Korespondenční Seminář z Programování

ZAČÁTEČNICKÁ KATEGORIE

28. ročník | **KSP-Z** | **Květen 2016**

Prázdniny se blíží, ale než si jich začnete užívat plnými doušky, věnujte pár chvil přečtení řešení úloh poslední série tohoto ročníku KSP-Z. Gratulujeme všem, kteří získali nějaké body! Pokud se vám nějaká praktická úloha nevydařila, můžete ji zkusit vyřešit i po termínu.

A jako obvykle se nás nebojte zeptat, pokud vám cokoliv není úplně jasné. Obrátit se na nás můžete přes fórum na našich stránkách nebo e-mailem na *ksp@mff.cuni.cz*.

Řešení čtvrté série začátečnické kategorie 28. ročníku KSP

28-Z4-1 Půdorys

Důležitou součástí řešení bylo uvědomit si, že hledané obdélníky mají jednu zajímavou vlastnost – body ležící na jedné straně obdélníka mají vždy stejnou jednu souřadnici. Takže svislé hrany obdélníka mají stejnou x-ovou souřadnici a vodorovné zase y-ovou.

My musíme pro každý bod na vstupu ověřit, zda leží na nějaké hraně obdélníka. K tomu potřebujeme nejdříve znát souřadnice těchto hran. To už zvládneme jednoduše.

Všechny body na levé svislé hraně budou mít nejmenší x-ovou souřadnici, body na pravé svislé hraně budou mít největší x-ovou souřadnici. Stačí nám tedy pouze projít x-ové souřadnice všech bodů a zapamatovat si minimum a maximum. Pro vodorovné hrany analogicky s y.

Teď už jenom ověříme, že každý bod leží na nějaké hraně jednoduchým porovnáním jeho souřadnic se zjištěnými souřadnicemi hran.

Program (Python 3):
`http://ksp.mff.cuni.cz/viz/28-Z4-1.py`

Martin Šerý

28-Z4-2 Vykopávky

Je zřejmé, že když mají mít všechny čtyři obdélníkové oblasti stejný součet, bude to právě součet všech políček vydělený čtyřmi. Abychom ho tedy spočítali, projdeme si nejdřív jednou všechna čísla a sečteme je. Zatím nezáleží na tom, v jakém pořadí je projdeme.

Dále si všimneme, že část nahoře od bodu S (všichni se tu mají potkat) a část dole budou mít stejný součet, a to právě polovinu součtu celkového. Budeme tedy procházet čísla ještě jednou, tentokrát po řádcích, budeme je opět sčítat, ale zastavíme se, když se dosčítáme k polovině celkového součtu. Zapamatujeme si, kolik řádků jsme zatím prošli, to bude totiž výška obdélníku, kterou chceme zjistit.

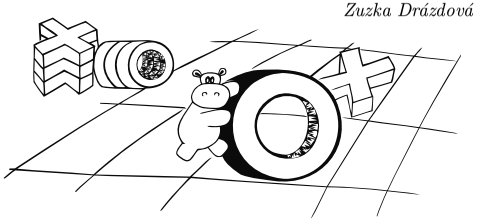
Do třetice všeho dobrého projdeme čísla ještě jednou, tentokrát po sloupcích. Opět je budeme sčítat, dokud se nedostaneme k polovině celkového součtu. Počet zatím prošlých sloupců nám dá hledanou šířku obdélníku.

Protože jsme čísla na vstupu prošli třikrát, složitost bude $\mathcal{O}(RS)$, což je počet řádků vynásobený počtem sloupečků, tedy počet všech políček.

Ještě se zamyslíme, jak by se dala uspořit paměť – co kdybychom si nemuseli pamatovat všech R čísel? Vždy potřebujeme počítat součty celých řádků a celých sloupců. Takže by stačilo si při načítání vstupu průběžně sčítat všechna čísla v každém řádku a sloupci, a pamatovat si jen tyto údaje, kterých je $R + S$.

