

### Řešení třetí série začátečnické kategorie 38. ročníku KSP

#### 38-Z3-1 Osobní pokladna

Zadání po nás chce vyhodnotit výraz zapsaný v notaci, kde jsou operátory psány za operandy místo mezi operandy, jak bychom mohli být zvyklí. Této notaci se běžně říká postfixová notace nebo RPN (*Reverse Polish notation*), ale to pro vyřešení úlohy vědět nepotřebujeme, zadání totiž popisuje i to, jakým způsobem se operátory vlastně vyhodnocují; hladově zleva.

Jinými slovy: budeme procházet vstup a když narazíme na operand (tedy číslo), někde si ho uložíme. Pokud narazíme na operátor, potřebujeme znát poslední dvě čísla, s nimi operaci provést a výsledek (tedy opět číslo) si zase uložit. Máme slíbeno, že se po nás nebude chtít dělit nulou a že na konci nám zbyde právě jedno číslo, které je výsledkem a o takové chybové stavy se tedy nemusíme starat.

Zbývá vymyslet jak uložit čísla tak, abychom měli jednoduchý přístup k těm, která jsme uložili jako poslední. K tomu se nabízí použít zásobník, který přesně toto splňuje – při odebrání prvku dostaneme ten nejpozději přidaný. V Pythonu k tomu můžeme použít obyčejný seznam a metodu `.append()` pro přidání prvku a `.pop()` pro odebrání prvku.

Na konec je ještě dobré rozmyslet si pořadí operandů.  $2\ 3$  – totiž znamená  $2 - 3$  v běžné notaci. Do zásobníku bychom tedy vložili postupně 2 a 3 a při odebrání ze zásobníku bychom dostali nejdříve 3; první odebraný prvek ze zásobníku je tedy druhým operandem a výsledek je  $-1$  (a ne 1 při popleteném pořadí).

A jak to bude rychlé? Celý vstup projdeme jednou a na každém operátoru nebo operandu strávíme konstantně času, výsledná složitost je tedy  $\mathcal{O}(n)$ , kde  $n$  je délka vstupu.

Program (Python):

<http://ksp.mff.cuni.cz/viz/38-Z3-1.py>

Úlohu připravili: Honza Černohorský,  
Ondra Machota, David „Dejwut“ Pacák

#### 38-Z3-2 Největší dort

Na vstupu máme  $K$  bodů a hledáme mezi nimi čtyři takové, že tvoří obdélník rovnoběžný s osami s co největším obsahem. Jinými slovy, hledáme čtveřici bodů tvaru  $(x_1, y_1)$ ,  $(x_1, y_2)$ ,  $(x_2, y_1)$ ,  $(x_2, y_2)$  a zajímá nás, jakou největší hodnotu výrazu  $|x_1 - x_2| \cdot |y_1 - y_2|$  můžeme dosáhnout.

Nejjednodušší řešení je pomocí čtyř vnořených for-cyklů vyzkoušet všechny možné čtveřice bodů. Pro každou čtveřici zjistíme, zda tvoří rohy obdélníka, a pokud ano, spočítáme jeho obsah. Během algoritmu si budeme udržovat obdélník s doposud největším obsahem. Takové řešení má složitost  $\mathcal{O}(K^4)$ .

Můžeme přijít se zlepšovákem: rozmyslíme si, že když zkusíme nějaké tři vrcholy obdélníka, tak umíme jednoznačně určit, kde musí ležet čtvrtý vrchol – popřípadě, že naše tři body jsou slepá ulička a obdélník z nich postavit nejde. Sta-

čí nám tedy zkusit všechny trojice bodů, dopočítat čtvrtý bod, a otestovat, zda se nachází mezi našimi  $K$  body. To umíme v konstantním čase, pokud použijeme hešování (v Pythonu `set` nebo `dict`), případně v čase  $\mathcal{O}(\log K)$ , pokud si na začátku body seřadíme (například primárně podle  $x$ -ové souřadnice a sekundárně podle  $y$ -ové) a pak použijeme binární vyhledávání. Celková časová složitost je tedy  $\mathcal{O}(K^3)$ , případně  $\mathcal{O}(K^3 \log K)$ .

