Korespondenční Seminář z Programování

34. ročník | **KSP** | **Červen 2022**

Vzorová řešení páté série třicátého čtvrtého ročníku KSP

34-5-1 Lehátka

Není zřejmé, jak optimální vzdálenost najít „přímo“, ale pro danou minimální vzdálenost mezi hrochy snadno ověříme, zda ji umíme dodržet. Můžeme postupovat následujícím způsobem.

Binárně vyhledáme optimální vzdálenost hrochů. Víme, že tato hodnota bude určitě někde mezi nulou a vzdáleností krajních lehátek. Vyzkoušíme, zda umíme dodržet vzdálenost uprostřed mezi těmito hodnotami. Pokud ano, můžeme zkoušet vyšší hodnoty. Pokud ne, musíme zkusit nižší. Postupně půlíme interval zkoušených hodnot, dokud nenajdeme samotné optimum.

Jak přesně ověříme, že danou vzdálenost umíme dodržet? Můžeme opět použít binární vyhledávání. Určitě si nepohoršíme, když prvnímu hrochovi věnujeme první lehátko. Binárním vyhledáváním zjistíme, jaké nejbližší lehátko je v požadované vzdálenosti nebo dál. Tím jsme našli lehátko pro druhého hrocha a postupujeme podobně dál, dokud nezjistíme, že máme místo pro všechny hrochy, nebo nám došla lehátka, a požadovaná vzdálenost je tak příliš vysoká.

Pokud jako V označíme vzdálenost mezi krajními lehátky, celkový čas běhu našeho algoritmu bude $\mathcal{O}(H \log V \log L)$, protože binárně vyhledáváme v intervalu délky V a pro každou zkoušenou vzdálenost přiřadíme (až) H hrochům binárním vyhledáváním lehátka z L možností.

Program (C++):
`http://ksp.mff.cuni.cz/viz/34-5-1.cpp`

Program (Python 3):
`http://ksp.mff.cuni.cz/viz/34-5-1.py`

Úlohu připravili: Jirka Kalvoda, Martin Koreček, Jirka Setnička

34-5-2 Veletrh

Označíme N počet vrcholů grafu (místností), M počet hran mezi nimi, U počet událostí a $T_{\max}$ čas konce poslední události. Hledáme trasu (nějaký plán toho, kdy se chceme přesunout kam a jak dlouho tam počkat) s co nejvyšším *ziskem* – tak budeme říkat počtu minut událostí, kterých jsme během plánu zúčastnili.

Prohledávání časoprostoru

Nejprve nadefinujeme $\delta_{t,v}$, což je zisk z t-té minuty strávené ve vrcholu v. Tedy 1, pokud tam zrovna probíhá událost, jinak 0.

Nyní budeme počítat čísla $Z_{t,v}$: maximální možný zisk trasy, která na konci t-té minuty ($0 \leq t \leq T_{\max}$) skončí ve vrcholu v. Přitom $Z_{t,v} = -\infty$, pokud žádná taková trasa neexistuje.

Nejprve prozkoumáme případ $t = 0$, tedy první minutu veletrhu. Tu můžeme trávit pouze ve vrcholu 0, protože z jiného nestihneme přijít. Takže $Z_{t,0} = \delta_{t,0}$ a $Z_{t,v} = -\infty$ pro všechna $v \neq 0$.

Nyní uvažujme $t > 0$. Optimální trasa může končit buďto minutou čekání ve vrcholu v, anebo přesunem z nějakého vrcholu u. Pokud končí čekáním, máme z trasy zisk $Z_{t-1,v} + \delta_{t,v}$. Pokud trasa končí přesunem po hraně délky $\ell_{u,v}$, zisk je roven $Z_{t-\ell_{u,v},u}$ (což je $-\infty$, pokud $t < \ell_{u,v}$). Zkombinováním obou možností dostaneme:

$$Z_{t,v} = \max(Z_{t-1,v}, \max_u Z_{t-\ell_{u,v},u}).$$

Jak dlouho nám bude trvat spočítat všechna $Z_{t,v}$? Pro každý čas t procházíme všechny vrcholy v, pro každý vrchol v zkoumáme všechny do něj vedoucí hrany. Celkem tedy pro každé t projdeme každou hranu právě jednou. Výpočet všech $Z_{t,v}$ tedy trvá $\mathcal{O}(T_{\max} \cdot (N + M))$.

