Korespondenční Seminář z Programování

ZAČÁTEČNICKÁ KATEGORIE

26. ročník | **KSP-Z** | **Červenec 2014**

Náš premiérový ročník KSP-Z se chýlí ke konci. Ještě vám ale dlužíme vzorová řešení úloh čtvrté série a zejména závěrečnou výsledkovou listinu. Kdybyste některé řešení nemohli pochopit nebo chtěli vysvětlit jakýkoli detail, nebojte se nás zeptat na našem fóru nebo emailem na *ksp@mff.cuni.cz*.

Děkujeme za všechna vaše řešení a těšíme se na vás v příštím ročníku, ať už znovu v KSP-Z nebo v hlavní kategorii KSP. Pěkné prázdniny!

Řešení čtvrté série začátečnické kategorie 26. ročníku KSP

26-Z4-1 Vražedná čísla

Místo samotných vražedných čísel zaměříme svou pozornost na jejich zbytky po dělení číslem Q. Aby se nám s nimi pohodlně pracovalo, zavedeme si následující (vcelku obvyklé) značení: je-li x nějaké celé číslo, bude $x \bmod Q$ jeho zbytek po dělení Q. Tento zbytek je také celé číslo a leží mezi 0 a $Q-1$.

Nyní se podívejme na součet nějakých dvou čísel $x+y$. Kdy je tento součet dělitelný Q? Buď tehdy, jsou-li obě čísla také dělitelná Q (čili $x \bmod Q = y \bmod Q = 0$), nebo dají-li jejich zbytky dohromady Q, tedy

$$(x \bmod Q) + (y \bmod Q) = Q.$$

Počítejme nejprve dvojice prvního druhu. K tomu nám stačí spočítat, kolik z vražedných čísel je dělitelných Q. Pokud jich je z_0, můžeme utvořit přesně $z_0 \cdot (z_0-1)/2$ dvojic. Proč právě tolik? Představme si, že na chvíli budeme odlišovat, které číslo ve dvojici je první a které druhé. Máme z_0 možností, jak zvolit první číslo, a pak z_0-1 možností, jak doplnit druhé (o jedničku méně proto, že nesmíme totéž číslo použít dvakrát). Teď jsme ovšem každou dvojici započítali dvakrát, takže výsledek ještě vydělíme dvěma.

Pokračujme dvojicemi druhého druhu. Označme z_i počet vražedných čísel, která dávají zbytek i. Dvojici druhého druhu získáme tak, že spárujeme číslo se zbytkem 1 s číslem se zbytkem $Q-1$, nebo 2 s $Q-2$, atd. Takže napočítáme celkem

$$z_1 z_{Q-1} + z_2 z_{Q-2} + \ldots$$

dvojic. Kde se přesně zastavíme? Za $z_{Q/2} z_{Q/2}$ už pokračovat nesmíme, protože bychom opět dvojice počítali podruhé. Ale i samotný člen $z_{Q/2} z_{Q/2}$ je zákeřný – objeví se jen tehdy, je-li Q sudé číslo, ale pokud tomu tak je, jsme ve stejné situaci jako u dvojic prvního druhu, neboť kombinujeme čísla se stejným zbytkem. Takových dvojic bude $z_{Q/2} \cdot (z_{Q/2}-1)/2$.

Pojďme shrnout, co jsme vymysleli. Pro liché Q je výsledek roven

$$\frac{z_0 \cdot (z_0-1)}{2} + z_1 z_{Q-1} + z_2 z_{Q-2} + \ldots + z_{\frac{Q-1}{2}} z_{\frac{Q+1}{2}},$$

pro sudé Q pak

$$\frac{z_0 \cdot (z_0-1)}{2} + z_1 z_{Q-1} + z_2 z_{Q-2} + \ldots + \frac{z_{Q/2} \cdot (z_{Q/2}-1)}{2}.$$

Program pouze spočítá, kolik čísel dává který zbytek, a pak to dosadí do našeho kouzelného vzorečku. Obojí jistě stihne v lineárním čase a lineární paměti.