Program (Python 3):
`http://ksp.mff.cuni.cz/viz/28-Z4-2.py`

Zuzka Drázdová



28-Z4-3 Mocniny

Přímočarým řešení Kevinova problému je si pro každé číslo x spočítat jeho druhou a třetí mocninu, a pak ověřit, že i ony leží v naší posloupnosti. To můžeme udělat třeba sekvenčním vyhledáním x^2 a x^3. Zapamatujeme si právě ta x, pro která jsme našli dané mocniny. Ze zapamatovaných x už jenom stačí najít minimum.

Toto řešení má kvadratickou složitost $\mathcal{O}(N^2)$, kde N je počet čísel na vstupu. Pro každé číslo projdeme všechna ostatní a zjistíme, jestli mezi nimi je i jeho druhá a třetí mocnina.

Rychlejší řešení získáme úpravou toho přímočarého. Nejvíce času strávíme hledáním x^2 a x^3. Tento krok můžeme urychlit tím, že si na začátku celou posloupnost setřídíme. Poté už můžeme mocniny hledat pomocí binárního vyhledávání.[1]

Navíc nám stačí se zastavit u prvního x v setříděné posloupnosti. Toto x je totiž první pro které jsme našli mocniny, takže všechna další čísla už budou pouze větší.[2]

Setřídění nám zabere $\mathcal{O}(N \log N)$ času. Binární vyhledání jednoho prvku má logaritmickou složitost a jelikož hledané x může být téměř až na konci setříděné posloupnosti, provedeme jej pro každý prvek. Tj. celé vyhledávání trvá $\mathcal{O}(N \log N)$ času, což je stejné jako složitost setřídění, bude to tedy i výsledná složitost.

Program (Python 3):
`http://ksp.mff.cuni.cz/viz/28-Z4-3-bin.py`

[1] `http://ksp.mff.cuni.cz/viz/kucharky/zakladni-algoritmy`
[2] V Pythonu se hodí použít `set`, který je přímo dělaný na podobné vyhledávání.

Pokud váš programovací jazyk disponuje datovou strukturou pro množinu, často pojmenovanou `set`, můžete ji s výhodou použít. Ta si interně udržuje čísla setřízená a vlastně na nich binární vyhledávání provádí – jen nemá čísla uložená v seznamu. Rychlé řešení pak vyjde velmi elegantně na jeden řádek.

Program (Python 3):
`http://ksp.mff.cuni.cz/viz/28-Z4-3-set.py`

Martin Šerý

28-Z4-4 Čtyřková

Úloha byla trochu podlá, protože zjistit odpověď pro jedno číslo je vlastně stejně těžké, jako ji zjistit pro všechna najednou. Pojďme se na to podívat.

Začínáme s jednou čtyřkou. Jaká čísla jsme schopni vyrobit, pokud použijeme čtyřky dvě? Jsou to $8 = 4 + 4$, $0 = 4 - 4$, $16 = 4 * 4$ a $1 = 4/4$. Protože tato čísla s jednou čtyřkou určitě nevyrobíme, zapamatujme si o nich že jdou vyrobit nejlépe se dvěmi.

A co když si dovolíme další čtyřku navíc? Pak se stačí podívat na ta čtyři čísla, která umíme vyrobit se dvěma čtyřkami, a zkusit k nim všemi čtyřmi operacemi přidat třetí. Tím, že se operace vyhodnocují zleva doprava, snadno spočítáme výsledek nezávisle na tom, jak jsme dokázali vyrobit číslo původní.

Vytvoříme si tedy pole výsledků `v` a pro každé číslo od 0 do 9 999 si do něj uložíme třeba -1, protože jej zatím vyrobit neumíme. Jen pro číslo 4 víme, že jej umíme vyrobit za pomocí jedné čtyřky.

Pak si pořídíme frontu čísel, která jsme dokázali vyrobit, ale ještě z nich neprozkoumali možnosti rozšíření o další čtyřku. Na začátku bude obsahovat jen tu první čtyřku.