Pak už stačí vyzkoušet všechny vrcholy, kde jsme mohli být v poslední minutě veletrhu, a najít ten s maximálním ziskem. To nám časovou složitost nezkazí.

Nakonec musíme vymyslet, jak optimální trasu vypsat. Při výpočtu $Z_{t,v}$ si budeme pamatovat, pro jaký konec trasy se nabylo maxima – tedy zda jsme čekali ve v, nebo přišli z nějakého u. Pomocí toho pak můžeme optimální trasu rekonstruovat od konce k začátku. To opět složitost nepokazí.

Naše první řešení tedy pracuje v čase $\mathcal{O}(T_{\max} \cdot (N + M))$. Pojďme ho překonat – zejména se chceme zbavit závislosti na rozsahu časů.

Program (C++):
`http://ksp.mff.cuni.cz/viz/34-5-2-sim.cpp`

Předvýpočet vzdáleností

Zbavíme se omezení, že hrany vedou jenom mezi některými vrcholy. Pro každou dvojici vrcholů spočítáme jejich vzdálenost, tedy délku nejkratší cesty. Na to se hodí například Floydův-Warshallův algoritmus pracující v čase $\mathcal{O}(N^3)$. Najdete ho v kuchařce o dynamickém programování.

Od této chvíle můžeme předpokládat, že se lze odkudkoliv kamkoliv dostat přímo. Jen musíme při zavěrečném vypisování trasy umět přesuny rozložit na jednotlivé hrany – to je ale snadné, pokud si ve F.-W. algoritmu budeme pamatovat předchůdce vrcholu na nejkratší cestě.

Lemma o trpělivosti

Nyní ukážeme, že pokud nějakou událost navštívíme, nemusíme z ní nikdy netrpělivě odcházet před koncem. Přesněji řečeno, alespoň jedna z optimálních tras zůstane na každé navštívené události až do konce.

Důkaz provedeme tak, že vezmeme nějakou optimální trasu a postupně ji budeme upravovat, než bude „trpělivá“. Dokud není, vybereme nějakou událost u_i, na které nezůstane do konce. Trasu upravíme, aby na této události vydrželo o minutu déle. Tím pádem přijde o minutu později na následující událost u_{i+1}. Celkově se tedy zisk nezměnil. Jen pozor na případ, kdy jsme na u_{i+1} trávili jen její poslední minutu – tehdy u_{i+1} z trasy odstraníme a z u_i jdeme rovnou na u_{i+2}. Tam nepřijdeme později než v původní trase.

Opakováním tohoto postupu zařídíme, aby trasa na každé události čekala až do konce.

Rychlejší řešení

Naše rychlejší řešení nebude zkoumat všechny polohy v časoprostoru (dvojice vrchol a čas), nýbrž jenom konce událostí. Opět nás bude zajímat maximální zisk, jakého umíme dosáhnout trasou, která končí daným koncem události.

Události procházíme v pořadí časů začátků. Mějme událost u ve vrcholu v probíhající v časovém intervalu $\langle t_1, t_2\rangle$. Probereme všechny konce událostí, ze kterých dokažeme přijít na u.

To uděláme tak, že projdeme všechny vrcholy v'. Nechť u' je událost ve v' s časovým intervalem $\langle t'_1, t'_2\rangle$. Pokud z jejího konce odcházíme do v, přijdeme tam v čase $t'_2 + 1 + d$, kde d je předpočítaná vzdálenost z v' do v. Z události u tedy stihneme posledních $p = t'_2 + d - t_2$ minut. Pokud vyjde $p \leq 0$, nemá tedy smysl z u' na u chodit.

Stačilo by tedy projít všechny vrcholy v' a události u' v nich. Všechny u' jsme už dříve zpracovali, takže známe maximální zisky tras končících v nich. Stačí k nim připočítat zisk z části události u, kterou stihneme, a vybrat maximum přes všechny u'. Jen nesmíme zapomenout, že předchozí událost může být také ve v, takže do maxima ještě započítáme zisk z předchozí události ve v plus zisk z celé u.

