# Korespondenční Seminář z Programování

**35. ročník** | **KSP** | **Říjen 2022**

## Vzorová řešení první série třicátého pátého ročníku KSP

### 35-1-1 Hroší přeslička

Celou situaci převedeme do řeči teorie grafů. Horu si představíme jako graf, který je stromem. Křižovatky budou odpovídat vrcholům grafu, cestičky pak hranám mezi nimi. Vrchol hory odpovídá kořenu stromu.

Místo toho, abychom postupně zkoušeli všechny trasy a zjišťovali jejich nejvyšší bod, zamysleme se nad tím, kolikrát je daný vrchol nejvyšším bodem nějaké trasy.

Nejprve rozeberme situaci, kdy je uvažovaný vrchol kořenem. Nechť kořen $r$ sousedí s $N_r$ vrcholy, označme je $v_1$ až $v_{N_r}$. Velikost podstromu začínajícího ve $v_i$ označme $c_i$, do podstromu počítáme i $v_i$ samotný. Hodnota $c_i$ také odpovídá počtu všech vrcholů, ze kterých vede cesta do $r$ přes $v_i$.

Kolikrát je kořen nejvyšším bodem nějaké trasy? Zcela jistě, pokud nějaká trasa vede přes kořen, pak je kořen jejím nejvyšším vrcholem. Zajímá nás tedy počet cest, které vedou přes kořen.

První možnost je, že trasa v kořenu končí. V takovém případě začíná v podstromu jednoho ze synů kořene $v_1, \ldots, v_{N_r}$. Počet všech vrcholů v nich zjistíme sečtením velikostí těchto podstromů:

$$\sum_{i=1}^{N_r} c_i$$

Stejný počet dostaneme, pokud nám trasa začíná v kořenu a končí v jednom z vrcholů.

Jinak trasa v kořenu nekončí, ani nezačíná. Pak musí krajní vrcholy trasy spadat do různých podstromů. Kdyby totiž spadaly do stejného podstromu pod $v_i$ a kořen by ležel na trase, znamenalo by to, že hrana vedoucí z kořenu do $v_i$ by byla na trase dvakrát, což ze zadání není možné. Stejně tak platí, že pokud leží koncové vrcholy v různých podstromech, pak kořen na trase být musí. Trasa vede mezi různými podstromy a mezi nimi může trasa vést pouze přes kořen.

Takových tras je:

$$\sum_{i=1}^{N_r} \sum_{j=0; j\neq i}^{N_r} c_i c_j$$

Počáteční vrchol můžeme vybrat z libovolného podstromu, k němu koncový pouze z podstromů nějakého jiného syna.

Dvojitá suma se nám ale nelíbí – nelze ji rychle spočítat. Proto ji zkusme spočítat nějak jinak. Vezměme všechny cesty, které nemají kořen jako počáteční ani koncový vrchol (dovolíme i cesty, které začínají a končí stejným vrcholem). Těch je $\left(\sum_{i=1}^{N_r} c_i\right)^2$. Tímto bychom ale započítali navíc cesty, které mají jak koncový, tak počáteční vrchol v témže podstromu. Těch je $\sum_{i=1}^{N_r} c_i^2$. Pokud je tedy od původního počtu odečteme, dostaneme počet tras, které procházejí přes kořen, ale nemají jej jako krajní vrchol:

$$\left(\sum_{i=1}^{N_r} c_i\right)^2 - \sum_{i=1}^{N_r} c_i^2$$

Celkově tak umíme z velikostí podstromů synů spočítat, kolikrát se kořen vyskytne jako nejvyšší vrchol nějaké trasy:

$$\left(\sum_{i=1}^{N_r} c_i\right)^2 - \sum_{i=1}^{N_r} c_i^2 + 2\sum_{i=1}^{N_r} c_i$$

Nyní se zamysleme, jak by se situace změnila pro vrchol $u$, který není kořenem. Kromě případů, které jsme již rozebrali, se nyní mohlo stát, že alespoň jeden z krajních vrcholů trasy není v podstromu $u$. Pak ale $u$ nemůže být nejvyšším vrcholem trasy – jeho otec musí být rovněž na trase.

Zajímají nás tedy pouze takové dvojice, kde oba vrcholy leží v podstromu $u$. Nyní jako $v_i$ označíme podstromy synů $u$. Zbytek výpočtu bude stejný jako pro kořen.

Umíme tedy pro každý vrchol ve stromu spočítat, kolikrát je nejvyšším bodem trasy. Jak to spočítat efektivně pro celý strom?

Můžeme to udělat upraveným prohledáváním do hloubky – před návratem z rekurze vrátíme velikost daného podstromu. Tu spočítáme snadno jako součet velikostí podstromů synů plus jedna. Stejně tak před tím, než se vrátíme, můžeme spočítat, kolikrát daný vrchol bude nejvyšším na trase, a případně upravit udržované globální maximum (a v jakém vrcholu to maximum je).

Jaká je časová složitost takového algoritmu?

V každém vrcholu trávíme čas lineární vzhledem k počtu synů. Jenže celkový počet synů přes všechny vrcholy nemůže být větší, než kolik je vrcholů, tedy celková složitost je lineární.

Jelikož si pamatujeme strukturu celého grafu, je i paměťová složitost lineární.

*Úlohu připravili: Vojta Káně, Kristýna Petrlíková, Ondra Sladký*

### 35-1-2 Vaření lektvarů

Nejprve proveďme následující pozorování – pokud chce Demivirtos začít vařit nějaký lektvar v nějakém čase, ale mohl by jej začít dříve, pak si určitě nepohorší, když jej začne vařit dříve. Následující lektvar by totiž mohl nechat vařit ve stejný čas, anebo dříve a tak dále až do posledního lektvaru, čímž by dovaření posledního lektvaru mohl pouze uspíšit. To ale znamená, že se mu vyplatí každý lektvar uvařit, co nejdříve to lze, tedy hned, jakmile se dovařily jeho prerekvizity.

Zbývá tedy pro každý lektvar zjistit, kdy se dokončí, pokud všechny lektvary dáme vařit hned, jak to půjde. Na to použijme následující algoritmus – upravené hledání topologického uspořádání.[1]

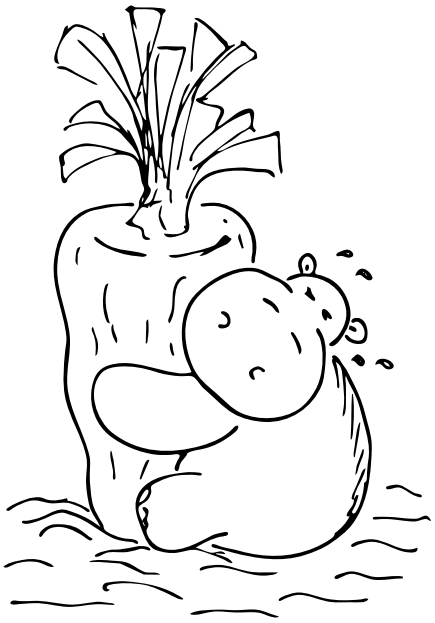
[1] `http://ksp.mff.cuni.cz/viz/kucharky/grafy`

V průběhu si udržujme množinu všech lektvarů, které nemají žádné nehotové prerekvizity. Její stav na začátku můžeme zjistit tak, že projdeme všechny prerekvizity a následně si poznamenáme ty lektvary, které nemají žádnou prerekvizitu. U těch zbylých si poznamenáme, kolik prerekvizit mají.

