Korespondenční Seminář z Programování

ZAČÁTEČNICKÁ KATEGORIE

34. ročník | **KSP-Z** | **Říjen 2021**

Řešení první série začátečnické kategorie 34. ročníku KSP

34-Z1-1 Letopočty

Zřejmé řešení by bylo postupně pro každý rok, který Kevin na planetě stráví, spočítat počet dní, a nakonec všechna tato čísla sečíst. Pro přestupný rok se stačí dívat, jestli je číslo dělitelné 3 a zároveň není dělitelné 48. Tím získáme řešení, které je lineární v počtu dní, které Kevin na planetě stráví. Takto šly získat až 4 body, ale pak už rozdíl mezi počátečním a koncovým rokem byl příliš veliký na to, aby šel tento postup aplikovat.

Určitě nás napadne, že to půjde zlepšit. Přeci když přesně známe, kolikátý každý rok je přestupný (a kdy jsou výjimky), tak se nemusíme dívat na každý rok zvlášť.

Nejdříve si spočítáme, kolik dní by Kevin na planetě strávil, kdyby měl každý rok 42 dní. Konkrétně by to bylo $42 \cdot (y - x + 1)$ dní. Dále musíme započítat všechny přestupné roky, tedy přičíst den za každý násobek trojky v rocích, ve kterých se Kevin na planetě nachází. Tím jsme ale započetli až moc přestupných let – protože 48 je násobkem 3, tak jsme započítávali mezi přestupné roky i ty, které ve skutečnosti byly nepřestupné. Za každý násobek 48 tedy musíme jeden den odečíst. Tento postup už vede do cíle, jen musíme zjistit, kolik přesně je kterých násobků.

Číslo prvního přestupného roku zjistíme jako $x_p = 3 \cdot \lceil x/3 \rceil$. Výraz $\lceil x/3 \rceil$ značí horní celou část z $x/3$, tedy nejmenší celé číslo větší nebo rovno $x/3$. Celý výraz pak značí nejmenší celé číslo větší nebo rovno x, které je násobkem trojky. Například pro $x = 100$ bychom dostali 102. V programovacích jazycích je pro celou část často dostupná funkce `ceil`.

Obdobným způsobem zjistíme číslo posledního přestupného roku, tentokráte jako $y_p = 3 \cdot \lfloor y/3 \rfloor$. Zde používáme dolní celou část neboli `floor`.

Teď už můžeme spočítat celkový počet přestupných roků jako $(y_p - x_p)/3 + 1$. Zkuste si rozmyslet, že tento postup bude fungovat i tehdy, pokud během Kevinova pobytu žádný přestupný rok nenastane.

Počet výjimek (čísel dělitelných 48, tedy roků nepřestupných) spočítáme úplně stejně, jen 3 nahradíme za 48.

Zbývá z výsledků spočítat odpověď. Celkový počet dní je tedy $42 \cdot (y - x + 1)$+ počet přestupných dní − počet výjimečných dní.

Jaká je časová složitost vylepšeného řešení? Počet kroků algoritmu vůbec nezáleží na číslech roků na vstupu. Je tedy časová složitost konstantní? Pokud předpokládáme, že jsou čísla roků rozumně velká na to, abychom s nimi dokázali počítat, pak ano. Jinak bychom museli násobení a dělení zpracovávat ručně, pak by byla složitost lineární v počtu cifer.

Program (Python 3):
`http://ksp.mff.cuni.cz/viz/34-Z1-1.py`

Úlohu připravili: Robert Gemrot, David Klement

34-Z1-2 Čísla domů

Domy si můžeme představit jako uspořádané dvojice pozice a času dostavby. Pozicí máme na mysli informaci o kolikátý dům v řadě se jedná.

Když si tyto uspořádané dvojice setřídíme podle data, tak snadno můžeme zrekonstruovat celou stavbu a domům přiřadit čísla.

Domy totiž byly postaveny v pořadí, v jakém jsou v tomto setříděném poli, stačí jim tedy jen postupně přidělit čísla.