Tento algoritmus by běžel v čase $\mathcal{O}(N^3 + U^2)$ – nejdříve by v čase $\mathcal{O}(N^3)$ předpočítal vzdálenosti. Pak by v $\mathcal{O}(U \log U)$ setřídil události. A nakonec by pro každou událost zkoumal až U předchůdců.

Ještě rychlejší řešení

Zkoumání tolika předchůdců si můžeme ušetřit. Kdykoliv zpracujeme událost, poznamenáme si ke všem vrcholům, kdy se do nich umíme dostat z konce aktuální události. Každý vrchol si tedy bude pamatovat možné časy příchodů z minulých událostí. Když pak zpracováváme další událost, podíváme se na časy příchodu do jejího vrcholu a odebíráme od začátku seznamu ty příchody, které jsou dostatečně brzké. Všimneme si, že žádný z nich se nevyplatí použít pro pozdější událost ve stejném vrcholu.

Za každou událost vytvoříme příchody do všech vrcholů. Celkem tedy vznikne $\mathcal{O}(UN)$ příchodů. Vytvořit i odebrat jeden příchod stojí konstantní čas. Kromě toho trávíme každou událostí už jenom konstantní čas.

Celkem tedy výpočtem strávíme čas $\mathcal{O}(N^3 + U \log U + UN)$. Do toho se jistě vejde i závěrečné vypsání optimální trasy. Toto řešení je lepší než předchozí, kdykoliv se v každém vrcholu koná aspoň jedna událost.

Program (C++):
`http://ksp.mff.cuni.cz/viz/34-5-2.cpp`

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

34-5-3 Jednoznačné cestování

Nejprve zkusme přiřadit hranám ceny tak, aby žádné dvě cesty neměly stejnou cenu. Cesta v našem případě bude posloupnost po sobě jdoucích hran, která nenavštíví žádný vrchol vícekrát. Zatím nebudeme hledět na to, aby nejlevnější cesta byla skutečně i tou nejkratší. Všimneme si, že různé cesty mají různou množinu hran, přes kterou prochází. Tedy určitě když každá množina hran bude mít unikátní celkovou cenu, tak máme vyhráno. Nyní si můžeme pomoct binárním zápisem čísel. Když i-té hraně přiřadíme cenu 2^i, tak součet libovolných cen bude v binárním zápise obsahovat jedničky na pozicích příslušných hran. Tedy když dostaneme do ruky nějakou celkovou cenu, zvládneme poznat, za které hrany platíme pouhým pohledem na binární zápis ceny.

Jak ale zařídit, aby nejlevnější cesta byla skutečně nejkratší? Když ke každé ceně hrany přičteme nějaké opravdu velké číslo, tak se rozdíly v cenách hran mohou stát zanedbatelné. Projít k hran bude stát pouze k-krát moc a nějaké drobné, zatímco projít $k + 1$ hran bude stát $k + 1$ krát moc a už nebude záležet na tom, jaké drobné jsou k tomu navíc. V původních cenách může celá cesta stát nejvýše:

$$2^0 + 2^1 + 2^2 + \cdots + 2^{m-1} = 2^m - 1.$$

Validní řešení získáme, když k cenám přičteme 2^m. Cena i-té hrany tedy bude $2^m + 2^i$. Snadno nahlédneme, že stále dvě různé cesty mají různou celkovou cenu.

Největší použitá cena je $2^m + 2^{m-1} \in \Theta(2^m)$ a časová složitost algoritmu je evidentně lineární.

