a máme inverzi.

$$\mathbf{A}^{-1}=\frac{1}{2c}\mathbf{B}=\frac{1}{2c}\begin{pmatrix}1 & 1\\ -1 & 1\end{pmatrix}$$

Pro $c=0$ inverze existovat nemůže, protože taková matice $\mathbf{A}$ každý obrázek zmenší do jediného bodu.

Úlohu připravili: David Klement, Michal Kodad, Martin „Medvěd“ Mareš



Výsledková listina druhé série třicátého pátého ročníku KSP

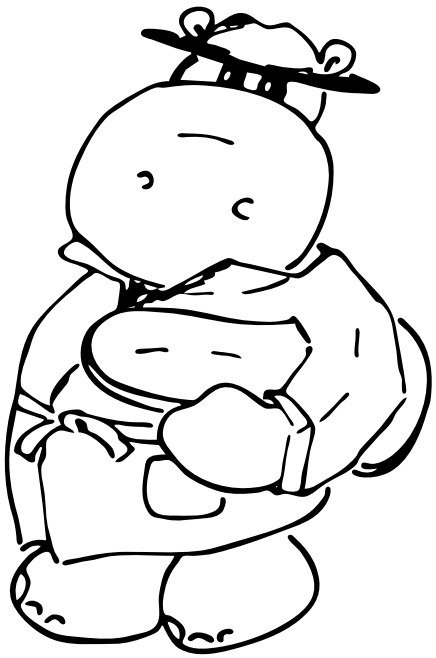
	řešitel	*škola*	*ročník*	*sérií*	*2-1*	*2-2*	*2-3*	*2-4*	*2-S*	*2-X1*	*série*	*KSP-X*	*celkem*
0.					11	12	10	12	15	10	60,0	20,0	120,0
1.	Benjamin Swart	MensaG	4	7	11	12	10	12	14,5	10	59,5	18,0	119,5
2.–3.	Vojtěch Lančarič	SPŠG Třebešín	4	7	11	12	10	12	14,5		59,5	0,0	117,5
	Štěpán Mikéska	GJarošeBO	4	2	11	12	10	12	15	8	60,0	9,0	117,5
4.	David Kolář	GJírovcČB	4	7	11	10	10	10	13,5		54,5	0,0	112,5
5.	Viktor Helmich	GTMannaPH	4	2	11	10	10	7	14		52,0	0,0	109,0
6.	Kateřina Doubková	GNAlejíPH	4	3	11	12	10	12	13,5		58,5	0,0	108,0
7.	Erik Ježek	SPŠSmíchov	1	2	11	12	10	12	14,5		59,5	5,0	107,0
8.	Filip Prášil	GJiříPoděb	4	3	11	12	10	10	15		58,0	1,0	106,0
9.	Jáchym Kouba	GJŠkodyPŘ	3	7	11	12	10	12	13,5		58,5	0,0	105,5
10.	Ondřej Pupík	GRožnovPR	3	4	11	12	10	12	13,5		58,5	0,0	101,5
11.–12.	Patrik Číhal	SŠKKamPard	3	6	11	4	10	12	13,5	9	50,5	17,0	99,0
	Marek Raška	GTři	4	2	11	12	10	10	11,5		54,5	0,0	99,0
13.	Zuzana Aubrechtová	GHeyrovPH	4	6	2	12	10	12	8		44,0	0,0	93,0
14.	Kryštof Tahal	GUBalvanJN	4	2	11	5	1	12	13,5		42,5	0,0	90,5
15.	Albert Bakoč	GZborovPH	2	6	11	7	7	10	9		44,0	0,0	89,0
16.	Matúš Púll	GZborovPH	3	5	11	7	9	10	7,5		44,5	0,0	88,5
17.	Anna-Kristina Migel	GNAlejíPH	0	2	11	12	10	12			45,0	6,0	88,0
18.	Honza Kocourek	ParkLane	3	3	6	2	10	8	12,5		38,5	0,0	85,5
19.	Patrik Prítrský	GGrössBA	2	2	11	12	10	9			42,0	1,0	84,0
20.	Adam Červenka	GJarošeBO	4	2	6	12	1	9	14		42,0	0,0	83,5
21.	Adam Jahoda	GKepleraPH	4	4	6	5	10	12	13,5		46,5	0,0	79,5
22.	Daniel Culliver	GZborovPH	3	4	11	7	10	10	13,5		51,5	0,0	75,5
23.	Vít Kaděra	G Wicht	1	2	11		10	8	4		33,0	0,0	73,0
24.	Michael Jarvis	GŠpitálsPH	1	2	11	8	10				29,0	0,0	68,5
25.	Adam Kolník	SSŠVTPraha	4	11		12	10	12			34,0	0,0	65,0
26.	Jakub Ondroušek	GTomkovaOL	3	12	11		10				21,0	0,0	62,5
27.	Robert Klimt	GDobříš	3	2	4		10				14,0	0,0	62,0
28.	Petr Slonek	GJarošeBO	4	2	11						11,0	0,0	59,5
29.	Martin Šindelář	GGrössBA	3	2	11	0	10	8			29,0	0,0	56,5
30.	Jakub Konc	GPároNitra	4	1	11	12	10		15		48,0	0,0	48,0
31.	Petr Němec	G Wicht	1	2	11						11,0	0,0	46,0
32.	Jan Slíva	MensaG	2	6							0,0	0,0	45,0
33.–34.	Viktor Číhal	SPŠSmíchov	3	6							0,0	0,0	42,0
	Kryštof Marek	SGPCE	3	3	11	10	10	11			42,0	0,0	42,0
35.	Jakub Hampl	GMělník	3	3	6	10		11			27,0	0,0	41,5
36.	Jan Ševeček	G UherBrod	1	2	2		10				12,0	0,0	40,0
37.	Kateřina Vomelová	GÚstavníPH	3	4	6	3	1	8			18,0	0,0	38,0
38.	Erik Sabol	GČeskoliPH	3	3	6						6,0	2,0	36,5
39.	Jakub Binter	GČeskáČB	0	1							0,0	0,0	30,0
40.	Oto Skýpala	GJŠkodyPŘ	−1	2	6			8			14,0	0,0	28,0
41.	Julie Krejčí	PraKonz	3	3	4						4,0	0,0	26,0
42.	Finley Stuart	GPísnickáPH	2	2	0						0,0	0,0	24,0
43.	Richard Dobíšek	MensaG	2	1							0,0	0,0	22,0
44.	Martin Skýpala	GJŠkodyPŘ	3	1	11		10				21,0	0,0	21,0
45.	Lucian Poljak	GJŠkodyPŘ	1	2	2	8					10,0	0,0	20,0
46.–47.	Janek Hlavatý	GJirsíkaČB	4	10							0,0	0,0	19,0
	Adam Houdek	SOŠ Březová	−2	1	11			8			19,0	0,0	19,0
48.–49.	Petr Dymanus	GŠpitálsPH	3	1	6	6	1	3			16,0	0,0	16,0
	Ondřej Novák	G Brandýs	0	2	0				7		7,0	0,0	16,0
50.	Michal Martínek	GÚstavníPH	2	3							0,0	0,0	15,0
51.	Radomír Budínek	GŘíč	4	1							0,0	0,0	12,0
52.	Tomáš Kazimír	GNPr	3	1	11						11,0	0,0	11,0
53.	Vladimír Sklenár	GTerVans	3	7				3			3,0	0,0	10,5
54.–56.	David Bojko	GMělník	1	2	6		1	3			10,0	0,0	10,0
	Jakub Kopčil	GMikulášPL	4	3							0,0	0,0	10,0
	Jan Kotovský	GPísnickáPH	4	10							0,0	0,0	10,0