Dodejme ještě, že ve většině programovacích jazyků si musíme dávat pozor na zbytky po dělení záporného čísla. Často totiž vyjdou záporně: například $(-8) \bmod 7 = -1$. V programu je proto jistější psát $((x \bmod Q) + Q) \bmod Q$, což pro kladné x neuškodí a pro záporné to výsledek posune do kladné hodnoty, která se nám hodí více.

Program (C):
`http://ksp.mff.cuni.cz/viz/26-Z4-1.c`

Program (Python 3):
`http://ksp.mff.cuni.cz/viz/26-Z4-1.py`

Ondřej Hlavatý & Martin „Medvěd“ Mareš

26-Z4-2 Sbírání vajíček

Kam se má Kevin s košíkem postavit, aby se Zuzka co nejméně naběhala?

Na přímce je vzdálenost bodů a a b absolutní hodnota jejich rozdílu, $|a-b|$. Třeba body 3 a 10 jsou vzdálené $|3-10| = 7$.

Hledáme místo, které má nejmenší možný součet dvojnásobků vzdáleností od zadaných bodů na přímce. Proč dvojnásobků? Jednou počítáme cestu od Kevina k vajíčku a jednou od vajíčka ke Kevinovi. Obě jsou stejně dlouhé. Když chceme vybrat místo s nejmenším součtem, násobení dvěma ale vůbec nemusíme uvažovat. Tak to bude pro nás při dokazování příjemnější. Každý sčítanec v součtu je násoben dvěma, můžeme dvojku vytknout před součet. Vždy, když porovnáme dvě ušlé vzdálenosti, dvojku z nerovnice odstraníme, protože $2a < 2b$ je to samé jako $a < b$.

Pro každé místo na zahradě tedy můžeme sečíst vzdálenosti od vajíček, a vybrat to místo, kde je nejnižší součet. Toto nám dá správný výsledek, ale bude to pomalé. Možných míst je velmi mnoho.

Jak zjistit rychleji, kam postavit Kevina?

Představme si, že už máme zjištěný součet pro nějaké místo A, od tohoto místa je 5 vajíček napravo a 10 nalevo. Teď Kevina posuneme o 3 políčka doleva do bodu B, mezi A a B se bez újmy na obecnosti žádné vajíčko nenachází. (Kdyby tam bylo, za B zvolíme to nejbližší místo, kde to vajíčko je, a tak mezi A a B už žádné nebude. B už sice nebude o tři políčka od A, ale to nevadí, následující odstavec bude platit obdobně i pro jinou vzdálenost než 3.)

Jak se změní součet? Zvětší se o třikrát tolik, kolik bylo vajíček od A vpravo, to je o $3 \cdot 5$. Zároveň se ale zmenší o třikrát tolik, kolik vajíček bylo od A nalevo, to je o $3 \cdot 10$, protože k nim se Kevin přiblížil. Součet od B je tedy menší než od A, takže to je o něco vhodnější místo.

Tímto postupem tedy pro každý bod, od kterého je více vajíček na jedné straně než na druhé, umíme najít místo s lepším součtem vzdáleností. Žádné takové tedy nebude to správné.

Nejlepší je naopak místo, které má stejně vajíček nalevo jako napravo. Je-li vajíček lichý počet, je toto místo jen jedno, Kevin by si měl stoupnout přímo na prostřední vajíčko. Pokud jich je sudý počet, je to jakékoliv místo mezi dvěma prostředními vajíčky, tedy mezi vajíčky číslo $N/2$ a $N/2 \pm 1$ (plus, pokud indexujeme od jedné, mínus, pokud od nuly). Vybrat můžeme třeba jedno z nich. Prvek na prostřední pozici se obvykle nazývá *medián* posloupnosti.

A je to jasné. Jako řešení stačí vypsat medián. Existuje zaručeně lineární algoritmus na jeho nalezení, je popsán v kuchařce.[1] Nám ale stačí čísla uložit do pole, setřídit v čase $\mathcal{O}(N \log N)$, sáhnout doprostřed a vypsat medián.

