Korespondenční Seminář z Programování

ZAČÁTEČNICKÁ KATEGORIE

33. ročník | **KSP-Z** | **Říjen 2020**

Řešení první série začátečnické kategorie 33. ročníku KSP

33-Z1-1 Kontrola závorkových programů

Pokud by se v Kevinově programu vyskytovaly pouze kulaté závorky, řešení problému by bylo poměrně přímočaré. Mohli bychom procházet program po znacích a průběžně si počítat počet dosud nespárovaných závorek. Na počátku by počet nespárovaných závorek byl roven 0. Vždy, když bychom narazili na otevřenou závorku, přičetli bychom k aktuálnímu počtu nespárovaných závorek jedničku a vždy, když bychom narazili na zavírací závorku, počet bychom o jedna snížili. Snadno nahlédneme, že ve správně uzávorkovaném programu počet nespárovaných závorek v průběhu výpočtu nikdy neklesne pod 0 a na konci výpočtu bude tento počet roven 0, tj. nezbudou nám žádné nespárované závorky.

Problém s tímto postupem nastane v momentě, kdy dovolíme v programu více různých druhů závorek. Potom už nám nestačí počítat otevřené a uzavřené závorky jako v předchozím postupu, ale musíme dbát i na jejich druh. Ve správném uzávorkování totiž nemůžeme spárovat chlupatou závorku s kulatou a podobně. V takovém případě budeme muset použít malinko obecnější postup. Jak na to půjdeme? Tentokrát si nebudeme pamatovat pouze počet nespárovaných závorek, ale budeme si ukládat celou posloupnost nespárovaných závorek, na které narazíme (tj, včetně jejich druhu).

Dokud na vstupu čteme nějakou otevírací závorku, uzávorkování nás netrápí. Ve chvíli, kdy narazíme na nějakou zavírací závorku, potřebujeme ověřit, že poslední přečtená otevírací závorka byla stejného druhu jako právě přečtená zavírací závorka.

K zapamatování si dosud přečtených otevíracích závorek využijeme datovou strukturu zvanou *zásobník*. Zásobník nám umožňuje si věci ukládat a následně je v opačném pořadí odebírat (tj. první odebírám to, co jsem na zásobník uložila jako poslední). Nejjednodušší implementace je asi pomocí dostatečně velkého pole a počítadla, kolik závorek právě v zásobníku máme – toto číslo nám dá index poslední vložené závorky v poli. Při odebírání ze zásobníku přečteme závorku na této pozici a odečteme od počítadla jedničku, při přidávání do zásobníku přičteme k počítadlu jedničku a zapíšeme přidanou závorku na pozici podle počítadla. V některých jazycích na to dokonce existují zkratky, například jako `append` a `pop` v Pythonu.

Pokaždé, když narazím na otevírací závorku, uložíme si ji na vršek zásobníku. Ve chvíli, kdy narazím na zavírací závorku, odeberu ze zásobníku poslední uloženou závorku a zkontroluji, jestli jsou otevírací a zavírací závorka stejného druhu. Pokud mají závorky různý druh, můžu rovnou vytisknout odpověď `NE`, jinak pokračuji dál v kontrole. Posledním krokem je kontrola prázdnosti zásobníku, protože teprve potom víme, že máme správné uzávorkování programu a můžeme tedy vytisknout odpověď `ANO`.

Během odebírání závorek ze zásobníku musíme dávat pozor na situaci, když bychom chtěli odebrat další otevírací závorku, ale v zásobníku by již žádná nebyla. V takovém případě nám ale počet zavíracích závorek překročil počet otevíracích a program tedy není uzávorkovaný správně, takže vypíšeme `NE` a přeskočíme na další řádek vstupu.

Jakou bude mít náš algoritmus časovou složitost? Vzhledem k tomu, že se na každou závorku podíváme nejvýše dvakrát, bude časová složitost lineární vzhledem k délce vstupu.