Následně vždy vezmeme číslo x z fronty a podíváme se na výsledek operace $x + 4 = y$. Pokud jsme y ještě neviděli, ve `v[y]` bude stále mínus jednička. V tom případě toto musí být nejkratší způsob, jak y vyrobit, a můžeme do `v[y]` uložit `v[x] + 1`. Současně y přidáme do fronty.

To samé provedeme pro ostatní operace. Dáme si při tom pozor, abychom přeskočili dělení, pokud x není dělitelné čtyřmi.

Ve výpočtech jsme trochu zanedbali to, že Sářina kalkukačka umí počítat jen s čtyřciferными čísly. Jak bylo napovězeno v zadání, s výhodou použijeme operaci modulo. Každé y proženeme jednoduchým výrazem:

$$y_2 = (y_1 + 10\,000) \bmod 10\,000$$

Přičítat můžeme bezpečně, i když je číslo kladné, protože se 10 000 modulením zase odečte.

Až frontu vyprázdníme, určitě jsme našli všechna čísla, která vyrobit jdou, a v poli `v` máme výsledky, které chceme vracet. Stačí se do `v` podívat na správné místo a vypsat výsledek.

Pokud trochu znáte grafové algoritmy, můžete si všimnout, že popsaný způsob je vlastně prohledávání grafu do šířky. V tomto grafu jsou vrcholy čísla a z každého vedou tři nebo čtyři hrany za operace.

Můžeme prozradit, že každé číslo takto vytvořit jde, a dokonce je nejkratší výraz až na dvě čísla jednoznačný. Pokud byste chtěli zároveň výraz vypsat, můžete si do `v` uložit i x a operaci, kterou jste y vyrobili, a z těchto pak výraz zpětně poskládat.

Algoritmus má trochu překvapivě konstantní časovou složitost. Nejprve proběhne předvýpočet, který není závislý na vstupu, a tedy je z principu konstantní. Po přečtení vstupu už se jen podíváme na správné místo do pole. Pokud bychom neměli pevně zadaný rozsah 10 000 čísel, ale nějaké nejvyšší číslo K, časová složitost by byla lineární s K. Stejně tak paměťová.

Aby byl váš výpočet ještě rychlejší, mohli jste použít trik. Na co výsledky počítat při každém startu programu znovu, když si je můžete spočítat jednou a uložit pro příště například do souboru? Pro získání plného počtu bodů to ale vůbec nebylo nutné, 10 000 čísel spočítáte popsaným způsobem i v Pythonu bleskově.

Program (Python 3):
`http://ksp.mff.cuni.cz/viz/28-Z4-4.py`

Ondra Hlavatý

28-Z4-5 Vyhlazení GPS

Úlohu si trochu upravíme – místo toho, abychom počítali průměr, budeme počítat pouze součet posledních K hodnot. Každé číslo pak jen před vypsáním vydělíme K, takže vyřešíme původní zadání, ale přemýšlet se o tom bude lépe.

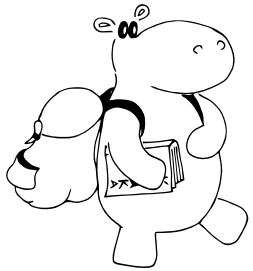
Použijeme myšlenku pohyblivého okénka. Prohlasíme na chvíli, že K je 3. Známe součet $S_1 = x_1 + x_2 + x_3$, a dostaneme x_4. Jak z nich spočítat součet $S_2 = x_2 + x_3 + x_4$?

Každý vidí, že stačí odečíst x_1 a přičíst x_4. Důležité je, že K může být úplně libovolné a vždy stačí nejstarší číslo odečíst a nové přičíst, tedy provést konstantní počet operací.

Pořídíme si proto pole, do kterého si budeme ukládat posledních K čísel, a jejich součet si budeme počítat průběžně. Vždy první číslo z pole vyhodíme a na konec jiné přidáme.

