Korespondenční Seminář z Programování

ZAČÁTEČNICKÁ KATEGORIE

26. ročník **KSP-Z** **Květen 2014**

Přinášíme vám vzorová řešení úloh třetí série KSP-Z. Pokud jste z nějaké úlohy nedostali plný počet bodů, určitě řešení dané úlohy prozkoumejte, a i pokud jste body získali všechny, stejně doporučujeme se na řešení podívat. Často vám může poskytnout mírně odlišný náhled na problém, což je vždy ku prospěchu.

Jsme rádi za všechna vaše odevzdaná řešení a doufáme, že KSP-Z zachováte přízeň i v letošní poslední, čtvrté sérii. Kdybyste některé řešení nemohli pochopit či potřebovali vysvětlit jakýkoli detail, nebojte se nás zeptat na našem fóru nebo emailu *ksp@mff.cuni.cz*.

Řešení třetí série začátečnické kategorie 26. ročníku KSP

26-Z3-1 Zámky labyrintu

Stejně tak, jako zadání znělo jednoduše, řešení bude též jednoduché.

Nejprve si musíme rozmyslet, na jaké hodnoty bychom potenciálně měli nastavit a, b a c.

Proměnná a musí splňovat (jak bylo uvedeno v zadání) rovnost $b - a = c - b \Rightarrow a = -(c - b - b) = 2b - c$. Pro b a c vyjdeme ze stejné rovnice a vyjde nám, že $b = \frac{c+a}{2}$ (zde se jen musí ověřit, že toto číslo je celé) a $c = 2b - a$.

Tedy stačilo spočítat tyto hodnoty, a pak zjistit, kde je v absolutní hodnotě nejmenší rozdíl s původní hodnotou.

Program (C):
`http://ksp.mff.cuni.cz/viz/26-Z3-1.c`

Vojta Sejkora

26-Z3-2 Čarodějova šifra

Než jsme se mohli pustit do samotného šifrování mřížkou, bylo nutné poradit si se situací, kdy jsme měli text kratší, než kolik byla velikost mřížky ($K \times K$). Dalo by se to řešit nějakou soustavou podmínek až přímo při šifrování, ale proč si to komplikovat?

Nejjednodušší řešení jsou často ta nejlepší – prostě si na začátku doplníme text na požadovanou délku opakováním písmen „KSP“. Na technické detaily, stejně jako na detaily načítání vstupu se můžete podívat do vzorových programů na konci tohoto řešení.

Tím se dostáváme k hlavní části celé úlohy, k šifrování mřížkou. Řešme nejdříve jednodušší případ bez otáčení a pak zkusme stejný princip zkombinovat s otáčením.

Máme tedy dvourozměrné pole s mřížkou a text. Budeme si udržovat pozici v textu – jaké písmeno bychom měli zapsat dál – a budeme postupně po sloupcích a řádcích procházet dvourozměrné pole s mřížkou (vnější cyklus přes řádky, vnitřní přes sloupce, abychom šli správně zleva doprava řádek po řádku). Vždy, když narazíme na díru, zapíšeme na stejnou pozici ve výsledném dvourozměrném poli písmenko, které je zrovna na řadě (a posuneme ukazatel na další). Tak postupně projdeme celou mřížku a máme jednu čtvrtinu úkolu splněnou.

Teď by se nám líbilo mřížku otočit a to celé provést znova. Můžeme si buď celou mřížku nakopírovat a otočit, nebo můžeme transformovat souřadnice až při jejím otáčení. Je důležité si uvědomit, že když mřížku otočíme doprava (po směru hodinových ručiček), je to to samé, jako když souřadnice transformujeme na druhou stranu, doleva. Když se ptáme, co je na souřadnicích $[r, s]$ v mřížce otočené o jedna doprava, můžeme náš dotaz přeložit na ekvivalentní dotaz: co se nachází na souřadnicích otočených o jedna doleva v původní mřížce.

Ať už se rozhodneme pro jeden nebo druhý způsob, budeme potřebovat nějaký přepočet souřadnic pro otočení, neboli na jaké pozici se octne to, co bylo původně na souřadnicích $[r, s]$. Ukážeme vzorce pro otočení doprava, pro otočení doleva je to stejné, jen odzadu (jedno otočení doleva je to samé jako tři otočení doprava). Indexujeme tabulku od nuly, takže máme sloupce i řádky s indexy $0, \ldots, K - 1$.

