Korespondenční Seminář z Programování

ZAČÁTEČNICKÁ KATEGORIE

33. ročník **KSP-Z** **Prosinec 2020**

Řešení druhé série začátečnické kategorie 33. ročníku KSP

33-Z2-1 Stánkař

Co je na úloze se stánkařem a jeho nedočkavými zákazníky hezké, je to, že jde řešit několika různými jednoduchými způsoby, a ač to všechno budou simulace („přehrajeme si“, co se u stánku bude dít), tak podstatou vycházejí z různých myšlenek.

Prvním způsobem, který vám chceme předvést, je **simulace podle času**. Pořídíme si cyklus, ve kterém budeme skákat po celých minutách od času 0 dál a při tom budeme zkoumat, co se v ten čas děje se zákazníky (mohli bychom tento postup nazvat i více matematicky *iterací podle času*).

Zákazníky si můžeme na začátku načíst do pole (abychom se mohli průběžně dívat na čas dalšího zákazníka a porovnávat ho s časem simulace) a současně si budeme v jiném poli držet frontu zákazníků stojících u stánku. V každém kroku simulace potřebujeme:

- Přidat do fronty zákazníky, kteří přišli v tento čas,
- odstranit z fronty všechny zákazníky, kteří již čekali příliš dlouho a odešli,
- obsloužit jednoho zákazníka (pokud je fronta zákazníků u stánku neprázdná),
- zvednout čas o 1 a pokračovat v dalším kroku simulace.

Při programování simulace je dobré si rozmyslet, kdy chceme simulaci ukončit a vydat výsledek. V našem případě to bude, když už nám nebudou zbývat žádní zákazníci (ani v poli zákazníků, ani ve frontě).

Taková simulace se dá naprogramovat celkem jednoduše a na naši úlohu s přehledem stačí. Problém by ale mohla mít ve chvíli, kdyby mezi zákazníky byly obrovské mezery a simulace by musela udělat miliony kroků „naprázdno“.

Druhým způsobem, který ukážeme, je **simulace iterující přes zákazníky**. Jak už její název napovídá, tak nebude „skákat“ po jednotlivých minutách a během toho obsluhovat zákazníky, ale bude „skákat“ po zákaznících a během toho kontrolovat čas.

Bude nám stačit držet si nejdřívější čas, ve kterém můžeme obsloužit dalšího zákazníka (což na začátku bude čas 0). Pak v každém kroku simulace načteme dalšího zákazníka v pořadí. Pokud bude nejbližší čas obsloužení dalšího zákazníka…

- … dříve, než přijde tento zákazník ke stánku, tak ho určitě zvládneme obsloužit a nejdřívější čas obsloužení dalšího zákazníka nastavíme na minutu po příchodu tohoto.
- … méně než 10 minut po příchodu tohoto zákazníka, tak ho také stihneme obsloužit a nejdřívější čas obsloužení dalšího zákazníka zvedneme o jednu minutu.
- … větší nebo rovno 10 minutám po příchodu tohoto zákazníka, tak ho obsloužit nezvládneme, zákazník odejde z fronty a my budeme pokračovat s dalším zákazníkem.

Toto řešení má o něco složitější myšlenku, na implementaci je však jednodušší, než to první. Stále se jedná o simulaci, ale už mu nebudou dělat problémy velké mezery (a zvládne tak vždy odpovědět v lineárním čase vzhledem k velikosti vstupu.

Program (Python 3 – simulace podle času):
`http://ksp.mff.cuni.cz/viz/33-Z2-1-cas.py`

Program (Python 3 – iterace přes zákazníky):
`http://ksp.mff.cuni.cz/viz/33-Z2-1-zakaznici.py`

Úlohu připravili: Jirka Setnička, Vašek Šraier, Eliška Vítková

33-Z2-2 Kulinářský hlavolam

Pojďme nejprve rozmyslet přímočaré řešení – vyzkoušíme všechny možnosti, jak rozestavět hrnce na plotýnky, a z nich vybereme tu, která splňuje všechny požadavky. Takový přístup jistě funguje, ale samotných rozestavění je až $N!$, což už je příliš. A to ani neuvažujeme čas potřebný k ověření validity rozestavění.

