Korespondenční Seminář z Programování

ZAČÁTEČNICKÁ KATEGORIE

31. ročník | **KSP-Z** | **Červen 2019**

Řešení čtvrté série začátečnické kategorie 31. ročníku KSP

31-Z4-1 Nejosamělejší kamarád

Nejlehčí úloha série v sobě neskrývala žádná překvapení, ale alespoň trochu se zamyslet bylo potřeba, přímočaré „dřevorubecké“ řešení nestačilo.

Takovým „dřevorubeckým“ řešením by mohlo být to, které si pro každého kamaráda samostatně najde nejbližšího jiného a pak mezi všemi kamarády najde maximum (tedy toho nejosamělejšího). Toto řešení pro každého z N kamarádů projde všech $N-1$ zbývajících kamarádů a celkový čas bychom tak mohli vyčíslit jako $\mathcal{O}(N^2)$ – to by na menší vstupy ještě stačilo, ale na největší vstupy to již bude příliš pomalé.

Trik, který nám řešení vylepší, je setřídění kamarádů podle pozice. Pokud jste zaslechli něco o třídění nebo jste se podívali třeba do naší kuchařky o třídících algoritmech,[1] tak tušíte, že rychlé třídící algoritmy umí seřadit seznam N prvků v čase $\mathcal{O}(N \log N)$. V Pythonu můžete použít takové rychlé třídění, které už naprogramoval někdo jiný, jednoduchým zavoláním funkce `sorted` na seznam.

A jak nám setřídění kamarádů pomůže? Díky němu víme, že pro každého kamaráda je jeho nejbližší jiný kamarád v setříděném seznamu buď těsně nalevo nebo těsně napravo od něj. Pak nám stačí jenom kamarády jednou projít, pro každého si v konstantním čase spočítat vzdálenost k nejbližšímu jinému kamarádovi, a přitom si průběžně udržovat maximum. Tím se dostaneme na čas $\mathcal{O}(N)$ za tento průchod a $\mathcal{O}(N \log N)$ celkově i s tříděním.

Program (Python 3):
`http://ksp.mff.cuni.cz/viz/31-Z4-1.py`

Jirka Setnička

31-Z4-2 Závažíčka na druhou

Začněme pozorováním: pro každou velikost závaží je jednoznačně dáno, na kterou váhu přijde a jak ovlivní rozdíl vah. Když použijeme závaží váhy 2^i, tuto váhu k rozdílu přičteme či odečteme v závislosti na sudosti či lichosti i. Obecně, i-tá váha (počítáno od nuly) ovlivní rozdíl hodnotou $(-2)^i$. Výstup úlohy určuje, které mocniny -2 je třeba použít, aby jejich součet byl roven požadované hodnotě na vstupu. Tohle není nic jiného než zápis čísla na vstupu v *mínus dvojkové soustavě*, takže úloha je vlastně převod čísel do této soustavy.

Jak to provést? Všimněme si další věci: $(-2)^0 = 1$ je jediná lichá mocnina (mínus) dvojky, tedy pokud je číslo na vstupu liché, určitě se v jeho zápisu v mínus dvojkové soustavě objeví. Pokud je vstup kladný, můžeme do výstupu hodnotu 1 rovnou zahrnout, ale pokud je záporný, tak její použití posune výstup směrem do kladných čísel, což je opačně, než chceme. V takovém případě do výstupu zahrneme i následující mocninu $(-2)^1 = -2$, neboť $-2 + 1 = -1$. Přesněji, nemusíme použít přímo -2, ale úplně stačí, když celková hodnota zbytku bitů výstupu bude o jedna menší než číslo na vstupu, neboť musí kompenzovat první bit, jehož hodnota je nyní jedna.