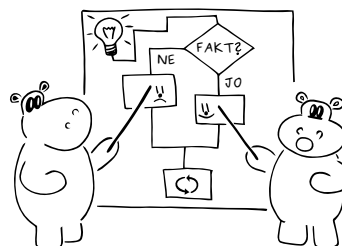
Tento postup umíme ještě vylepšit, když si uvědomíme, že nám stačí zkusit jen dva body: levý dolní a pravý horní roh obdélníka. Zbylé dva body z nich už umíme dopočítat: pokud je levý dolní roh na souřadnicích  $(x_1, y_1)$  a pravý horní na  $(x_2, y_2)$ , pak levý horní roh je na  $(x_1, y_2)$  a pravý dolní na  $(x_2, y_1)$ . Jejich existenci umíme opět ověřit buď hešováním, nebo binárním vyhledáváním. Tím se dostaneme na časovou složitost  $\mathcal{O}(K^2)$ , resp.  $\mathcal{O}(K^2 \log K)$ .

Dobrym zvykem je ověřit, že náš algoritmus funguje, tedy, že vypíše platný obdélník a ten bude mít největší možný obsah. Na jednu stranu algoritmus určitě vypíše nějaký platný obdélník: dva rohové body jsou z naší množiny bodů a u obou dopočítaných bodů v algoritmu explicitně kontrolujeme, zda existují. Na druhou stranu, vezmeme-li obdélník  $O$  s největším obsahem, náš algoritmus ho najde, neboť zkouší všechny dvojice bodů, takže natrefí i dvojici (levý dolní roh  $O$ , pravý spodní roh  $O$ ).

Program (Python):

<http://ksp.mff.cuni.cz/viz/38-Z3-2.py>

Úlohu připravili: Daniel Culliver,  
Riša Hladík, Jirka Kvapil, Vladimír Sklenář



#### 38-Z3-3 Polární express

Na vstupu dostaneme rozměry mřížky  $R$  a  $S$ , počáteční pozici lokomotivy  $x, y$  a vagóny neboli body přejezdu  $N$ . Naším úkolem je vymyslet cestu, která projde všechny vagóny. Cestu zapisujeme pomocí znaků  $>$ ,  $<$ ,  $v$  a  $^$ , které značí pohyb doprava, doleva, dolů a nahoru. Cesta nemusí být nejkratší, ale musí mít méně než  $10^6$  kroků.

Triviální řešení je pustit vlak tak, aby prošel všechna políčka mřížky a vrátil se zpět na start. To zaručí, že posbíráme všechny vagóny bez nehody. Nevýhoda: cesta má délku  $R \cdot S$ , což může být více než  $10^6$ .

Jak to ale vylepšit? Klíčem je projít body po řádcích tak, aby vlak nemohl vrazit do sebe. K tomu využijeme, že mřížka je toroidální: když se vlak dostane na pravý okraj, může

pokračovat zleva, a když se vlak dostane na spodní okraj, může pokračovat shora. Díky tomu neexistuje okraj mapy a vlak se může pohybovat bez omezení.

Pro jednodušší práci se souřadnicemi posuneme lokomotivu na  $(0, 0)$  a všechny vagony posuneme o stejnou hodnotu jako lokomotivu. Tady využijeme toroidální vlastnost mřížky – když posuneme lokomotivu o  $x$  doprava, posuneme všechny vagony také o  $x$  doprava. Tímto trikem získáme nejen jednodušší souřadnice, ale i informaci o tom, jak daleko jsou vagony vpravo a dolů od lokomotivy.

Nyní seřadíme vagony, lexikograficky podle řádků a sloupců. V každém řádku pak postupujeme takto: pokud jsme ve sloupci  $c$ , navštívíme všechny vagony v daném řádku, které jsou v sloupcích větších než  $c$ , a pak se vrátíme přes torus až do posledního vagonu co může být  $c - 1$ . V posledním bodě řádku se pak posuneme dolů a opakujeme tento postup pro další řádek. Tímto způsobem zajistíme, že se vlak nikdy nesrazí sám se sebou.