Chceme ale odpovídat v původním pořadí, připravíme si tedy pole odpovědí a to postupně naplníme. Projdeme setříděné dvojice a do pole odpovědí na index pozice domu napíšeme jeho číslo. Nakonec už jen stačí pole odpovědí vytisknout.

Jak ale jednoduše třídit data a časy? Povšimneme si nádherné vlastnosti formátu, ve kterém je zadaný vstup – stačí ho totiž třídit jako řetězce. Díky tomu, že všechny části mají jasně danou délku a jsou uspořádány dle jejich důležitosti, tak vše funguje.

Časová složitost tohoto algoritmu je $\mathcal{O}(N \log N)$. Paměťová je $\mathcal{O}(N)$.

Program (Python 3):
`http://ksp.mff.cuni.cz/viz/34-Z1-2.py`

Program (C++):
`http://ksp.mff.cuni.cz/viz/34-Z1-2.cpp`

Úlohu připravili: Filip Hejsek, Jirka Kalvoda

34-Z1-3 Hádání čísel

Nejpřímočařejším řešením by bylo ptát se postupně na čísla od jedné, dokud neuhodneme to správné. V nejhorším případě ale může hledané číslo být až R, takže by to mělo lineární časovou složitost $\mathcal{O}(R)$.

Čísla ale nemusíme zkoušet po jednom. Když se zeptáme na nějaké číslo a není to to správné, dozvíme se také, kterým směrem máme hledat dál. Například když se na začátku zeptáme na číslo x a dostaneme odpověď `Uber`, čísla x až R už nemusíme řešit a stačí dále prohledávat jenom čísla 1 až $x - 1$. Když si číslo x budeme volit strategicky, v každém kroku si umíme prohledávaný úsek výrazně zmenšit.

Když bude x blízko jednoho okraje, je sice možné, že tím vyřadíme velký úsek najednou, ale v nejhorším případě bude vždy hledané číslo na špatné straně a vyřadíme jenom malý kousek. Zato když si zvolíme x uprostřed, budou úseky na obou stranách stejně dlouhé a vždy tedy vyřadíme alespoň polovinu čísel.

Algoritmus by tedy vypadal takto: Pamatujeme si okraje úseku, který nám zbývá k prohledání, na začátku jsou to $k_1 = 1$ a $k_2 = R$. Zeptáme se na $x = \frac{k_1 + k_2}{2}$ (s celočíselným dělením) a, pokud x není to správné, zmenšíme si podle toho rozsah. Pokud je moc velké, nastavíme si k_2 na $x - 1$ a,

pokud je moc malé, nastavíme si k_1 na $x + 1$. Toto opakujeme, dokud buď neuhodneme správné číslo, nebo nezbyde v rozsahu jenom jedno číslo ($k_1 = k_2$), které už musí být to správné.

Jelikož po každém kroku je rozsah minimálně o polovinu menší (a když dojdeme na rozsah 1, tak se zastavíme), tak když se na to podíváme v opačném pořadí (v každém kroku zpět se velikost rozsahu alespoň zdvojnásobí), tak původní velikost rozsahu (R), musela být nejméně dva na počet kroků. Z toho vyplývá, že kroků mohlo být maximálně $\log_2 R$. Protože každý krok trvá konstantní čas a má jeden dotaz, časová složitost i počet dotazů budou $\mathcal{O}(\log R)$. Paměť je konstantní – používáme jen dvě proměnné.

(Pozn.: I pokud bychom volili x jinak, ale pořád s konstantním poměrem velikostí úseků na obou stranách, časová složitost by byla stejná, jenom by se zhoršila konstanta. Například pro $x = \frac{k_1+2k_2}{3}$ je ten delší úsek dvě třetiny rozsahu místo poloviny, takže by se počítal logaritmus o základu 1.5 místo 2.)

Tomuto postupu se říká *binární vyhledávání* a obecně se používá, když potřebujeme najít pozici konkrétního prvku v seřazeném poli pomocí porovnávání.