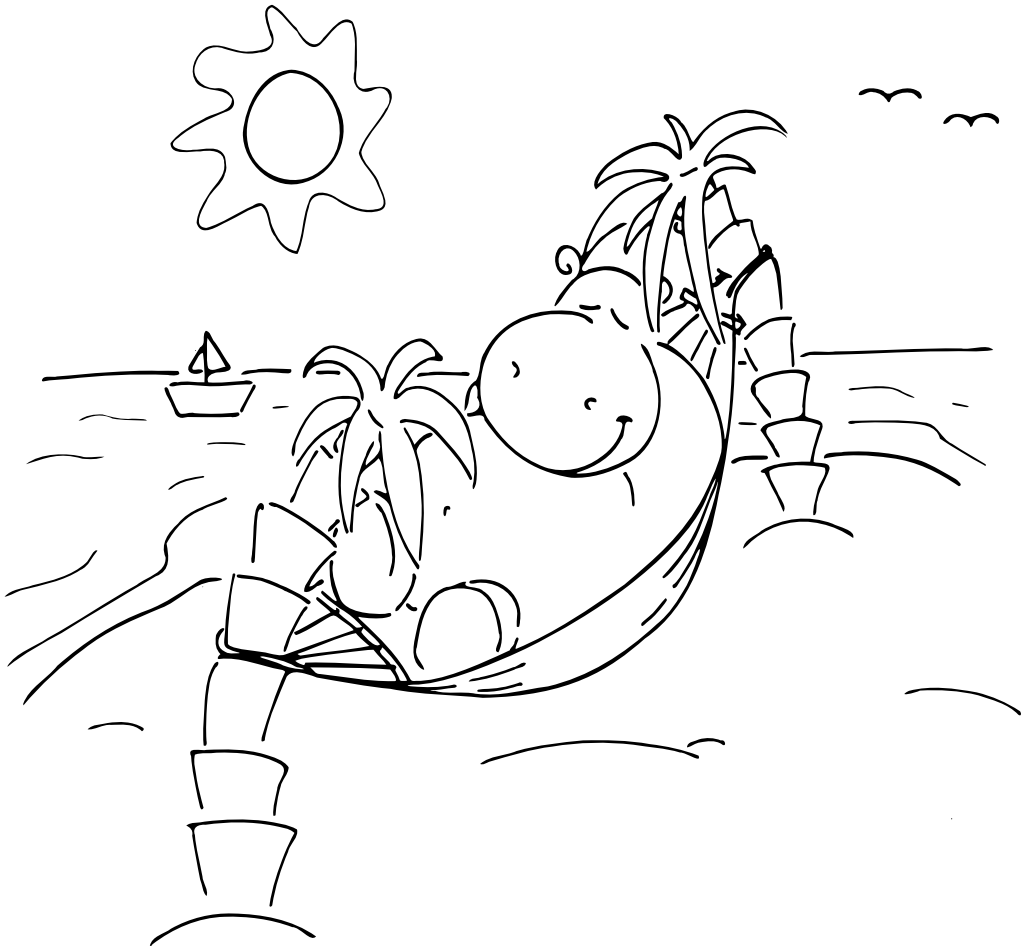
Počkat, tady něco nehraje. Vyhodit první číslo z pole nám zabere spoustu času, protože musíme ta ostatní posunout!

Představte si, že byste si mohli pořídit pole zatočené do kruhu, takový prstenec. Někdy se mu proto anglicky říká *ring buffer*, cyklický buffer. Prstenec bude stát před námi na stole a místo abychom my točili s prstencem, budeme obcházet okolo stolu.



Tolik představa, teď jak to implementovat. Pole zůstane rovné a bude mít K prvků jako předtím. V prvním kole budeme pracovat s prvním prvkem s indexem 0. V druhém s druhým… až v K-tém s K-tým s indexem $K - 1$. Potom by se nám hodilo, abychom pokračovali znovu od začátku. K tomu nám poslouží stará známá funkce modulo. V i-tém kole budeme pracovat s prvkem s indexem $i \bmod K$.

Přejeme Ti hezké prázdniny plné slunečných dní, zážitků



…a hrochů!