Program (Python 3):
`http://ksp.mff.cuni.cz/viz/33-Z1-1.py`

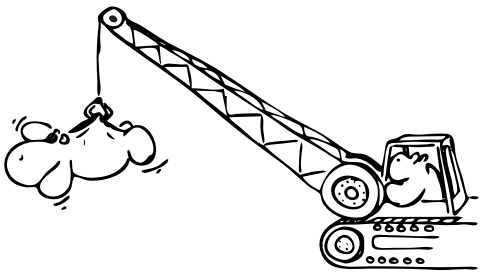
Maruška Kalousková & Klárka Tauchmanová

33-Z1-2 Sobotní den železnice

Nejprve je důležité si uvědomit, že nás vůbec nemusí zajímat, ke kterému vlaku patří daný příjezd/odjezd, ani jaký příjezd a odjezd patří k sobě. Zajímá nás pouze to, že v určitém čase došlo k příjezdu/odjezdu vlaku a tedy že se změnil aktuální počet vlaků vyskytujících se na nádraží.

K tomu se nám bude hodit vyrobit si seznam událostí. Do tohoto seznamu si budeme ukládat jednotlivé časy, kdy došlo k nějaké události (tj. příjezdu či odjezdu vlaku). Protože ale potřebujeme umět rozlišit mezi situací, kdy vlak přijel a kdy odjel, budeme si ke každému času ukládat také druh události. Nejjednodušší bude reprezentovat seznam jako pole, do kterého při čtení vstupu vložíme za každý vlak obě s ním spojené události. Takto vyrobený seznam událostí si následně setřídíme podle času tak, abychom měli události seřazené chronologicky za sebou.

Zjištění aktuálního počtu vlaků na nádraží pak bude probíhat následovně: postupně budeme procházet setříděný seznam událostí a přepočítávat aktuální počet vlaků vyskytujících se na nádraží v daném čase. Na začátku výpočtu bude počet vlaků vyskytující se na nádraží roven 0. Pokaždé, kdy dojde k příjezdu nějakého vlaku, se počet vlaků o jedna zvýší. V případě odjezdu vlaku se počet vlaků naopak o jedna sníží. Protože nás zajímá čas, kdy se na nádraží vyskytuje nejvíce vlaků, budeme si průběžně udržovat dosud nalezené maximum počtu vlaků na nádraží. Vždy, když číslo aktuálního počtu vlaků na nádraží změníme, zjistíme, jestli se náhodou nejedná o adepta na maximum a pokud ano, přepíšeme dosud nalezené maximum za nové maximum.



Jediným háčkem, na který si musíme při počítání dávat pozor, je to, že do fotografie musíme započítat jak vlaky, které v aktuálním čase přijíždějí, tak i vlaky, které v aktuálním čase odjíždějí. Pro každý čas tedy chceme nejprve zpracovat

všechny příjezdy vlaků v daném čase a až poté zpracovávat odjezdy vlaků.

Jaká bude časová složitost našeho algoritmu? Vytvoření seznamu událostí a následné hledání maximálního počtu vlaků umíme v čase lineárním k délce vstupu. Nejvíce času strávíme tříděním seznamu událostí. Pokud použijeme nějaký z rychlejších třídících algoritmů,[1] strávíme tříděním čas $\mathcal{O}(N \log N)$.

Program (Python 3):
`http://ksp.mff.cuni.cz/viz/33-Z1-2.py`

Maruška Kalousková & Klárka Tauchmanová

33-Z1-3 Petrův zmatený výlet

Pro vyřešení úlohy je dobré postupovat od cílového políčka. Zkontrolujeme všechna sousední políčka a každé, jehož rozcestník ukazuje na cíl, si uložíme. Potom projdeme všechny ještě nenavštívené sousedy těchto uložených políček a opět si uložíme ty, jejichž rozcestníky ukazují na uložená políčka. Tímto způsobem postupně prozkoumáme všechna políčka, ze kterých se dá dostat do cíle (protože ukazují na políčko, které ukazuje na políčko ... které ukazuje na cíl).

Na ukládání použijeme datovou strukturu zásobník (stack). Do ní si budeme ukládat všechna políčka, z nichž se dá dostat do cíle a jejichž sousedy jsme zatím neprozkoumali.

