Korespondenční Seminář z Programování

ZAČÁTEČNICKÁ KATEGORIE

31. ročník | KSP-Z | Prosinec 2018

Řešení první série začátečnické kategorie 31. ročníku KSP

31-Z1-1 Zuzka a poník

Měli jste za úkol vypsat v sekundách délku každého kola, které Zuzka oběhla s poníkem. Délka jednoho kola je čas, který uplyne mezi dvěma stisknutími stopek: časy stisknutí v digitálním formátu dostal Váš program na vstupu. Nejjednodušším přístupem k řešení bylo si převést digitální formát na počet sekund od začátku měření: v takovémto formátu už je možné dva časy stisknutí stopek od sebe odečíst a získat čas kola.

O něco komplikovanější bylo přijít na to, jak správně (a jednoduše) převést formát stopek na počet sekund. To se může lišit v závislosti na programovacím jazyku, který používáte.

V oblíbeném Pythonu je třeba použít funkci `input` pro čtení řádku a získaný řetězec rozdělit podle dvojteček zavoláním funkce `split(':')`. Například pokud je na vstupu řádek `00:03:13`, tak zavolání `input().split(':')` vrátí tříprvkové pole `['00', '03', '13']`. Prvky pole jsou ale zatím řetězce, proto je třeba každý z nich převést na číslo pomocí funkce `int`. Teprve potom získáme počet hodin, minut a sekund na stopkách, s nímž můžeme pracovat dál.

V jazyce C a podobných nízkoúrovňových jazycích se s řetězci většinou nepracuje snadno. V Céčku nás ale zachrání funkce `scanf`, díky které můžeme ze vstupu přímo dostat konkrétní počty hodin, minut a sekund. Pokud si například zadefinujeme celočíselné proměnné `hod`, `min` a `sec`, můžeme voláním `scanf("%d:%d:%d", &hod, &min, &sec)` do nich přímo načíst správná čísla. Všimněte si, že první parametr funkce popisuje formát řádku (`%d` vyjadřuje, že na jeho místě se nachází celé číslo).

Ať už správná čísla získáme jakkoliv, získat počet sekund od začátku měření je snadné. Stačí sečíst počet sekund na stopkách, počet minut vynásobený šedesáti a počet hodin vynásobený 3 600.

Celý program pak probíhá následovně. Nejprve si načteme počet kol N. Potom provádíme N cyklů, kdy v každé iteraci načteme čas stopek, převedeme ho na počet sekund od počátku měření, odečteme od počtu sekund v předešlém cyklu a rozdíl vypíšeme. Jenom v první iteraci použijeme jako předchozí čas nulu, protože první kolo začalo na počátku měření (na stopkách byl nulový čas).

Zbývá určit časovou a paměťovou složitost celého algoritmu. Časová složitost je $\mathcal{O}(N)$, protože provádíme N cyklů a čas na zpracování jednoho řádku je konstantní (všechny řádky na vstupu mají omezenou délku). Paměťová složitost programu je ale konstantní – uvědomte si, že vstup čteme postupně, nemáme důvod uchovávat všechny časy kol v paměti.

Přikládáme dva zdrojové kódy v Pythonu. První program je delší a podrobnější, druhý využívá různých vlastností Pythonu ke zkrácení kódu.

Program (Python 3, podrobný):
`http://ksp.mff.cuni.cz/viz/31-Z1-1-delsi.py`

Program (Python 3, kratší):
`http://ksp.mff.cuni.cz/viz/31-Z1-1-kratsi.py`

Program (C):
`http://ksp.mff.cuni.cz/viz/31-Z1-1.c`

Kuba Maroušek

31-Z1-2 Ukradený jezdec

Hledání nejkratší cesty figurky na šachovnici je ve skutečnosti hledání nejkratší cesty v grafu. Pokud jste se s grafy ještě nepotkali, doporučujeme se s nimi seznámit v naší kuchařce,[1] určitě se vám ještě někdy budou hodit.

Můžeme si představit, že každé políčko šachovnice je vrcholem a z každého vrcholu vede až T hran do jiných – každý povolený tah je jedna hrana z každého vrcholu. Pozor, tyto hrany jsou orientované, tedy pokud vede hrana z jednoho políčka do druhého, nemusí nutně vést i opačným směrem! Příkladem jsou třeba povolené tahy pěšáka, který se smí pohybovat pouze dopředu, nikdy se nevrací.

Pro řešení úlohy se hodí použít prohledávání do šířky. Algoritmus bude obsahovat frontu políček, které má zpracovat, a tu budeme pomocí smyčky procházet. Na počátku bude ve frontě pouze startovní políčko. V každé iteraci smyčky odebereme první políčko z fronty a z něj zkusíme skočit na jiné políčka pomocí každého povoleného tahu. Pokud nějaký skok, který jsme vyzkoušeli, vede na políčko, které jsme ještě nikdy nenavštívili, tak toto výsledné políčko přidáme na konec fronty.