Dominik, Dominik, Filip,
Karry, Katka, Kuba, Marek,
Medvěd, Michal, Ondra, Toman,
Janka, Jenda, Jirka,
Marek, Martin, Martin,
Petr, Vašek, Vojta, Zuzka

Nechť se tedy průběžný součet jmenuje S a udržujeme cyklický buffer B. Jak S, tak všechny prvky B budou na začátku nulové. V i-tém kole odečteme od S prvek $B[i \bmod K]$ a následně na jeho místo vložíme nové číslo x_i, a zpátky ho přičteme do S.

Pokud by se nám nelíbilo, že počítadlo kroků i roste do nekonečna a může přetéci, budeme ho průběžně modulit. To totiž ničemu nevadí, my ho k jinému účelu nepotřebujeme.

Takto se nám podařilo udržovat průběžný součet s konstantní časovou složitostí na jeden krok. K celému výpočtu potřebujeme paměť velikosti $\mathcal{O}(K)$, kterou si ale vynulujeme na začátku, a v každém kroku přistupujeme pouze k jednomu prvku.

Program (Python 3):
`http://ksp.mff.cuni.cz/viz/28-Z4-5.py`

Ondra Hlavatý

28-Z4-6 Klučičí drby

Nejprve si uvědomíme, že drb se vždy přestane šířit – kluků je konečně mnoho a každý říká drb právě jednomu, takže časem se určitě dostane k někomu, kdo už jej zná.

Navíc víme, že o přestávce je stejný drb sdělován nejvýše jednou (říkáme ho pouze svému nejlepšímu kamarádovi). Proto se každý drb dostane do skupiny kamarádů, kteří si drby předávají do kruhu. Drb se zastaví tehdy, když se jej dozví všichni z nějakého cyklu.

Zkusme pro každý drb spočítat, kolik kluků jej bude znát. Uděláme to přesně tak, jak se drby šíří. Začneme u našeho nejlepšího kamaráda, poté je na řadě nejlepší kamarád toho našeho, pak jeho nejlepší kamarád, … A tak dál, dokud nenarazíme na nějakého, který už drb zná.

A abychom věděli, kolik kluků se ho dozvědělo, stačí si postupně nejlepší kamarády počítat (v každém kroku přičteme jedničku k počítadlu).

Jak poznáme, že jsme se právě dostali k někomu, kdo už drb zná? Budeme si kluky značit. Pořídíme si třeba pole plné nul, kde máme pro každého kluka jedno políčko. Vždy, když se někdo dozví drb, uložíme si do jeho políčka jedničku. Před šířením dalšího drbu si celé pole vynulujeme.

Složitost tohoto řešení je kvadratická vzhledem k počtu kluků ve třídě – každý drb sledujeme zvlášť, drbů je stejně jako kluků. Drb se navíc může rozšířit až ke všem klukům.

Ale jde to ještě zrychlit! Všimněte si, že často počítáme vícekrát, jak dlouho se šíří nějaký drb od stejného kluka. Mohli bychom si tedy pro každého kluka pamatovat, ke kolika dalším se od něj drb dostane. Tomuto číslu budeme říkat *společenský dopad*.

Jak společenské dopady získat a jak si je pamatovat? Jednoduše. Jako paměť nám poslouží další pole. Začneme opět simulovat šíření nějakého drbu. Postupně procházíme a počítáme nejlepší kamarády jako v prvním řešení, počítadlo si označíme jako K.

Rozdílem oproti kvadratickému řešení bude, že se zastavíme i tehdy, řekneme-li drb někomu, jehož společenský dopad už známe (budiž to číslo L). $K + L$ nám udává celkový počet kluků, kteří se dozví právě simulovaný drb.

Skočíme zpátky na začátek a projdeme stejnou cestu znovu, ale budeme u toho ukládat dopady jednotlivých kluků – první má $K+L$, druhý $K+L-1$, další $K+L-2$, … Těm, kteří už mají společenský dopad spočítaný, ho nepřepisujeme, dokonce u prvního takového se opět zastavíme.

