Korespondenční Seminář z Programování

ZAČÁTEČNICKÁ KATEGORIE

30. ročník | **KSP-Z** | **Červenec 2018**

Milí řešitelé a řešitelky!

Zpožděný mezistátní motorový osobní vlak KSP-30 právě přijíždí na první kolej. Ve vlaku je řazen vůz na přepravu hrochů, vzorové řešení 4. série a závěrečná výsledková listina. Všem účastníkům se omlouváme za zpoždění vlaku, ale i celého letošního ročníku KSP. Doufáme, že nám zachováte přízeň i napřesrok (a že KSP bude opět fungovat obvyklým způsobem), a třeba zkusíte řešit i hlavní kategorii. Zatím přejeme krásné prázdniny a těšíme se na ty z vás, kdož přijedou na podzimní soustředění. Vaši organizátoři.

Řešení čtvrté série začátečnické kategorie 30. ročníku KSP

30-Z4-1 Statistika sprintů

Na začátek je vhodné uvědomit si, že místo nejnižšího průměru stačí hledat nejnižší součet (sumu). To proto, že průměr k-tice čísel $(a_1, \ldots, a_k)$ je $(a_1 + \ldots + a_k)/k$ a k je kladná konstanta. Přeformulujeme tedy zadání na hledání nejnižší sumy.

Triviální řešení může vypadat takto: Spočítáme sumy pro k-tice $(x_0, \ldots, x_{k-1})$, $(x_1, \ldots, x_k)$, … Sumy si budeme ukládat do pole velikost $n-k$ a index v poli bude ukazovat první prvek k-tice. Toto pole poté projdeme a najdeme v něm minimum (průběžně si udržujeme nalezené minimum a příslušný index).

Toto řešení poběží v čase $\mathcal{O}((n-k)\cdot k)$, protože $(n-k)$-krát sčítáme k-tici čísel. Pokud k je výrazně menší než n, můžeme výraz zjednodušit na $\mathcal{O}(nk)$.

Problém triviálního řešení je v tom, že mnohokrát sčítáme stejná čísla. Obrázek níže ukazuje, jaké čtveřice sčítáme (vždy jeden obdélník přísluší jedné čtveřici). Čím tmavší pozadí, tím vícekrát se daný prvek přičte (16 se přičte ve třech čtveřicích).

Všimněme si, že když přecházíme od k-tice s indexem i ke k-tici s indexem $i+1$, tak jediné, v čem se suma změní, je to, že vypadne prvek i a přibude prvek $i+k$.

Sumy tedy nemusíme počítat pokaždé znovu: jakmile známe i-tou sumu, můžeme z ní v konstantním čase spočítat $(i+1)$-ní – stačí jeden prvek odečíst a jeden přičíst. Jedině pro pozici 0 stále musíme poctivě nasčítat k prvků.

Tím vylepšíme předchozí řešení na časovou složitost $\mathcal{O}(n)$, protože na každý prvek sáhneme maximálně dvakrát.

Program (Python 3):
`http://ksp.mff.cuni.cz/viz/30-Z4-1.py`

Vojta Sejkora

30-Z4-2 Klíče od tělocvičny

Základní myšlenka je jednoduchá. Budeme chtít postupně vytvořit seznamy S_i obsahující všechny členy s klíčem kvality i (pro i od 0 do K). Potom jen sečteme jejich délky a dostaneme výsledek.

S_0 je prostě seznam vedoucích klubu, ten dostaneme přímo na vstupu.

Nyní bychom chtěli vytvořit seznam S_1. To uděláme tak, že projdeme seznam S_0, pro každého z členů v tomto seznamu se podíváme na jeho známé a ty přidáme do S_1. Pak můžeme obdobným způsobem projít S_1 a známé přidat do S_2, atd.

Ale má to dva háčky:

- Některý z těchto známých už možná klíč má. Například pokud se znají dva vedoucí A a B, při procházení známých A narazíme na B, ale nechceme ho přidat do S_1 (protože už má klíč kvality 0, tak mu nebudeme dávat horší).
- Potřebujeme umět rychle zjistit seznam známých daného člena.

S prvním problémem se vypořádáme tak, že si pro každého člověka poznamenáme, jestli už dostal klíč. K tomu si pořídíme pole D délky N, kde $D[i]$ bude 1, pokud už jsme členovi i dali klíč (zařadili ho do některého seznamu S_i), jinak 0.