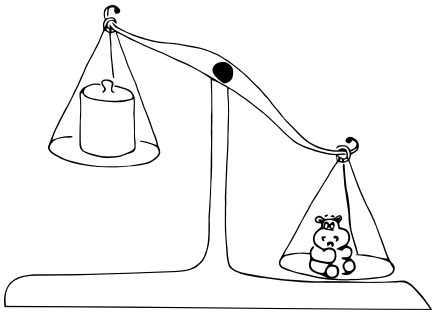
Umíme určit hodnotu prvního (nejmenšího) bitu výstupu. Pokud byl vstup sudý, první bit jsme ponechali na nule, takže celková hodnota zbytku bitů výstupu musí být rovna vstupu. Pokud byl vstup lichý, tak od zbytku jeho bitů chceme, aby součet jejich mocnin byl roven vstupu mínus jedna, a to ať už byl vstup kladný nebo záporný. Hodnota zbytku bude díky tomuto postupu vždy sudá.

Jak určit další bity výstupu? Zde přichází třetí pozorování: není třeba se tímto problémem zabývat, máme už totiž vše, co potřebujeme. Víme, jakou celkovou hodnotu mají mít ostatní bity, této hodnotě budeme říkat *zbytek*, ale umíme určit jen hodnotu bitu na mocnině $(-2)^0$. Určení hodnoty bitu $(-2)^1$ provedeme tak, že hodnotu *zbytku* posuneme o jednu mocninu doprava, tedy vydělíme ji -2, a její zápis v mínus dvojkové soustavě připojíme před již spočítaný bit. Nyní odpovídá bit $(-2)^1$ hodnotě $\frac{(-2)^1}{-2} = (-2)^0$ a můžeme na něj použít stejný postup jako na první bit. Toto opakujeme, dokud není požadovaná hodnota *zbytku*, tedy i všech zatím nezpracovaných bitů, nulová.

V každém kroku algoritmu se hodnota *zbytku* zmenší v absolutní hodnotě na polovinu, takže těchto kroků může algoritmus provést nejvýše logaritmicky mnoho, než tato hodnota klesne na nulu (dělíme celočíselně). Časová složitost na převod jednoho čísla do mínus dvojkové soustavy je tedy $\mathcal{O}(logN)$, kde N je velikost převáděného čísla, neboli je lineární s délkou binárního zápisu tohoto čísla.

Program (C++):
`http://ksp.mff.cuni.cz/viz/31-Z4-2.cpp`

Kuba Pelc



31-Z4-3 Probíhání bludištěm

Úlohu s místnostmi v bludišti můžeme zkusit vyřešit hrubou silou tak, že si pro každou místnost odsimulujeme všech K kroků, ale to narazí už třeba při zadání obsahujícím dvě místnosti a K v řádu miliard – prosté odsimulování je příliš pomalé, musíme vymyslet něco lepšího.

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

Než se pustíme do vymýšlení rychlejšího řešení úlohy, tak nejprve udělejme krok stranou a podívejme se na labyrint místností trošku jinak. Prvním krůčkem může být představit si labyrint jako *graf*, neboli jako vrcholy (místnosti) a hrany spojující tyto vrcholy (dveře). Kdo se v grafech moc neorientujete, můžete pro lepší vysvětlení nahlédnout do grafové kuchařky.[2]

S různými grafy jste se už v minulých sériích mohli v některých úlohách potkat, ale tento graf má několik speciálních vlastností. První vlastností je, že hrany jsou *orientované* (takže si je můžeme představit jako šipky mezi místnostmi), a druhá vlastnost je, že do každé místnosti právě jedna šipka vchází a právě jedna z ní vychází. Když se nad tím zamyslíme víc, tak lehce uvidíme, že šipky mezi místnostmi tak musejí tvořit nějaké okružní trasy (jednu nebo více), těm budeme říkat *cykly*. Musí to tak být, protože když z nějaké místnosti začneme, tak skončit můžeme opět jenom v ní – z každé jiné místnosti existuje pokračování do zatím nepoužité místnosti (kdyby dveře vedly do místnosti, do které jsme již vstoupili odjinud, tak by tato místnost musela mít dva vstupy, což nemá) a místností není nekonečně. Takže dříve či později dojdeme opět do místnosti, odkud jsme začali cyklus procházet.