Předchozí řešení můžeme vylepšit drobnou optimalizací – ceny nebudeme přiřazovat hranám ale vrcholům. Ovšem my vybíráme poplatky pouze na hranách, jak si zaúčtovat průchody za vrcholy? Můžeme polovinu zaúčtovat při vstupu a polovinu při odchodu z vrcholu. Všimneme si, že až na první a poslední vrchol vybereme za každou cestu správně. Ovšem nám stačí, aby byly ceny unikátní jenom mezi každou dvojicí vrcholů, takže to vlastně znamená, že z každé ceny odečteme jenom konstantu. Tím se jednoznačnost neporuší. Drobný problém je, že touto úpravou budeme chtít účtovat neceločíselné ceny, což zadání zakazuje. Ovšem když ceny vynásobíme dvěma, tedy za každou hranu zaúčtujeme součet okolních vrcholů, tak se také nic nepokazí. Na hranu spojující vrcholy s indexy x a y budeme účtovat $(2^n + 2^x) + (2^n + 2^y) = 2^x + 2^y + 2^{n+1}$

Největší použitá cena je nejvýše $2^{n+1} + 2^n + 2^{n-1} \in \Theta(2^n)$ a časová složitost algoritmu je stále lineární. Všimneme si, že kdyby graf obsahoval multihrany (dvě různé hrany mezi stejnou dvojicí vrcholů), tak už toto řešení nefunguje, kdežto řešení před optimalizací stále ano. Můžete si rozmyslet, jak řešení opravit, aby fungovalo i pro multihrany.

V předchozích úvahách jsme se snažili zajistit, aby každá cesta mezi dvojicí vrcholů měla unikátní cenu. Ovšem zadání po nás chce pouze to, aby nejlevnější cesta byla unikátní. Pojďme využít této vlastnosti na nalezení algoritmu, který generuje hranám menší ceny, byť bude běžet pomaleji.

Využijeme předchozí myšlenku o přičtení velkého čísla ke každé hraně. Budeme se tedy starat jen o nejlevnější cesty mezi dvojicemi vrcholů, protože nejlevnější cesta musí být po přičtení nutně nejkratší.

Začneme s grafem, který neobsahuje žádné vrcholy. Postupně do něj budeme přidávat hrany s nějakou cenou tak, aby nejlevnější z nejkratších cest mezi každou dvojicí vrcholů v aktuálním grafu byla stále unikátní.

Po přidání hrany jsou pro každou dvojici vrcholů dva kandidáti na nejlevnější cestu mezi nimi – původní nejlevnější cesta a nějaká cesta využívající aktuálně přidanou hranu. O původní nejkratší víme, že je jednoznačná.

Dále si ukážeme, že nejlevnější cesta využívající přidanou hranu je také jednoznačná. Nikdy se nevyplatí stejnou hranu při jedné z možností projít jedním směrem a při jiné v opačném směru, protože takovéto cesty budou nutně delší

než kdybychom došli na jeden konec hrany a pak bez použití přidané hrany pokračovali do cíle. Když ovšem existuje kratší trasa, trasy přes přidanou hranu nemusíme řešit. Ze startu na začátek hrany je nejlevnější cesta unikátní, z konce hrany do cíle taky, takže celá nejlevnější trasa využívající novou hranu musí být také unikátní. Zajímá nás tedy jenom, jak nastavit cenu přidávané hrany tak, aby nejlevnější cesty přes hranu a bez hrany nebyly stejně drahé. Pokud jsou pro danou dvojici cesty jinak dlouhé či jedna z možností vůbec neexistuje, nemůže nastat žádný problém.

Dvojic vrcholů ovšem máme jen $\frac{n(n-1)}{2}$, takže existuje nejvýše $\frac{n(n-1)}{2}$ cen nové hrany takových, že jejich použitím nebudou všechny nejlevnější cesty jednoznačné. Proto zvládneme vybrat číslo mezi 0 a n^2 takové, že se nic nerozbije. Cena cest může být nejvýše $(n-1) \cdot n^2 < n^3$. Tedy když každé hraně přiřadíme cenu mezi n^3 a $n^3 + n^2$, nemusíme řešit, jestli se jedná o nejkratší cestu.