Vzorové řešení je na pět řádků, viz ukázkový program.

Program (Python 3):
`http://ksp.mff.cuni.cz/viz/26-Z4-2.py`

Dominik Macháček

26-Z4-3 Hra Othello

Pojďme si nejprve rozmyslet, jak bychom takovouto úlohu řešili s tužkou a papírem. K tomu se nám hodí nakreslit si diagram všech možných průběhů hry (viz obrázek vpravo).

Na začátku má Kevin na výběr ze tří možných tahů, poté Petr ze dvou a poslední tah už je Kevinovi předurčen. Zadání říká, že oba hráči hrají optimálně. Jak z toho poznáme, kdo kam potáhne?

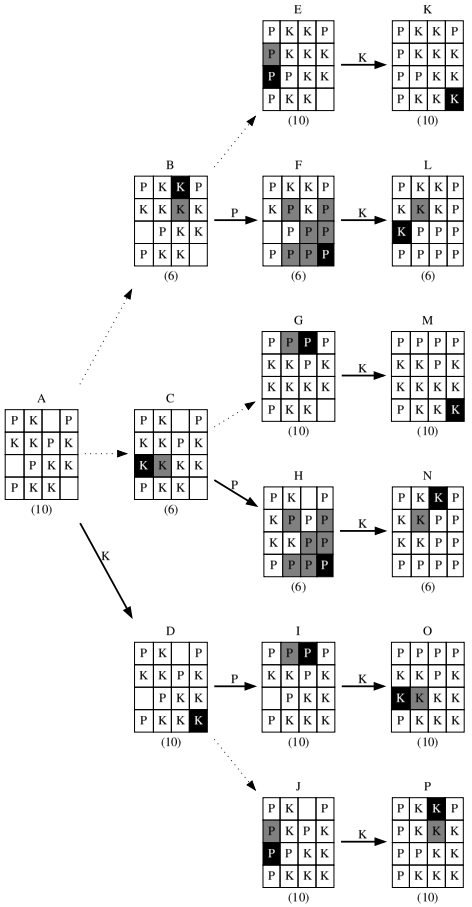
Začněme jednodušší otázkou: jak by se úloha řešila, kdyby zbývaly už jen dva tahy do konce (a na řadě byl tedy Petr)? Petr má na výběr za dvou možností, kam táhnout. Vybere si pochopitelně tu, ve které získá Kevin na konci *méně* kamenů. Např. v pozici[2] C si vybere tah do pozice H, ze které Kevin musí táhnout do N a skončit tak s 6 kameny (číslo pod deskou v závorce), kdežto kdyby si vybral G, bude mít Kevin 10 kamenů. Tučné šipky v našem schématu značí vždy tah, který by si hráč z dané pozice vybral.

Nyní pro každou z pozic dva tahy před koncem (B–D) víme, kam Petr potáhne a jak hra dopadne. Proto si k těmto pozicím můžeme poznamenat *ohodnocení* $o(X)$, tedy číslo říkající, s kolika kameny Kevin skončí, vyjdeme-li z této pozice a oba hráči budou hrát optimálně. Ohodnocení pozic najdete v našem diagramu jako čísla v závorce pod deskou.

Nás ovšem zajímá výsledek celé hry, tedy ohodnocení pozice A. No ale to už je teď snadné určit! Víme, že pokud si Kevin vybere například tah do C, získá $o(\mathrm{C}) = 6$ kamenů (neb ohodnocení přesně popisuje, jak dopadne zbytek hry od C do konce). Kevin si tedy samozřejmě vybere tah s *nejvyšším* ohodnocením. Jinými slovy $o(\mathrm{A}) = \max(o(\mathrm{B}), o(\mathrm{C}), o(\mathrm{D}))$.