Postupovat budeme od cíle. Všechna políčka, která s ním sousedí, prozkoumáme. Pokud se na některém nachází rozcestník ukazující na cíl, tak ho uložíme do zásobníku. Každé políčko, které vložíme do zásobníku, si označíme na mapě znakem `#` (protože víme, že se z něj dá dojít do cíle).

Potom už jen tento postup zopakujeme pro všechna políčka, která vyjmeme ze zásobníku. Zkontrolujeme všechny sousedy (kromě těch, kteří už jsou označeni znakem `#`). Pokud se na sousedním políčku nachází rozcestník ukazující na aktuální políčko, tak to sousední políčko uložíme do zásobníku a označíme ho `#`.

Až bude zásobník prázdný, stačí nám jen znovu projít celou mapu, všechna písmena SJVZ změnit na znak `.` (tečka) a tuto mapu vypsat.

Algoritmus prochází vstupní mapu, každé políčko je do zásobníku vloženo maximálně jednou. Některá políčka mohou být navštívena až čtyřikrát (můžeme se na ně podívat až ze čtyř sousedů). Celkově bude časová složitost lineární vzhledem k velikosti mapy na vstupu – $\mathcal{O}(N)$ kde $N = R \cdot S$.

Program (Python 3):
`http://ksp.mff.cuni.cz/viz/33-Z1-3.py`

Lucka Vomelová

33-Z1-4 Zuzka a černé kočky

Zuzčina mapa popisuje neorientovaný ohodnocený graf. K vyřešení naší úlohy to není potřeba vědět, ale pokud grafy neznáte, určitě doporučujeme si o nich něco zjistit, např. v naší kuchařce o grafech.[2] Postupu, který použijeme, se říká prohledávání do šířky a kuchařka se jím také zabývá.

Všimněme si, že parita (lichost) počtu černých koček, na které jsme natrefili, se mění jen při průchodu ulicí s lichým počtem koček. Použijeme drobný trik.

Představíme si, že každá křižovatka má dvě verze, sudou a lichou. Podle toho mapku pomyslně „překreslíme“: budeme mít dvojnásobný počet křižovatek i ulic. Ulice se sudým počtem koček překreslíme na jednu ulici mezi sudými verzemi původních křižovatek a druhou ulici mezi těmi lichými. „Lichá“ ulice se přemění na dvě ulice vedoucí vždy z liché verze jedné křižovatky do sudé verze druhé. V takto upravené mapě se chceme dostat ze sudé verze startovní křižovatky do sudé verze té cílové.

Pro každou vzniklou křižovatku si navíc uložme, jakou ulicí do ní umíme ze startu přijít, abychom později uměli trasu zrekonstruovat. To můžeme zapsat například do pole. Ze začátku budou tyto záznamy prázdné. Teď si pořiďme frontu a vložme do ní pro začátek jen sudou verzi startovní křižovatky.

Dokud fronta není prázdná, odebereme z ní křižovatku, která je na řadě a podíváme se na všechny křižovatky, do kterých odtud vede ulice. Ty, které jsme zatím nenavštívili, přidáme do fronty a nastavíme, že aktuální ulice je ta, po které se sem dá dostat. Takovým postupem zajistíme, že navštívíme všechny dostupné křižovatky. Když narazíme na sudou verzi cílové křižovatky, můžeme vypsat, kudy jsme se tam dostali a máme vyhráno. Pokud se fronta vyprázdní aniž bychom cíl našli, kýžená cesta neexistuje.

Jak vypsat nalezenou trasu? Kdybychom postupovali jednoduše z cílové křižovatky podle jejích záznamů, dostaneme trasu pozpátku. Chceme-li normální popis trasy, můžeme použít zásobník, na který postupně křižovatky obrácené trasy vložíme a poté odebíráním křižovatek získat skutečnou trasu ve správném pořadí (tzn. při praktickém programování využít rekurzivní funkci, která vysleduje trasu pozpátku a při návratu vypisuje křižovatky). Nebo si můžeme k ulicím, po kterých jsme přišli pamatovat také délku dosavadní trasy, a poté celou trasu zapsat do pole příslušné velikosti.