Program (Python 3):
`http://ksp.mff.cuni.cz/viz/34-Z1-3.py`

Úlohu připravili: Jan Adámek, Jirka Kalvoda, Martin „Medvěd“ Mareš

34-Z1-4 Šíření zpráv

Zamysleme se nejprve, jak bychom úlohu řešili ručně – nejspíše bychom procházeli log zpráv podle času, zkontrolovali, jestli odesílatel už o úspěchu věděl a pokud ano, poznamenali si uvědomělost i u příjemce. Pak už stačí jen ve správném pořadí vypsat ID všech informovaných lidí.

Výhoda této myšlenky je, že zaručeně funguje. Nepřevedli jsme si úlohu na žádný jiný problém, děláme prostě přesně to, co po nás chce zadání. Bohužel ale k plnohodnotnému algoritmu chybí vyjasnit několik detailů. Pojďme je doplnit.

Jak si pamatovat, kdo už o úspěchu ví? Pořiďme si pole a nově informované prostě vložme nakonec. Hledání obnáší celé pole projít a podívat se, jestli se v něm hledaná osoba nachází. Takové řešení je ale dost pomalé – každé z M hledání trvá až $\mathcal{O}(N)$ kroků, tedy celkově $\mathcal{O}(MN)$. Navíc musíme na konci ještě seznam ID seřadit. Paměti pak spotřebujeme $\mathcal{O}(M)$.

Když je N malé

Co kdyby N bylo rozumně velké? To nám zadání neslibuje, ale je zajímavé (i z hlediska zisku bodů) se nad touto variantou zamyslet. O ID máme slíbeno, že to je celé čisla od 1 do N. Mohli bychom si tedy pořídit pole délky N a na i-té pozici (nebo $i - 1$, číslujeme-li od nuly) si uchovávat stav informovanosti člověka s ID i. Jak rychlé to je teď? Potřebujeme čas $\mathcal{O}(N)$ na pořízení pole, $\mathcal{O}(1)$ na každé z M vyhledávání, tedy celkově jsme se dostali na čas $\mathcal{O}(N+M)$. Paměti naše pole zabírá $\mathcal{O}(N)$.

Tato složitost zní hezky – vždyť je lineární, ale má to háček. N totiž není omezené velikostí vstupu. Mohlo by se stát, že v logu máme jen 2 různé uživatele, kteří mají ID 1 a 10^9. Pak je vstup maličký, ale náš program spotřebuje řádově gigabajty paměti a přípravou pole stráví zbytečně mnoho času.

Zrychlení přečíslováním ID

Jak to opravit? Můžeme před zpracováním logů nejprve přečíslovat ID. Průchodem logu si uložíme do pole jednotlivá ID odesílatele i adresáta, která jsou na jednotlivých řádcích. Pak toto pole seřadíme. Seřazení pole vyřešíme v čase $\mathcal{O}(M \log M)$ – v poli máme přesně $2M$ záznamů.

Máme nyní v ruce seřazené pole, které obsahuje všechna možná ID. Můžeme tedy každé ID přečíslovat na pozici jeho prvního výskytu v tomto poli. Jelikož je pole seřazená, můžeme tuto pozici najít binárním vyhledáváním. Přečíslování ID u jednoho řádku logu tedy zabere $\mathcal{O}(\log M)$ času. Tím změníme N na $2M$ a můžeme použít řešení pro rozumně velké N, které bude potřebovat $\mathcal{O}(M)$ času i prostoru.

Chceme-li získat původní nepozměněné ID, stačí se podívat v poli na číslo na pozici přečíslovaného ID. Celý algoritmus s přečíslováním tudíž potrvá čas $\mathcal{O}(M \log M)$ a spotřebuje paměť $\mathcal{O}(M)$.

Zrychlení množinou

Můžeme se dostat čas $\mathcal{O}(M \log M)$ i tak, že pole v kvadratickém řešení nahradíme *množinou*. Ta umí rychle ukládat prvky a odpovídat, zda je prvek uložen. Zde budou prvky množiny jednotlivá ID. Pak místo hledání v poli se zeptáme, zda je ID v množině a místo přidávání ID na konec jej do množiny uložíme. Na konci pak z množiny vytáhneme (a seřadíme, pokud už nejsou seřazeny) všechny klíče.