Tomu, co jsme právě popsali, se obvykle říká *minimaxový algoritmus*, a je to asi nejznámější herní algoritmus vůbec. Myšlenka je jednoduchá: dosažitelným pozicím postupně směrem od koncových přiřazujeme ohodnocení, které říká, s jakým „skóre“ hra skončí, budou-li oba hráči hrát optimálně. Jeden hráč (v našem případě Kevin) usiluje o co nejvyšší skóre, druhý o co nejnižší. Roli skóre v naší hře zaujímá počet Kevinových kamenů.

Pozici můžeme ohodnotit ve chvíli, kdy známe ohodnocení všech pozic, na které z ní lze táhnout, a ohodnocení je buď minimem, nebo maximem (proto „minimax“) z ohodnocení těchto pozic, podle toho, který hráč je na tahu.



Nyní už zbývá jen to naprogramovat. Postup lze schématicky znázornit např. takto:

1. Pro každou ze tří možností prvního Kevinova tahu:
2. Pro každou ze dvou možností Petrova tahu:
3. Vytvoř novou kopii desky.
4. Odehraj na ní tyto dva tahy a nutný Kevinův poslední.
5. Spočítej Kevinovy kameny na zaplněné desce (skóre).
6. Vyber ze těchto dvou možných skóre minimum.
7. Vyber z těchto tří minim to největší a prohlas ho za výsledek.

[1] `http://ksp.mff.cuni.cz/viz/kucharky/rozdel-a-panuj`

[2] Pozicí zde rozumíme kompletní stav hry v daný okamžik, tedy umístění a barvu všech kamenů, spolu s informací, kdo je na tahu.

Závěrečná výsledková listina začátečnické kategorie 26. ročníku KSP

	řešitel	*škola*	*ročník*	*sérií*	*Z4-1*	*Z4-2*	*Z4-3*	*Z4-4*	*Z4-5*	*Z4-6*	*série*	*celkem*
0.					8	10	10	12	12	14	66,0	264,0
1.	Miroslav Šerý	GValašKlob	1	4	8	10	10	12	12	5	57,0	237,5
2.	Jan Tománek	GPelhřimov	3	4	8	10	1	12			31,0	224,0
3.	Jakub Pelc	G_UherBrod	0	4	8	10	10	12			40,0	223,5
4.	Václav Fabík	ZŠKřídloBO	−1	4	8						8,0	190,0
5.	Jonáš Malena	SŠJeštědLI	4	3							0,0	164,5
6.	Václav Končický	GSOŠ_FrMís	3	4	8	0					8,0	140,0
7.	Přemysl Šťastný	GŽamberk	0	3							0,0	139,0
8.	Michal Töpfer	G_DrJPekMB	1	3	6	10	10		10	14	50,0	138,0
9.	Pavel Mikuš	GMělník	3	4	4	10	10	8			32,0	112,0
10.	František Zajíc	G_Nymburk	1	3							0,0	102,0
11.	Lucie Studená	GKepleraPH	4	2							0,0	95,0
12.	Jakub Lukeš	GNAlejíPH	1	4	4	3					7,0	88,0
13.	Antonín Teichmann	GJeronýmLI	4	2							0,0	72,0
14.	Petr Šíma	GKlatovy	1	4	4						4,0	70,0
15.	Jiří Vozár	G_UherBrod	2	1							0,0	59,5
16.	Michal Převrátil	GKlatovy	1	2							0,0	58,0
17.–18.	Antonín Bruštík	G_UherBrod	3	4	8						8,0	54,0
	Jan Burda	G_Holice	−1	3	6						6,0	54,0
19.	Marek Vitula	GJarošeBO	3	2							0,0	51,0
20.	Jakub Heyduk	SŠP_ČB	4	2							0,0	48,0
21.	Jakub Tětek	CírkG_Plzeň	0	2	6						6,0	47,0
22.	Václav Trpišovský	GOpenGaBab	−3	3							0,0	44,0
23.	Lukáš Fruněk	GLesníZlín	1	1							0,0	40,0
24.–26.	Josef Gajdůšek	SŠKKamPard	1	1							0,0	39,0
	Stanislav Lukeš	GPísnickáPH	1	1							0,0	39,0
	Radovan Švarc	G_ČTřebová	3	2							0,0	39,0
27.	Daniel Šerý	G_RožnovPR	2	2							0,0	37,0
28.–29.	Jan Horák	GŠumperk	3	3							0,0	34,0
	Viktor Kovařík	G_UherBrod	3	3							0,0	34,0
30.	Milan Malina	GMikulášPL	1	2							0,0	30,0
31.	Jan Václavek	GUnOrl	2	1	6	10		12			28,0	28,0
32.	Jan Vargovský	GSPŠFrenšt	4	1							0,0	25,0
33.–34.	Ivana Krumlová	GJarošeBO	1	2							0,0	23,0
	Vojtěch Václavík	GSOŠ_FrMís	4	2							0,0	23,0
35.–36.	Tomáš Michna	SPŠEOstrava	4	2							0,0	20,0
	Benedikt Žour	G_UherBrod	−1	2							0,0	20,0
37.	Janek Hlavatý	ZŠ_DukelČB	−5	3							0,0	19,0
38.–39.	Štěpán Košan	GKlatovy	2	1							0,0	18,0
	Aneta Šťastná	GOmskPha	4	1							0,0	18,0
40.	David Karlík	G_UherBrod	3	2							0,0	16,0
41.	Zdeněk Pavlátka	GMikulášPL	2	2							0,0	15,5
42.	David Dvořáček	G_UherBrod	3	2	6						6,0	14,0
43.	Martin Jílek	GKlatovy	2	1							0,0	13,0
44.–46.	Ivona Hrivová	GŽilina	4	3							0,0	11,0
	Dominik Krasula	GKrnov	1	1							0,0	11,0
	Jan Vozár	G_UherBrod	0	3	0						0,0	11,0
47.	Michaela Bačová	G_UherBrod	3	2							0,0	9,0
48.	Petr Pacner	GBroumov	2	1							0,0	7,0
49.	Tereza Bohumská	GPísnickáPH	−1	1							0,0	2,0
50.	Martin Majtán	G_SNPPiešť	1	1							0,0	1,0