Nemůžeme to nějak zrychlit? Pro každý hrnec je přece snadné určit všechny plotýnky, na kterých stát nesmí – to jsou ty menší než on sám a ty, na kterých už nějaký hrnec stojí. Jak si ale vybrat, na kterou použitelnou plotýnku hrnec umístit, aby nám tam později nepřekážel? Snadno, nalezneme nejmenší, která postačuje. Hrnce, které jsou větší než ploténka, by na ni stejně nepasovaly a těm menším jsme zase nijak neuškodili – ty lze postavit i na jakoukoli větší. Nápad můžeme také převrátit a ke každé plotýnce hledat co největší hrnec, který se na ni vejde.

Jak dlouho běží tento algoritmus? Pro každý hrnec zkusíme projít všechny plotýnky (nebo naopak), na což potřebujeme $\mathcal{O}(NK)$ kroků. To je mnohem lepší než naše původní řešení, ale stále může být příliš pomalé pro dlouhé vstupy.

Pojďme zrychlit tu variantu, kdy na každou plotýnku stavíme největší možný hrnec. Hrozně moc času strávíme hledáním správného hrnce. To je pochopitelné, když přicházejí plotýnky libovolně. Ani pořadí, ve kterém vybíráme hrnce, nemá moc řád, ale s tím se dá něco dělat. Pojďme si seřadit plotýnky od největší k nejmenší a v tom pořadí je procházet. Teď už vždy hledáme největší hrnec velikostí menší roven předchozímu. Ale protože předchozí jsme už použili, je to vlastně vždy největší nepoužitý hrnec.

Jak toho využít? Seřaďme si i hrnce od největšího k nejmenšímu. Teď už je řešení snadné. Vezměme vždy první hrnec, postavme jej na první plotýnku a následně oba odeberme (nebo prostě přeskočme). Pokud náhodou narazíme na dvojici, která na sebe nepasuje, víme, že to určitě ani jinak nejde a úloha nemá řešení. Takový případ nám dokonce zadání nepovoluje.

Jak dlouho to nakonec trvá? Nejrychlejší řadící algoritmy, které známe a dají se použít na čísla libovolného rozsahu,

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

zaberou $\mathcal{O}(N \log N)$ času. Podrobnosti lze najít např. v naší kuchařce.[1] V našem případě to udělá $\mathcal{O}(N \log N + K \log H)$, což, protože K je nejvýše N, odpovídá $\mathcal{O}(N \log N)$.

Program (Python 3):
`http://ksp.mff.cuni.cz/viz/33-Z2-2.py`

Úlohu připravili: Vojta Káně, Jirka Sejkora

33-Z2-3 Rostoucí funkce

Představme si, jak bychom tento problém řešili ručně, bez počítače. Můžeme si prostě tipnout nějaké číslo x a spočítat si funkční hodnotu $f(x)$ v tomto bodě. Pokud bude hledaný výsledek větší než toto $f(x)$, musíme x trochu zvýšit. Stejně tak i naopak, pokud jsme výsledek přestřelili, musíme si tipnout menší x. Tento postup samozřejmě funguje jen v případě, že je funkce rostoucí. To však máme v zadání slíbeno.

Podle čeho se budeme rozhodovat, jaké x si tipnout? Na začátku máme dolní odhad 1 a horní odhad 100. Zkusme si tipnout x uprostřed tohoto úseku. Tento střed spočítáme jako aritmetický průměr obou odhadů. Pokud zjistíme, že hledaná hodnota je větší než $f(x)$, víme, že x musí být větší než střed. Díky tomu můžeme použít tento střed jako svůj vylepšený dolní odhad. Obdobně, pokud by byla hledaná hodnota menší než $f(x)$, použijeme střed jako nový horní odhad.

Tento krok opakujeme a postupně vylepšujeme své dolní a horní odhady. Můžeme si všimnout, že při každém kroku se rozdíl mezi nimi zmenší na polovinu. Díky tomu budou naše tipy přesnější a přesnější. Jakmile si tipneme takové x, že se $f(x)$ od hledané hodnoty liší o méně než setinu, máme vyhráno.

Pozor na to, že zaokrouhlujeme pouze $f(x)$. Samotné x zaokrouhlovat nesmíme, protože jej mnohdy potřebujeme určit přesněji. Hodnota $f(x)$ se totiž může změnit o více než jednu setinu i v případě, že se x změní jen maličko. Pro polynom z úlohy například platí $f(25) = 50780$ a $f(25.0001) \doteq 50780.67$.