Způsob, jak vytvořit množinu, je mnohem pokročilejší a zde jej uvádět nebudeme. Můžete si jej však přečíst v kuchařkách o binárních vyhledávacích stromech[1] nebo hešování.[2]

Úlohu připravili: Vojta Káně, Martin Koreček, Kristýna Petrlíková

[1] `http://ksp.mff.cuni.cz/viz/kucharky/vyhledavaci-stromy`
[2] `http://ksp.mff.cuni.cz/viz/kucharky/hesovani`

Výsledková listina první série začátečnické kategorie 34. ročníku KSP

	řešitel	*škola*	*ročník*	*sérií*	*Z1-1*	*Z1-2*	*Z1-3*	*Z1-4*	*série*	*celkem*
0.					8	10	12	14	44,0	44,0
1.–2.	Zuzana Aubrechtová	GHeyrovPH	3	1	8	10	12	14	44,0	44,0
	Richard Tichý	SG Kladno	0	1	8	10	12	14	44,0	44,0
3.	Tomáš Janovec	GMnichHrad	4	7	8	10	12	12,5	42,5	42,5
4.–17.	Olga Cinková	ArcibisGPH	2	6	8	8	12	14	42,0	42,0
	Viktor Číhal	SPŠSmíchov	2	3	8	10	12	12	42,0	42,0
	Štěpán Fröde	GDobruška	2	1	8	10	12	12	42,0	42,0
	Jakub Hampl	GMělník	2	1	8	10	12	12	42,0	42,0
	Matěj Hošek	GVolgogrOS	0	2	8	10	12	12	42,0	42,0
	Jáchym Kouba	GJŠkodyPŘ	2	1	8	10	12	12	42,0	42,0
	Stanislav Kozák	G Holice	4	2	8	10	12	12	42,0	42,0
	Marek Maškarinec	SPŠEMasLI	1	4	8	10	12	12	42,0	42,0
	Kryštof Maxera	GJírovcČB	1	8	8	10	12	12	42,0	42,0
	Anna-Kristina Migel	GNAlejíPH	−1	1	8	10	12	12	42,0	42,0
	Ján Plachý	G VBN Prie	4	1	8	10	12	12	42,0	42,0
	Jakub Podskalský	SSŠVTPraha	2	1	8	10	12	12	42,0	42,0
	Jan Prosecký	GNoMěsNMor	3	1	8	10	12	12	42,0	42,0
	Matěj Strnad	SPŠJičín	1	6	8	10	12	12	42,0	42,0
18.	Adam Kuča	PORG Krč	4	1	8	10	12	11	41,0	41,0
19.–24.	Kryštof Latka	PORG Krč	4	4	8	10	12	10	40,0	40,0
	Filip Neubauer	AkademGPH	2	1	8	10	12	10	40,0	40,0
	Vít Olšovec	GPřípotoPH	0	1	8	10	12	10	40,0	40,0
	Michal Pavlíček	MendelGOP	4	1	8	10	12	10	40,0	40,0
	Tomáš Pražák	GJSeiferPH	1	3	8	10	12	10	40,0	40,0
	Jakub Smolík	GEBenešeKL	4	5	8	10	12	10	40,0	40,0
25.	Jakub Mikeš	GJŠkodyPŘ	4	4	8	8	12	11	39,0	39,0
26.–29.	Honza Kocourek	ParkLane	2	2	8	8	12	10	38,0	38,0
	Lukáš Linek	GOpatovPHA	−2	1	8	10	12	8	38,0	38,0
	Jaromír Obitko	ZS6 Kladno	0	1	8	8	12	10	38,0	38,0
	Matúš Púll	GZborovPH	2	1	8	10	12	8	38,0	38,0
30.	Adam Jahoda	GKepleraPH	3	3	4	10	12	11	37,0	37,0
31.	Svatava Šimečková	GJarošeBO	0	2	4	10	12	10	36,0	36,0
32.	Václav Kouřil	GTachov	4	1	4	8	12	9	33,0	33,0
33.–46.	Pavel Altmann	GMikulášPL	3	8	8	10	12		30,0	30,0
	Alexandr Bihun	GJírovcČB	2	1	8	10	12		30,0	30,0
	Matúš Duchyňa	GGrössBA	3	1	8	10	12		30,0	30,0
	Nikolay Fomichev	SSŠVTPraha	3	1	8	10	12		30,0	30,0
	Jan Hlavsa	GMělník	4	6	8	10	12		30,0	30,0
	Adam Kolník	SSŠVTPraha	3	5	8	10	12		30,0	30,0
	Kryštof Marek	SGPCE	2	4	8	10	12		30,0	30,0
	Jakub Nevařil	G UherBrod	4	13	8	10	12		30,0	30,0
	Nikol Poláková	GMetodovaBA	3	1	8	10	12		30,0	30,0
	Vladimir Sklenár	GTerVans	2	1	8	10	12		30,0	30,0
	Vojtěch Skyba	G UherBrod	4	4	8	10	12		30,0	30,0
	Daniel Šoltýs	GTřeKošice	4	4	8	10	12		30,0	30,0
	Ivan Trenčanský	GLSáru	3	1	8	10	12		30,0	30,0
	Vojtěch Venzara	GMělník	4	6	8	10	12		30,0	30,0
47.–51.	Samuel Dembinný	SPŠ Kladno	1	1	8	8	12		28,0	28,0
	Lukáš Létal	GJŠkodyPŘ	3	6	8	8	12		28,0	28,0
	Milan Savickij	SPŠSmíchov	2	1	8	8	12		28,0	28,0
	Ondřej Stupka	GVolgogrOS	2	1	8	8	12		28,0	28,0
	Tadeáš Zíka	SPŠSmíchov	1	1	8	8	12		28,0	28,0
52.–54.	David Pacák	G Brandýs	1	1	4	10	12		26,0	26,0
	Alexandra Seďová	GVídeňskBO	1	1	6	8	12		26,0	26,0
	Oto Skýpala	GJŠkodyPŘ	−2	4	4	10	12		26,0	26,0
55.	Jan Straka	VOŠ Žďár	2	1	4	8	12	1	25,0	25,0