Pořád se nám může stát, že se simulace zastaví kvůli podmínce z prvního řešení (přestane se i šířit drb). Připomeňme, že drb se zastaví, když se dostane do cyklu, a dozví-li se ho někdo z cyklu, dozví se ho i všichni ostatní z něj. To znamená, že společenský dopad všech kluků na cyklu bude stejný – délka cyklu (drb se z tohoto kruhu nikdy nikam dál nedostane).

Místo, kde se v cyklu zastavíme, je zároveň i to místo, kde do něj vstoupíme. Zapamatujeme si ho. Opět skočíme na začátek a ukládáme K, $K-1$, $K-2$, … Jakmile znovu narazíme na cyklus, přestaneme odečítat jedničku a ukládáme stejné číslo (právě délku cyklu).

Poznámka na konec: Tuto vylepšenou simulaci postupně spustíme pro každý drb (nemusí totiž existovat kluk, jehož společenský dopad postihne všechny).

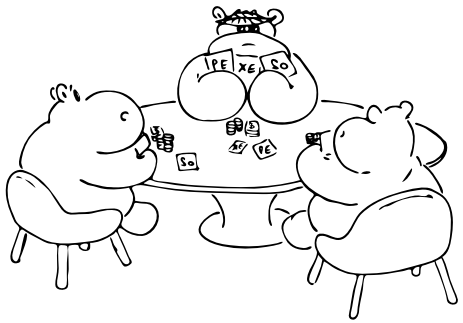
Zrychlili jsme řešení? Simulujeme opět všechny drby a simulace může trvat lineárně s počtem kluků ve třídě. Jenže čím déle trvá, tím více společenských dopadů spočítáme. Trvá-li K kroků, zapamatujeme si tím i K dosud neznámých dopadů. To ovšem znamená, že už je žádná další simulace nebude počítat – od každého kluka šíříme drb pouze jednou.

Dohromady uděláme tolik kroků, kolik je ve třídě kluků. Celková časová složitost je lineární, paměťová také.

Pokud by nás zajímala časová složitost jedné simulace, můžeme o ní tvrdit, že je amortizovaně konstantní. Sice můžou některé trvat dlouho, ale v průměru (když pustíme simulaci pro všechny drby) nám jedna trvá pouze konstatně dlouho.