Takovéto uložení v poli je důležité, protože nám umožňuje rychle (v konstantním čase) testovat, zda už daný člověk má klíč. Kdybychom místo toho měli seznam lidí, kteří dostali klíč, museli bychom pro ověření jednoho člověka celý tento seznam projít, což by trvalo čas $\mathcal{O}(N)$.

Ještě zbývá jedna otázka: jak zařídit, abychom uměli rychle zjistit seznam známých nějakého člena? Jednoduše: prostě si budeme pro každého člena seznam známých pamatovat. Tyto seznamy vytvoříme při načítání vstupu: když načteme řádek popisující známost lidí i a j, přidáme j do seznamu známých i a zároveň i do seznamu známých j.

Můžeme si například pořídit seznam Z, jehož prvky budou jednotlivé seznamy známých – $Z[i]$ bude seznam známých i-tého člověka, $Z[i][0]$ bude první známý i-tého člověka, atd.

Na vstup se taky můžeme dívat jako na graf, jehož vrcholy tvoří členové klubu a hrany známosti mezi nimi. Potom výše popsaný seznam Z není nic jiného než reprezentace grafu seznamem sousedů a náš algoritmus je vlastně upravené prohledávání grafu do šířky (liší se tím, že začínáme prohledávat z několika vrcholů současně). Pokud vám ještě tyto pojmy nic neříkají, dočtete se o nich v naší grafové kuchařce.[1]

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

Program (Python 3):
`http://ksp.mff.cuni.cz/viz/30-Z4-2.py`

Filip Štědronský

30-Z4-3 Uhlazovací válec

Překvapivě, v této úloze nebylo vůbec nic záludného. Pokud znáte libovolný algoritmus na průchod grafu nebo čtvercové mřížky, mohli jste ho s úspěchem použít. Nutným předpokladem pro takto jednoduché řešení je ovšem konstantní velikost desky (v našem případě 3×3), bez něj je úloha výrazně složitější.

Pojďme si připomenout prohledávání do hloubky. Na začátku si na zásobník vložíme levý horní roh, a postupně budeme vytahovat možná umístění válce. Navštívená políčka si musíme někam poznamenat, abychom je nenavštěvovali vícekrát. Tomuto účelu poslouží třeba další mřížka booleovských proměnných. Poté započítáme všechna políčka, která válec na tomto umístění pokrývá – i toto započítání si ale musíme u každého políčka poznamenat, odděleně od navštívení. Na závěr pro každého ještě nenavštíveného souseda zjistíme, zda se na dané místo dá válec posunout, a pokud ano, vložíme jej na vrchol zásobníku.

Pokud bychom chtěli být extra úsporní, stačí nám ověřit nepřítomnost překážek těsně za hranami plochy zabrané válcem. Protože je však v naší úloze válec konstantně velký (a prakticky velmi malý), nemusíme se s tím zatěžovat a můžeme klidně kontrolovat celou plochu čtyřikrát.

Jestliže jste místo explicitního zásobníku (např. v seznamu) využili rekurze a zásobníku volání funkcí, mohli jste narazit na to, že ve vyšších programovacích jazycích je cena za volání funkce příliš vysoká. Pokud to jde tak snadno jako v této úloze, doporučujeme se rekurzi vyhnout, a to zejména v Pythonu.

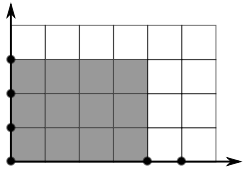
Takto upravené prohledávání si zachovává svou časovou složitost, která je lineární s počtem políček, která prohledáváme. Znovu připomínáme, že je ve skutečnosti násobena velikostí plochy, která se však díky konstantní velikosti „schová do $\mathcal{O}$-čka“.

Program (Python 3):
`http://ksp.mff.cuni.cz/viz/30-Z4-3.py`

Ondra Hlavatý

30-Z4-4 Ohrazení zahrádky

O kolících na x-ové ose budeme říkat, že jsou *vodorovné*, kolíkům na y-ové ose budeme říkat *svislé*. Podle zadání jsme na vstupu dostali seznamy vodorovných a svislých kolíků X a Y délky m a n – konkrétně v seznamu X jsou x-ové souřadnice všech vodorovných kolíků a v seznamu Y jsou y-ové souřadnice všech svislých kolíků.