Jak ovšem najít, kterou cenu hrany můžeme použít? Pro každou hranu bohužel budeme muset procházet všechny dvojice vrcholů. Můžeme si v průběhu celého algoritmu udržovat tabulku $n \times n$, kde budeme mít pro každou dvojici vrcholů minimální cenu cesty mezi nimi v aktuálním grafu, případně informaci, že žádná cesta zatím neexistuje. Když budeme přidávat hranu, projdeme všechny dvojice vrcholů. Za pomoci tabulky spočteme minimální cenu cesty mezi nimi používající přidanou hranu, kdyby její cena byla n^3. Stačí se podívat na informace o cestách ze startu a cíle do krajních vrcholů hrany. Pokud je o k levnější než původní cesta, víme, že problém nastane jen tehdy, když cena za průchod přidanou hranou bude $n^3 + k$. Pro každou z možných cen hrany si můžeme udržovat, jestli je nebo není problematická. Když spočtená cena leží v přípustném intervalu, poznačíme si jen, že tuto cenu nemůžeme použít. Pak jen sekvenčním průchodem najdeme nejlevnější nezakázanou cenu a s takovouto cenou hranu přidáme. Všechny dvojice vrcholů pak ještě jednou projdeme a případně upravíme minimální cenu cesty mezi nimi, pokud to přes nově přidanou hranu vychází lépe.

Na ceny hran se používají čísla v $O(n^3)$. Časová složitost algoritmu je $O(n^2 m)$. Paměťová složitost je $O(n^2)$.

Úlohu připravili: Jirka Kalvoda, Kristýna Petrlíková

34-5-4 Dělení ve velkém

Pokud si rozmyslíme, co dělá zdrojový program v zadání, tak chceme umět pro pole velikosti N spočítat součet celočíselných dělení přes všechny dvojice v poli. A to rychleji než kvadraticky vzhledem k velikosti pole.

V řešení využijeme toho, že hodnoty v poli nejsou větší než nějaké rozumně malé C. Můžeme tedy snadno zjistit pro každou hodnotu od 0 do C, kolikrát se mezi prvky vyskytuje. Dále také prvky setřídíme, což vzhledem k malým hodnotám můžeme udělat pomocí přihrádkového třídění.

Nyní pro každou hodnotu a mezi 0 a C chceme zjistit, jakým celkovým součtem přispějí všechny podíly, které mají a ve jmenovateli. Všimněme si, že obecně nemůže být příliš mnoho různých výsledků, kterých jednotlivé podíly mohou nabývat. Konkrétně to mohou být hodnoty 0 až C/a: jedná se o nejvýše $1 + \frac{C}{a}$ různých hodnot. Pro každou z nich jsme schopni říct, kolikrát přispěje do celkového výsledku. To je kolikrát se hodnota a vyskytuje krát kolik hodnot může být v čitateli, abychom získali odpovídající výsledek.

Jediné, co zatím neumíme spočítat, je počet čitatelů, které při jmenovateli a dají nějaký konkrétní výsledek, označme jej b. To jsou všechna čísla x z původního pole, pro která platí:

$$ab \leq x < a(b+1)$$

Stačí tedy umět zjistit počet hodnot v poli menších než nějaká konkrétní hodnota. Jakmile toto umíme, tak počet hodnot v intervalu, je počet hodnot menších než $a(b+1)$, od čehož odečteme počet hodnot menších než ab.

Zjistit počet hodnot menších než nějaké k ale umíme snadno. Stačí v uspořádané posloupnosti binárně vyhledávat k. Tím najdeme první výskyt k v posloupnosti, případně nejnižší vyšší číslo než k. To, na jakém indexu v seřazeném poli se tato hodnota nachází, je pak rovno počtu prvků v posloupnosti menších než k.

Jakmile sečteme počty čitatelů krát počty jmenovatelů přes všechny hodnoty jmenovatelů a přes všechny možné výsledky, dostaneme celkový výsledek.

Zatím ale vůbec není jasné, že jsme si polepšili. Mohlo by se zdát, že hodnot jmenovatelů je lineárně vzhledem k hodnotám a stejně tak možných výsledků a tedy že binární vyhledávání provádíme celkem kvadraticky-krát. Ukažme, že situace se dá zanalyzovat tak, že skutečná složitost algoritmu vyjde lépe.