1. otočení doprava: $[r, s] \to [s, K - 1 - r]$
2. otočení doprava: $[r, s] \to [K - 1 - r, K - 1 - s]$ (vlastně první otočení aplikované dvakrát)
3. otočení doprava: $[r, s] \to [K - 1 - s, r]$ (to samé, jen třikrát)

Teď již máme všechny potřebné stavební kameny. Stačí jen čtyřikrát provést zapsání přes mřížku a mezitím otáčet (ve vzorových programech používáme variantu s transformací souřadnic). Díky vlastnostem mřížky slíbeným v zadání jsme zapsali na každé políčko výsledné tabulky právě jeden znak a tím jsme splnili čarodějovo zadání, stačí mu tedy mřížku odevzdat (vypsat) a můžeme jít dovádět do Labyrintu snů.

Program (Python):
`http://ksp.mff.cuni.cz/viz/26-Z3-2.py`

Program (C):
`http://ksp.mff.cuni.cz/viz/26-Z3-2.c`

Jirka Setnička

26-Z3-3 Hádanka

Nejdříve si připomeneme kritérium dělitelnosti devíti, které asi všichni znají: číslo je dělitelné devíti beze zbytku právě tehdy, když ciferný součet jeho zápisu (v desítkové soustavě) je také dělitelný devíti. Například číslo 738 je dělitelné devíti právě proto, že ciferný součet $7 + 3 + 8 = 18$ devíti dělitelný je.

Než se pustíme do samotné úlohy, dovolím si drobnou otázku. Zamýšleli jste se někdy nad tím, proč toto kritérium funguje? Pokud ne, ukážeme si to. Vezmeme si naše známé číslo 738, budeme ho postupně dělit devíti a budeme sledovat přenosy mezi řády.

Rozepíšeme si číslo po řádech na $7 \cdot 100 + 3 \cdot 10 + 8 \cdot 1$ a budeme ho postupně zkoušet dělit od největšího řádu (jako

bychom popořadě dělili jednotlivé členy čísly 900, 90 a 9). Dělení 900 nám nic nezmění, ale už dělení 90 je zajímavé. To nám zruší první člen a z každé stovky nám zbude právě jedna desítka, tedy přičteme 7 k desítkám (dostáváme tak $10 \cdot 10 + 8 \cdot 1$). Když budeme dál dělit 9, zmizí nám desítkový člen a z každé desítky nám zůstane právě jedna jednička, dostaneme tedy výraz $18 \cdot 1$ a u něho už dělitelnost devíti snadno poznáme.

Asi jste si už všimli jisté pravidelnosti. Je to dáno tím, že číslo v n-ární (přesněji desítkové) soustavě dělíme číslem $n - 1$ (tedy devítkou). Z každého řádu se nám tak přenese do nižšího právě takové číslo, které v něm je. A všechny tyto přenosy se pak shromáždí v jednotkovém řádu, což je přesně ekvivalentní cifernému součtu.

Potom, co jsme si dokázali dělitelnost devíti, přejděme už k řešení samotné úlohy. V podstatě nám stačí na místa otazníků doplnit taková čísla, aby nám ciferný součet vyšel dělitelný devíti. Zároveň ale chceme, aby číslo bylo co nejmenší možné, takže chceme umisťovat co nejmenší čísla do velkých řádů.

S výjimkou pozice na začátku čísla, kde musí být alespoň jednička, můžeme jinak zkusit všude místo otazníků doplnit nuly. Pokud nám potom vyjde ciferný součet dělitelný devíti, vyhráli jsme, pokud ale ne, bude ještě nutné číslo upravit.

Stačí si ale spočítat, kolik nám schází do nejbližšího dalšího čísla dělitelného devíti, to je maximálně osm. Takže nám stačí nějaké zapsané číslo (nulu nebo jedničku) zvětšit o osm. A protože chceme výsledné číslo co možná nejmenší, provedeme to u nejméně významného řádu – u nejpravější pozice s otazníkem.