Aby Zuzka zatlučením posledního kolíku vytvořila obdélníkovou zahradu rovnoběžnou s osami, musí ho zatlouct přesně napravo od nějakého svislého kolíku a zároveň přesně nahoru nad nějaký vodorovný kolík. Jinými slovy, Zuzka může kolík zatlouct jen na pozice tvaru $[x, y]$, kde x je prvek pole X a y je prvek pole Y. V takovém případě bude mít zahrada objem $x \cdot y$ jednotek.

Můžeme tedy vyzkoušet všechny možnosti, jak vybrat nějaké x z pole X a k němu y z pole Y. Pro každou možnost pak můžeme spočítat $x \cdot y$, ověřit, zda tento součin nepřekračuje Q, a pokud nepřekračuje, porovnat ho s dosavadním maximem a případně součin prohlásit za nové maximum.

Takový algoritmus bude mít časovou složitost $\mathcal{O}(mn)$, protože máme $m \cdot n$ možností na výběr x a y. Paměťová složitost je $\mathcal{O}(m + n)$, protože nám stačí pamatovat si jen X a Y.

Zrychlujeme…

S takto pomalým řešením se ale (samozřejmě) nespokojíme. V první řadě si všimneme, že náš algoritmus pracuje stejně dobře, ať už jsou na vstupu prvky X a Y uvedené v jakémkoliv pořadí. Zkusíme tedy prvky na začátku vzestupně seřadit (za to nic nedáme, oproti $\mathcal{O}(mn)$ je čas na řazení zanedbatelný) a uvidíme, zda neumíme seřazení nějak využít.

Jak se náš algoritmus chová na seřazeném vstupu? Nejprve vezme nejmenší x a k němu bere všechna y od nejmenšího, pak pokračuje o něco větším x' atd. Pro konkrétní x jsou nejdřív součiny $x \cdot y$ příliš malé (už máme lepší maximum), pak jsou chvíli „tak akorát“ (mají potenciál zvýšit dosavadní maximum) a pak jsou zase příliš velké (přesahují Q).

Chtěli bychom umět přeskakovat příliš malé a příliš velké součiny. Je zřejmé, že jakmile pro konkrétní x přeroste $x \cdot y$ horní mez Q, budou i všechny další součiny příliš velké a můžeme přeskočit až na zpracování dalšího x. S příliš malými součiny ale podobný trik udělat neumíme.

Pomůžeme si úpravou našeho algoritmu – místo abychom pro jedno x procházeli y od nejmenšího, dokud je součin v mezích, půjdeme naopak od největších y, dokud součin meze překračuje. Jakmile totiž narazíme na první součin $x \cdot y$ nepřekračující Q, můžeme s ním zkusit vylepšit dosavadní maximum a přeskočit na další x – všechny ostatní součiny budou menší, takže s nimi maximum nezlepšíme.

Zdá se, že jsme se dostali do stejné situace, jako předtím – víme, kdy už máme skončit, ale nevíme, kde máme začít. Tentokrát si ale umíme pomoct: pro konkrétní x nebudeme procházet všechna y od největšího, ale začneme tam, kde jsme skončili s minulým x'. Jelikož x se zvětšují, platí, že pokud byl nějaký součin $x' \cdot y$ příliš velký, bude i součin $x \cdot y$ příliš velký. Přeskakujeme tedy jen ty y, které pro nás již v minulém kole byly příliš velké, a tím spíš pro nás budou příliš velké teď.

Jaké má toto řešení časovou složitost? Na začátku řadíme obě pole, což zvládáme v $\mathcal{O}(n \log n + m \log m)$. V druhé části sice provádíme m cyklů a v každém můžeme projít až $\Theta(n)$ různých y, ale není těžké si rozmyslet, že dohromady vyzkoušíme jen $\Theta(n + m)$ různých kombinací x a y: mezi každými dvěma kombinacemi se buď x zvýší nebo y sníží a druhá složka zůstane stejná. Dohromady se ale x může zvýšit nejvýše m-krát a y se může snížit jen n-krát, celkově tedy může být jen $n + m$ kombinací. Časová složitost druhé části je tedy $\mathcal{O}(n+m)$, celková časová složitost je $\mathcal{O}(n \log n + m \log m)$. Paměťová složitost zůstává $\mathcal{O}(n+m)$.