Program (Python 3):
`http://ksp.mff.cuni.cz/viz/28-Z4-6.py`

Katka Zákravská & Jenda Hadrava



Výsledková listina čtvrté série začátečnické kategorie 28. 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.	Roman Bujdák	G JM Galanta	2	4	8	10	10	12	11	14	65,0	236,5
2.	Václav Brož	GZborovPH	1	4	8	10	10	12	11	9	60,0	233,0
3.	Pavel Koch	GTěš	4	4	8	10	10	12	12	9	61,0	226,7
4.	Daniel Skýpala	GTomkovaOL	−2	4	8	10	10	12	11	14	65,0	226,5
5.	Jakub Šťastný	G BO-Řeč	1	4	8	10	10	2	11	8	49,0	214,5
6.	Jiří Moravčík	GUHradiště	2	8	8	10	10	12	6	7	53,0	180,0
7.	Jiří Löffelmann	GLitoměřPH	2	4	8	10	10	12	11		51,0	150,5
8.	Vojtěch Hudec	G_ČTřebová	2	3	8	10	10		11	9	48,0	144,0
9.	Dennis Pražák	GJirsíkaČB	1	4	8		10		11		29,0	136,0
10.	Michael Bausano	GTěš	4	3							0,0	133,5
11.	Vendula Kuchyňová	GMLerchaBO	2	3							0,0	129,0
12.	Martin Bencko	GOhradníPH	−1	4	8	6	10	2	0	0	26,0	128,5
13.	Vojtěch Březina	GCoubTábor	−1	4	8		10				18,0	122,0
14.	Anna Hollmannová	GSRandyJN	−1	4	8	10	10		6	3	37,0	119,5
15.–16.	Jakub Jirkal	GJungmanLT	1	8	8	10	10				28,0	115,0
	Václav Pavlíček	ZŠ Ždírec nD	0	4			4				4,0	115,0
17.	Tomáš Terem	GTajBanBys	4	6							0,0	106,0
18.	Ondřej Gonzor	G Brandýs	−1	3	8	10	10		10	3	41,0	104,5
19.	Lukáš Riedel	G Bílina	4	3	0						0,0	104,0
20.	Antonín Prantl	G Strakon	3	3	8	10	10	2	10		40,0	101,0
21.	Michal Rickwood	G_ČTřebová	2	3				2			2,0	95,5
22.	Pavel Turinský	G Brandýs	3	4	8	10	10	12	12	14	66,0	84,0
23.	Jonáš Havelka	GJírovcČB	0	2		10	7	6			23,0	82,0
24.	Martin Picek	GJirsíkaČB	1	4							0,0	81,5
25.–26.	Vojtěch Káně	G Brandýs	0	4	8		10		12		30,0	80,0
	Jáchym Solecký	PORGPha	3	2							0,0	80,0
27.–28.	Tomáš Domes	MendelG_OP	3	2	8	10	10				28,0	78,5
	Samuel Schneider	GTajBanBys	4	4							0,0	78,5
29.–30.	Jan Kaifer	GČesBrod	0	7	8	10					18,0	77,5
	Michaela Štolová	G Sokolov	4	6	8	10	10				28,0	77,5
31.	Ondřej Krsička	Integra BO	0	4	8	4	7	2			21,0	76,0
32.	Jindřich Dítě	ZŠKom2ZS	0	4					4		4,0	71,5
33.	Josef Pospíšil	GÚstavníPH	2	4			3				3,0	69,0
34.	Petr Dedek	MensaG	0	2							0,0	66,0
35.	Radoslav Hašek	G_Čáslav	2	4	8		7				15,0	62,0
36.	Vojtěch Kuchař	ZŠ Sobotka	−1	4		10					10,0	60,9
37.	Roman Solař	GJarošeBO	4	2							0,0	60,0
38.	Tomáš Trojan	G Cheb	0	7							0,0	58,5
39.–40.	Adam Husník	GArabskáPH	2	1							0,0	58,0
	Pavel Souček	G_Nymburk	4	5							0,0	58,0
41.	Ľuboš Kolumber	SpojŠ Popr	4	6							0,0	56,5
42.–43.	Michael Olšavský	GNadKavaPH	1	1							0,0	56,0
	Petra Štefaníková	GOlgHavl	4	3							0,0	56,0
44.–45.	David Nápravník	GLitoměřPH	3	3							0,0	54,0
	Jiří Sejkora	GVoděraPH	4	4	8		10				18,0	54,0
46.	Vít Gaďurek	Neuvedená	1	5							0,0	53,0
47.	David Blažek	SPŠÚžlabPH	3	1							0,0	52,0
48.–49.	Dominik Krasula	GKrnov	3	4							0,0	48,0
	Václav Šraier	GČeskoliPH	3	2							0,0	48,0
50.	Tomáš Novotný	G BO-Řeč	2	1							0,0	47,0
51.	Andrej Čermák	G JF Šaľa	2	4							0,0	46,0
52.	Janek Hlavatý	GJirsíkaČB	−3	9							0,0	44,3
53.	Michal Szymik	G Wicht	2	2							0,0	44,0