Dá se také nahlédnout, že cesta, kterou takto najdeme, je ta nejkratší možná (alespoň co do počtu prošlých ulic). To ale není v úloze hlavní a formálnější důkaz této skutečnosti naleznete ve zmíněné kuchařce.

Každou ulici i křižovatku navštívíme maximálně jednou, jen je jich dvojnásobek těch původních. Složitost tak vyjde $\mathcal{O}(2K + 2U) = \mathcal{O}(K + U)$, kde K značí počet křižovatek a U počet ulic. I rekonstrukce cesty se do tohoto času vejde, máme tak lineární algoritmus vůči délce vstupu.

Martin Koreček

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

Výsledková listina první série začátečnické kategorie 33. 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.–3.	Robert Gemrot	GKomHavíř	4	10	8	10	12	14	44,0	44,0
	Robert Jaworski	GÚstavníPH	3	18	8	10	12	14	44,0	44,0
	Jakub Ondroušek	GTomkovaOL	1	9	8	10	12	14	44,0	44,0
4.	Dominik Farhan	GMikulášPL	4	4	8	10	12	13,5	43,5	43,5
5.–8.	Jan Hlavsa	GMělník	3	1	8	10	12	4	34,0	34,0
	Kryštof Maxera	GJírovcČB	0	3	8	10	12	4	34,0	34,0
	Zdeněk Pezlar	GJarošeBO	3	1	8		12	14	34,0	34,0
	Vojtěch Venzara	GMělník	3	1	8	10	12	4	34,0	34,0
9.–11.	Vojtěch Gaďurek	PORGPha	4	1	8	9	12	3	32,0	32,0
	Veronika Jůzková	MensaG	3	5	8	10	12	2	32,0	32,0
	Pavel Šrytr	GMělník	4	1	8	10	12	2	32,0	32,0
12.	Tomáš Kašpárek	G FrýdlNOs	3	1	4	10	12	4	30,0	30,0
13.	Adam Húšťava	EupSchoolLux	3	12	8	7	12		27,0	27,0
14.–16.	Jonáš Dej	G Wicht	2	1	8	10	8		26,0	26,0
	Tomáš Janovec	GMnichHrad	3	2	8	10	8		26,0	26,0
	Thomas Riedle	BRG APP	2	5	8	2	12	4	26,0	26,0
17.	Jonáš Bína	ZŠŠtáflovaHB	−4	2	8	2	12		22,0	22,0
18.	Jan Kotyk	G Kolín	4	1	8		12		20,0	20,0
19.	Yahor Herashchanka	ZŠ Turnov	0	1	8	4,7	4	3	19,7	19,7
20.–23.	Klára Grinerová	GZborovPH	4	1	8	2	8		18,0	18,0
	Tomáš Chabada	SPŠEMasLI	4	5	8	10			18,0	18,0
	Lukáš Létal	GJŠkodyPŘ	2	1	8	10			18,0	18,0
	Erik Sabol	GČeskoliPH	1	4	8	2	8		18,0	18,0
24.–25.	Martin Fof	MendelGOP	3	1	8	2	4		14,0	14,0
	Honza Kocourek	ParkLane	1	1		10	4		14,0	14,0
26.	Ondřej Piroutek	GČeskoliPH	3	1	8	2		1	11,0	11,0
27.–29.	Daniel Mencl	GMělník	3	1	8	2			10,0	10,0
	Jan Najman	SPSEPard	4	8	8	2			10,0	10,0
	Jakub Nevařil	G UherBrod	3	11	8	2			10,0	10,0
30.	Radek Bláha	GČeskáČB	−1	1	8	1			9,0	9,0
31.	Jan Machourek	GBBr	0	3	8				8,0	8,0
32.	Matěj Hošek	GVolgogrOS	−1	1	7	0			7,0	7,0
33.	Martin Bulíř	SPŠEMasLI	4	1	2	4			6,0	6,0
34.	Matěj Strnad	ZŠRiegraSM	0	3	2	2			4,0	4,0
35.–36.	Olga Cinková	ArcibisGPH	1	1		2			2,0	2,0
	Antonín Musil	PORGPha	4	3		2			2,0	2,0
37.	Marek Maškarinec	GFXŠaldyLI	0	1	1				1,0	1,0