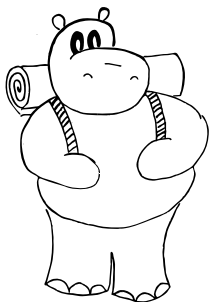
Pohyb mezi body je pak jednoduchý: mezi dvěma sousedními body spočítáme rozdíl v poloze a to nám řekne, kolik kroků musíme udělat doprava a dolů. Přitom využijeme toroidní vlastnost mřížky – když musíme jít záporně doprava, půjdeme místo toho doprava přes levý okraj, a podobně pro dolů.

Řešení má časovou složitost  $\mathcal{O}((N \log N) + N \cdot \max(R, S))$ . Třídění zabere  $\mathcal{O}(N \log N)$  plus čas na vypisování cesty, která může mít délku až  $2 \cdot N \cdot \max(R, S)$ , protože v nejhorším případě můžeme projít všechny vagony a mezi nimi ujít vzdálenost  $R$  nebo  $S$ .

Program (Python):

`http://ksp.mff.cuni.cz/viz/38-Z3-3.py`

Úlohu připravili: Ondra Sedláček,  
Vladimír Sklenář, Dan Skýpala



---

---

### 38-Z3-4 Nákup v poslední chvíli

---

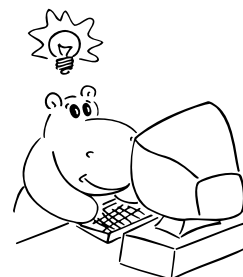
---

Na vstupu jsou časy příchodu každého zákazníka  $a_1, \dots, a_n$ . Nejjednodušší způsob, jak přijít na optimální počet stanic  $K$ , je postupně si simulovat, jestli zákazníkům bude stačit  $k = 1, k = 2, k = 3, \dots$ , než na optimum narazíme. V celém řešení budeme předpokládat, že přesun zákazníků je instantní. Nejprve si seřadíme seznam časů příchodu a pak spustíme simulaci pro  $k$  stanic:

Pro tuto simulaci si vyrobíme pole  $S$  o  $k$  prvcích, kde si budeme pamatovat, v jaký čas bude každá stanice volná – na začátku jsou všechny volné v čase 0. Pak iterujeme seřazený seznam příchodů. Pro příchod  $a_i$  najdeme minimum  $c$  v poli  $S$ , respektive čas kdy bude nějaká stanice nejdříve volná. Pokud je  $c \geq a_i + 15$ , tedy právě přichodí zákazník musí čekat déle než patnáct minut, víme, že  $k$  stanic nám nebude stačit a spustíme novou simulaci pro

$k + 1$ . Jinak můžeme zákazníka k této stanici umístit, ta pak dalšího zákazníka může obsloužit v čase  $\max(c, a_i) + 5$ ; tedy v moment, kdy je stanoviště připraveno nebo kdy přijde zákazník (kterákoli ze situací nastane později) plus pět minut. Touto hodnotou nahradíme  $c$  v seznamu a můžeme se posunout k dalšímu zákazníkovi. Pokud takto úspěšně odsimulujeme i posledního zákazníka, našli jsme to nejlepší  $K$ , neboť všechny simulace předtím musely selhat.

Jak jsme na tom s časovou složitostí? V nejhorším případě, kdy všichni zákazníci přijdou nahuštění v jeden čas, budeme potřebovat  $K = n/3$  stanic (za patnáct minut u jedné stanice obsloužíme tři zákazníky). Simulací tedy provedeme  $\mathcal{O}(n)$ -krát. Jedna simulace musí projít všech  $n$  zákazníků, zároveň pro každého zákazníka hledáme minimum v seznamu dlouhém  $k = n/3$ . Společně s počátečním tříděním v čase  $\mathcal{O}(n \log n)$  vyjdeme s monstrózní složitostí  $\mathcal{O}(n^3)$ . Zbytečně hodně času mrháme na opakovaném hledání minima a slepém zkoušení různých  $k$ . Jak to zvládnout lépe?