Abychom z běhu funkce získali potřebnou cestu, vytvoříme si také pole velikosti $N \times N$, kde budou pro každé políčko uloženy souřadnice políčka, z kterého jsme na něj skočili. Toto pole si také bude pro každé políčko pamatovat, zda jsme ho už navštívili: pokud ne, tak pro toto políčko bude v poli uložena nějaká speciální hodnota, pokud ano, tak zde budou souřadnice výchozího políčka skoku.

Pokaždé, když provedeme nějaký skok a přidáme ho do fronty, tak také zapíšeme do tohoto pole souřadnice počátečního políčka skoku na index koncového políčka skoku. Po nalezení nějaké cesty ji pomocí tohoto pole umíme rekonstruovat prostě tak, že se podíváme na výchozí místo tahu, z kterého jsme se dostali na cíl, pak na výchozí místo tohoto tahu a tak dále, dokud takto nedojdeme na počáteční místo figurky.

Toto řešení bude vracet nejkratší cesty, bude ovšem i rychlé? Pojďme se zamyslet nad obsahem fronty: na začátku bude obsahovat jediné políčko (start), na které známe nejkratší cestu délky 0. Z tohoto políčka skočíme a přidáme do fronty nějaký počet dalších políček, na která nyní známe cestu délky 1. Rychlejší cesta na ně pochopitelně existovat nebude a taky nebudou existovat žádná jiná políčka,

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

na která by se dalo dostat cestou délky 1. Nyní zpracujeme všechna tato políčka s cestou délky 1 a vygenerujeme nová s cestou délky 2. Opět můžeme nahlédnout, že na tato políčka rychlejší cesta neexistuje, protože bychom na ně museli skočit z políčka, na které se umíme dostat pomocí cesty délky nejhůře 0 (aby byla cesta skutečně rychlejší, musí mít délku nejvýše 1), a všechny možné skoky z políček s cestou délky 0 už jsme prošli. Obecně, kdykoliv skočíme na ještě nenavštívené políčko, doskákali jsme na něj nejkratší možnou cestou délky D, protože v předchozím kroku jsme už doskákali na všechna možná políčka, na které se lze dostat nejkratší cestou délky $D-1$.

Toto řešení tedy bude vracet nejkratší cestu a bude i rychlé. Každé políčko totiž navštívíme pouze jednou, tudíž zpracujeme nejhůře $N \times N$ políček.

Proč nepoužít prohledávání do hloubky?

Kromě prohledávání do šířky můžeme procházet grafem také pomocí prohledávání do hloubky, neboli užitím rekurze. Pro tuto úlohu rekurze k dostatečně rychlému řešení nepovede. Pojďme se podívat proč.

Vyjdeme z implementace z kuchařky, která navštívené vrcholy označuje a už se na ně nikdy nevrací. Tento algoritmus prostě zastavíme, jakmile narazíme na políčko cíle, a vrcholy ze zásobníku ve správném pořadí vypíšeme. Toto sice bude validní cesta, nicméně nebude to vždy cesta nejkratší, ale prostě ta první cesta, na kterou algoritmus narazí.

Řešení nyní upravíme takto: místo binární informace, jestli jsme už políčko navštívili, si budeme pamatovat pro každé políčko délku nejkratší cesty ze startu na toto políčko. V algoritmu si budeme pamatovat délku cesty na aktuální vrchol (třeba spolu s každým vrcholem v zásobníku). Vždy, když chceme zkusit do zásobníku přidat nějaký vrchol, tak se podíváme na tuto informaci a vrchol přidáme jen tehdy, pokud je současná nejkratší cesta na tento vrchol delší než cesta, kterou bychom nyní vytvořili.

Toto řešení bude správně vracet nejkratší cestu, ale bude příšerně pomalé. Má povoleno každý vrchol navštívit vícekrát, takže když bude mít obzvláště smůlu, může projít až exponenciálně mnoho (v závislosti na počtu vrcholů) různých cest, než nalezne tu nejkratší cestu do cíle. Toto rekurzivní řešení je tedy mnohem pomalejší než naše lineární řešení používající prohledávání do šířky.

Program (Python 3):
`http://ksp.mff.cuni.cz/viz/31-Z1-2.py`

Kuba Pelc

31-Z1-3 Průnik kvádrů

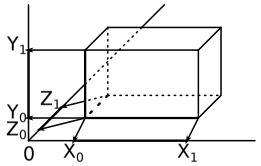
V této úloze jsme dostali za úkol spočítat obsah průniku kvádrů. Pro začátek si zjednodušme zadání a pojďme si nejprve rozmyslet, jak spočítat průnik dvou uzavřených intervalů. Označíme si proto první interval $[a_1, b_1]$ a druhý $[a_2, b_2]$.