Počítejme, kolikrát celkem zavoláme binární vyhledávání. To je:

$$\sum_{a=0}^{C} 1 + \frac{C}{a} = \sum_{a=0}^{C} 1 + \sum_{a=0}^{C} \frac{C}{a} = C + 1 + C \sum_{a=0}^{C} \frac{1}{a}$$

Zbývající suma je harmonická řada. O té ale víme, že její součet lze shora omezit logaritmem z počtu prvků, které sčítáme.

Celkově tak dostaneme, že binární vyhledávání zavoláme $\mathcal{O}(C \log C)$ krát. Jelikož binární vyhledávání je časově nejnáročnější operace, kterou v každém kroku provádíme, celková časová složitost je $\mathcal{O}(C \log C \log N)$, jelikož binární vyhledávání trvá $\mathcal{O}(\log N)$, což pro malé hodnoty vychází lépe než kvadratický algoritmus ze zadání.

Paměťová složitost je $\mathcal{O}(C + N)$. Poznamenejme, že to by šlo zlepšit na $\mathcal{O}(N)$, pokud bychom si ukládali počty pouze u hodnot, které se v poli vyskytují a pro ostatní bychom krok přeskakovali.

Úlohu připravil: Ondra Sladký

Výsledková listina páté série třicátého čtvrtého ročníku KSP

	řešitel	*škola*	*ročník*	*sérií*	*5-1*	*5-2*	*5-3*	*5-4*	*5-S*	*série*	*KSP-X*	*celkem*
0.					9	12	12	12	15	60,0	36,0	300,0
1.	Benjamin Swart	MensaG	3	5	9	12	8	13	15	57,0	34,5	294,0
2.	Daniel Skýpala	GTomkovaOL	4	25	9	12	8	13	15	57,0	12,0	291,5
3.	Robert Jaworski	GÚstavníPH	4	12	9	12	8	9	15	53,0	16,0	272,5
4.	Jáchym Kouba	GJŠkodyPŘ	2	5	9	3		11	15	38,0	4,0	214,0
5.	David Kolář	GJírovcČB	3	5	9	12		13	15	49,0	0,0	213,5
6.	Jakub Ondroušek	GTomkovaOL	2	10	3	3			15	21,0	0,0	200,5
7.	Jan Slíva	MensaG	1	5	9	12	9			30,0	0,5	160,5
8.	Patrik Číhal	SŠKKamPard	2	4	9	12	1	11	15	48,0	0,0	158,0
9.	Viktor Číhal	SPŠSmíchov	2	5	9	12	8	12		41,0	0,0	152,5
10.	Vojtěch Lančarič	SPŠG Třebešín	3	5	9	12	6	7		34,0	0,0	139,0
11.	Lukáš Tomoszek	GTři	4	4						0,0	4,0	136,4
12.	Adam Kolník	SSŠVTPraha	3	9						0,0	8,0	122,5
13.	Jan Kotovský	GPísnickáPH	3	9						0,0	0,0	120,5
14.	Lukáš Veškrna	GKepleraPH	4	9						0,0	0,0	115,5
15.	Alex Olivier Michaud	GJírovcČB	3	4						0,0	0,5	109,0
16.	Vladimír Sklenár	GTerVans	2	5				1	15	16,0	4,0	105,5
17.	Prokop Randáček	GFXŠaldyLI	3	9						0,0	0,0	80,0
18.	Ján Plachý	G VBN Prie	4	5	0					0,0	4,0	77,5
19.	Vít Skalický	GPísnickáPH	4	22						0,0	0,0	77,0
20.	Jan Černohorský	G Brandýs	4	5					15	15,0	0,0	74,5
21.	Richard Tichý	SG Kladno	0	3						0,0	0,5	69,0
22.	Petr Šícho	GKepleraPH	4	5	9					9,0	0,0	64,0
23.	Kryštof Maxera	GJírovcČB	1	6			1			1,0	0,0	61,0
24.	Jakub Mikeš	GJŠkodyPŘ	4	3						0,0	0,0	54,5
25.	Jiří Kvapil	GTomkovaOL	4	21	9					9,0	0,0	53,5
26.	Matúš Púll	GZborovPH	2	3	9				15	24,0	0,0	47,2
27.	Patrik Herman	GTomkovaOL	3	8						0,0	0,0	44,5
28.	Eliška Macáková	CENADA BA	2	4						0,0	0,0	43,5
29.	Adam Kuča	PORG Krč	4	3						0,0	1,0	42,0
30.	Daniel Šoltýs	GTřeKošice	4	7	9					9,0	0,0	41,0
31.	Zuzana Aubrechtová	GHeyrovPH	3	4					15	15,0	0,0	40,5
32.	Matúš Duchyňa	GGrössBA	3	3						0,0	0,0	36,0
33.	Albert Bakoč	GZborovPH	1	4					15	15,0	0,0	35,0
34.	Ivan Trenčanský	GLSáru	3	3						0,0	0,0	33,0
35.	Jonáš Dej	G Wicht	3	2						0,0	0,0	29,0
36.	Nikolay Fomichev	SSŠVTPraha	3	2						0,0	0,0	26,0
37.–38.	Jakub Hampl	GMělník	2	1	9				15	24,0	0,0	24,0
	Petr Hladík	GMikulášPL	4	5	3					3,0	0,0	24,0
39.	Veronika Jůzková	MensaG	4	6						0,0	0,0	23,0
40.–41.	Martin Belluš	GGrössBA	3	1						0,0	0,0	22,0
	Filip Siviček	GTimLučen	3	1						0,0	0,0	22,0
42.	Bobur Toshtemirov	GMikulášPL	3	2						0,0	0,0	21,0
43.–45.	Adam Jahoda	GKepleraPH	3	2	0		1	11		12,0	0,0	18,0
	Honza Kocourek	ParkLane	2	1						0,0	0,0	18,0
	Karel Procházka	GPBystrica	4	1						0,0	0,0	18,0
46.	Kateřina Vomelová	GÚstavníPH	2	2	9		1	2		12,0	0,0	17,8
47.	Michal Martínek	GÚstavníPH	1	2	9					9,0	0,0	16,0
48.	Jakub Švojgr	GČeskáČB	3	1						0,0	0,0	14,0
49.–51.	Daniel Culliver	GZborovPH	2	2	1					1,0	0,0	12,0
	Michal Pavlíček	MendelGOP	4	2						0,0	0,0	12,0
	Ondřej Skácel	GTomkovaOL	3	6						0,0	0,0	12,0
52.–55.	Michal Bernat	GZborovPH	2	1						0,0	0,0	11,0
	Vojtěch Franc	GArabskáPH	2	1						0,0	0,0	11,0
	Matouš Mišta	GTomkovaOL	1	1						0,0	0,0	11,0
	Filip Neubauer	AkademGPH	2	2	0					0,0	0,0	11,0