	řešitel	*škola*	*ročník*	*sérií*	*2-1*	*2-2*	*2-3*	*2-4*	*2-S*	*2-X1*	*série*	*KSP-X*	*celkem*
57.	Dominik Malý	GBSučany	4	1	6		1				7,0	0,0	7,0
58.–61.	Vojtěch Bízek	GPísnickáPH	3	1		5	1				6,0	0,0	6,0
	Adam Bureš	SPŠ Přerov	3	1	6						6,0	0,0	6,0
	Samuel Lipovský	GBSučany	4	1	6						6,0	0,0	6,0
	Tomáš Macholda	GCoubTábor	4	1	6						6,0	0,0	6,0
62.–64.	Vojtěch Procházka	MensaG	2	1			5				5,0	0,0	5,0
	Matěj Smetana	AkademGPH	2	1							0,0	0,0	5,0
	Jan James Soukup	GKlatovy	4	1							0,0	0,0	5,0
65.	Jáchym Löwenhöffer	GEvolutionJM	2	1							0,0	0,0	4,0
66.–71.	Martin Dobruský	SŠKKamPard	4	1							0,0	0,0	3,0
	Lída Kačenková	GBudějovPH	4	1				3			3,0	0,0	3,0
	Miroslav Kolouch	GJírovcČB	3	1							0,0	0,0	3,0
	Andrea Mikulová	BGOstrava	4	3	0						0,0	0,0	3,0
	Vít Mitáš	GPolička	1	2	2						2,0	0,0	3,0
	Petr Šišlák	GZborovPH	2	1							0,0	0,0	3,0
72.–73.	Alexandr Bihun	GJírovcČB	3	1							0,0	0,0	2,0
	Jakub Vlček	GPříbor	4	1							0,0	0,0	2,0

Výsledková listina KSP-X po druhé sérii třicátého pátého ročníku

	řešitel	*škola*	*ročník*	*sérií*	*1-X1*	*2-X1*	*celkem*
0.					10	10	20,0
1.	Benjamin Swart	MensaG	4	7	8	10	18,0
2.	Patrik Číhal	SŠKKamPard	3	6	8	9	17,0
3.	Štěpán Mikéska	GJarošeBO	4	2	1	8	9,0
4.	Anna-Kristina Migel	GNAlejíPH	0	2	6		6,0
5.	Erik Ježek	SPŠSmíchov	1	2	5		5,0
6.	Erik Sabol	GČeskoliPH	3	3	2		2,0
7.–8.	Filip Prášil	GJiříPoděb	4	3	1		1,0
	Patrik Prítrský	GGrössBA	2	2	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.



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/*