Průnik těchto intervalů, pokud se protínají, bude jistě opět interval, jejž označíme $[a, b]$. Jaká bude hodnota a? Aby mohl průnik začít, musí taktéž začít oba základní intervaly. To znamená, že jistě $a \geq a_1, a \geq a_2$. Navíc tento interval začne v jednom z těchto bodů. Podobně pro konec průniku b. Ale co když se intervaly nijak neprotínají? To může nastat jen v případě, když jeden z intervalů skončí dříve, než druhý začne.

Tento způsob můžeme jednoduše rozšířit na zjištění průniku n intervalů. Pokud máme intervaly $[a_1, b_1], [a_2, b_2], \ldots, [a_n, b_n]$, stejnou úvahou jistě bude pro jejich průnik platit $a \geq a_i$ pro $1 \leq i \leq n$. Stejně pro b. Taktéž bude fungovat i situace, když je průnik prázdný.

Dostali jsme tak způsob, jak spočítat průnik n intervalů. Nejprve spočítáme $a = \min(a_1, a_2, \ldots, a_n)$, stejně tak $b = \max(b_1, b_2, \ldots, b_n)$. Pokud zjistíme, že $a > b$, prohlásíme průnik za prázdný.

Nyní využijme toho, že umíme spočítat průnik intervalů k tomu, abychom spočítali průnik kvádrů. Tak, jak máme kvádry zadané, jsou všechny rovnoběžné vůči sobě a jejich průnik je zase kvádr rovnoběžný s ostatními. Dále si vyzkoušejme, že „stlačením“ kvádru na přímku rovnoběžnou s jednou jeho hranou dostaneme úsečku shodnou s tou hranou. Když toto uděláme pro všechny kvádry, jejich průnikem bude úsečka – hrana průniku. Díky tomu můžeme průnik kvádru najít pro každou stranu zvlášť a výsledky zkombinovat zpět do kvádru.



To umíme vyřešit právě přes průnik intervalů – každá hrana je určena intervalem! Po spočítání tak dostaneme trojici intervalů $[X_0, X_1], [Y_0, Y_1], [Z_0, Z_1]$. Ta nám určuje kvádr, jenž má levý spodní přední bod na souřadnicích (X_0, Y_0, Z_0) a pravý horní zadní bod na souřadnicích (X_1, Y_1, Z_1).

Zbývá spočítat objem tohoto kvádru. To je ale jednoduché – intervaly nám prozrazují jednotlivé délky stran, což je rozdíl koncového a počátečního bodu. Tudíž $V = (X_1 - X_0) \cdot (Y_1 - Y_0) \cdot (Z_1 - Z_0)$. Vše zkombinujeme a dostáváme správný algoritmus, který běží v čase $\mathcal{O}(n)$, kde n je počet všech kvádrů.

Program (Python 3):
`http://ksp.mff.cuni.cz/viz/31-Z1-3.py`

Vašek Končický

31-Z1-4 Piškvorky naslepo

Jako první by nás mohlo napadnout udržovat si hrací plochu jako dvojrozměrné pole T, kde na každé pozici $T[i][j]$ je znak, který se právě vyskytuje na odpovídajícím políčku v hrací ploše; pro značení prázdného políčka můžeme používat třeba znak mezery. Každý tah pak jednoduše odsimulujeme: zapíšeme na příslušnou pozici příslušný znak a pak zkontrolujeme, zda náhodou hráč, který je právě na tahu, nevyhrál.

Jak ale pole T sestrojit, když je plocha, na jaké hráči hru hrají, neomezená? Plocha sice neomezená je, ale určitě se nic nestane, když budeme uvažovat jen nejmenší obdélníkovou oblast, do které někdo zahrál. Můžeme si tedy všechny dotazy uložit někam do pole a předtím, než je zpracujeme, si nejdřív spočítat největší řádek R a sloupec S, na které jeden z hráčů zahraje, a vytvořit pole odpovídající velikosti a vyplnit ho mezerami.