Nyní vždy vezmeme jeden lektvar, který nemá žádnou prerekvizitu – máme zaručeno, že takový existuje, jinak by na sebe dva nebo více lektvarů navzájem čekalo. Dáme jej vařit v čase, kdy dovařila nejpozdější z jeho prerekvizit – to si budeme ukládat a průběžně aktualizovat u každého lektvaru – a poznamenáme si, že skončí v čase, kdy začal, plus doba jeho vaření. Nyní u každého lektvaru, jehož prerekvizitou daný lektvar je (to si pamatujeme u daného lektvaru), snížíme počet zbývajících prerekvizit o jedna a případně upravíme, kdy se dovaří nejpozdější prerekvizita.

Zbývá aktualizovat množinu lektvarů, které již nemají neuvařené prerekvizity. Jenže to mohou být pouze ty, které měly jako prerekvizitu lektvar, který se právě uvařil, jinak bychom je přidali již předtím. Stačí tedy pouze u těchto zkontrolovat, zda jsou splněné jejich prerekvizity (to poznáme tak, že počet zbývající prerekvizit je nula). Pokud ano, pak je přidáme mezi lektvary, které lze hned uvařit.

Jakmile dovaříme všechny lektvary, stačí je znovu projít a najít ten, který se dovaří nejpozději.



A jakou to má složitost?

Na začátku jen procházíme všechny lektvary a prerekvizitové vztahy, na konci všechny vrcholy – to obojí má lineární velikost vzhledem k počtu lektvarů plus celkovému počtu prerekvizit. Zbývá analyzovat prostřední část – jenže tam na každý lektvar sáhneme pouze jednou a aktualizace lektvarů, kterým je prerekvizitou, můžeme naúčtovat prerekvizitovému vztahu, na který sáhneme také pouze jednou. Tedy i tato část má lineární časovou složitost vzhledem k délce vstupu.

V paměti máme uložené všechny lektvary a všechny prerekvizity, tedy i prostorová složitost je lineární.

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

*Úlohu připravili: Jan Adámek, Říša Hladík, Jirka Kalvoda, Michal Kodad, Dan Skýpala, Ondra Sladký*

## 35-1-3 Letadélko

*Protože počet otazníků byl ze zadání velmi malý, označme si ho $K$ a počítejme časovou složitost i vzhledem k němu.*

První, co nás napadne, je vyzkoušet všechny možnosti, jak mohlo letadlo letět. Všech možností je $2^K$, a když každou doplníme a zkontrolujeme, dostaneme počet validních možností. Tento algoritmus bude ale ukrutně pomalý: $\mathcal{O}(N \cdot 2^K)$

**Zlepšujeme**

Teď tedy budeme muset přijít na něco lepšího – všimněme si, že pokud letadlo měnilo několikrát výšku, nezáleží, jak přesně tyto výšky měnilo, ale stačí jen vědět, v jaké výšce skončilo (za předpokladu, že se neutopilo).

Z toho můžeme všechny lety, které skončí ve stejné výšce, seskupit a chovat se k nim stejně.

Označme si $d[i][h]$ počet letů, které po $i$ změnách výšek skončí ve výšce $h$.

Pojďme si najít zřejmé hodnoty. Letadlo začíná ve výšce 0: $d[0][0] = 1$, a pokud výška nulová není: $d[0][h] = 0$. A nikdy nesmí klesnout pod hladinu: $d[i][-h] = 0$.

Nyní nám jen zbývá vymyslet, jak vypočítat $d[i+1]$ z $d[i]$. Pokud jsme v letu stoupli, tak všechny lety zvýšily výšku o 1: $d[i+1][h] = d[i][h-1]$, pokud jsme naopak klesli: $d[i+1][h] = d[i][h+1]$. A zbývá nám vyřešit poslední případ, kdy nevíme, co se stalo. Pak se mohlo stát obojí: $d[i+1][h] = d[i][h-1] + d[i][h+1]$.

Potom, co toto všechno spočítáme, chceme, aby letadlo skončilo v nulové výšce, tedy odpověď je $d[N][0]$.

A jakou má toto složitost? Máme $N$ sloupců, letadlo může za let vystoupat až do výšky $N$ a výpočty jednotlivých polí provedeme v konstantním čase. Celkem $\mathcal{O}(N^2)$.

**Zlepšujeme ještě jednou**

Všimněme si ale, že rozptyl výšek, ve kterých se letadlo může v jeden časový okamžik nacházet, je velmi malý, totiž $2K$. Většina tabulky tak bude zaplněná nulami. Když si budeme udržovat okénko velikosti $2K$, tak nám to zredukuje časovou složitost na $\mathcal{O}(NK)$.

**A ještě jednou**

Nicméně naše řešení můžeme ještě zlepšit. Pokud totiž na naší pozici není otazník, tak se celý sloupec pouze posune o 1 nahoru nebo dolů, a tedy ho můžeme zachovat (jen ořízneme utopená letadla – protože jsme klesli v aktuálním kroku jen o jedna, tak může být utopené letadlo jen jedno). Vypočítat nové okénko velikosti $2K$ potřebujeme pouze na každém otazníku, tedy $K$-krát. Tím zlepšíme časovou složitost na $\mathcal{O}(N + K^2)$.

Program (Python):
`http://ksp.mff.cuni.cz/viz/35-1-3.py`

*Úlohu připravili: Jirka Kalvoda, Dan Skýpala*

## 35-1-4 Atlas zlomků

Máme najít všechny zlomky $a/b \in \langle 0, 1\rangle$, které jsou v základním tvaru a $1 \leq b \leq N$. Zlomek $0/1$ vypíšeme zvlášť, pak už se můžeme spolehnout na to, že čitatel je nenulový. Postačí tedy najít všechny uspořádané dvojice $(a, b)$ takové, že $1 \leq a \leq b \leq N$ a čísla $a$ a $b$ jsou nesoudělná (to znamená, že jejich největší společný dělitel je 1).

**Dřevorubecké řešení**

Můžeme si vzít všechny dvojice $(a, b)$ s $1 \leq a \leq b \leq N$, pro každou z nich spočítat největšího společného dělitele (NSD), a pokud vyjde 1, dvojici vypsat.

Jak počítat NSD? K tomu se hodí *Euklidův algoritmus.* Zde ho popíšeme jen stručně, detaily najdete v kuchařce o teorii čísel.[2] Jednodušší verze algoritmu opakovaně od většího čísla odečítá menší, a až se vyrovnají, prohlásí výsledek za NSD. Výpočet se určitě zastaví, protože každým krokem se zmenší součet $a + b$ aspoň o 1. Výsledek je správně, neboť každý krok zachovává množinu všech společných dělitelů, a tedy i NSD.

Algoritmus může potřebovat až $a + b$ kroků, což v našem případě je $\mathcal{O}(N)$. To je hodně, ale lze to jednoduše zrychlit. Pokud by třeba bylo $a = 999$ a $b = 7$, opakovaným odčítáním $b$ od $a$ nakonec spočítáme $999 \bmod 7 = 5$. Místo odčítání tedy můžeme větší číslo rovnou vymodulit menším. Jen pozor na to, že takto uděláme o krok navíc, takže algoritmus se nezastaví s $a = b$, nýbrž až s $a = 0$ – tehdy je NSD rovno $b$.

V kuchařce dokazujeme, že modulící verze Euklidova algoritmu provede $\mathcal{O}(\log(a + b))$ průchodů. Tím pádem naše řešení vyzkouší $\mathcal{O}(N^2)$ dvojic $(a, b)$, pro každou z nich v čase $\mathcal{O}(\log(2N)) = \mathcal{O}(\log N)$ spočítá NSD a případně dvojici vypíše. Celková časová složitost tedy činí $\mathcal{O}(N^2 \log N)$.