Program (Python 3):
`http://ksp.mff.cuni.cz/viz/30-Z4-4.py`

Říša Hladík

30-Z4-5 Sběr jablek

Kolem každého stromu je nějaký interval, ve kterém jsou jablka nejblíže k němu, a všude jinde bude nějaký jiný strom blíž. Takový interval je z každé strany ohraničený středem mezi naším stromem a nejbližším stromem z dané strany (nebo není ohraničený, když tam žádný další strom není). Když si stromy setřídíme, můžeme si jedním průchodem vyrobit takový interval pro každý strom – jako hranici vždy použijeme střed mezi dvěma stromy.

Teď potřebujeme pro každé jablko najít, do kterého spadá intervalu. Jablka si také můžeme setřídit podle polohy a pak je postupně projít. Budeme je procházet zároveň s intervaly, do kterých můžou spadat – pro každé jablko si posuneme „kurzor“ na interval tak, aby do něj spadalo. Pak se podíváme, jaký je tam strom, a vypíšeme, jak je od jablka daleko. Když budeme mít hranice intervalů v poli `hranice_intervalu` (které začíná mezi prvním a druhým intervalem a končí nekonečnem), tak by to v kódu mohlo vypadat takto:

```
nh = 0  # Nejbližší hranice
for jablko in jablka:
  # Posuneme si hranici tak, aby byla vždy
  # za jablkem. Někdy se tento cyklus nemusí
  # vůbec vykonat.
  while hranice_intervalu[nh] < jablko:
      nh += 1

  # Strom je na stejné pozici jako jeho interval
  vzdalenost = abs(stromy[nh] - jablko)
  print(vzdalenost)
```

Nalezení hranic i průchod jablky bude potřebovat řádově stejně času jako je stromů, respektive jablek. Setřídění polí provedeme nějakým rozumným algoritmem z knihovny (třeba MergeSort nebo QuickSort) a to bude trvat $\mathcal{O}(N \log N)$ času pro jablka i stromy. Když budeme mít N stromů a M jablek, budeme potřebovat na celý algoritmus $\mathcal{O}(N \log N + M \log M)$ času.

Program (Python 3):
`http://ksp.mff.cuni.cz/viz/30-Z4-5.py`

Standa Lukeš

30-Z4-6 Analýza nadávek

Máme najít v *textu* délky N podřetězec délky K (budeme mu říkat *slovo*), který se vyskytuje co nejvíckrát zopakovaný bezprostředně za sebou. Navíc máme slíbeno, že K je mnohem menší než N.

Většinou se hodí začít úplně primitivním algoritmem, který je vlastně jenom překladem zadání. Prostě vyzkoušíme všechna možné slova a pro každé z nich spočítáme, kolikrát po sobě se opakuje. Na to stačí porovnat ho s následujícími K znaky, pak s dalšími K znaky a tak dále.

Jak bude takový „dřevorubecký“ algoritmus rychlý? Máme celkem $N - K + 1$ slov, každé z nich můžeme porovnávat s až N dalšími znaky. To dává časovou složitost $\mathcal{O}((N - K + 1) \cdot N) = \mathcal{O}(N^2)$.

Nic moc, řeknete si. Ale jde to snadno zrychlit: Pokud se nějaké slovo S zopakovalo t-krát, tak se žádné jiné slovo začínající během prvních $t - 1$ opakování slova S nemůže opakovat víc než t-krát. Nejpozději po t-tém opakování se totiž zarazíme o stejnou neshodu znaků, o jakou jsme se zarazili u slova S.

Když jsme tedy pro slovo na pozici i našli t opakování, nemusíme příště zkoušet slovo na pozici $i + 1$, ale stačí popoběhnout na pozici $i + tK - 1$. Například pro $K = 4$ a text

`OKOPOKOPOKOPAKOPAKOPAKOPAKOPA`

objevíme 4 výskyty slova `OKOP` a pak pokračujeme od `KOPA`. (`KOPO`, `OPOK` ani `POKO` se nemohou opakovat víckrát než `OKOP`.)