Zbývá umět kontrolovat, že jeden z hráčů vyhrál. Nejjednodušší je projít celou tabulku ve všech čtyřech směrech (tj. po řádcích, sloupcích, a obou diagonálách) a průběžně

	řešitel	*škola*	*ročník*	*sérií*	*Z1-1*	*Z1-2*	*Z1-3*	*Z1-4*	*Z1-5*	*Z1-6*	*série*	*celkem*
39.–44.	Radim Buráň	G UherBrod	4	0	8			10			18,0	18,0
	Jakub Komárek	GUHradiště	4	0	8		10				18,0	18,0
	Jan Koška	GJírovcČB	−1	0	8		10				18,0	18,0
	Jan Šulíček	SPSEPard	2	0	8		10				18,0	18,0
	Matěj Volf	GCoubTábor	1	0	8		10				18,0	18,0
	Radek Zavřel	SPŠSmíchov	3	0	8		10				18,0	18,0
45.–46.	Ondřej Hráček	GOlgHavl	2	0	8		8				16,0	16,0
	Patrik Rosenberg	GJarošeBO	−2	0	1	3		12			16,0	16,0
47.	Lucie Kunčarová	GVolgogrOS	3	0	8	1			5		14,0	14,0
48.–49.	Alexandr Čelakovský	G UherBrod	3	0	8	1	4				13,0	13,0
	Jan Jeníček	GNAlejíPH	3	0	8		5				13,0	13,0
50.–51.	Patrik Herman	GTomkovaOL	0	0	8	1		2			11,0	11,0
	Ondra Müller	GTurnov	2	0	8	1		2			11,0	11,0
52.–53.	Jiří Bleha	SPSEPard	2	0	8			2			10,0	10,0
	Bohdan Kopčák	GNAlejíPH	3	0	8		2				10,0	10,0
54.	Ondřej Polanecký	SPŠPísek	2	0	8			0,7			8,7	8,7
55.–72.	Martin Boček	MendelGOP	0	0	8						8,0	8,0
	Tomáš Černý	GArabskáPH	3	0	8						8,0	8,0
	Evgenia Golubeva	GJosefskPH	4	0	8						8,0	8,0
	Janek Hlavatý	GJirsíkaČB	0	0	8						8,0	8,0
	Milan Jiříček	SPŠPísek	2	0	8						8,0	8,0
	Radim Kopunec	G UherBrod	−1	0	8						8,0	8,0
	Filip Krul	SPŠSmíchov	3	0	8						8,0	8,0
	Ondřej Martínek	G UherBrod	4	0	8						8,0	8,0
	Antonín Musil	PORGPha	2	0	8						8,0	8,0
	Dávid Oravec	G DubNVáh	4	0	8						8,0	8,0
	Václav Pavlíček	SPSEPard	3	0	8	0					8,0	8,0
	Jan Piroutek	GŠpitálsPH	3	0	8		0				8,0	8,0
	Jakub Profota	GŘíč	4	0	8						8,0	8,0
	Matej Stencel	GPošKošice	2	0	8						8,0	8,0
	Jan Škoula	GBenesov	3	0	8						8,0	8,0
	Filip Vaculík	G UherBrod	4	0	8						8,0	8,0
	Jiří Vlček	GFXŠaldyLI	3	0	8						8,0	8,0
	Michal Zacek	MensaG	2	0	8						8,0	8,0
73.	Vojtěch Frőmmel	SPŠEMasLI	3	0	5						5,0	5,0
74.	Petr Kroča	G UherBrod	−2	0	0	0	0,3	4			4,3	4,3
75.	Tomas Kocian	GTurnov	3	0	2						2,0	2,0
76.	Tomáš Hájek	G UherBrod	4	0	1						1,0	1,0

matfyz

KSP pro vás připravují studenti Matematicko-fyzikální fakulty Univerzity Karlovy.

Webové stránky: *https://ksp.mff.cuni.cz/*
E-mail: *ksp@mff.cuni.cz*
Diskusní fórum: *https://ksp.mff.cuni.cz/forum/*

Chcete-li s námi komunikovat bezpečně, můžete si ověřit náš HTTPS certifikát – jeho SHA1 fingerprint je: `E9:DB:EE:C6:62:BC:14:DE:09:E4:E8:97:DC:36:0E:87:B3:50:B0:01`.

si počítat, kolik stejných symbolů za sebou jsme v daném směru viděli. Řečeno kódem (pro řádky, ostatní směry by se psaly podobně):

```
for r in range(R):
        znak = " "
        pocet = 0
        for s in range(S):
                if T[r][s] != znak:
                        pocet = 0
                        znak = T[r][s]
                pocet += 1
                if pocet >= 5 and znak in "XO":
                        # výhra
```

Takové řešení má časovou složitost $\mathcal{O}(NRS)$, kde N je počet tahů, a paměťovou složitost $\mathcal{O}(RS)$, neboť pro každý tah projdeme celou tabulku.

Šetříme čas i paměť

V předchozím řešení stačilo, aby zlomyslní organizátoři přidali na vstup jeden tah na políčko $[1\,000\,000, 1\,000\,000]$, a náš program by v lepším případě spadl, v horším případě by předtím ještě zabral veškerou volnou paměť našeho počítače. Obecně se snažíme stranit algoritmů, jejichž časová a paměťová složitost závisí na hodnotě čísel na vstupu, a upřednostňujeme algoritmy, jejichž složitost závisí pouze na velikosti vstupu.