Teď bychom se již mohli vrhnout přímo na řešení, ale dovolíme si ještě udělat druhý krůček stranou a podíváme se na náš graf s cykly ještě jinak. Takovýto graf totiž popisuje nějakou *permutaci*, neboli promíchání prvků. Jeden krok této permutace si můžeme představit tak, že do každé místnosti postavíme člověka s číslem odpovídajícím číslu místnosti a každého necháme právě jednou projít dveřmi. Přesně tak máme vlastně i zadaný vstup.

Poznámku o permutacích jsme tu ale uvedli spíše na rozšíření obzorů, nám teď bohatě stačí představa, že místnosti tvoří několik cyklů. Když si pro každý cyklus najdeme všechny místnosti, které ho tvoří, a spočítáme si velikost cyklu, můžeme finální místnost po K krocích určit jednoduše. Předně pokud je K větší než velikost cyklu, tak můžeme vzít jenom zbytek po dělení K velikostí cyklu a pak si jen odkrokujeme zbylý počet kroků po cyklu. Toto řešení potřebuje čas $\mathcal{O}(N)$ pro nalezení cyklů a pak si pro každou z N místností odkrokuje nejvýše N kroků, čímž dostaneme finální časovou složitost $\mathcal{O}(N^2)$.

Proč si ale kazit časovou složitost tím, že musíme po cyklu krokovat? Když máme cyklus uložený v nějakém seznamu, velikost cyklu je S a máme udělat K kroků, tak pro i-tou místnost cyklu určíme výslednou lokaci jako $cyklus[(i + K) \bmod S]$ (kde mod je operace modulo, neboli zbytek po dělení). To pro každou místnost umíme udělat v konstantním čase, takže dosáhneme výsledného času $\mathcal{O}(N)$.

Implementovat to můžeme tak, že si pořídíme cyklus přes všechny místnosti. Když potkáme ještě nezpracovanou místnost, tak z ní zahájíme procházení cyklu a pak si pro každou místnost na cyklu spočteme výslednou lokaci a označíme ji za zpracovanou.

Program (Python 3):
`http://ksp.mff.cuni.cz/viz/31-Z4-3.py`

Jirka Setnička

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

31-Z4-4 Ohnivý únik

V úloze hledáme něco jako nejkratší cestu mezi nějakými místy. Na to se nabízí použít klasické grafové postupy, především *BFS*, tedy prohledávání do šířky. Na bludiště se můžeme dívat jako na graf: jednotlivá volná políčka jsou vrcholy a každé políčko je propojené hranou se všemi svými sousedy, kteří nejsou zeď. Při programování ale není nutné tento graf explicitně vytvářet, můžeme stále pracovat jen s dvojrozměrným polem, které odpovídá bludišti. Vrchol je popsán souřadnicemi svého políčka. Zda z jednoho políčka vede hrana do druhého zjistíme tak, že se podíváme, jestli jedno z nich není zdí.

Pro prohledávání bludiště do šířky budeme potřebovat frontu (detaily viz kuchařka).[3] Při procházení našeho bludiště si do ní budeme ukládat souřadnice políček, která chceme zpracovat. Jakmile při prohledávání poprvé nalezneme políčko s východem, tak platí, že do žádného jiného políčka s východem se nelze dostat rychleji. Ve frontě se totiž nyní nacházejí jen políčka, která jsou stejně daleko nebo dále od počátku než aktuální. Časová složitost prohledávání do šířky je $\mathcal{O}(n + m)$, kde n je počet vrcholů a m počet hran. V tomto grafu ale platí, že počet hran m je asi jen $2n$ (každému vrcholu přiřadíme nejvýše jednu hranu dolů a jednu doleva), takže celková časová složitost je $\mathcal{O}(n+2n) = \mathcal{O}(n)$.

Slupky

Všimněme si, že prohledávání do šířky zpracovává vrcholy po *slupkách*. Slupka je tvořena všemi vrcholy, které jsou od počátečního stejně daleko. Nultá slupka je jen počáteční vrchol, první slupkou jsou jeho sousedé, druhou sousedé jeho sousedů. Obecně slupka $i + 1$ jsou všichni sousedé slupky i vyjma vrcholů, které už patří do slupky $i - 1$.