**Kešování mezivýsledků**

Předchozí řešení počítá NSD mnoha dvojic. Přitom určitě mnohokrát dojde ke stejnému mezivýsledku. Pomůžeme si tedy myšlenkou z dynamického programování a budeme si ve dvojrozměrném poli $D$ pamatovat, pro které dvojice jsme už NSD spočítali, a nebudeme je počítat znovu. Jelikož se všechny mezivýsledky pohybují mezi 0 a $N$, bude pole $D$ zabírat prostor $\mathcal{O}(N^2)$.

Postačí nám odčítací verze Euklidova algoritmu. Zapíšeme ji rekurzivně:

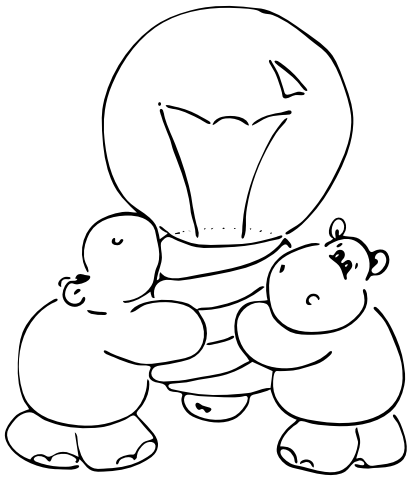
**Algoritmus** NSD$(a, b)$

1. Pokud $D[a, b] = 0$: ◁ NSD$(a, b)$ *ještě neznáme*
2. Pokud $a = b$: $D[a, b] = a$.
3. Pokud $a > b$: $D[a, b] =$ NSD$(a - b, b)$.
4. Pokud $a < b$: $D[a, b] =$ NSD$(a, b - a)$.
5. Vrátíme $D[a, b]$.

Představme si, že jsme tuto funkci postupně zavolali pro $K$ různých dvojic. Ona sama se nejspíš rekurzivně zavolala pro spoustu dalších dvojic. Ale pouze v $\mathcal{O}(N^2)$ případech se může stát, že políčko $D[a, b]$ ještě není vyplněné, takže rekurze bude pokračovat. V ostatních případech se hned vrátíme – tento čas můžeme naúčtovat tomu, kdo nás zavolal, což je buďto vyplňování políčka, nebo jedno z $K$ původních volání „zvenku“.

Celkem tedy výpočty NSD strávíme čas $\mathcal{O}(N^2 + K)$. V naší úloze je ovšem $K$ také $\mathcal{O}(N^2)$, takže celý algoritmus doběhne v čase $\mathcal{O}(N^2)$.

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



**Euklidův algoritmus pozpátku**

Úloha po nás chce optimalizovat časovou složitost vzhledem k počtu vypsaných zlomků – budeme mu říkat třeba $Z$. Jak dobré je v tomto ohledu předchozí řešení? To záleží na tom, jak velké je $Z$ vzhledem k $N^2$. Pokud je výrazně menší, trávíme spoustu času dvojicemi $(a, b)$, které se nakonec ukáží být soudělné, takže je zahodíme. Tak raději vymyslíme, jak se soudělným dvojicím rovnou vyhnout.

Výpočet NSD pro každou nesoudělnou dvojici skončí ve dvojici $(1, 1)$. Co kdybychom zkusili provádět Euklidův algoritmus pozpátku a kdykoliv bychom drželi v ruce nějakou dvojici $(a, b)$, prozkoumali bychom všechny možnosti, odkud se do ní algoritmus mohl dostat. To jsou jenom dvojice $(a + b, b)$ a $(a, a + b)$.

Budeme pokračovat rekurzivně. Jakmile $a$ nebo $b$ překročí $N$, rekurzi zastavíme – všechny další dvojice v této větvi výpočtu by byly ještě větší. Ještě si všimneme, že na každý zlomek narazíme právě jednou, protože Euklidův algoritmus má v každém kroku jednoznačně určený následující krok.

Jediný háček je v tom, že nevynucujeme $a \leq b$, takže vypsané zlomky mohou být větší než 1. Všimněte si, že tuto podmínku nelze vynucovat průběžně, jelikož ke dvojicím s $a \leq b$ může být potřeba dojít přes dvojice s $a > b$. Naštěstí stačí zlomky větší než 1 prostě nevypisovat – jelikož tvoří nejvýše polovinu všech vygenerovaných zlomků, časovou složitost si tím nezhoršíme.

Program bude vypadat takto:

**Algoritmus** Zlomky$(a, b)$

1. Pokud $a > N$ nebo $b > N$, vrátíme se.
2. Pokud $a \leq b$, vypíšeme zlomek $a/b$.
3. Zavoláme Zlomky$(a + b, b)$.
4. Zavoláme Zlomky$(a, a + b)$.

Výpočet odstartujeme zavoláním Zlomky$(1, 1)$.

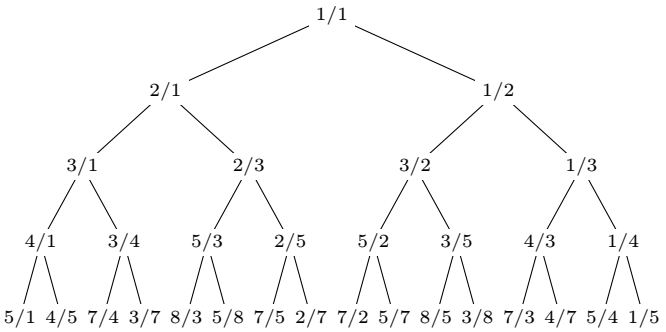
Časovou složitost odhadneme snadno. Každé volání funkce, které prošlo přes podmínku v 1. kroku, vypíše jeden zlomek. Těmito voláními tedy strávíme čas $\mathcal{O}(Z)$. Ostatní volání se hned vrátí, takže je můžeme naúčtovat tomu, kdo je zavolal.

Tak jsme získali jednoduchý algoritmus s časovou složitostí $\mathcal{O}(Z)$, což je zjevně optimální.

**Calkinův-Wilfův strom**

Mimochodem, všechna volání naší funkce Zlomky můžeme znázornit následujícím stromem. Říká se mu Calkinův-

Wilfův strom, a jak jsme dokázali, obsahuje každý kladný zlomek právě jednou.




**Trocha asymptotiky na závěr**

Ještě jsme nezjistili, jaký je vztah mezi $Z$ a $N^2$, tedy jak daleko je kešovací algoritmus od optima. V této poněkud matematičtější sekci dokážeme, že $Z \geq cN^2$ pro nějakou konstantu $c > 0$. Takže $N^2$ je ve skutečnosti $\mathcal{O}(Z)$ a kešovací algoritmus je (aspoň asymptoticky) optimální.

Označme $M$ množinu všech dvojic $(a, b)$ takových, že $1 \leq a, b \leq N$. Zjevně je $|M| = N^2$. Dále buď $S \subseteq M$ podmnožina obsahující všechny soudělné dvojice. Dokážeme, že $|S| \leq dN^2$ pro nějakou konstantu $0 < d < 1$. Nesoudělných tedy bude alespoň $(1-d)N^2$, z toho s $a \leq b$ aspoň $(1-d)/2 \cdot N^2$.

Pro libovolné prvočíslo $2 \leq p \leq N$ označíme $S_p$ množinu těch dvojic $(a, b) \in M$, v nichž je jak $a$, tak $b$ dělitelné prvočíslem $p$. Všechny dvojice v $S_p$ jsou tedy soudělné, a naopak každá soudělná dvojice leží v alespoň jedné množině $S_p$. Platí tedy $S = \bigcup_p S_p$, a proto $|S| \leq \sum_p |S_p|$, kde $\sum_p$ sčítá přes všechna prvočísla mezi 2 a $N$.