Samotný minimax je jen několik vnořených cyklů. Dále potřebujeme umět simulovat průběh hry. K tomu si desku uložíme jako dvourozměrné pole znaků a pro každý tah ji upravujeme přímo následujíce pravidla. Ta jsou v zadání už víceméně v algoritmické podobě, pročež je stačí jen „přeložit“ do vašeho oblíbeného programovacího jazyka.

Drobný zádrhelem mohlo být například, jak zařídit „pro každý z osmi směrů proveďte následující“. Tady se hodí úplně stejný trik, jaký jsme použili v úloze o piškvorkách:[3] každý směr lze popsat dvojicí čísel `dr`, `dc` $\in \{-1, 0, 1\}$, která nám říkají, o kolik se při pohybu v daném směru změní číslo řádku a sloupce. Např. pohyb vlevo je popsán dvojicí $(0, -1)$ a šikmo vpravo nahoru $(-1, 1)$. Podrobněji ve zdrojáku.

Program (C):
`http://ksp.mff.cuni.cz/viz/26-Z4-3.c`

Poznámka: V automatickém vyhodnocování odevzdaných řešení byla chyba: za správnou odpověď jsme nebrali počet kamenů, když oba hrají optimálně, nýbrž nejvyšší vůbec dosažitelný počet kamenů. To odpovídá tomu, že Kevin hraje optimálně a Petr „anti-optimálně“, neboli v druhém tahu vybírá maximum namísto minima. Poté, co nás na tuto skutečnost upozornil jeden řešitel na fóru, upravili jsme vyhodnocování, aby přijímalo obě varianty, a upozornili všechny, kdo se pokoušeli úlohu řešit a řešení jim nebyla uznána.