Tím pádem každý znak textu prozkoumáme nejvýše K-krát, takže složitost algoritmu klesne na $\mathcal{O}(NK)$. To je, pokud zanedbáme závislost na K, jistě nejlepší možné.

Program (Python 3):
`http://ksp.mff.cuni.cz/viz/30-Z4-6-nk.py`

Lineární řešení

Také vám vrtá hlavou, jestli existuje algoritmus, který by byl rychlý i pro velké K? Tak tady je.

O znaku na i-té pozici řekneme, že je *nadějný,* pokud je stejný jako znak na $(i - K)$-té pozici. Všimneme si, že pokud se nějaké slovo vyskytne dvakrát za sebou, všechny znaky jeho druhého výskytu jsou nadějné. A naopak: pokud najdeme K po sobě jdoucích nadějných znaků, znamená to, že se nějaké slovo vyskytlo podruhé. Platí to i obecněji: slovo se vyskytne t-krát za sebou právě tehdy, když je v textu $(t - 1)K$ po sobě jdoucích nadějných znaků.

Hezky je to vidět na následujícím příkladu ($N = 22$, $K = 4$, nadějné znaky jsou označené hvězdičkami):

```
ABCDABCEDBCEDBCEDBXYDB
....***..*********..**
```

Existuje úsek 9 po sobě jdoucích nadějných znaků, takže se nějaké slovo (`BCED` nebo `CEDB`) vyskytuje $\lfloor 9/4 \rfloor + 1 = 3$ krát za sebou.

Našemu algoritmu tedy stačí najít nejdelší úsek po sobě jdoucích nadějných znaků. To jistě zvládne v čase $\mathcal{O}(N)$ nezávisle na K.