Pokud budeme zpracovávat vrcholy ze začátku fronty, které jsou ve slupce i, budeme přidávat na konec fronty vrcholy slupky $i + 1$, dokud ze začátku fronty vrcholy i nedojdou. Nyní je celá fronta plná vrcholů slupky $i + 1$, které budou při zpracování do fronty přidávat vrcholy slupky $i+2$ a tak dále. Ve frontě budou tedy vždy přítomny pouze dvě slupky, jedna, jejíž vrcholy zpracováváme a druhá, jejíž vrcholy do fronty přidáváme.

Celý výpočet *BFS* jde tedy rozdělit na jednotlivé slupky. Fronta lze nahradit dvěma seznamy: z prvního postupně odebíráme vrcholy slupky i, do druhého přidáváme vrcholy slupky $i + 1$. Jakmile nám dojdou vrcholy v prvním seznamu, tak oba seznamy jednoduše prohodíme a pokračujeme ve výpočtu. Toto zpracování jedné slupky odpovídá pohnutí o jeden krok či jednu hranu na hledané nejkratší cestě.

A co oheň?

Zatím jsme se zabývali jen hledáním nejkratší cesty do nějakého východu a oheň jsme ignorovali. Provedeme následující: budeme hledat nejkratší cestu do nějakého východu pomocí výše popsané varianty *BFS* se dvěma seznamy, které odpovídají slupkám. Políčka, která hoří, budeme považovat za zeď. Po zpracování každé slupky provedeme jeden krok simulace ohně. Oheň budeme také simulovat pomocí slupkového prohledávání do šířky. Do nulté slupky ohně vložíme všechna políčka, která na počátku hoří.

V programu budeme ve smyčce postupně simulovat jeden tah osoby, poté jeden tah ohně. Jeden tah odpovídá zpra-

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

cování jedné slupky daného *BFS*. Každý vrchol, kam se dostane prohledávání ohně, označíme v mapě jako oheň, a to už v okamžiku, kdy toto políčko přidáme do příští slupky.

Pokud bychom při *BFS* osoby přidali do příští slupky políčko sousedící s ohněm, tak by při simulaci ohně toto políčko pod námi začalo hořet, takže jej nelze použít. Toto můžeme opravit například tím, že při odebírání políček ze slupky při *BFS* osoby zkontrolujeme, zda políčko nezačalo hořet, a v takovém případě jej rovnou zahodíme.

Tím zajistíme, že nikdy nebudeme zpracovávat cestu, která projde ohněm (do zapáleného políčka nikdy nevkročíme a políčko, které začne hořet přímo pod námi, zahodíme v dalším kroku). A jak dlouho nám to trvá? Obě *BFS* zpracují každé políčko nejvýše jednou, takže i dohromady bude jejich časová složitost lineární vzhledem k velikosti bludiště.

Program (C++):
`http://ksp.mff.cuni.cz/viz/31-Z4-4.cpp`

Kuba Pelc

31-Z4-5 Naplnění nádob

Máme n nádob a p příkazů k nalévání vody. Každý příkaz je určený trojicí čísel (a_i, b_i, ℓ_i), která říká, že do všech nádob s čísly a_i až b_i máme nalít po ℓ_i litrech vody.

Představíme si na chvilku, že příkaz je jenom jeden: (a, b, ℓ). Tehdy bychom mohli na a-tou nádobu pověsit cedulku s nápisem „toto je první nádoba, do které se má nalévat ℓ litrů“ a na $(b+1)$-ní nádobu cedulku „od této nádoby dál už ℓ litrů nenalévej“. Pak postačí projít se podél řady nádob: když potkáme první cedulku, začneme do všech nádob, kolem kterých projdeme, nalévat ℓ litrů. S naléváním skončíme, jakmile potkáme druhou cedulku.