Zmíněnému přístupu se říká binární vyhledávání.[2] Lze jej použít, pokud vstup vždy roste nebo klesá – například k hledání v setříděném poli. Jeho časová složitost je logaritmická vzhledem k rozsahu čísel, který musíme prohledat.

Výše zmíněné řešení bylo naprosto dostačující. Ještě ale nastíníme jedno možné vylepšení: Při ručním hledání máme určitý pojem o tom, jak blízko výsledku jsme. Pokud vyšel dolní odhad jenom trochu mimo, zatímco horní odhad je od výsledku daleko, bylo by lepší zkoušet tipovat čísla blíže dolnímu odhadu. Jak tohle říct programu? Využijeme lineární interpolace. To znamená, že si představíme, že mezi oběma odhady má funkce podobu úsečky. Pro úsečku, respektive přímku, už umíme k dané hodnotě zpětně dopočítat x. Takto spočítané x použijeme jako nový tip. Bohužel, pro některé funkce by tento postup mohl být příliš pomalý, proto se vyplatí střídat takovéto tipy a dělení úseku napůl pomocí binárního vyhledávání.

Program (Python 3):
`http://ksp.mff.cuni.cz/viz/33-Z2-3.py`

Úlohu připravili: David Klement, Michal Kodad, Martin „Medvěd“ Mareš, Kuba Pelc, Jirka Setnička

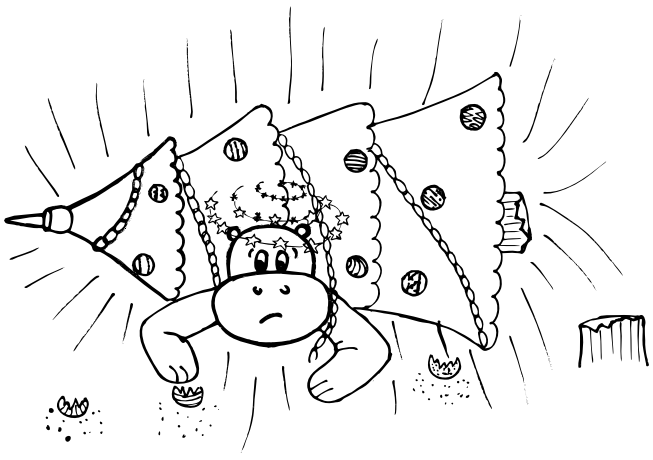
33-Z2-4 Měření plachty

Pro každé dva stěžně je plocha plachty mezi nimi jednoznačná. Zkusme nejdříve nastavit levý i pravý stěžeň na ty krajní a uložme si je jako zatím nejlepší řešení. Vzdálenost mezi nimi je největší možná, takže najdeme-li větší plachtu, bude určitě vyšší. Proto vyšší ze stěžňů ponechme a posuňme se směrem k němu od toho nižšího (pokud mají oba stejnou výšku, můžeme klidně posunout obě pozice). Argument o zvyšování plachty stále funguje a proto můžeme tento krok opakovat. Pokaždé přitom změníme záznam nejlepšího řešení, pokud jsme našli lepší.

Jakmile má náš levý stěžeň stejnou pozici jako pravý, skončíme a vrátíme uložené nejlepší řešení. Díky našemu argumentu přitom máme jistotu, že jsme nezanedbali lepší možnost.

V každém kroce jen zjistíme plochu, která odpovídá aktuálním pozicím stěžňů, a porovnáme ji s dosud nejlepším výsledkem, což trvá konstantní čas. Vzdálenost mezi stěžni se přitom mezi kroky vždy o jedna sníží, kroků tak nebude více, než je stěžňů. Algoritmus tedy poběží lineárně dlouho vůči velikosti vstupu.