	řešitel	*škola*	*ročník*	*sérií*	*5-1*	*5-2*	*5-3*	*5-4*	*5-S*	*série*	*KSP-X*	*celkem*
56.	Pavel Jordán	GPOA Znojmo	3	3						0,0	0,0	10,0
57.–60.	Kateřina Doubková	GNAlejíPH	3	1	9					9,0	0,0	9,0
	Julie Krejčí	???	2	1	9	0				9,0	0,0	9,0
	Kryštof Marek	SGPCE	2	2	9					9,0	0,0	9,0
	Ondřej Pupík	GRožnovPR	2	2	0					0,0	0,0	9,0
61.–62.	Martin Fof	MendelGOP	4	1						0,0	0,0	8,0
	Jakub Kopčil	GMikulášPL	3	2	0					0,0	0,0	8,0
63.–64.	Ondřej Machota	G Brandýs	4	2	1					1,0	0,0	6,0
	Jonáš Menšík	GJŠkodyPŘ	0	1						0,0	0,0	6,0
65.	Marek Maškarinec	SPŠEMasLI	1	1						0,0	0,0	4,0
66.–67.	Andrea Mikulová	BGOstrava	3	1	0		1	2		3,0	0,0	3,0
	Jan Šuráň	GZborovPH	4	1						0,0	0,0	3,0
68.	Jakub Surga	ParkLane	4	6						0,0	0,0	1,0

Výsledková listina KSP-X po páté sérii třicátého čtvrtého ročníku

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

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

matfyz

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/*