Kolik prvků má jedna množina $S_p$? Ve dvojici $(a, b) \in S_p$ musí být jak $a$, tak $b$ násobkem $p$. Celkem tedy máme nejvýše $N/p$ možností pro $a$ a stejně tolik pro $b$. Proto $|S_p| \leq (N/p)^2$.

O velikosti množiny $S$ tedy víme, že $|S| \leq \sum_p (N/p)^2 = N^2 \cdot \sum_p 1/p^2$. Tato nerovnost platí i tehdy, sčítáme-li přes úplně všechna prvočísla. Suma $\sum_p 1/p^2$ bude kýžená konstanta $d$, ovšem musíme ukázat, že leží ostře mezi 0 a 1.

Pomůžeme si jinou sumou, známou pod názvem Basilejský problém:[3] $\beta = \sum_{n=1}^{\infty} 1/n^2$. O ní se ví, že má součet $\beta = \pi^2/6 \doteq 1.645$. To neumíme dokázat bez pokročilé matematické analýzy. Vystačíme si proto se slabším odhadem $\beta \leq 2$, který ponecháme jako cvičení: ukažte indukcí podle $k$, že součet prvních $k$ členů řady je menší nebo roven $2 - 1/k$.

Pro naši konstantu $d$ jistě platí, že $d \leq \beta$. To ale nestačí na $d < 1$. Ovšem když si uvědomíme, že $1/1^2$ a $1/4^2$ jsou v $\beta$, ale nejsou v $d$, dostaneme $d \leq \beta - 1 - 1/16 \leq 2 - 1 - 1/16 = 15/16$. A to už naše požadavky splňuje.

*Úlohu připravili: Jirka Kalvoda, Martin „Medvěd“ Mareš, Dan Skýpala*

## 35-1-X1 Rejstřík zlomků

Pokračujme v úvahách z řešení úlohy 35-1-4. Hodil by se nám podobný strom, ale uspořádaný podle hodnot zlomků, abychom je uměli vyjmenovat v rostoucím pořadí. Tak si ho pořídíme.

**Strom zlomkovník**

Nejprve definujeme *mediant* dvou zlomků $a/b$ a $c/d$. To je zlomek $(a+c)/(b+d)$. Všimneme si, že pokud platí $a/b < c/d$, pak mediant leží mezi nimi:

$$\frac{a}{b} < \frac{a+c}{b+d} < \frac{c}{d}$$

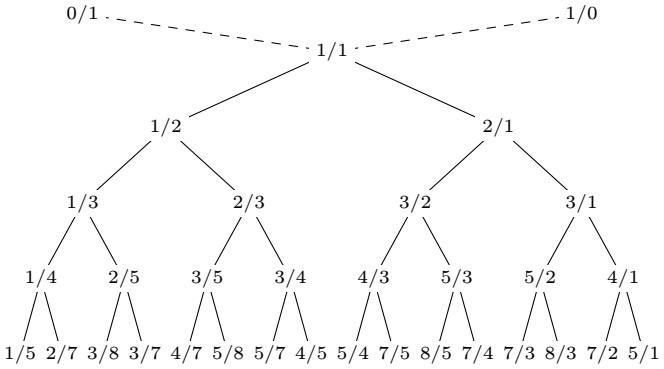
(snadno ověříme vynásobením společným jmenovatelem).

Pomocí mediantů budeme vytvářet posloupnosti zlomků. Posloupnost $A_0$ obsahuje zlomky $0/1$ a $1/0$ (ten druhý je naše soukromá reprezentace nekonečna). Posloupnost $A_{i+1}$ získáme z $A_i$ tak, že mezi každé dva zlomky vložíme jejich mediant:

$$A_1 = \frac{0}{1}, \frac{1}{1}, \frac{1}{0}$$
$$A_2 = \frac{0}{1}, \frac{1}{2}, \frac{1}{1}, \frac{2}{1}, \frac{1}{0}$$
$$A_3 = \frac{0}{1}, \frac{1}{3}, \frac{1}{2}, \frac{2}{3}, \frac{1}{1}, \frac{3}{2}, \frac{2}{1}, \frac{3}{1}, \frac{1}{0}$$

Z vlastností mediantů plyne, že každá posloupnost $A_i$ je ostře rostoucí.

Vytváření posloupností vkládáním mediantů můžeme také popsat stromem. Říká se mu *Sternův-Brocotův strom* nebo také *strom zlomkovník* (rostou na něm přeci zlomky). Jeho prvních pár hladin vypadá takto:




Každý vnitřní vrchol stromu je mediantem svého levého a pravého předka – když jdeme z vrcholu nahoru, první vrchol, do nějž přijdeme zleva, je pravý předek; symetricky levý předek (buď levý nebo pravý předek je otec, druhý předek leží někde výše).

Posloupnost $A_i$ je tedy sjednocením prvních $i$ hladin stromu. Navíc pro libovolný vrchol $v$ platí, že všechny zlomky v levém podstromu pod $v$ jsou menší než zlomek ve $v$, a ten je menší než všechny zlomky v pravém podstromu. Zlomkovník je tedy uspořádaný jako vyhledávací strom, a proto se v něm žádné racionální číslo neopakuje.

Všechny zlomky ve stromu bychom mohli vypsat v rostoucím pořadí následující rekurzivní funkcí (nenechte se zmást tím, že běží nekonečně dlouho; v praxi bychom samozřejmě rekurzi včas zařízli, a tak vypsali vhodný konečný podstrom). Její parametry jsou zlomek $\ell_a/\ell_b$ coby levý předek vypisovaného podstromu a zlomek $p_a/p_b$ coby jeho pravý předek.

**Algoritmus** ZLOMKOVNÍK$(\ell_a/\ell_b, p_a/p_b)$

1. $a \leftarrow \ell_a + p_a$, $b \leftarrow \ell_b + p_b$
2. ZLOMKOVNÍK$(\ell_a/\ell_b, a/b)$ ◁ *levý podstrom*
3. Vypíšeme $a/b$.
4. ZLOMKOVNÍK$(a/b, p_a/p_b)$ ◁ *pravý podstrom*

[3] https://cs.wikipedia.org/wiki/Basilejsk%C3%BD_probl%C3%A9m

Rekurzi spustíme zavoláním ZLOMKOVNÍK$(0/1, 1/0)$.

Brzy dokážeme, že ve stromu leží všechny zlomky a jsou v základním tvaru. Předtím si ale rozmyslíme, jak pomocí zlomkovníku vyřešit zadanou úlohu.

**Řešení úlohy**

Zlomkovník budeme používat jako vyhledávací strom. Je trochu nepraktické, že je to strom nekonečný. Jelikož však všechny cesty do „malých“ zlomků s čitatelem i jmenovatelem z $\{1, \ldots, N\}$ vedou jen přes další malé zlomky, stačí se prostě při prohledávání stromu zastavovat o velké zlomky.

Stačí tedy umět vypsat nejmenších $K$ klíčů ve vyhledávacím stromu. Na to si pořídíme rekurzivní funkci, která bude postupně vypisovat klíče a vrátí, kolik jich vypsala:

**Algoritmus** PRVNÍCHK$(v, K)$

1. Pokud $v = \emptyset$, vrátíme 0.
2. $i \leftarrow$ PRVNÍCHK$(\mathit{levý_syn}(v), K)$
3. Pokud $i < K$:
4. Vypíšeme $\mathit{klíč}(v)$.
5. $i \leftarrow i + 1$
6. Pokud $i < K$:
7. $i \leftarrow i +$ PRVNÍCHK$(\mathit{pravý_syn}(v), K - i)$
8. Vrátíme $i$.