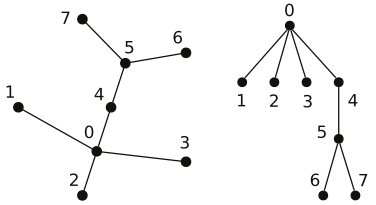
Pročež si pamatujte: pokud jste přesvědčeni, že vaše řešení je správné, a odevzdávátko ho odmítá, je dobré nás na to upozornit. I organizátor si občas nepřečte pořádně zadání…

Filip Štědronský

26-Z4-4 Hlídači v labyrintu

Připomeňme si, že v informatické hantýrce se náš Labyrint nazývá *stromem,* křižovatky označujeme jako *vrcholy* a chodby jsou *hrany.*

Univerzálním receptem na tři čtvrtiny stromových úloh je strom si *zakořenit* a úlohu řešit rekurzivně od kořene. Nejsnáze si to ukážeme na obrázku – vlevo je původní strom a vpravo jeho zakořeněná verze.



Jeden libovolný vrchol (v našem případě 0) prohlásíme za *kořen.* Tím nám ve stromu přirozeně vzniknou směry „nahoru“ (ke kořeni) a „dolů“ (od kořene). Z tohoto způsobu kreslení je také vidět, proč se těmto grafům říká stromy (až na to, že informatici mají zvláštní zvyk kreslit je kořenem vzhůru).

Z každého vrcholu u vede právě jedna cesta do kořene (kdyby dvě, tvořily by dohromady cyklus, a ty ve stromech nejsou). Nejbližší vrchol na této cestě (tedy vrchol „těsně nad“ u) nazýváme *otcem* u. Např. vrchol 4 je otcem vrcholu 5. Naopak vrcholy 6 a 7 označujeme jako *syny* vrcholu 5.

Ještě se nám bude hodit pojem *podstromu.* Podstrom pod u (značíme $T(u)$) je tvořen vrcholem u a všemi jeho (i nepřímými) potomky (syny, syny synů, …). Např. $T(4)$ je tvořen vrcholy 4, 5, 6, 7 (a hranami mezi nimi). Jako *podstromy vrcholu* budeme označovat podstromy pod každým z jeho synů. Např. podstromy vrcholu 0 jsou $T(1), \ldots, T(4)$.

Tím bychom měli z krku část „zakořenit“. Nyní samotné rekurzivní řešení, které obvykle spočívá v tom, že když řešíme úlohu pro nějaký strom T s kořenem u, vyřešíme ji nejprve rekurzivně pro všechny podstromy kořene a pak z těchto dílčích řešení nějak poskládáme řešení pro celé T. Pokud jste o rekurzi nikdy neslyšeli, doporučujeme si přečíst příslušnou kapitolu v naší kuchařce.[4]

Označme si $v_1, \ldots, v_k$ všechny syny u a zaveďme zkratku $T_1 := T(v_1), \ldots, T_k := T(v_k)$. Nyní máme dvě možnosti:

a) Do u postavíme hlídače. Tak máme postaráno o hrany uv_1 až uv_k a hlídače v každém podstromu T_i rozestavíme dle rekurzivně získaného optimálního řešení pro tento podstrom. To si určitě můžeme dovolit, neb v libovolném rozmístění hlídačů můžeme tu část, která je v T_i, nahradit za optimální rozmístění, a počet hlídačů tím nezvýšíme.

b) Do u nepostavíme hlídače. Pak musíme umístit hlídače do každého v_i. Tedy v rámci libovolného podstromu rozmístíme co nejméně hlídačů tak, aby alespoň jeden stál v jeho kořeni.

Z toho už je vidět, že potřebujeme, aby nám rekurze vrátila nejen optimální počet hlídačů. Výstupem našeho rekurzivního algoritmu spuštěného na strom T budou dvě čísla:

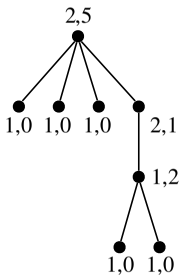
- Nejmenší počet hlídačů, kteří dokáží uhlídat chodby T, za předpokladu, že jeden stojí v kořeni ($K(T)$).
- Nejmenší počet hlídačů za předpokladu, že v kořeni žádný nestojí ($N(T)$).