Začneme opět se setříděným seznamem příchodů zákazníků. Místo toho, abychom museli pokaždé minimální čas pracně hledat, pořídíme si datovou strukturu, která tu těžkou práci odvede za nás, tedy minimovou haldou – ta za cenu pomalejších přidávání nových prvků zvládne minimálním časem odpovědět konstantně rychle. Na začátku si do haldy umístíme pouze jeden čas 0. Pak v jedné smyčce pro každý čas návštěvy  $a_i$  opakujeme:

Podíváme se na minimální prvek  $c$  v haldě. Pokud  $c \leq a_i + 15$ , stihne tato stanice obsloužit zákazníka včas. Odebere  $c$  z haldy a zpět do ní přidáme  $\max(c, a_i) + 5$ . Jinak je  $c > a_i + 15$ , tedy v haldě nemáme dost stanic. To jednoduše vyřešíme přidáním nového čísla  $a_i + 5$  (můžeme si to představit tak, že tam tato stanice čekala na tohoto zákazníka a dále ji pak mohou využít i ostatní). Odpovědí na otázku je pak počet čísel v haldě.

Toto bude fungovat, protože otvíráme novou stanici (přidáváme nové číslo do haldy) pouze v moment, kdy je to nutné. Jakmile otevřeme stanici  $K$  pro  $i$ -tého návštěvníka, všech ostatních  $K - 1$  stanic je dostupných až po čase  $a_i + 15$ , jinak řečeno  $k - 1$  zákazníků používá v čase  $a_i + 15$  nějakou stanici a na chudáka přichodícího v  $a_i$  by se nedostalo.

Časově jsme na tom o hodně lépe. Cyklus máme pouze jeden přes všechny zákazníky. V každé iteraci můžeme z haldy jednu hodnotu odebrat a jednu přidat, obojí v čase  $\mathcal{O}(\log n)$ . Kolik máme čísel (stanovišť) v haldě zjistíme průchodem v čase  $\mathcal{O}(n)$ . Celková časová složitost i s prvotním tříděním tedy vyjde  $\mathcal{O}(n \log n)$ . V paměti si musíme uchovat časy příchodu a časy dostupnosti stanic, kterých ale nebude nikdy více než zákazníků samotných, tudíž je prostorová složitost  $\mathcal{O}(n)$ .