Pojďme se tedy zbavit závislosti na R a S. Brzdí nás dvě věci: na začátku vytváříme pole velikosti $R \times S$, přestože z něj pak použijeme jen $\mathcal{O}(N)$ políček, a po každém tahu znovu procházíme celé pole, přestože je jasné, že pokud někdo teď vyhrál, muselo to být díky nově přikreslenému symbolu.

Místo pole T tedy použijeme nějakou „děravou“ strukturu, třeba hešovací tabulku. Ta navenek funguje podobně jako pole, ale namísto čísla ji můžeme indexovat třeba i dvojicí čísel nebo řetězcem. Pro nás je však důležitější, že obsahuje-li hešovací tabulka právě N položek, zabírá jen $\mathcal{O}(N)$ paměti, nezávisle na tom, jaké jsou hodnoty, kterými ji indexujeme. Za toto celé však platíme tím, že doba přístupu k jednomu prvku je $\mathcal{O}(1)$ pouze v průměrném případě – máme-li smůlu, může přístup trvat i $\mathcal{O}(N)$ času.

Pouhým nahrazením pole za hešovací tabulku si zlepšíme paměťovou složitost z $\mathcal{O}(RS)$ na $\mathcal{O}(N)$ – nesmíme už ovšem hešovací tabulku na začátku vyplňovat mezerami, namísto toho se v celém algoritmu při přístupu k nějakému políčku hešovací tabulky budeme vždy ptát, zda se v ní dané políčko vůbec nachází, a pokud ne, chápeme ho jako prázdné.

Zlepšit časovou složitost je také snadné: namísto, abychom procházeli celou tabulku $[0,0] \ldots [R-1, S-1]$, stačí, když projdeme čtvercovou oblast $[i-4, j-4] \ldots [i+4, j+4]$, kde $[i, j]$ je souřadnice políčka, na které právě někdo zahrál. Tak bude jedno zkontrolování výhry trvat čas $\mathcal{O}(9^2) = O(1)$. (Samozřejmě, kdybychom chtěli být ještě o něco rychlejší, můžeme výhru kontrolovat chytřeji a neprocházet zbytečně všechna políčka v oblasti, ale pomohli bychom si jenom o konstantu.)

Takto upravené řešení má v průměrném případě časovou složitost $\mathcal{O}(N)$ a paměťovou taktéž a můžeme si rozmyslet, že lépe to jít nemůže.

Program (Python 3):
`http://ksp.mff.cuni.cz/viz/31-Z1-4.py`

Říša Hladůk

31-Z1-5 Fotka zastupitelů

Chceme najít v posloupnosti celých čísel nejdelší souvislý úsek, v němž se žádné číslo nevyskytuje dvakrát. Nejjednodušší řešení, které se nabízí, je projít všechny takové úseky. Pro každý možný začátek budeme zkoušet přidávat další čísla a před přidáním každého čísla vždy celý úsek projdeme a zkontrolujeme, jestli se číslo v daném úseku už nenachází. Tento algoritmus je správný, ale pomalý. Pro každý začátek provedeme až $\mathcal{O}(n^2)$ kroků, celkem tedy $\mathcal{O}(n^3)$.

Jde to rychleji

Všimneme si, že pokud zrovna procházíme úsek mezi i a j a zprava přidáme číslo, které už v tomto úseku je (řekněme na pozici k), žádný platný úsek, který začíná mezi i a k, nebude delší než $j-i$. Nemůžeme totiž přidat číslo na pozici $j+1$, dokud máme to samé číslo i na pozici k. Musíme tedy nejdříve vyhodit číslo na pozici k, abychom mohli získat dalšího kandidáta na nejdelší úsek.

Pořídíme si tedy dva ukazatele. Pravý ukazatel bude vždy značit pravý konec právě prohlížené posloupnosti, levý ukazatel bude značit levý konec. Nejprve budeme posouvat pravý ukazatel. Po každém posunutí zkontrolujeme, zda se nové číslo už jednou mezi levým a pravým ukazatelem nachází, nebo ještě ne. Pokud ne, můžeme posouvat pravý ukazatel dále. Pokud tam číslo již jednou je, budeme posouvat levý ukazatel, dokud na dané číslo nenarazíme a nevyhodíme ho z posloupnosti.