Pokud tato čísla známe pro všechny podstromy, snadno je spočítáme i pro T:

$$N(T) = K(T_1) + \ldots + K(T_k)$$
$$K(T) = \min(K(T_1), N(T_1)) + \ldots + \min(K(T_k), N(T_k))$$

Ještě musíme vyřešit okrajový případ stromu, který žádné podstromy nemá (je tvořen jediným vrcholem, takovému říkáme *list*). Tam je to ale jednoduché, neb takový strom neobsahuje žádné chodby, a tudíž žádné hlídače nepotřebuje ($N(T) = 0, K(T) = 1$).

Pro ilustraci ukážeme hodnoty K a N pro strom výše:



[3] `http://ksp.mff.cuni.cz/viz/26-Z1-2/reseni`
[4] `http://ksp.mff.cuni.cz/viz/kucharky/zakladni-algoritmy`

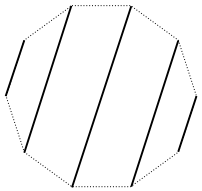
K uhlídání stromu tedy stačí dva hlídači (umístění ve vrcholech 0 a 5).

Program (Python 3):
`http://ksp.mff.cuni.cz/viz/26-Z4-4.py`

Filip Štědronský

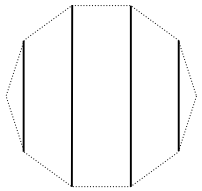
26-Z4-5 Podávání rukou

Zamyslíme se nad úlohou zvlášť pro případy, kdy je N liché a kdy sudé. Nejprve třeba pro N sudé. Pokud si představíme lidi jako vrcholy pravidelného N-úhelníku a podání ruky jako úsečku mezi nimi, je vidět, že aby nikdo v nějakém taktu nezahálel, musí všechny úsečky vést rovnoběžně. Je tedy $N/2$ možností, jak takovým způsobem podávání rukou udělat v různých natočeních.



Tím si ale nepodaly ruku všechny dvojice! Třeba se sousedem ob jedno místo si nikdo ruku nepodal. Obecně si nepodala ruku žádná dvojice dvou lidí taková, že je mezi nimi lichý počet lidí (jinými slovy mezi každými dvěma lidmi byl sudý počet lidí, jak je vidět z obrázku). Zato ale platí, že si podala ruku každá dvojice lidí, kteří mají mezi sebou sudý počet lidí (to totiž odpovídá nějakému natočení rovnoběžek).

Abychom doplnili dosud nevyřešené dvojice, navrhneme ještě další schéma podávání – v něm budou vždy dva lidi proti sobě, kteří si s nikým ruku nepodají, a ostatní si budou podávat ruce zase rovnoběžně. Opět máme $N/2$ možností, jak toto schéma natočit, a opět žádné nepropojí jednoho člověka dvakrát se stejným. Teď je akorát potřeba ukázat, že bylo nezbytné, aby v těchto taktech dva lidi zaháleli. Ti, co zaháleli, byli vždy sevřeni svými oběma sousedy, kteří si navzájem podávali ruku. Tím je vždy odřízli od zbytku světa. Přitom si ale ruce museli podat, proto toto odříznutí bylo nezbytné.

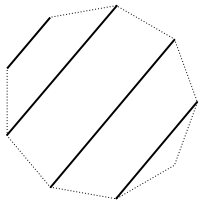


Nikdy si žádná dvojice nepodala ruku dvakrát a každý si podal s někým ruku $(N-1)$-krát (to je přesně tolik, kolikrát měl). Navíc jsme ukázali, že proběhlo jenom tolik zahálení, kolik opravdu muselo, proto pro sudé N může proběhnout jenom N taktů a lépe to nejde.

Teď podobně vyřešíme úlohu pro liché N: Tady bude muset být v úplně každém taktu jeden zahálející. Jeho sousedi si mohou spolu podat ruku, jeho sousedi ob jednoho si mohou podat ruku a tak dále, takže takto si každý podá s někým ruku.