Všechno, co potřebujeme, je několikrát projít všechny cifry čísla. Takže pro číslo dlouhé N cifer zabere náš postup čas $\mathcal{O}(N)$. Nahlédněte do vzorových programů.

Program (Python):
`http://ksp.mff.cuni.cz/viz/26-Z3-3.py`

Program (C):
`http://ksp.mff.cuni.cz/viz/26-Z3-3.c`

Jirka Setnička

26-Z3-4 Tvar labyrintu

Struktuře křižovatek a cest mezi nimi se v informatice říká *graf*, a takto speciálnímu grafu bez cyklů potom *strom.* Problém ze zadání je tedy problém nalezení nejdelší cesty (trasy) ve stromu.

Představme si křižovatky a slepé cesty jako uzlíky a cesty mezi nimi jako provázky odpovídající délky. Potom celý labyrint uchopme za křižovatku s číslem 0 a nechme provázky s uzlíky volně spadnout dolů. Pokud si to dokážete představit, můžete si všimnout následujících faktů. Nejdelší cesta bude vždy končit ve slepé chodbě – pokud do křižovatky vedou alespoň dvě chodby a jednou z nich přijdeme, tak není důvod se zastavovat, když můžeme druhou odejít. Další pozorování je, že uzlík, který spadnul nejníže (je tedy nejvzdálenější od křižovatky 0), bude určitě na kraji té nejdelší cesty. Kdybychom poté celý strom chytili za tento nejhlubší uzlík a zase nechali ty ostatní volně spadnout dolů, už bychom snadno nalezli nejdelší cestu.

Jak to ale udělat algoritmicky? Nalézt nejhlubší uzlík zvládneme průchodem grafu do hloubky, problém je ale s „přetočením“ stromu pod nejhlubší uzlík. Proto si vysvětlíme snadněji naprogramovatelný postup.

K tomu si zavedeme několik pojmů, vyjdeme u nich z představy zavěšených uzlíků. *Podstrom* začínající v nějakém uzlíku u je strom obsahující tento uzlík a vše, co visí pod ním, uzlík u je *kořenem* tohoto podstromu. *Svislá cesta* je pak taková trasa, která v tomto zavěšení vede pouze dolů. Stojí za všimnutí, že libovolnou jinou cestu můžeme získat složením dvou svislých cest.

Při zpracování každé křižovatky uvažujeme pouze podstrom s kořenem v této křižovatce. Spočítat chceme dvě věci: jednak délku nejdelší svislé cesty, jednak délku nejdelší cesty procházející kořenem (tedy zpracovávanou křižovatkou).

Rekurzivně získáme délky nejdelších svislých cest vedoucích ze sousedních křižovatek, k nim přičteme vzdálenost k příslušné křižovatce. Maximum z těchto hodnot představuje délku nejdelší svislé cesty, součet dvou největších hodnot je pak délkou nejdelší cesty procházející přes kořen. Pro úplnost dodejme, že nejdelší svislá cesta vycházející ze slepé uličky má délku 0.

Pokaždé, když počítáme cestu procházející přes kořen, porovnáme navíc výsledek s globálně udržovaným maximem, a je-li větší, toto maximum upravíme. Po zpracování celého stromu v něm tak máme hledanou délku nejdelší cesty ve stromu.

Ještě bychom si měli rozmyslet složitost. Do každé křižovatky určitě přijdeme jen jednou (do každé vede shora maximálně jedna chodba). Jejím zpracováním strávíme jednak nějaký konstantní čas, jednak nějaký další konstantní čas za každou chodbu, která z křižovatky vede dolů. Všimněme si ovšem, že když do každé křižovatky vede nejvýše jedna chodba, je chodeb nejvýše N. Dohromady tak potřebujeme čas $\mathcal{O}(N)$. Podobně paměťová složitost je lineární.

Program (C):
`http://ksp.mff.cuni.cz/viz/26-Z3-4.c`

Ondra Hlavatý & Karolína „Karryanna“ Burešová

26-Z3-5 Ceny na střelnici

Pomalu, ale jistě

Hledáme v zadané posloupnosti a co nejdelší úsek, ve kterém jsou všechny prvky různé. Zkusíme nejdřív najít nejdelší takovýto úsek, který začíná na prvním prvku, tedy zcela vlevo.