Totéž funguje, je-li příkazů víc. Za každý příkaz (a_i, b_i, ℓ_i) vyrobíme dvě cedulky: počáteční umístíme na a_i-tou nádobu, koncovou na $(b_i + 1)$-ní nádobu. Pak se zase projdeme kolem řady nádob a doléváme. Průběžně si udržujeme proměnnou D, která nám říká, kolik vody máme do nádob dolévat. Když potkáme počáteční cedulku pro i-tý příkaz, zvýšíme D o ℓ_i. Když potkáme koncovou cedulku, tak D o ℓ_i snížíme.

Postup můžeme ještě zjednodušit: na cedulky můžeme psát prostě $+\ell_i$ a $-\ell_i$ podle toho, jak máme změnit D. A pokud se sejde více cedulek na tomtéž místě, můžeme jejich hodnoty sečíst.

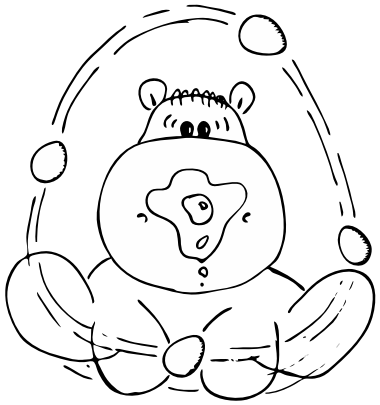
Nyní algoritmus popíšeme detailněji:

- Nejprve vytvoříme pole $c[1 \ldots n + 1]$, v němž si budeme pro každou nádobu pamatovat součet všech cedulek pověšených na ni. Na počátku výpočtu to budou samé nuly. (Pole indexujeme až do $n+1$, protože občas potřebujeme pověsit cedulku za poslední nádobu.)
- Projdeme všechny příkazy a pro každý příkaz (a_i, b_i, ℓ_i) zvýšíme $c[a_i]$ o ℓ_i a snížíme $c[b_i + 1]$ o ℓ_i.
- Projdeme všechny nádoby a doléváme. Inicializujeme D na 0. Pak pro $i = 1, \ldots, n$ opakujeme: k D přičteme $c[i]$ a do i-té nádoby nalijeme D litrů. Jelikož do této nádoby už nic dalšího nenalijeme, můžeme rovnou vypsat, že v ní na konci bude D litrů.

Jak rychlý tento algoritmus bude? Inicializace pole trvá čas $\mathcal{O}(n)$, jeho vyplnění podle příkazů $\mathcal{O}(p)$ a konečně projití nádob zabere $\mathcal{O}(p)$. Celkem tedy $\mathcal{O}(n+p)$. Rychleji to jistě nejde, protože tento čas je potřebný na samotné přečtení vstupu a vypsání výstupu.

Paměť spotřebujeme na uložení pole c a na konstantní počet pomocných proměnných. Vstup a výstup si nemusíme pamatovat: vstup zpracováváme jeden příkaz po druhém a už se k nim nevracíme, výstup vypisujeme průběžně. Prostorová složitost tudíž činí $\mathcal{O}(n)$.

Martin „Medvěd“ Mareš



31-Z4-6 Sčítání škod

Zadání úlohy nás navádí k tomu, abychom vytvořili nějaký graf – pokud si potřebujete grafovou terminologii připomenout, nahlédněte do naší grafové kuchařky.[4] Máme políčka, která na sobě nějak závisí. Zkusíme tedy políčka prohlásit za vrcholy grafu a závislosti za hrany. Pokud je například v poli `A4` zapsáno `=A2+A1`, natáhneme orientované hrany z vrcholů `A2` a `A1` do vrcholu `A4`. Je důležité, aby hrany byly orientované, jinak bychom ztratili přehled o tom, které políčko se má vyhodnotit před kterým.

Máme orientovaný graf, tak co teď? Chceme najít nějaké pořadí políček se vzorci (to jsou ta, do kterých vedou nějaké hrany z jiných políček). Pro jednoduchost najdeme pořadí všech políček, tedy i těch, ve kterých je zapsané pouze číslo. Vlastně nám to vůbec nebude vadit, protože jak ukážeme dále, všechna tato políčka zařadíme na začátek, takže když budeme chtít vypsat pořadí samotných políček se vzorci, přeskočíme akorát počáteční část a začneme vypisovat až od prvního políčka se vzorcem.