To, kdo zrovna bude zahálet, vždy určí natočení rovnoběžek. Pokaždé bude zahálet někdo jiný, takže rovnoběžky budou natočeny pokaždé jiným směrem. Z toho plyne, že si žádný člověk nepodá ruku s nikým dvakrát, protože každý je od něj jiným směrem. Natočení se vystřídalo celkem N a každý jenom v jednom zahálel, takže si každý musel podat ruku se všemi.

Zahálel každý jenom jednou a podle stejného argumentu, jako jsme použili výše, to lépe nejde.



Martin Španěl

26-Z4-6 Překreslení obrázku

Začněme nejprve lehčí variantou. Kevin může obarvit všechny souvislé úseky černé barvy, které mají délku alespoň K. Kratší úseky neobarví nikdy, protože by tím přetáhl na bílá políčka.

Stačí tedy postupně číst vstup a pamatovat si délku aktuálního černého úseku. Pokud jsme na bílé, délka bude nulová. Jakmile nějaký černý úsek skončí, nebo nastane konec vstupu, porovnáme jeho délku s číslem K. Pokud je délka úseku větší, připočteme ji k počtu obarvitelných políček.

Celková časová složitost tohoto řešení je $\mathcal{O}(S)$, lineární s velikostí vstupního obrázku. Paměti spotřebujeme jenom konstantně, $\mathcal{O}(1)$.

Těžší varianta

Nejprve si předpočítáme, na které pozice můžeme štětec umístit, a kde bychom již přetahovali. Uděláme to tak, že pro každé políčko obrázku spočítáme velikost největšího čtverce s pravým dolním rohem v daném políčku.

Obrázek projdeme postupně po řádcích zleva doprava. Pokud jsme na bílém políčku, zapamatujeme si nulu. Pokud jsme na černém, podíváme se o jedno políčko doleva, nahoru a šikmo doleva nahoru. Na nich jsme již hodnotu spočítali dříve. Ze zapamatovaných čísel vezmeme minimum, přičteme k němu jedničku a dostaneme správný výsledek na aktuální pozici.

Například pro obrázek:

```
Č B B Č Č Č
Č Č Č Č Č B
B Č Č Č Č B
B B Č Č Č B
```

Předpočítáme:

```
1 0 0 1 1 1
1 1 1 1 2 0
0 1 2 2 2 0
0 0 1 2 3 0
```


Jak z toho ale zjistíme, kolik políček můžeme obarvit štětcem velikosti $K \times K$? Nejsnazší bude si celý obrázek překreslit a na závěr políčka jen přepočítat. Kreslení však musíme udělat chytře, abychom některá políčka nepřemalovávali mockrát a nepokazili si tím časovou složitost.

Vynulujeme všechna čísla menší než K. Tím nám zůstanou nenulová ta políčka, která můžeme obarvit pravým dolním rohem štětce.

Pro $K = 2$ jsou to pouze následující políčka:

```
0 0 0 0 0 0
0 0 0 0 2 0
0 0 2 2 2 0
0 0 0 2 3 0
```

Nyní obrázek projdeme v přesně opačném pořadí, čili z pravého dolního rohu. Při tomto průchodu všechna čísla postupně „rozšíříme“ doleva nahoru. Pokud je na aktuálním políčku nenulové číslo, tak na políčko vlevo, nahoru a šikmo vlevo nahoru napíšeme číslo o jedna nižší. Při tomto přepisování akorát nikdy nesmíme číslo snížit, vždy je zapíšeme jen pokud tím hodnotu zvýšíme.

```
0 0 0 1 1 0
0 1 1 1 2 0
0 1 2 2 2 0
0 0 1 2 3 0
```

Tím jsme celý obrázek překreslili. Stačí už jenom spočítat obarvená – nenulová políčka. Těžší variantu jsme tedy vyřešili v čase a prostoru $\mathcal{O}(RS)$, tedy lineárním s celkovým počtem políček obrázku.

Program (C):
`http://ksp.mff.cuni.cz/viz/26-Z4-6.c`

Jenda Hadrava

Pěkné prázdniny!