Tímto způsobem určitě jednou narazíme na nejdelší úsek, v němž se žádné číslo nevyskytuje dvakrát. Jistě totiž jednou pravým ukazatelem dojdeme na pravý konec hledaného úseku. Protože jsme levým ukazatelem hýbali jen v případě, že se v posloupnosti vyskytovalo nějaké číslo dvakrát, určitě nyní levý ukazatel ukazuje na začátek hledané posloupnosti. Pokud bychom se ho pokusili posunout doleva a úsek prodloužit, narazili bychom na číslo, které se v daném úseku už jednou vyskytuje (případně na začátek posloupnosti).

Tento algoritmus je o něco rychlejší než ten první. Pravý i levý ukazatel projdou celou posloupnost pouze jednou, ovšem při každém posunu pravého ukazatele procházíme celý již vybraný úsek, abychom zjistili, jestli se v něm nové číslo vyskytuje. Celkový čas tedy bude $\mathcal{O}(n^2)$.

Trik s polem

Ukážeme, že úloha jde vyřešit i v lineárním čase. Protože máme politické strany očíslované od 0 do $n-1$, můžeme si vyrobit pole délky n, do kterého si budeme ukládat, kde jsme dané číslo naposledy viděli.

Tedy pokaždé, když posuneme pravý ukazatel, podíváme se na nové číslo a přistoupíme na `pole[nové číslo]`. Pokud na tomto indexu v poli nenalezneme žádný záznam, znamená to, že v aktuálně vybrané posloupnosti mezi levým a pravým ukazatelem toto číslo ještě není, a zapíšeme tam pozici pravého ukazatele. Pokud však nějaký záznam nalezneme, číslo v posloupnosti už jednou máme. Budeme tedy posouvat levý ukazatel a pokaždé, když z posloupnosti nějaké číslo vyhodíme, vymažeme jeho záznam v poli.

Takto tedy umíme zjistit, jestli se v posloupnosti nějaké číslo nachází, v konstantním čase. Celkem nám tedy stačí $\mathcal{O}(n)$ času na projití celé posloupnosti a nalezení nejdelšího souvislého úseku, ve kterém se žádná čísla neopakují.

A co kdyby bylo číslování jiné?

Na závěr se můžeme zamyslet ještě nad jedním případem:

co kdyby nebyly politické strany očíslovány takhle pěknými přirozenými čísly, ale nějakými ošklivými, velkými nebo zápornými?

Jedna možnost je si čísla v čase $\mathcal{O}(n \log n)$ seřadit a přečíslovat. Algoritmus tak už nepoběží v lineárním čase, ale stále rychleji než předchozí algoritmy.

Druhá možnost je pomoct si hešovací tabulkou.[2] V ideálním případě to bude fungovat úplně stejně jako s polem s přirozenými indexy. Občas se nám může stát, že se dvě čísla zahešují na stejnou hodnotu a budeme muset projít celé pole stejně jako u druhého algoritmu, u dobrých hešovacích funkcí se to ale neděje často a průměrná časová složitost tedy bude stále lineární.

Zuzka Urbanová

31-Z1-6 Trasa demonstrací

Jak bylo řečeno v zadání úlohy, město tvoří grafovou strukturu zvanou strom. Každá křižovatka je vrcholem a každá ulice je hranou. Pokud jste na stromy ještě nenarazili, doporučujeme přečíst si o nich v naší grafové kuchařce.[3]

Chceme vlastně významné vrcholy stromu rozdělit do párů takovým způsobem, aby se cesta mezi vrcholy v jednom páru nepřekrývala s žádnou jinou cestou mezi vrcholy jiného páru. Okrajové křižovatky, které nejsou označené jako významné, výsledek nijak neovlivní (kdyby na ně některý průvod zašel, ocitl by se ve slepé uličce a nezbývalo by mu nic jiného, než se vrátit zpět na nějaké rozcestí).

Nejdříve nazveme nějaký libovolný vrchol stromu jeho kořenem. V tomto nově zakořeněném stromě platí, že všechny významné okrajové křižovatky jsou listy, tedy s výjimkou kořene v případě, že jsme za kořen vybrali nějakou významnou okrajovou křižovatku.

Pojďme se podívat, jak se v tomto nově zakořeněném stromě budou chovat podstromy zavěšené pod některým z jeho vrcholů. O libovolném takovém podstromě můžeme říct, že je se zbytkem stromu spojen právě jednou hranou. Z toho vyplývá, že nejvýše jedna významná křižovatka z tohoto podstromu může být spárována s jinou, která se nachází mimo tento podstrom (kdyby bylo s vrcholy zbytku stromu spárováno více křižovatek v tomto podstromě, znamenalo by to, že cesty mezi nimi by se nutně potkávaly právě na této hraně).