Jak rychlá tato funkce je? Představme si cestu ve stromu mezi kořenem a posledním vypsaným klíčem. Vše nalevo od této cesty bylo vypsáno. Vše napravo je moc velké a vůbec jsme tam nezalezli. Některé vrcholy na cestě jsme vypsali, některé nevypsali, ale těch nevypsaných je nejvýš tolik, kolik je hloubka stromu – v našem případě tedy nejvýš $N$.

Navštívíme tedy $\mathcal{O}(N+K)$ vrcholů, pokaždé v konstantním čase spočítáme, jaký zlomek se tam nachází. Algoritmus má tedy časovou složitost $\mathcal{O}(N+K)$, a to je $\mathcal{O}(K)$, neboť máme slíbeno, že $K \geq N$.

Mimochodem, kdyby bylo $K$ malé, stejně si umíme poradit: zlomky na nejlevější cestě ve stromu jsou $1/i$, takže bychom rovnou mohli skočit správně hluboko a vypisovat odtamtud. To by dalo složitost $\mathcal{O}(K)$ nezávisle na $N$.

**Zlomky jsou v základním tvaru**

Zbývá ověřit, že zlomkovník má slíbené vlastnosti.

Indukcí podle $i$ budeme dokazovat, že pro každé dva zlomky $a/b$ a $c/d$, které sousedí v posloupnosti $A_i$, platí:

$$bc - ad = 1.$$

Z toho ihned plyne, že každý společný dělitel čísel $a$ a $b$ musí také dělit $bc - ad$, pročež jediní společní dělitelé jsou 1 a $-1$. Čísla $a$ a $b$ jsou tedy nesoudělná. Analogicky jsou nesoudělná i $c$ a $d$.

Nyní slíbená indukce. Pro $A_0$ tvrzení evidentně platí. Když z $A_i$ vyrábíme $A_{i+1}$, mezi každé sousední zlomky $a/b$ a $c/d$ vkládáme $(a+c)/(b+d)$. Tím jsme vytvořili dvě nové dvojice sousedních zlomků. Pro první z nich má platit:

$$b(a + c) - a(b + d) = 1,$$

což je po roznásobení totéž jako

$$ab + bc - ab - ad = 1.$$

To platí, jelikož obě $ab$ se navzájem odečtou a $bc - ad = 1$ je indukční předpoklad.

Podobně pro druhou dvojici sousedních zlomků chceme

$$(b + d)c - (a + c)d = 1,$$

což je opět ekvivalentní s $bc - ad = 1$.

Všechny zlomky ve zlomkovníku jsou tedy v základním tvaru.

**Cesty ve zlomkovníku**

Zlomkovník nám také dává další způsob, jak zapisovat racionální čísla – cestami z kořene. Označíme-li L krok doleva a P krok doprava, bude například LPPL $= 5/7$. Ukážeme, jak převádět mezi zlomky a cestami.

Jelikož zlomkovník je vyhledávací strom, můžeme každému vrcholu přiřadit nějaký interval $(\ell_a/\ell_b, p_a/p_b)$. Všechny zlomky v podstromu leží v tomto intervalu, všechny zlomky mimo podstrom leží mimo interval. Přitom $\ell_a/\ell_b$ a $p_a/p_b$ jsou levý a pravý předek vrcholu.

Intervaly budeme popisovat pomocí matic $2 \times 2$:

$$\begin{pmatrix} \ell_b & p_b \\ \ell_a & p_a \end{pmatrix}$$

Z intervalu přiřazeného vrcholu můžeme spočítat zlomek, který v tomto vrcholu bydlí. Je to mediant levé a pravé meze. Popíšeme ho funkcí $z$, jež maticím přiřazuje zlomky:

$$z\left(\begin{pmatrix} \ell_b & p_b \\ \ell_a & p_a \end{pmatrix}\right) = \frac{\ell_a + p_a}{\ell_b + p_b}.$$

Nyní popíšeme, co se děje při procházení po cestě. Začneme v kořenu:

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

Odbočíme-li doleva, nahradíme pravou mez intervalu aktuálním zlomkem. To lze zapsat násobením matic:

$$\begin{pmatrix} \ell_b & p_b \\ \ell_a & p_a \end{pmatrix} \cdot \mathbf{L} = \begin{pmatrix} \ell_a & \ell_b + p_b \\ \ell_a & \ell_a + p_a \end{pmatrix}, \text{ kde } \mathbf{L} = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}.$$

Podobně pro krok doprava:

$$\begin{pmatrix} \ell_b & p_b \\ \ell_a & p_a \end{pmatrix} \cdot \mathbf{P} = \begin{pmatrix} \ell_b + p_b & p_b \\ \ell_a + p_a & p_a \end{pmatrix}, \text{ kde } \mathbf{P} = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}.$$

Složitější cestu tedy můžeme popsat tak, že vezmeme matici $\mathbf{I}$ (to je jednotková matice), postupně ji zprava násobíme maticemi $\mathbf{L}$ a $\mathbf{P}$ pro jednotlivé kroky cesty, a nakonec funkcí $z$ získáme zlomek na konci cesty.

Například

$$z(\mathbf{ILPPL}) = z(\mathbf{LPPL}) = z\left(\begin{pmatrix} 3 & 4 \\ 2 & 3 \end{pmatrix}\right) = \frac{2+3}{3+4} = 5/7.$$

Jelikož zlomkovník je binární vyhledávací strom, můžeme v něm zadaný zlomek vyhledávat rekurzivně: kdykoliv přijdeme do vrcholu se zlomkem větším než hledaný, pokračujeme doleva; když s menším, pak doprava. Můžeme to zapsat následovně, přičemž $a/b$ je hledaný zlomek a $\mathbf{V}$ matice reprezentující interval aktuálního vrcholu (na počátku je $\mathbf{V} = \mathbf{I}$):

**Algoritmus** HLEDEJZLOMEK$(a/b, \mathbf{V})$

1. Opakujeme:
2. $v_a/v_b = z(\mathbf{V})$
3. Pokud $a/b = v_a/v_b$:
4. Nalezli jsme a skončíme.
5. Pokud $a/b < v_a/v_b$:
6. HLEDEJZLOMEK$(a/b, \mathbf{VL})$
7. Jinak:
8. HLEDEJZLOMEK$(a/b, \mathbf{VP})$

Pokud se zlomek $a/b$ ve stromu vyskytuje, tento algoritmus ho najde. Pokud by se nevyskytoval, algoritmus poběží donekonečna a bude se nořit do čím dál užších intervalů obsahujících $a/b$. (Mimochodem, pokud bychom algoritmus nechali hledat iracionální číslo, najde nekonečnou cestu konvergující k zadanému číslu, na které leží čím dál lepší racionální aproximace. To ale nebudeme dokazovat.)

Na hledání se dá také dívat jinak: místo abychom postupně upravovali matici $\mathbf{V}$, budeme upravovat zlomek $a/b$.