Program (Python 3):
`http://ksp.mff.cuni.cz/viz/30-Z4-6-n.py`

Martin „Medvěd“ Mareš

Výsledková listina čtvrté série začátečnické kategorie 30. ročníku KSP

	řešitel	*škola*	*ročník*	*sérií*	*Z4-1*	*Z4-2*	*Z4-3*	*Z4-4*	*Z4-5*	*Z4-6*	*série*	*celkem*
0.					8	10	10	12	12	14	66,0	264,0
1.	Ondřej Jamelský	G Cheb	0	0	8	10	10	12	12	14	66,0	263,0
2.	Petr Aubrecht	GHeyrovPH	3	0	8	10	10	12	12	14	66,0	254,0
3.	Petr Budai	G JGJ PH	1	0	8	10	10	12	12	14	66,0	239,0
4.	Michal Bravanský	GBílovec	0	0	8	10	10	12	12	13	65,0	233,0
5.–6.	Daniel Skýpala	GTomkovaOL	0	0	8	10	10	12	11	12	63,0	226,0
	Jiří Kvapil	GTomkovaOL	0	0	8	10	10	12	12	11	63,0	226,0
7.	Dalibor Kramář	G BO-Řeč	3	0	8	10	10	12	12	14	66,0	169,0
8.	Janek Hlavatý	GJirsíkaČB	−1	0	8	10	10	12	12	14	66,0	150,0
9.–10.	Ondřej Hráček	GOlgHavl	1	0	8	3		12	12		35,0	143,0
	Jan Kotovský	GPísnickáPH	−1	0	8	10	4	3	12		37,0	143,0
11.	Terézia Strišovská	GJHroncaBA	2	0	8	10		6			24,0	140,0
12.	Filip Kastl	GKepleraPH	2	0	8	1			12	13	34,0	138,7
13.	Ondřej Bleha	GBNěmcovHK	3	0							0,0	138,0
14.	Vojtěch Káně	G Brandýs	2	0	8	10	10	12	12	14	66,0	137,0
15.	Michal Mlčoch	G UherBrod	3	0	8			6			14,0	123,5
16.	Martin Bencko	GOhradníPH	1	0	8			6	6		20,0	118,0
17.	Klára Tauchmanová	GOhradníPH	4	0							0,0	115,0
18.	Robert Jaworski	GÚstavníPH	0	0	8	10					18,0	113,0
19.–20.	Albert Kučera	GNadŠtolPH	2	0	8	10	4	6	6	4	38,0	110,0
	Jakub Komárek	GUHradiště	3	0	8	10					18,0	110,0
21.	Lukáš Gáborik	GTajBanBys	1	0	8			12	12		32,0	101,0
22.	Lucie Vomelová	GŠpitálsPH	2	0							0,0	100,0
23.	Václav Zvoníček	GJarošeBO	2	0	8	10					18,0	88,3
24.	Martin Zmitko	G FrýdlNOs	2	0	8	1					9,0	85,0
25.	Jan Vodstrčil	G VMýto	1	0							0,0	82,0
26.	Jaroslav Paška	ŠPMNDaGB	3	0							0,0	79,0
27.	Radim Buráň	G UherBrod	3	0	0			6			6,0	78,0
28.	Michal Kodad	SPŠSmíchov	2	0							0,0	74,0
29.	Jan Hartman	GChodoviPH	2	0	8	10	10	6	12		46,0	72,0
30.	Adam Húšťava	EupSchoolLux	0	0	8	7				12	27,0	71,7
31.	Radoslav Hašek	GČáslav	4	0	8			12			20,0	70,0
32.	Kristina Galikova	ŠPMNDaGB	3	0							0,0	68,7
33.	David Klement	GNAlejíPH	2	0	8	10	10				28,0	68,0
34.	Matěj Volf	GCoubTábor	0	0	8			6			14,0	66,0
35.–36.	Jan Černý	BiGy Žďár	2	0							0,0	62,0
	Jakub Šuráň	GStrážnice	3	0							0,0	62,0
37.	Jakub Nevařil	G UherBrod	0	0							0,0	60,0
38.	Martin Kostrubanič	GČáslav	4	0	8	10	10	6			34,0	58,0
39.–40.	Jiří Tlamicha	GŘíč	1	0							0,0	52,0
	Vojtěch Žák	GŠpitálsPH	2	0							0,0	52,0
41.	Erik Berta	GAlejKošice	3	0							0,0	51,0
42.–43.	Vojtěch Poupa	CírkG Plzeň	0	0							0,0	49,0
	Jiří Vlček	GFXŠaldyLI	2	0							0,0	49,0
44.–46.	Jakub Ferenčík	GDašickáPA	0	0							0,0	47,0
	Matyáš Lorenc	GJungmanLT	4	0							0,0	47,0
	Jan Piroutek	GŠpitálsPH	2	0							0,0	47,0
47.	Michal Starý	GNoMěsNMor	4	0							0,0	46,0
48.–49.	Marie Kalousková	GNAlejíPH	2	0	8	7			12		27,0	45,0
	Ondřej Wrzecionko	GTěš	3	0	8			6			14,0	45,0
50.	Alexandra Géciová	GJHroncaBA	2	0	8	10		6			24,0	40,0
51.	Adam Verner	SPŠ Prosek	3	0	4						4,0	38,0
52.–53.	Jiří Bleha	SPSEPard	1	0	4	1		6			11,0	37,0
	Dávid Oravec	G DubNVáh	3	0							0,0	37,0
54.–55.	Anna Hollmannová	GSRandyJN	1	0					11		11,0	36,0
	Vojtěch Michal	GNVPláníPH	3	0							0,0	36,0
56.	Jan Koška	GJírovcČB	−2	0							0,0	35,0
57.	Radim Kopunec	G UherBrod	−2	0							0,0	34,0