	řešitel	*škola*	*ročník*	*sérií*	*Z4-1*	*Z4-2*	*Z4-3*	*Z4-4*	*Z4-5*	*Z4-6*	*série*	*celkem*
54.	Radek Jančík	GJarošeBO	3	3							0,0	43,0
55.	Ondřej Cach	ZŠPolab	0	3							0,0	40,5
56.–62.	Ondřej Baumgartner	GMost	4	1							0,0	40,0
	Ondřej Borýsek	GJarošeBO	3	1							0,0	40,0
	Hana Hladíková	GNadKavaPH	2	1							0,0	40,0
	Vojtěch Lanz	GZborovPH	2	1							0,0	40,0
	Jakub Matěna	GČeskoliPH	4	5							0,0	40,0
	Pavel Svoboda	ZŠJílovsPH	0	4	8						8,0	40,0
	Lucien Šíma	PORGPha	4	1							0,0	40,0
63.–65.	Robert Jaworski	GÚstavníPH	−2	3		10					10,0	36,0
	Jan Mráz	G Holice	2	5							0,0	36,0
	David Žáček	GZborovPH	3	4							0,0	36,0
66.	David Pavlík	GJarošeBO	3	1							0,0	30,0
67.–68.	Jakub Suchánek	GOpatovPHA	2	1							0,0	28,0
	Marek Černoch	GFPValMez	0	1							0,0	28,0
69.	Alexej Popovič	SlovanGOL	4	2							0,0	27,5
70.	Ondrej Potúček	G_GolNitra	2	2				4			4,0	27,0
71.–72.	Ludmila Bujnovská	MendelG_OP	2	1							0,0	26,0
	Matěj Šmíd	SPŠÚžlabPH	2	3							0,0	26,0
73.	Jiří Kvapil	GTomkovaOL	−2	1							0,0	24,0
74.	Eliška Vlčinská	GHladnov	1	1	8	3,3	3,3			9	23,7	23,7
75.	Filip Čermák	MendelG_OP	2	1							0,0	18,5
76.–78.	Radim Buráň	G UherBrod	1	1							0,0	18,0
	Jakub Dostál	SlovanGOL	2	3							0,0	18,0
	Pavel Martinec	GLesníZlín	2	2				10			10,0	18,0
79.–82.	Jan Burda	G Holice	2	8							0,0	16,0
	Ladislav Töpfer	G DrJPekMB	1	1							0,0	16,0
	Jan Vaník	GRNK	1	1							0,0	16,0
	Adam Šánta	GJHroncaBA	1	1							0,0	16,0
83.	Milan Kubala	GTajBanBys	4	3							0,0	15,0
84.	Jan Vozár	G UherBrod	2	9							0,0	13,0
85.–87.	Vanda Hendrychová	GHeyrovPH	4	1							0,0	12,0
	Jakub Růžička	G_Nymburk	1	1							0,0	12,0
	Lenka Vincenová	GTomkovaOL	2	1							0,0	12,0
88.	Michal Mlčoch	G UherBrod	1	1							0,0	10,0
89.–93.	Martin Hofbauer	G BO-Řeč	1	1							0,0	8,0
	Filip Matějka	GZborovPH	2	1							0,0	8,0
	Václav Plavec	GTep	3	1							0,0	8,0
	Daniel Pluskal	G BO-Řeč	2	3							0,0	8,0
	Martin Zima	G Holice	2	3							0,0	8,0
94.–95.	Dominik Karas	GTěš	4	2							0,0	7,0
	Roman Ondráček	GBoskovice	2	6				7	0		7,0	7,0
96.	Matěj Hudec	CírkG Plzeň	4	4							0,0	6,5
97.–98.	Josef Martínek	GPelhřimov	2	1							0,0	5,0
	Robert Wiesner	GJosefskPH	4	1							0,0	5,0
99.–100.	Martin Hubata	GMikulášPL	0	1							0,0	2,0
	Daniel Perout	GJarošeBO	−1	1							0,0	2,0
101.–103.	Tomáš Baják	ZŠ VBílovice	−1	2							0,0	1,0
	Rajmund Hruška	GPošKošice	3	2							0,0	1,0
	Martin Kolář	GÚstavníPH	3	1							0,0	1,0
104.–108.	Lenka Duongová	SvobChebŠ	−2	1							0,0	0,5
	Martin Mikšík	SPŠ Bo	1	1							0,0	0,5
	Filip Novotný	GJMasar_JI	0	1							0,0	0,5
	Petr Zahradník	GaSOŠ ÚL	1	1							0,0	0,5
	Petr Šícho	GKepleraPH	−2	1							0,0	0,5

Blahopřejeme k získaným bodům a doufáme, že jste si z letošního ročníku odnesli spoustu cenných zkušeností. Těšíme se na vaše řešení v příštím ročníku!