Uvědomíme si, že když hledáme takové pořadí, tak určitě můžeme jako první vyhodnotit políčka se samotnými čísly – vrcholy, do kterých nevede žádná hrana. Vyhodnocení políčka v naší grafové reprezentaci provedeme tak, že příslušný vrchol z grafu vymažeme, stejně jako všechny hrany, které z něj vedou. Tím se nám můžou objevit další vrcholy, do kterých už nepovede hrana, tak je pak můžeme vyhodnotit také. Je to jistě korektní krok, protože vrchol, do kterého nevede žádná hrana, nezávisí buď na žádném jiném políčku, a nebo na už vyhodnocených políčkách (a tedy smazaných z grafu).

Jak bude vypadat celý algoritmus? Nejprve projdeme všechny vrcholy grafu a ty, do kterých nevede žádná hrana, si zařadíme do fronty. Postupně vrcholy z fronty odebíráme. Vždy se podíváme na každou hranu vedoucí z odebraného vrcholu, smažeme ji, a pokud to byla jediná hrana vedoucí do následujícího vrcholu, tak jej také přidáme do fronty.

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

Zastavíme se, když bude fronta prázdná. Buď jsme prošli celý graf a našli správné pořadí, nebo nám v grafu nějaké vrcholy zůstaly, ale do každého vede hrana. V tom případě se v grafu vyskytuje cyklus. To znamená, že nějaká políčka v tabulce na sobě závisejí navzájem a nejde z nich vybrat jedno, které bychom mohli vyhodnotit jako první. Pokud tedy existuje nějaké pořadí, ve kterém můžeme políčka vyhodnotit, náš algoritmus ho najde, a pokud neexistuje, tak to taky odhalíme.

Jaká bude časová složitost našeho algoritmu? Graf vytvoříme v lineárním čase vůči počtu políček. Pak ho jednou v lineárním čase projdeme, abychom zařadili do fronty vrcholy, které na žádných jiných nezávisí – políčka s čísly. Když pak budeme graf procházet, tak každý vrchol nejvýše jednou zařadíme do fronty a jednou ho z ní vymažeme. Než ho ale vymažeme, musíme se podívat na všechny hrany, které z něj vedou. Těch ale může být až $\mathcal{O}(n)$, pokud na tomto vrcholu závisí všechny ostatní. Můžeme se na to však podívat z druhé strany, tedy spočítat, kolikrát se na každý vrchol podíváme při mazání jiného vrcholu. Protože každý vrchol závisí nejvýše na dvou dalších, tak se na něj podíváme nejvýše dvakrát, než ho zařadíme do fronty. Přestože nám tedy mazání některých vrcholů může trvat dlouho, dohromady bude časová složitost algoritmu lineární, $\mathcal{O}(n)$.