	řešitel	*škola*	*ročník*	*sérií*	*Z4-1*	*Z4-2*	*Z4-3*	*Z4-4*	*Z4-5*	*Z4-6*	*série*	*celkem*
58.–59.	Michael Kozel	GZborovPH	4	0							0,0	29,0
	Dennis Pražák	GJirsíkaČB	3	0							0,0	29,0
60.–64.	Martin Cmiel	GOlgHavl	1	0							0,0	28,0
	Ondřej Daniš	GFPValMez	3	0							0,0	28,0
	Dominik Dinh	GNVPláníPH	3	0							0,0	28,0
	Jindřich Dítě	VOSPŠŽďár	2	0							0,0	28,0
	Filip Hejsek	GPísnickáPH	1	0							0,0	28,0
65.–66.	Ondřej Gonzor	G Brandýs	1	0							0,0	26,0
	Jan Kučera	SŠKlatovskáPL	1	0							0,0	26,0
67.	Vladimír Chudý	ZŠRonov	1	0							0,0	25,0
68.–72.	Ondřej Buček	GJarošeBO	4	0							0,0	23,0
	Evgenia Golubeva	GJosefskPH	3	0							0,0	23,0
	Tomáš Husák	GLitoměřPH	4	0							0,0	23,0
	Jan Najman	SPSEPard	1	0	4	3	0	6		10	23,0	23,0
	Marco Souza de Joode	GNadŠtolPH	1	0	4	5		6	8		23,0	23,0
73.–74.	Petr Hýbl	G UherBrod	2	0	4						4,0	22,0
	Zuzana Urbanová	GFXŠaldyLI	4	0							0,0	22,0
75.–76.	Štěpán Henrych	GŽat	2	0							0,0	20,0
	Filip Krul	SPŠSmíchov	2	0							0,0	20,0
77.	Tomáš Sláma	GTurnov	3	0							0,0	19,0
78.–84.	Karel Balej	GRokycany	3	0							0,0	18,0
	Robert Gemrot	GKomHavíř	1	0	8	10					18,0	18,0
	Matej Hockicko	TAPoprad	4	0							0,0	18,0
	Patrik Janko	SPŠSmíchov	2	0							0,0	18,0
	František Kmječ	G Brandýs	2	0							0,0	18,0
	Dávid Šutor	GTerVans	3	0							0,0	18,0
	Jakub Ucháč	ŠMaVVzt	2	0							0,0	18,0
85.	Jan Škoula	GBenesov	2	0	8	5	4				17,0	17,0
86.–87.	Jakub Hroník	GJiříPoděb	4	0							0,0	16,0
	Bronislav Růžička	GRokycany	−1	0							0,0	16,0
88.	Antonín Rousek	GDašickáPA	0	0							0,0	13,0
89.–92.	Erik Řehulka	ŠPMNDaGB	2	0							0,0	11,0
	Martin Sobotka	GLitoměřPH	2	0							0,0	11,0
	Jan Štěch	GJirsíkaČB	1	0							0,0	11,0
	Magdaléna Turinská	SZŠ Brandýs	1	0					11		11,0	11,0
93.	Petr Kubec	G Kolín	2	0							0,0	10,0
94.–96.	Tomáš Kratschmer	GMikulášPL	2	0							0,0	9,0
	Antonín Musil	PORGPha	1	0							0,0	9,0
	Kateřina Vokálová	G Kolín	2	0							0,0	9,0
97.–101.	Patrik Baláš	SPSEPard	4	0	8						8,0	8,0
	Václav Kelímek	SPŠBruntál	3	0							0,0	8,0
	Vojtěch Krupka	GJungmanLT	4	0							0,0	8,0
	Adéla Návratová	ZŠ MTyrš	0	0							0,0	8,0
	Martin Zimen	GJMasarJI	3	0							0,0	8,0
102.	Tomáš Dostál	MendelGOP	3	0	4						4,0	6,7
103.–105.	Daniela Hrbáčová	G Wicht	4	0							0,0	4,0
	Vít Novotný	GVoděraPH	3	0							0,0	4,0
	Michal Zacek	MensaG	1	0	4						4,0	4,0
106.	Vojtěch Kuchař	VOŠJičín	1	0							0,0	3,0
107.	Vojtěch Hronek	SPŠPísek	1	0							0,0	2,0
108.–111.	Lukáš Caha	GZborovPH	4	0							0,0	1,0
	Jan Hřebenář	20. ZŠ PL	−1	0							0,0	1,0
	Michal Rod	GJirsíkaČB	3	0							0,0	1,0
	Jakub Zemek	GUHradiště	3	0							0,0	1,0

KSP pro vás připravují studenti Matematicko-fyzikální fakulty Univerzity Karlovy.

Webové stránky: *https://ksp.mff.cuni.cz/*

E-mail: *ksp@mff.cuni.cz*

Diskusní fórum: *https://ksp.mff.cuni.cz/forum/*

Chcete-li s námi komunikovat bezpečně, můžete si ověřit náš HTTPS certifikát – jeho SHA1 fingerprint je: `E9:DB:EE:C6:62:BC:14:DE:09:E4:E8:97:DC:36:0E:87:B3:50:B0:01`.