	řešitel	*škola*	*ročník*	*sérií*	*Z1-1*	*Z1-2*	*Z1-3*	*Z1-4*	*série*	*celkem*
56.	Jáchym Löwenhöffer	GEvolutionJM	1	1	4	8		10	22,0	22,0
57.	Veronika Jůzková	MensaG	4	10		8	12		20,0	20,0
58.–60.	Viktor Čubík	G UherBrod	4	2	8	10			18,0	18,0
	Alexander Mateides	GJirsíkaČB	3	4	8	10			18,0	18,0
	Petr Starý	GJírovcČB	0	1	8	10			18,0	18,0
61.	Vít Mitáš	GPolička	0	1	0	8		9	17,0	17,0
62.–64.	Matěj Kříž	GDašickáPA	4	1	8	8			16,0	16,0
	Arnošt Polák	PORG Krč	4	2	8	8			16,0	16,0
	Erik Sabol	GČeskoliPH	2	9	8	8			16,0	16,0
65.	Jáchym Tuma	G FrýdlNOs	1	1	2		12		14,0	14,0
66.	Robin Kovar	GPŠ Praha	0	1	4	8			12,0	12,0
67.	Thomas Riedle	BRG APP	3	10	8	2			10,0	10,0
68.–70.	Adam Húšťava	EupSchoolLux	4	15	8				8,0	8,0
	Zara Karakaya	TAPoprad	4	1	8				8,0	8,0
	Marek Švajda	G UherBrod	4	1	8				8,0	8,0
71.	Jakub Kopčil	GMikulášPL	3	4	0,7	2,7	4		7,3	7,3
72.	Michal Mík	SSŠVTPraha	1	1	6				6,0	6,0
73.–76.	Albert Bakoč	GZborovPH	1	1	4				4,0	4,0
	Jáchym Hájek	GBNěmcovHK	−1	4	4				4,0	4,0
	Robert Klimt	GDobříš	2	1	4				4,0	4,0
	Matyas Oliva	G UherBrod	4	2	4				4,0	4,0
77.–78.	Janek Hlavatý	GJirsíkaČB	3	22	2				2,0	2,0
	Kateřina Vomelová	GÚstavníPH	2	1	2				2,0	2,0