Úlohu připravili: Jakub Hampl, Adam Jahoda

## Výsledková listina třetí série začátečnické kategorie 38. ročníku KSP

	<i>řešitel</i>	<i>škola</i>	<i>ročník</i>	<i>sérií</i>	<i>Z3-1</i>	<i>Z3-2</i>	<i>Z3-3</i>	<i>Z3-4</i>	<i>série</i>	<i>celkem</i>
0.					8	10	12	14	44,0	132,0
1.	Štěpán Koupil	GDašickáPA	-1	8	8	7	12	12	39,0	126,0
2.	Michal Eliáš	GMLerchaBO	2	3	8	10	12	5	35,0	123,0
3.	Lukáš Stanovský	PORGPha	1	13	8	10	5	9	32,0	120,0
4.	Lukáš Benke	PORGPha	-1	8	8	10	12	10	40,0	118,0
5.	Lukas Hrtan	GArabskáPH	3	3	8	10	12	2,5	32,5	101,5
6.-7.	Julie Klementová	GPísek	4	8	8	10	1	7	26,0	97,0
	Ondra Lupíšek	SPŠSmíchov	2	13	8	10			18,0	97,0
8.	Honza Stanovsky	GZborovPH	1	12	8	10	12		30,0	90,0
9.-10.	Rudolf Krzystek	GNAléjíPH	3	2	8	10	12	9	39,0	83,0
	Lukáš Mihola	GJŠkodyPŘ	1	7	8	8	5	2	23,0	83,0
11.	Vojta Vávra	GTomkovaOL	0	4	8				8,0	82,0
12.	Dexter David Nezveda	MensaG	-3	8	8		1	9,5	18,5	80,0
13.	Kryštof Ham	SPŠEOstrava	1	3	8	10	5	5,5	28,5	78,0
14.	Adéla Kocmanová	GČesLípa	3	3	8	10	12		30,0	70,0
15.	František Nouza	GRNK	4	9	8		12		20,0	68,0
16.	Filip Černý	GFXŠaldyLI	4	6					0,0	67,0
17.	Filip Jančík	GJŠkodyPŘ	1	4	8	10			18,0	64,0
18.	Andrej Fritsch	GKepleraPH	1	2	8	10	1		19,0	63,0
19.-20.	Ondřej Hebek	G VMýto	1	2					0,0	60,0
	Vít Příbyl	GŠpitálsPH	1	2	8	10	12		30,0	60,0
21.	Stanislav Šimeček	GMLerchaBO	2	2					0,0	57,3
22.-23.	Šimon Swaczyna	SlovanGOL	1	3	8		5		13,0	55,0
	Adam Urx	PORGPha	-1	2	8	10	0	1	19,0	55,0
24.	Jindřich Kaplický	GČelákov	4	2					0,0	53,0
25.	Ondřej Hůlka	GOMskáPH	4	3	2,7				2,7	47,7
26.	Jiří Bešta	GHodonín	3	1					0,0	42,0
27.-28.	Markéta Gašová	GBudějovPH	4	8					0,0	40,0
	Jakub Thomitzek	MendelGOP	1	1					0,0	40,0
29.-31.	Matěj Hošek	GVolgogrOS	4	22	8	10			18,0	38,0
	Tomáš Kvapil	PORGPha	0	2	8				8,0	38,0
	Jan Tůma	GSla	4	2					0,0	38,0
32.	Martin Maláč	PORGPha	1	7	1				1,0	37,0
33.	Bruno Ambrož	GZborovPH	2	1					0,0	35,0
34.	Samuel Poláček	GTerVans	3	5					0,0	31,0
35.-36.	Štěpán Hrdý	GLEpařovJČ	4	1					0,0	30,0
	Jan Pavel Škoda	GOpenGaBab	3	5					0,0	30,0
37.	Alexandra Gauchet	GKadaň	4	6					0,0	29,0
38.	Natálie Harvánková	G Chrudim	3	3	8	1	0	4,5	13,5	28,5
39.	Martin Žalud	GFXŠaldyLI	2	1	8	10	9		27,0	27,0
40.	Rozárka Michálková	GČelákov	4	2	8	4,7			12,7	26,7
41.	Matěj Příkryl	G Brandýs	2	3					0,0	26,0
42.	Sára Tyllová	GHermanek	2	3					0,0	24,0
43.	Matouš Skripnik	ZŠGLauder	3	2					0,0	22,0
44.	Lumi Yamori	GFXŠaldyLI	2	1	8	10	3		21,0	21,0
45.-46.	Jakub Binter	GČeskáČB	3	11					0,0	20,0
	Marián Fedorco	GTomkovaOL	3	2					0,0	20,0
47.	Matej Kurbel	G AM Trnava	4	2					0,0	19,0
48.	Patrik Šenkýř	G Sokolov	2	1					0,0	18,7
49.-51.	Filip Dvořák	GMilevsko	3	8					0,0	18,0
	Kateřina Tlustá	ZSHK	4	4					0,0	18,0
	Radek Zach	SPŠSmíchov	4	6					0,0	18,0
52.	Kristýna Pospíšilíková	GPříbor	3	4					0,0	17,0
53.-59.	Filip Adam	GPísnickáPH	1	2					0,0	8,0
	Adam Houdek	SOŠ Březová	1	2					0,0	8,0
	František Hrubý	SPŠSmíchov	3	1					0,0	8,0
	Natálie Jochová	G MNám Třb	4	1	8		0		8,0	8,0
	Eliška Knopfová	1.ITGPH	1	1	8				8,0	8,0
	Vojtěch Pop	GJŠkodyPŘ	1	1	8				8,0	8,0
	Jan Václavek	GJarošeBO	4	3					0,0	8,0

	<i>řešitel</i>	<i>škola</i>	<i>ročník</i>	<i>sérií</i>	<i>Z3-1</i>	<i>Z3-2</i>	<i>Z3-3</i>	<i>Z3-4</i>	<i>série</i>	<i>celkem</i>
60.–61.	Michael Ambros	GTomkovaOL	3	17					0,0	3,0
	Tomáš Diessner	GZborovPH	4	5					0,0	3,0
62.–63.	Zbyněk Míka	GPatočkyPH	–2	1	1				1,0	1,0
	Michal Martínek	GÚstavníPH	4	5					0,0	1,0