To je velice jednoduché, stačí si nějak v paměti udržovat, které všechny různé prvky jsme už potkali. Pokud budeme mít druhy cen označené přirozenými čísly, tak se na to skvěle hodí pole. Vytvořme si pole c, ve kterém na začátku budeme mít všude nuly (to znamená, že jsme zatím nic nenavštívili), a pokud potkáme cenu označenou i, napíšeme do c nějaké nenulové číslo na $c[i]$.

No a pokud budeme chtít zapsat na místo, kde už ale něco nenulového je, znamená to, že jsme právě přečtenou hodnotu potkali už podruhé. Takže úseky od začátku na místo, na které se budeme dívat, od teď budou určitě obsahovat dvojici stejných prvků.

Ale tím nesmíme ukončit hledání, ještě je možnost, že existuje nějaký vyhovující úsek začínající někde dál než na prvním prvku. Nejjednodušší způsob by byl použít stejný algoritmus od druhého prvku, potom od třetího, potom od čtvrtého a tak dále až do N-tého. To by ale trvalo moc dlouho – algoritmus spustíme N-krát a může zkoumat až

N prvků, takže časová složitost bude $\mathcal{O}(N^2)$. Pro první úkol ale stačí, protože pokaždé algoritmus skončí po nejvýše padesáti krocích (delší úsek různých cen nemůže být). Proto bude časová složitost $\mathcal{O}(50N) = \mathcal{O}(N)$.

Optimalizace

My ho ale dokážeme výrazně zrychlit jednoduchým trikem: Když budeme do pole chtít napsat, že jsme potkali nějakou cenu x na pozici i, na $c[x]$ zapíšeme hodnotu i. Tedy o druhu ceny x budeme vědět nejen, že jsme ho potkali, ale i kde to naposledy bylo.

Algoritmus bude postupovat tak, že bude procházet posloupnost cen a u každé zjistí, kde začíná nejdelší úsek různých cen, který v ní končí. Pokud prozkoumáme novou cenu, tento začátek nejdelšího úseku zůstane buď stejný jako u předchozí, nebo se posune někam směrem doprava.

Když přijdeme k nějaké ceně x na pozici i, zjistíme, jestli je její poslední výskyt před nebo za začátkem nejdelšího úseku končícího na pozici $i-1$. Pokud je před ním, můžeme ho ignorovat a začátek bude pro i stejný jako pro $i-1$. Pokud bude za ním, posune se tím začátek nejdelšího úseku doprava těsně za poslední výskyt.

Tímto způsobem projdeme postupně celou posloupnost a budeme si udržovat zatím nejlepší délku úseku, jakou jsme dosud viděli. Spolu s ní si zapamatujeme i indexy jejího začátku a konce. Ty přepíšeme, jakmile potkáme nějakou lepší.

Proč to celé funguje?

Sluší se ještě dokázat, že jsme žádnou ještě lepší validní posloupnost nepřehlédli. Na každém prvku posloupnosti jsme znali nejdelší posloupnost, která na tomto místě končí (kdybychom se pokusili její začátek posunout o jedno místo doleva, objevila by se mezi prvky posloupnosti nějaká dvojice – právě na takové místo jsme začátek posouvali).

Konec posloupnosti jsme v průběhu algoritmu posouvali postupně o jedničku, proto, ať by nejlepší validní podposloupnost končila kterýmkoliv prvkem, tak bychom přes něj museli přejít a podposloupnost bychom tak našli.

Jak dlouho to celé poběží? Uděláme N kroků, kde krok je všechno, co uděláme mezi jednotlivými posunutími konce posloupnosti. Tam ale uděláme vždy konstantní počet operací, takže časová složitost bude $\mathcal{O}(N)$.

Program (Python):
`http://ksp.mff.cuni.cz/viz/26-Z3-5.py`

Martin Španěl

26-Z3-6 Horská dráha

Hlavním nástrojem, který využijeme v této úloze, je třídění. O rychlých třídících algoritmech se dočtete třeba v naší kuchařce.[1]

V prvním úkolu stačí všechny hodnoty, které si Kevin zapsal, vzestupně setřídit. Poté budeme zleva doprava hodnoty procházet a udržovat si přitom počítadlo, které bude obsahovat délku souvislého úseku složeného ze stejných hodnot, ve kterém se právě nacházíme. Pokud je následující hodnota stejná jako předchozí, počítadlo o jedna zvětšíme, jinak jej vynulujeme. Zároveň si v průběhu udržujeme dosavadní maximální hodnotu počítadla, kterou stačí aktualizovat vždy před vynulováním a pak na konci.