Také z toho vyplývá, že nějaká cesta průvodu půjde přes tuto hranu právě tehdy, když má podstrom lichý počet významných křižovatek. Když jich je lichý počet, jedna z nich určitě musí být spárována s nějakou mimo tento podstrom. Pokud se v něm nachází sudý počet významných křižovatek, tak budou všechny spárované jen s křižovatkami tohoto podstromu. Kdyby tomu bylo jinak a jedna křižovatka by byla spárována s nějakou mimo tento podstrom, tak by nám v podstromě zbyla jedna nespárovaná křižovatka, a více křižovatek se zbytkem stromu spárovat nemůžeme, jak jsme již ukázali.

Nyní se podíváme na nějaký vrchol stromu a na podstromy, které mají kořen v jeho synech.

- Pokud mají všechny tyto podstromy sudý počet významných křižovatek, tak budou všechny spárovány uvnitř těchto podstromů, přes náš vrchol nepůjde žádný průvod a podstrom pod tímto vrcholem má také sudý počet významných křižovatek.
- Pokud má právě jeden z podstromů synů lichý počet významných křižovatek a ostatní sudý, tak tato přebývající křižovatka bude spárována s nějakou mimo podstrom pod tímto vrcholem a její průvod bude procházet přes tento vrchol.
- Pokud mají právě dva podstromy lichý počet významných křižovatek, tak přebývající křižovatky z obou těchto podstromů budou spárovány navzájem a cesta mezi nimi půjde přes tento vrchol. Kdyby tomu bylo jinak, tak by přes tento vrchol musely jít dva průvody.
- Pokud je lichých podstromů více než dva, tak úloha nemá řešení. Přes tento vrchol by totiž musely procházet nejméně dva průvody.

Na řešení úlohy se také dá dívat z jiného úhlu. Strom si opět někde zakořeníme. Budeme postupně obarvovat jeho hrany, které se stanou součástí cesty nějakého průvodu. Pro každý vrchol kromě kořene se musíme rozhodnout, zda hranu vedoucí do jeho otce obarvíme, nebo ne. Také pro každý vrchol musíme nejdříve počkat, než skončí obarvování v podstromě pod tímto vrcholem. Poté nám opět mohou nastat čtyři různé situace:

- Pokud není obarvená žádná hrana ze syna do tohoto vrcholu, tak hranu do otce neobarvíme
- Pokud je obarvená právě jedna, tak hranu do otce také obarvíme (pokračujeme v budování cesty z nějaké významné křižovatky)
- Pokud jsou obarvené právě dvě, tak se v tomto vrcholu právě spojily cesty mezi dvěmi významnými křižovatkami a tím se spárovaly. Hranu do otce neobarvíme.
- Pokud se zde setkají více než dvě obarvené hrany, řešení neexistuje, v tomto vrcholu se nutně musejí průvody potkat.

Obarvování začneme významnými křižovatkami, nyní listy zakořeněného stromu. U těch nám nezbývá nic jiného, než hranu obarvit. Poté se přesuneme do jejich otců a provedeme obarvování, poté opět do jejich otců a tak dále.

Samotný algoritmus pro nalezení párování je stejný, ať už se na problém díváme jako na postupné dělení stromu na menší a menší podstromy nebo na obarvování hran. Jedná se o prohledávání do hloubky.

Vytvoříme si rekurzivní funkci, která se bude spouštět na nějakém podstromě a bude vracet číslo nespárovaného listu v tomto podstromě, pokud nějaký takový list existuje. Když nějaké číslo vrátíme, je to jako bychom hranu do otce obarvili a zároveň to znamená, že tento podstrom měl lichý počet významných křižovatek. Když žádné číslo nevracíme, hranu neobarvujeme a podstrom má sudý počet významných křižovatek.

Pokud je funkce spuštěna na listě, který je významnou křižovatkou, vrátí číslo tohoto listu. Ve všech ostatních případech funkce nejdříve spustí sama sebe na všech synech aktuálního vrcholu. Pokud ze synů dostala právě dvě čísla nespárovaných listů, tak je spáruje a číslo nespárovaného vrcholu nevrací. Pokud dostala číslo jedno nebo žádné, tak tento výsledek funkce vrátí. Pokud dostala čísel více než dvě, tak řešení neexistuje.

Pro nalezení spárovaní nám stačí tuto funkci spustit na kořenu stromu, což může být zcela libovolný vrchol. Pokud jako kořen vybereme významnou okrajovou křižovatku, tak nám funkce nutně musí vrátit číslo nějakého nespárovaného vrcholu (nebo říct, že řešení neexistuje), protože podstrom pod takovým kořenem jich z pohledu naší funkce obsahuje lichý počet (samotný kořen totiž neuvažuje). Tento přebývající vrchol prostě spárujeme s kořenem.