Všimneme si jedné zajímavé vlastnosti algoritmu. Pokud první krok vedl doprava, ve všech dalších krocích bude platit $\mathbf{V} = \mathbf{PX}$, kde $\mathbf{X}$ je nějaká matice závislá na aktuálním kroku. Budeme tedy $a/b$ porovnávat s $z(\mathbf{PX})$, které je ovšem pro

$$\mathbf{X} = \begin{pmatrix} \ell_b & p_b \\ \ell_a & p_a \end{pmatrix}$$

rovno

$$z(\mathbf{PX}) = z\left(\begin{pmatrix} \ell_b & p_b \\ \ell_a + \ell_b & p_a + p_b \end{pmatrix}\right) = \frac{\ell_a + \ell_b + p_a + p_b}{\ell_b + p_b} = \frac{\ell_a + p_a}{\ell_b + p_b} + 1 = z(\mathbf{X}) + 1.$$

Tedy $a/b < z(\mathbf{PX})$ právě tehdy, když $a/b < z(\mathbf{X}) + 1$, čili $a/b - 1 < z(\mathbf{X})$. V algoritmu tedy můžeme místo vynásobení matice $\mathbf{V}$ zleva $\mathbf{P}$ prostě od $a/b$ odečíst 1, neboli zavolat HledejZlomek$((a-b)/b, \mathbf{V})$.

Podobnou úvahu můžeme provést pro krok doleva a rekurzi na$(a/b, \mathbf{PV})$ nahradit za $(a/(b-a), \mathbf{V})$. Odvození nemusíme opakovat, stačí si uvědomit, že prohození kroků doleva a doprava v cestě způsobí prohození čitatele a jmenovatele v cílovém zlomku.

Teď už se v rekurzi matice $\mathbf{V}$ nemění a zůstává pořád rovná počáteční jednotkové matici. Test $a/b < z(\mathbf{V})$ je tedy $a/b < z(\mathbf{I})$, což je $a/b < 1$, tedy $a < b$.

To je ale přeci Euklidův algoritmus! Pokud je $a < b$, odečteme $a$ od $b$, jinak $b$ od $a$. V jednom případě jdeme doleva, v druhém doprava. Dokázali jsme tedy, že vyhledávání ve zlomkovníku je ekvivalentní s Euklidovým algoritmem. Tím pádem se hledání vždy zastaví, ale to může jen tehdy, pokud zlomek najde. Zlomkovník tedy obsahuje všechny zlomky.

**Kouzlo na závěr**

Podívejte se na nějakou hladinu zlomkovníku a srovnejte ji s odpovídající hladinou Calkinova-Wilfova stromu z předchozí úlohy. Obsahují tytéž zlomky, jen v jiném pořadí! Důvod je prostý: cesta k určitému zlomku v obou případech odpovídá průběhu Euklidova algoritmu, jen poprvé průběh zapisujeme popředu, podruhé pozpátku. Očíslujeme-li tedy zlomky na $k$-té hladině jednoho stromu $k$-bitovými binárními čísly, bude poloha zlomku v jednom stromu rovna poloze v druhém stromu zapsané pozpátku.

*Úlohu připravili: Jirka Kalvoda, Martin „Medvěd“ Mareš, Dan Skýpala*

## 35-1-S Lineární rovnice

**Úkol 1 – Robot na Marsu** [5b]:

Označme si vektory na vstupu $\mathbf{x}$, $\mathbf{y}$. Tyto vektory jsou tvořeny dvěma složkami: $\mathbf{x} = (x_1, x_2)^\top$, obdobně $\mathbf{y} = (y_1, y_2)^\top$.

Představme si, že se nacházíme na souřadnicích $(0,0)^\top$. Chceme se pohnout o $m$ metrů dopředu, tedy se dostat na souřadnice $\mathbf{v} = (m, 0)^\top$.

Potřebujeme nalézt lineární kombinaci vektorů $\mathbf{x}$ a $\mathbf{y}$ takovou, aby výsledný vektor byl roven vektoru $\mathbf{v}$. Tedy chceme nalézt čísla $a$ a $b$ taková, že:

$$a\mathbf{x} + b\mathbf{y} = \mathbf{v}$$

Nyní tuto rovnici můžeme rozepsat po složkách:

$$ax_1 + by_1 = v_1 = m$$

$$ax_2 + by_2 = v_2 = 0$$

Dostali jsme dvě rovnice o dvou neznámých. Tuto soustavu by šlo vyřešit pomocí Gaussovy eliminace, nicméně v tomto případě není těžké zapsat řešení explicitně. Z druhé rovnice vyjádříme $b$:

$$b = -\frac{ax_2}{y_2}$$

A dosadíme do první rovnice:

$$ax_1 + \left(-\frac{ax_2}{y_2}\right) y_1 = m$$

$$a\left(x_1 - \frac{x_2 y_1}{y_2}\right) = m$$

$$a = \frac{my_2}{x_1 y_2 - x_2 y_1}$$

Odtud lze dopočítat i $b$, povšimněte si symetrie:

$$b = -\frac{ax_2}{y_2} = -\frac{my_2}{x_1 y_2 - x_2 y_1}\frac{x_2}{y_2} = \frac{mx_2}{y_1 x_2 - y_2 x_1}$$

Sondě tedy řekneme, aby se pohla o $a$ metrů vpřed a o $b$ metrů doleva. Pohyby pro povel „o $m$ metrů doleva“ spočítáme obdobně.

Pro úplnost je třeba ukázat, že jmenovatel výše uvedených výrazů není nulový. Jsou-li vektory $\mathbf{x}$, $\mathbf{y}$ nenulové a mají-li různý směr, poté platí $x_1/x_2 \neq y_1/y_2$, tedy $x_1 y_2 \neq x_2 y_1$.

**Úkol 2 – Nezávislost a soustavy** [3b]:

Mějme $d$-rozměrné vektory $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$, chceme ověřit jejich lineární nezávislost. Prvně si musíme uvědomit, co nezávislost vlastně znamená. Zmiňovali jsme dvě definice, my použijeme tu druhou, tedy že rovnice

$$\sum_{i=1}^{n} a_i \mathbf{x}_i = \mathbf{0}$$

platí jedině, pokud $a_i = 0$ pro všechna $i$.

Stačí pracovat s touto rovnicí, konkrétně ji rozepíšeme po složkách. Potřebujeme hodně indexů, výrazem $(\mathbf{x}_i)_j$ značíme $j$-tou složku vektoru $\mathbf{x}_i$.

$$a_1(\mathbf{x}_1)_1 + a_2(\mathbf{x}_2)_1 + \ldots + a_n(\mathbf{x}_n)_1 = 0$$
$$\vdots$$
$$a_1(\mathbf{x}_1)_d + a_2(\mathbf{x}_2)_d + \ldots + a_n(\mathbf{x}_n)_d = 0$$

Opět dostáváme soustavu rovnic. Řešení $a_i = 0$ bude existovat vždy. Pokud se jedná o jediné řešení, pak jsou vektory nezávislé. Jinak existuje nenulové $a_i$, příslušný vektor můžeme převést na druhou stranu a koeficientem $a_i$ rovnici vydělit, čímž dostaneme vyjádření jednoho vektoru jako lineární kombinaci zbytku.

Jednoduché pozorování na závěr: Pokud $n > d$, pak vždy platí, že vektory jsou lineárně závislé.

**Úkol 3 – Algoritmus eliminace** [7b]:

Samotný algoritmus byl popsaný v zadání, bylo však nutné rozmyslet mnoho implementačních detailů.

Začněme reprezentací soustavy. Jistě dává smysl si ji uložit jako seznam rovnic. Pro rovnice samotné lze také použít seznamy, pak si budeme muset naimplementovat jejich násobení a sčítání. Také by šlo využít nějaký vektorový typ, který už tyto operace umí, v Pythonu například `numpy.ndarray`.