Výpočet se skládá ze dvou fází: třídění a následný průchod. První fáze nás stojí při použití rychlého třídícího algoritmu $\mathcal{O}(N \log N)$ času, druhá fáze už jen $\mathcal{O}(N)$, takže celková časová složitost tohoto algoritmu je pak $\mathcal{O}(N \log N)$.

Budeme předpokládat, že se horská dráha chová rozumně spojitě, tedy pokud jsme přejeli z výšky a do výšky b, ocitli jsme se přitom jednou v každé výšce mezi a a b. Navíc úsek mezi každými dvěma Kevinovými zápisy je ze zadání celý z kopce nebo celý do kopce, takže každý úsek mezi dvěma zápisy můžeme popsat intervalem, jehož krajními hodnotami jsou ony zapsané hodnoty.

Rádi bychom nyní řekli, že hledaná výška je taková, která je obsažena v nejvíce intervalech. To je ovšem zrádné, protože místa, ve kterých si Kevin zapisoval výšku, jsou zachycena ve dvou intervalech. Lokální výškové extrémy tedy určitě nebudou dobrými kandidáty na hledanou nejčastěji navštěvovanou výšku. Představme si třeba obyčejnou sinusoidu, na které by se Kevin vyskytl ve výšce 0 mnohem častěji než ve výšce 1. Raději bychom proto počítali s otevřenými intervaly.

Učiníme pozorování, které nám to umožní: představme si, že jsme již nalezli onu výslednou výšku, ale vyskytuje se na seznamu, tedy je koncovým bodem některých intervalů. Bez újmy na obecnosti předpokládejme, že se tato výška vyskytuje v nejvýše tolika lokálních údolích jako vrcholech (jinak pokračujeme obráceně). Pak ovšem můžeme tuto výšku zmenšit o dostatečně malinké číslo, čímž sice už nebude v údolích, ale za každý vrchol teď budeme mít dva výskyty této výšky v jeho blízkém okolí. Jeden bude nalevo a jeden napravo. Celkový počet výskytů se tedy nezmenšil a dostali jsme výšku, která leží pouze v otevřených intervalech (volbou dostatečně malého zmenšení jsme se mohli všem ostatním lokálním extrémům vyhnout).

Nyní nám stačí pracovat s otevřenými intervaly a budeme postupovat podobně jako v první úloze. Vezmeme si všechny konce intervalů, přidáme ke každému konci příznak, zda je to konec levý nebo pravý, a všechny konce vzestupně setřídíme.

Setříděné pole budeme procházet zleva doprava a udržovat si, v kolika jsme zrovna intervalech. Pokud narazíme na levý konec, počítadlo o jedna zvýšíme, pokud narazíme na pravý, o jedna jej zmenšíme. Přitom si udržujeme maximální hodnotu počítadla tentokrát ještě navíc s intervalem, ve kterém jí bylo dosaženo. Ten vznikne jako průnik nějakých intervalů, při výpočtu jej však snadno dostaneme jako největší levý a nejmenší pravý konec, který aktuálně máme.

Nakonec máme interval (a, b), který reprezentuje výšky, ve kterých jsme se nejčastěji objevili. Stačí tedy vypsat libovolnou z nich, třeba $(a+b)/2$, což je určitě číslo z vnitřku intervalu.

Časová složitost tohoto algoritmu je stejná jako v prvním úkolu, tedy $\mathcal{O}(N \log N)$.