Zuzka Urbanová

Výsledková listina čtvrté série začátečnické kategorie 31. 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.	Daniel Skýpala	GTomkovaOL	1	0	8	10	10	12	12	14	66,0	254,5
2.	Jiří Kvapil	GTomkovaOL	1	0	8	10	10	12	11	14	65,0	238,5
3.	Vladimír Chudý	G Chrudim	2	0	8	10	10	12	3	9	52,0	226,0
4.	Michal Bravanský	GBílovec	1	0	8	10	10	12	12	13	65,0	223,0
5.	Robert Gemrot	GKomHavíř	2	0	8	10	10	12	10	14	64,0	218,0
6.	Kristýna Petrlíková	VOŠJičín	1	0	8	10	10	12	12		52,0	215,0
7.	Jan Štěch	GJirsíkaČB	2	0	8	10	10	12	3	14	57,0	190,7
8.	Adam Bujdák	G JM Galanta	3	0	8	10	10	12	10	8	58,0	190,0
9.	Jakub Kopčil	GMikulášPL	0	0	8	10	10	12	9	13	62,0	183,5
10.	Martina Daňková	KŠpGym Bo	2	0	8		10	12			30,0	165,0
11.	Jakub Ondroušek	GTomkovaOL	−1	0	8	4	10	6			28,0	148,0
12.	Albert Kučera	GNadŠtolPH	2	0	8		7				15,0	132,3
13.	Petr Kolář	GMilevsko	3	0							0,0	122,5
14.	Robert Jaworski	GÚstavníPH	1	0							0,0	114,0
15.	Jan Kotovský	GPísnickáPH	0	0	6	6	10	12			34,0	112,0
16.	Kateřina Rosická	GKutnáHora	4	0							0,0	106,0
17.	Filip Kastl	GKepleraPH	3	0							0,0	104,5
18.	Vojtěch Žák	GŠpitálsPH	3	0	8						8,0	104,0
19.	Terézia Strišovská	GJHroncaBA	3	0	8						8,0	101,0
20.	Šimon Andrš	GKepleraPH	0	0							0,0	99,0
21.	Martin Bencko	GOhradníPH	2	0	8		7				15,0	96,0
22.	Jan Najman	SPSEPard	2	0	8	10					18,0	91,0
23.	Michal Mlčoch	G UherBrod	4	0							0,0	88,0
24.	Lucie Vomelová	GŠpitálsPH	3	0							0,0	80,5
25.	Janek Hlavatý	GJirsíkaČB	0	0							0,0	77,0
26.–27.	Adam Húšťava	EupSchoolLux	1	0	8						8,0	73,0
	Eric Valčík	G UherBrod	4	0							0,0	73,0
28.	Marie Kalousková	GNAlejíPH	3	0	8				3	10	21,0	71,0
29.	Ondřej Chlubna	GOrlová	2	0							0,0	70,0
30.	František Kmječ	StOlavVGS	3	0	8	10	10	12			40,0	68,0
31.	Vojtěch Kuchař	VOŠJičín	2	0	8						8,0	65,0
32.	Ondřej Sladký	GMikulášPL	2	0	8	10	10	12	10	14	64,0	64,0
33.	Karel Chwistek	MendelGOP	2	0	8	10	10	12	10	13	63,0	63,0
34.	Jan Piroutek	GŠpitálsPH	3	0	0						0,0	61,0
35.–36.	Jiří Bleha	SPSEPard	2	0	8	10	0				18,0	56,0
	Martin Zmitko	G FrýdlNOs	3	0							0,0	56,0
37.–38.	Patrik Baláš	SPSEPard	1	0							0,0	51,0
	Ondra Müller	GTurnov	2	0							0,0	51,0
39.	Jan Hlaváč	GNAlejíPH	3	0							0,0	48,0
40.	Petr Aubrecht	GHeyrovPH	4	0							0,0	46,0
41.	Kryštof Suchánek	GLesníZlín	3	0	8						8,0	44,0
42.	Kateřina Vokálová	G Kolín	3	0	8	3,3	5,3	12		9	37,7	43,7
43.	Jakub Nevařil	G UherBrod	1	0	8		7				15,0	42,0
44.	Martin Klimeš	GZábřeh	3	0	8	10	10	12			40,0	40,0
45.	Pavel Altmann	GMikulášPL	0	0			0				0,0	37,0
46.–47.	Petr Budai	G JGJ PH	2	0							0,0	36,0
	Tomáš Dostál	MendelGOP	4	0							0,0	36,0
48.	Tomáš Sláma	GTurnov	4	0							0,0	33,0
49.	Vojtěch Káně	G Brandýs	3	0	8		10			13	31,0	31,0
50.	Dalibor Kramář	G BO-Řeč	4	0							0,0	30,0
51.	Patrik Herman	GTomkovaOL	0	0	4						4,0	26,3
52.–53.	Radim Buráň	G UherBrod	4	0							0,0	26,0
	Jan Šulíček	SPSEPard	2	0							0,0	26,0
54.–55.	Marek Chwistek	MendelGOP	0	0	8		5	12			25,0	25,0
	Jiří Kruchina	GČeskoliPH	1	0							0,0	25,0