Časová složitost tohoto řešení bude lineární vzhledem k počtu vrcholů v, protože každý vrchol navštívíme právě jednou. Paměťová bude také lineární, protože si potřebujeme pamatovat, jak vypadá náš strom, a pro stromy platí, že počet hran je roven počtu vrcholů mínus jedna; také si ale musíme pamatovat zásobník rekurze, který může mít až řádově v položek v nejhorším případě.

Kuba Pelc

[2] http://ksp.mff.cuni.cz/viz/kucharky/hesovani
[3] http://ksp.mff.cuni.cz/viz/kucharky/grafy

Výsledková listina první série začátečnické kategorie 31. ročníku KSP

	řešitel	*škola*	*ročník*	*sérií*	*Z1-1*	*Z1-2*	*Z1-3*	*Z1-4*	*Z1-5*	*Z1-6*	*série*	*celkem*
0.					8	10	10	12	12	14	66,0	66,0
1.	Daniel Skýpala	GTomkovaOL	1	0	8	10	10	12	12	14	66,0	66,0
2.	Petr Kolář	GMilevsko	3	0	8	10	10	12	12	13,5	65,5	65,5
3.	Jiří Kvapil	GTomkovaOL	1	0	8	3	10	12	12	14	59,0	59,0
4.	Filip Kastl	GKepleraPH	3	0	8	10	10	6	11	13,5	58,5	58,5
5.	Vladimír Chudý	G Chrudim	2	0	8	10	10	12	9	5	54,0	54,0
6.	Kristýna Petrlíková	VOŠJičín	1	0	8	10	10	12	12		52,0	52,0
7.–8.	Michal Bravanský	GBílovec	1	0	8	10	10	8	8	2	46,0	46,0
	Martina Daňková	KŠpGym Bo	2	0	8	10	10	12	6		46,0	46,0
9.	Marie Kalousková	GNAlejíPH	3	0	8	10	10	2	12		42,0	42,0
10.–12.	Šimon Andrš	GKepleraPH	0	0	8	10	10	12			40,0	40,0
	Adam Bujdák	G JM Galanta	3	0	8	10	10	12			40,0	40,0
	Jan Hlaváč	GNAlejíPH	3	0	8	10	10	12			40,0	40,0
13.	Michal Mlčoch	G UherBrod	4	0	8	3	8	12	6	2	39,0	39,0
14.	Jakub Ondroušek	GTomkovaOL	−1	0	8	7	10	12	1		38,0	38,0
15.–16.	Ondřej Chlubna	GOrlová	2	0	8	3	10	8	8		37,0	37,0
	Jan Štěch	GJirsíkaČB	2	0	8	1	10	10	8		37,0	37,0
17.	Petr Budai	G JGJ PH	2	0	8	10			12	6	36,0	36,0
18.	Albert Kučera	GNadŠtolPH	2	0	8	10	10	5,3	1		34,4	34,4
19.–23.	Patrik Baláš	SPSEPard	1	0	8	3	10	12			33,0	33,0
	Robert Jaworski	GÚstavníPH	1	0	8	3	10	12			33,0	33,0
	Jan Najman	SPSEPard	2	0	8	3	10	12			33,0	33,0
	Tomáš Sláma	GTurnov	4	0	8	7	10	8			33,0	33,0
	Terézia Strišovská	GJHroncaBA	3	0	8	3	10	12			33,0	33,0
24.	Kateřina Rosická	GKutnáHora	4	0	8	10	10	4			32,0	32,0
25.	Vojtěch Žák	GŠpitálsPH	3	0	8		10	12			30,0	30,0
26.	Martin Bencko	GOhradníPH	2	0	8	3	10	8			29,0	29,0
27.–31.	Petr Aubrecht	GHeyrovPH	4	0	8	10	10				28,0	28,0
	Tomáš Dostál	MendelGOP	4	0	8		10	2	8		28,0	28,0
	Robert Gemrot	GKomHavíř	2	0	8	10	10				28,0	28,0
	František Kmječ	StOlavVGS	3	0	8	10	10				28,0	28,0
	Martin Zmitko	G FrýdlNOs	3	0	8	10	10				28,0	28,0
32.	Eric Valčík	G UherBrod	4	0	8		10	8			26,0	26,0
33.–34.	Adam Húšťava	EupSchoolLux	1	0	8		10		6		24,0	24,0
	Kryštof Suchánek	GLesníZlín	3	0	8		10	6			24,0	24,0
35.	Lucie Vomelová	GŠpitálsPH	3	0	8	3	10	2			23,0	23,0
36.	Jiří Heller	GNAlejíPH	3	0	8		10	2			20,0	20,0
37.–38.	Jan Kotovský	GPísnickáPH	0	0	8	6	1	4			19,0	19,0
	Jakub Nevařil	G UherBrod	1	0	8		5	6			19,0	19,0