Program C++:
`http://ksp.mff.cuni.cz/viz/26-Z3-6.cpp`

Mark Karpilovskij

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

Výsledková listina třetí série začátečnické kategorie 26. ročníku KSP

	řešitel	*škola*	*ročník*	*sérií*	*Z3-1*	*Z3-2*	*Z3-3*	*Z3-4*	*Z3-5*	*Z3-6*	*série*	*celkem*
0.					8	10	10	12	12	14	66,0	198,0
1.	Jan Tománek	GPelhřimov	3	3	8	10	10	12	12	11	63,0	193,0
2.	Jakub Pelc	G_UherBrod	0	3	8	10	10	12	12	2,5	54,5	183,5
3.	Václav Fabík	ZŠKřídloBO	−1	3	8	10	10	12	12		52,0	182,0
4.	Miroslav Šerý	GValašKlob	1	3	8	10	10	12	12	5	57,0	180,5
5.	Jonáš Malena	SŠJeštědLI	4	3	8	10	10	12	2	3	45,0	164,5
6.	Přemysl Šťastný	GŽamberk	0	3	8	0	10		2	3	23,0	139,0
7.	Václav Končický	GSOŠ_FrMís	3	3	8	10	10				28,0	132,0
8.	František Zajíc	G_Nymburk	1	3	0	0					0,0	102,0
9.	Lucie Studená	GKepleraPH	4	2							0,0	95,0
10.	Michal Töpfer	G_DrJPekMB	1	2	8	10	10	1	2	4	35,0	88,0
11.	Jakub Lukeš	GNAlejíPH	1	3	0						0,0	81,0
12.	Pavel Mikuš	GMělník	3	3	8	10	10	6			34,0	80,0
13.	Antonín Teichmann	GJeronýmLI	4	2							0,0	72,0
14.	Petr Šíma	GKlatovy	1	3	8		10				18,0	66,0
15.	Jiří Vozár	G_UherBrod	2	1							0,0	59,5
16.	Michal Převrátil	GKlatovy	1	2							0,0	58,0
17.	Marek Vitula	GJarošeBO	3	2							0,0	51,0
18.–19.	Jan Burda	G_Holice	−1	2	8	10	10		3		31,0	48,0
	Jakub Heyduk	SŠP_ČB	4	2							0,0	48,0
20.	Antonín Bruštík	G_UherBrod	3	3	8	10	10	1			29,0	46,0
21.	Václav Trpišovský	GOpenGaBab	−3	3					5		5,0	44,0
22.	Jakub Tětek	CírkG_Plzeň	0	1	8	10	7	4	12	0	41,0	41,0
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	7	10	10	12	0		39,0	39,0
	Radovan Švarc	G_ČTřebová	3	2							0,0	39,0
27.	Daniel Šerý	G_RožnovPR	2	2	0	8	6		4	3	21,0	37,0
28.–29.	Jan Horák	GŠumperk	3	3	8		10				18,0	34,0
	Viktor Kovařík	G_UherBrod	3	3	8	10	0				18,0	34,0
30.	Milan Malina	GMikulášPL	1	2							0,0	30,0
31.	Jan Vargovský	GSPŠFrenšt	4	1							0,0	25,0
32.–33.	Ivana Krumlová	GJarošeBO	1	2	8		7				15,0	23,0
	Vojtěch Václavík	GSOŠ_FrMís	4	2							0,0	23,0
34.–35.	Tomáš Michna	SPŠEOstrava	4	2	0						0,0	20,0
	Benedikt Žour	G_UherBrod	−1	2	0				2		2,0	20,0
36.	Janek Hlavatý	ZŠ_DukelČB	−5	3	8		1				9,0	19,0
37.–38.	Štěpán Košan	GKlatovy	2	1							0,0	18,0
	Aneta Šťastná	GOmskPha	4	1							0,0	18,0
39.	David Karlík	G_UherBrod	3	2							0,0	16,0
40.	Zdeněk Pavlátka	GMikulášPL	2	2	0	0					0,0	15,5
41.	Martin Jílek	GKlatovy	2	1							0,0	13,0
42.–44.	Ivona Hrivová	GŽilina	4	3			0				0,0	11,0
	Dominik Krasula	GKrnov	1	1							0,0	11,0
	Jan Vozár	G_UherBrod	0	2							0,0	11,0
45.	Michaela Bačová	G_UherBrod	3	2							0,0	9,0
46.	David Dvořáček	G_UherBrod	3	1							0,0	8,0
47.	Petr Pacner	GBroumov	2	1							0,0	7,0
48.	Tereza Bohumská	GPísnickáPH	−1	1							0,0	2,0
49.	Majtán Martin	G_SNPPiešť	1	1							0,0	1,0