	řešitel	*škola*	*ročník*	*sérií*	*Z4-1*	*Z4-2*	*Z4-3*	*Z4-4*	*Z4-5*	*Z4-6*	*série*	*celkem*
56.	Ondřej Polanecký	SPŠPísek	2	0	8						8,0	24,7
57.	Patrik Rosenberg	GJarošeBO	−2	0							0,0	23,0
58.	Radek Zavřel	SPŠSmíchov	3	0							0,0	22,0
59.–60.	Martin Boček	MendelGOP	0	0	8		3				11,0	20,0
	Jiří Heller	GNAlejíPH	3	0							0,0	20,0
61.	Jan Hartman	GChodoviPH	3	0							0,0	19,0
62.–68.	Dominik Farhan	GMikulášPL	2	0	8		10				18,0	18,0
	Jan Kaifer	GKepleraPH	3	0							0,0	18,0
	Jakub Komárek	GUHradiště	4	0							0,0	18,0
	Bohdan Kopčák	GNAlejíPH	3	0							0,0	18,0
	Jan Koška	GJírovcČB	−1	0							0,0	18,0
	Matěj Volf	GCoubTábor	1	0							0,0	18,0
	Vojtěch Zabořil	GTurnov	2	0							0,0	18,0
69.	Anna Hollmannová	GSRandyJN	2	0					4		4,0	17,0
70.–71.	Ondřej Hráček	GOlgHavl	2	0							0,0	16,0
	Lucie Kunčarová	GVolgogrOS	3	0	2						2,0	16,0
72.–73.	Michal Němec	GVídeňskBO	3	0							0,0	14,0
	Petr Kroča	G UherBrod	−2	0				0			0,0	14,0
74.–75.	Alexandr Čelakovský	G UherBrod	3	0							0,0	13,0
	Jan Jeníček	GNAlejíPH	3	0							0,0	13,0
76.	Ondřej Martínek	G UherBrod	4	0							0,0	12,0
77.–78.	Branislav Blažek	GŽilina	1	0							0,0	11,0
	Magdaléna Turinská	SZŠ Brandýs	2	0							0,0	11,0
79.	Dávid Oravec	G DubNVáh	4	0							0,0	10,7
80.–95.	Tomáš Černý	GArabskáPH	3	0							0,0	8,0
	Evgenia Golubeva	GJosefskPH	4	0							0,0	8,0
	Petr Hladík	GMikulášPL	1	0		0					0,0	8,0
	Milan Jiříček	SPŠPísek	2	0							0,0	8,0
	Radim Kopunec	G UherBrod	−1	0							0,0	8,0
	Filip Krul	SPŠSmíchov	3	0							0,0	8,0
	Lukáš Maga	GŽilina	2	0							0,0	8,0
	Antonín Musil	PORGPha	2	0							0,0	8,0
	Václav Pavlíček	SPSEPard	3	0							0,0	8,0
	Jakub Profota	GŘíč	4	0							0,0	8,0
	Matej Stencel	GPošKošice	2	0							0,0	8,0
	Jan Škoula	GBenesov	3	0							0,0	8,0
	Šárka Štěpánková	G Chrudim	2	0							0,0	8,0
	Filip Vaculík	G UherBrod	4	0							0,0	8,0
	Jiří Vlček	GFXŠaldyLI	3	0							0,0	8,0
	Michal Zacek	MensaG	2	0							0,0	8,0
96.–97.	Vojtěch Frőmmel	SPŠEMasLI	3	0							0,0	5,0
	Mark Smetanova	GLepařovJČ	2	0							0,0	5,0
98.	Klára Hloušková	G Kolín	3	0	2						2,0	3,0
99.–100.	Michal Janata	???	2	0	2						2,0	2,0
	Tomas Kocian	GTurnov	3	0							0,0	2,0
101.	Tomáš Hájek	G UherBrod	4	0							0,0	1,0