Je třeba se zamyslet i nad číselným datovým typem. V zadání jsou sice celá čísla, v průběhu výpočtu nám však mohou vzniknout i zlomky. Jako jednoduché řešení se nabízí použít desetinná čísla, pak si ovšem musíme dát pozor na zaokrouhlovací chyby. Hodnoty, které by správně měly být nulové, mohou být nepatrně větší či menší. Vzorové řešení v průběhu eliminace používá celá čísla, tento postup by ale nefungoval pro velké soustavy, kde by čísla ve výpočtech mohla rychle vzrůst.

Pokud soustava nemá jednoznačné řešení, někdy v průběhu eliminace se stane, že ve všech řádcích v aktuálním sloupci budou nuly. Poté není co eliminovat, takže daný sloupec přeskočíme.

To bude mít za důsledek, že koeficient u poslední neznámé v posledním řádku bude nulový. Podle toho tedy jde poznat typ řešení. Ještě se musíme podívat na pravou stranu v poslední rovnici, podle toho rozlišíme soustavu bez řešení a parametrizované řešení.

**Zpětná substituce**

Jakmile máme eliminaci hotovou, zbývá dopočítat řešení zpětnou substitucí. Doteď stačila celá čísla, nyní již bude nutné použít zlomky.

V jednoznačném případě je výpočet poměrně přímočarý. Stačí procházet rovnice od konce, v každé bude jediná neznámá, zbytek už známe. Převedeme tedy již spočítané neznámé na pravou stranu, čímž vyjádříme tu hledanou.

Parametrizovaný případ je zajímavější, jedna rovnice nám může přidat více nových neznámých. Poté musíme všechny až na jednu prohlásit za parametry.

Při dopočítávání neznámé musíme dopočítat i to, jak závisí na parametrech. K tomu se hodí mít neznámé reprezentované v rovnicovém zápisu a až na konci jej převést na vektorový.

Zbývá jen se zamyslet nad složitostí řešení. Chceme eliminovat polovinu koeficientů, kterých je $\mathcal{O}(d^2)$. Pokaždé násobíme a sčítáme celé řádky délky $\mathcal{O}(d)$, celkem tedy $\mathcal{O}(d^3)$. Výpočet řešení v jednoznačném případě je $\mathcal{O}(d^2)$, tolik spočítaných neznámých totiž převádíme na pravou stranu. Parametrizovaný případ se chová podobně, jen pracujeme s vektory místo čísel, takže $\mathcal{O}(d^3)$. Paměti potřebujeme vždy $\mathcal{O}(d^2)$.

**Gaussova-Jordanova eliminace**

Existuje způsob, jak si výpočet řešení trošku zjednodušit. Při eliminaci budeme vytvářet nuly nejen pod aktuálním řádkem, nýbrž i nad ním. Tento postup se nazývá *Gaussova-Jordanova eliminace.*

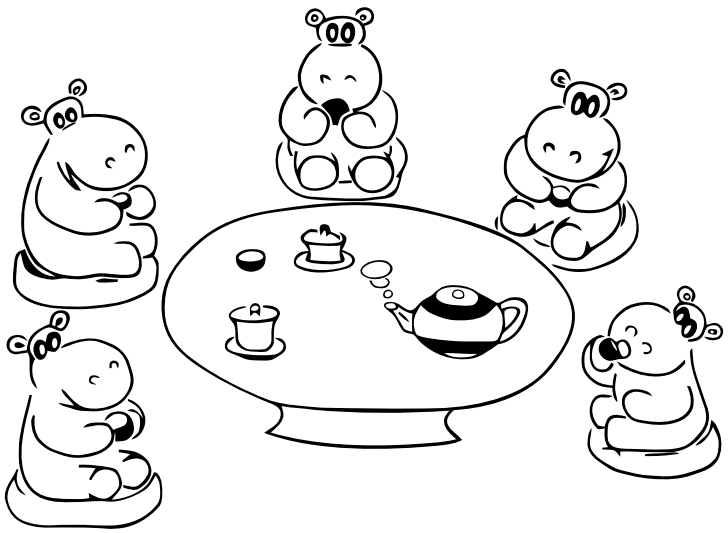
Je-li řešení jednoznačné, zbude v každé rovnici pouze jediný koeficient. Pak je vyjádření neznámé triviální, stačí podělit pravou stranu koeficientem.

Je-li řešení parametrizované, pak v rovnici navíc zbudou koeficienty parametrů. Stačí je tedy převést na pravou stranu a rovnou získáme vektorový zápis.

Kvůli vytváření nul i nad aktuálním řádkem provedeme dvakrát tolik operací než při Gaussově eliminaci. Asymptotická složitost ovšem zůstane stejná.