Úlohu připravili: Martin Koreček, Tom Sláma



[2] `http://ksp.mff.cuni.cz/viz/kucharky/binarni-vyhledavani`

Výsledková listina druhé série začátečnické kategorie 33. ročníku KSP

	řešitel	*škola*	*ročník*	*sérií*	*Z2-1*	*Z2-2*	*Z2-3*	*Z2-4*	*série*	*celkem*
0.					8	10	12	14	44,0	88,0
1.–2.	Robert Jaworski	GÚstavníPH	3	19	8	10	12	14	44,0	88,0
	Jakub Ondroušek	GTomkovaOL	1	10	8	10	12	14	44,0	88,0
3.	Dominik Farhan	GMikulášPL	4	5	8	10	12	9	39,0	82,5
4.	Robert Gemrot	GKomHavíř	4	11	8	10	12		30,0	74,0
5.	Jonáš Dej	G Wicht	2	2	8	10	12	14	44,0	70,0
6.–7.	Jan Hlavsa	GMělník	3	2	8	10	12	5	35,0	69,0
	Kryštof Maxera	GJírovcČB	0	4	8	10	12	5	35,0	69,0
8.	Vojtěch Venzara	GMělník	3	2	8	10	12		30,0	64,0
9.	Pavel Šrytr	GMělník	4	2	8	10	12		30,0	62,0
10.–11.	Tomáš Janovec	GMnichHrad	3	3	8	10	12	5	35,0	61,0
	Thomas Riedle	BRG APP	2	6	8	10	12	5	35,0	61,0
12.	Veronika Jůzková	MensaG	3	6	8	7	12		27,0	59,0
13.	Erik Sabol	GČeskoliPH	1	5	8	10	12	5	35,0	53,0
14.	Klára Grinerová	GZborovPH	4	2	8	10	12		30,0	48,0
15.–16.	Lukáš Létal	GJŠkodyPŘ	2	2	8	10	9		27,0	45,0
	Daniel Mencl	GMělník	3	2	8	10	12	5	35,0	45,0
17.	Martin Fof	MendelGOP	3	2	8	10	12		30,0	44,0
18.	Yahor Herashchanka	ZŠ Turnov	0	2	2	7	9	5	23,0	42,7
19.–20.	Štěpán Filipčík	GTomkovaOL	3	1	8	10	12	5	35,0	35,0
	Jakub Smolik	GEBenešeKL	3	1	8	10	12	5	35,0	35,0
21.	Zdeněk Pezlar	GJarošeBO	3	2	0				0,0	34,0
22.	Vojtěch Gaďurek	PORGPha	4	1					0,0	32,0
23.–26.	Jonáš Bína	ZŠŠtáflovaHB	−4	3	8				8,0	30,0
	Tomáš Kašpárek	G FrýdlNOs	3	1					0,0	30,0
	Adam Kolník	SSŠVTPraha	2	1	8	10	12		30,0	30,0
	Vít Novotný	GMělník	3	1	8	10	12		30,0	30,0
27.	Ondřej Piroutek	GČeskoliPH	3	2	6		12		18,0	29,0
28.–30.	Matouš Bohoněk	GMělník	3	1	8	7	12		27,0	27,0
	Adam Húšťava	EupSchoolLux	3	12					0,0	27,0
	Michal Žáček	MensaG	4	1	8	5	12	2	27,0	27,0
31.	Matěj Strnad	ZŠRiegraSM	0	4	8		12		20,0	24,0
32.–33.	Pavel Altmann	GMikulášPL	2	4	8	10	2		20,0	20,0
	Jan Kotyk	G Kolín	4	1					0,0	20,0
34.	Tomáš Chabada	SPŠEMasLI	4	5					0,0	18,0
35.	Honza Kocourek	ParkLane	1	1					0,0	14,0
36.	Olga Cinková	ArcibisGPH	1	2	2	7			9,0	11,0
37.–39.	Radek Bláha	GČeskáČB	−1	2	1				1,0	10,0
	Jan Najman	SPSEPard	4	8					0,0	10,0
	Jakub Nevařil	G UherBrod	3	11					0,0	10,0
40.–41.	Jan Machourek	GBBr	0	3					0,0	8,0
	Jan Soukup	GJiříPoděb	3	1	8				8,0	8,0
42.	Matěj Hošek	GVolgogrOS	−1	1					0,0	7,0
43.	Martin Bulíř	SPŠEMasLI	4	1					0,0	6,0
44.	Marek Maškarinec	GFXŠaldyLI	0	2	2				2,0	3,0
45.	Antonín Musil	PORGPha	4	3					0,0	2,0
46.	Vojtěch Čermák	SPŠEMasLI	2	3	1				1,0	1,0