Program (Python):
`http://ksp.mff.cuni.cz/viz/35-1-S3.py`

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



# Výsledková listina první série třicátého pátého ročníku KSP

| | *řešitel* | *škola* | *ročník* | *sérií* | *1-1* | *1-2* | *1-3* | *1-4* | *1-S* | *1-X1* | *série* | *KSP-X* | *celkem* |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0. | | | | | 9 | 10 | 12 | 14 | 15 | 10 | 60,0 | 10,0 | 60,0 |
| 1. | Benjamin Swart | MensaG | 4 | 6 | 9 | 10 | 12 | 14 | 15 | 8 | 60,0 | 8,0 | 60,0 |
| 2.–3. | David Kolář | GJírovcČB | 4 | 6 | 8,5 | 10 | 12 | 13 | 14,5 | | 58,0 | 0,0 | 58,0 |
| | Vojtěch Lančarič | SPŠG Třebešín | 4 | 6 | 9 | 10 | 12 | 13 | 14 | | 58,0 | 0,0 | 58,0 |
| 4. | Štěpán Mikéska | GJarošeBO | 4 | 1 | 9 | 10 | 12 | 13 | 13,5 | 1 | 57,5 | 1,0 | 57,5 |
| 5. | Viktor Helmich | GTMannaPH | 4 | 1 | 9 | 10 | 12 | 12 | 14 | 0 | 57,0 | 0,0 | 57,0 |
| 6. | Kateřina Doubková | GNAlejíPH | 4 | 2 | 9 | 10 | 7 | 10 | 13,5 | | 49,5 | 0,0 | 49,5 |
| 7. | Zuzana Aubrechtová | GHeyrovPH | 4 | 5 | 9 | 10 | 12 | 7 | 11 | 0 | 49,0 | 0,0 | 49,0 |
| 8.–9. | Patrik Číhal | SŠKKamPard | 3 | 5 | 9 | 10 | 12 | 4 | 13,5 | 8 | 48,5 | 8,0 | 48,5 |
| | Petr Slonek | GJarošeBO | 4 | 1 | 8 | 10 | 12 | 11 | 7,5 | | 48,5 | 0,0 | 48,5 |
| 10.–12. | Robert Klimt | GDobříš | 3 | 1 | 4,5 | 10 | 12 | 7 | 14,5 | | 48,0 | 0,0 | 48,0 |
| | Filip Prášil | GJiříPoděb | 4 | 2 | 9 | 10 | 12 | 2 | 15 | 1 | 48,0 | 1,0 | 48,0 |
| | Kryštof Tahal | GUBalvanJN | 4 | 1 | 4 | 10 | 12 | 8 | 14 | 0 | 48,0 | 0,0 | 48,0 |
| 13. | Erik Ježek | SPŠSmíchov | 1 | 1 | 7,5 | 10 | 12 | 10 | 8 | 5 | 47,5 | 5,0 | 47,5 |
| 14.–15. | Honza Kocourek | ParkLane | 3 | 2 | 5 | 10 | 12 | 7 | 13 | | 47,0 | 0,0 | 47,0 |
| | Jáchym Kouba | GJŠkodyPŘ | 3 | 6 | 9 | 10 | 12 | 3 | 13 | | 47,0 | 0,0 | 47,0 |
| 16.–17. | Albert Bakoč | GZborovPH | 2 | 5 | 8,5 | 10 | 12 | | 14,5 | | 45,0 | 0,0 | 45,0 |
| | Jan Slíva | MensaG | 2 | 6 | 9 | 10 | 12 | 14 | | | 45,0 | 0,0 | 45,0 |
| 18. | Marek Raška | GTři | 4 | 1 | 8,5 | 7 | 12 | 4 | 13 | | 44,5 | 0,0 | 44,5 |
| 19. | Matúš Púll | GZborovPH | 3 | 4 | 9 | 10 | 12 | | 13 | | 44,0 | 0,0 | 44,0 |
| 20.–21. | Anna-Kristina Migel | GNAlejíPH | 0 | 1 | 5 | 10 | 12 | 11 | 5 | 6 | 43,0 | 6,0 | 43,0 |
| | Ondřej Pupík | GRožnovPR | 3 | 3 | 9 | 10 | 12 | 2 | 10 | | 43,0 | 0,0 | 43,0 |
| 22.–23. | Viktor Číhal | SPŠSmíchov | 3 | 6 | 9 | 10 | 12 | 11 | | | 42,0 | 0,0 | 42,0 |
| | Patrik Prítrský | GGrössBA | 2 | 1 | 9 | 10 | 12 | 6 | 5 | 1 | 42,0 | 1,0 | 42,0 |
| 24.–25. | Adam Červenka | GJarošeBO | 4 | 1 | 9 | 10 | 12 | 0 | 10,5 | | 41,5 | 0,0 | 41,5 |
| | Jakub Ondroušek | GTomkovaOL | 3 | 11 | 7,5 | 10 | 12 | | 12 | | 41,5 | 0,0 | 41,5 |
| 26. | Vít Kaděra | G Wicht | 1 | 1 | 5 | 10 | 12 | 8 | 5 | | 40,0 | 0,0 | 40,0 |
| 27. | Michael Jarvis | GŠpitálsPH | 1 | 1 | 4,5 | 10 | 12 | 13 | | 0 | 39,5 | 0,0 | 39,5 |
| 28. | Petr Němec | G Wicht | 1 | 1 | 5 | 10 | 12 | 8 | | 0 | 35,0 | 0,0 | 35,0 |
| 29. | Adam Jahoda | GKepleraPH | 4 | 3 | 9 | 10 | 5 | 9 | | | 33,0 | 0,0 | 33,0 |
| 30. | Adam Kolník | SSŠVTPraha | 4 | 10 | 9 | 10 | 12 | | | | 31,0 | 0,0 | 31,0 |
| 31. | Erik Sabol | GČeskoliPH | 3 | 2 | 5 | 7 | 3 | 5 | 10,5 | 2 | 30,5 | 2,0 | 30,5 |
| 32. | Jakub Binter | GČeskáČB | 0 | 1 | 4 | 10 | 9 | 7 | 0 | | 30,0 | 0,0 | 30,0 |
| 33. | Jan Ševeček | G UherBrod | 1 | 1 | | 10 | 12 | 6 | | | 28,0 | 0,0 | 28,0 |
| 34. | Martin Šindelář | GGrössBA | 3 | 1 | 8,5 | 10 | 9 | | | | 27,5 | 0,0 | 27,5 |
| 35.–36. | Daniel Culliver | GZborovPH | 3 | 3 | | 10 | 1 | 2 | 11 | | 24,0 | 0,0 | 24,0 |
| | Finley Stuart | GPísnickáPH | 2 | 1 | 8 | 2 | 1 | 6 | 7 | 0 | 24,0 | 0,0 | 24,0 |
| 37.–38. | Richard Dobíšek | MensaG | 2 | 1 | 9 | 10 | 1 | 2 | | | 22,0 | 0,0 | 22,0 |
| | Julie Krejčí | PraKonz | 3 | 2 | | 7 | 9 | 1 | 5 | | 22,0 | 0,0 | 22,0 |
| 39. | Kateřina Vomelová | GÚstavníPH | 3 | 3 | | 2 | 12 | 6 | | | 20,0 | 0,0 | 20,0 |
| 40. | Janek Hlavatý | GJirsíkaČB | 4 | 10 | 9 | 10 | | | | | 19,0 | 0,0 | 19,0 |
| 41. | Michal Martínek | GÚstavníPH | 2 | 3 | | 10 | 5 | | | | 15,0 | 0,0 | 15,0 |
| 42. | Jakub Hampl | GMělník | 3 | 2 | | | 3 | 7 | 4,5 | 0 | 14,5 | 0,0 | 14,5 |
| 43. | Oto Skýpala | GJŠkodyPŘ | −1 | 1 | | | 12 | 2 | | | 14,0 | 0,0 | 14,0 |
| 44. | Radomír Budínek | GŘíč | 4 | 1 | | | 5 | 7 | | | 12,0 | 0,0 | 12,0 |
| 45.–47. | Jakub Kopčil | GMikulášPL | 4 | 3 | | 10 | 0 | | | | 10,0 | 0,0 | 10,0 |
| | Jan Kotovský | GPísnickáPH | 4 | 10 | | 10 | | | | | 10,0 | 0,0 | 10,0 |
| | Lucian Poljak | GJŠkodyPŘ | 1 | 1 | | 10 | | | | | 10,0 | 0,0 | 10,0 |
| 48. | Ondřej Novák | G Brandýs | 0 | 1 | | | 1 | | 8 | | 9,0 | 0,0 | 9,0 |
| 49. | Vladimír Sklenár | GTerVans | 3 | 6 | | | 3 | | 4,5 | | 7,5 | 0,0 | 7,5 |
| 50.–51. | Matěj Smetana | AkademGPH | 2 | 1 | | 5 | | | | | 5,0 | 0,0 | 5,0 |
| | Jan James Soukup | GKlatovy | 4 | 1 | 5 | | | | | | 5,0 | 0,0 | 5,0 |
| 52. | Jáchym Löwenhöffer | GEvolutionJM | 2 | 1 | 4 | | | | | | 4,0 | 0,0 | 4,0 |
| 53.–56. | Martin Dobruský | SŠKKamPard | 4 | 1 | | | 3 | | | | 3,0 | 0,0 | 3,0 |
| | Miroslav Kolouch | GJírovcČB | 3 | 1 | | | 3 | | | | 3,0 | 0,0 | 3,0 |
| | Andrea Mikulová | BGOstrava | 4 | 2 | 3 | | | | | | 3,0 | 0,0 | 3,0 |
| | Petr Šišlák | GZborovPH | 2 | 1 | | 2 | 1 | | | | 3,0 | 0,0 | 3,0 |

| | *řešitel* | *škola* | *ročník* | *sérií* | *1-1* | *1-2* | *1-3* | *1-4* | *1-S* | *1-X1* | *série* | *KSP-X* | *celkem* |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 57.–58. | Alexandr Bihun | GJírovcČB | 3 | 1 | | 2 | | | | | 2,0 | 0,0 | 2,0 |
| | Jakub Vlček | GPříbor | 4 | 1 | | 2 | | | | | 2,0 | 0,0 | 2,0 |
| 59. | Vít Mitáš | GPolička | 1 | 1 | | | 1 | | | | 1,0 | 0,0 | 1,0 |

## Výsledková listina KSP-X po první sérii třicátého pátého ročníku

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

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/*