Korespondenční Seminář z Programování

SOUTĚŽ KASIOPEA

27. ročník | **Vzorová řešení** | **Březen 2015**

Vážení a milí čtenáři, předkládáme vám autorská řešení úloh čtvrtého ročníku internetové soutěže Kasiopea. Veškeré informace o soutěži včetně výsledků, zadání a vzorových zdrojových kódů naleznete na stránce soutěže:
http://ksp.mff.cuni.cz/akce/kasiopea/2015/

V případě nejasností kolem řešení úloh či jiných podnětů nám neváhejte napsat na adresu *kasiopea@ksp.mff.cuni.cz*.

27-K-1 Po škole — 10 bodů

K této úloze snad ani není co psát. Stačilo jen každý řádek projet znak po znaku a připočíst jej do správné třídy znaků a na závěr vypsat výsledek. Časová složitost $\mathcal{O}(\textit{počet znaků na vstupu})$.

Karel Tesař

27-K-2 Kostičky — 10 bodů

Zamyslíme se, jak se mění počet odhalených stěn při postupném přidávání kostiček. Položíme-li kostičku přímo na stůl tak, aby se nedotýkala (myšleno celou stěnou, rohem se dotýkat může) žádné další kostičky, zvýší se počet odhalených stěn o pět.

Položíme-li kostičku na stěnu jiné kostičky, aniž by se dotýkala nějaké další kostičky, zvýší se počet odhalených stěn o čtyři (přibude pět nových odhalených stěn, ale ztratíme jednu stěnu kostičky, na kterou pokládáme).

Jiný způsob jak navýšit počet odhalených stěn o pět než ten jmenovaný neexistuje. Rozmístit co nejvíce kostiček tak, aby přidaly pět stěn a zbylé tak, aby přidaly čtyři stěny, je tudíž to nejlepší, co můžeme mít. Z toho již plyne, že jednou z možností, jak optimálně rozmístit kostičky, je položit nejvyšší možný počet kostiček na stůl tak, aby se nedotýkaly, a další kostičky pokládat na tyto kostičky.

Snadno si ověříte, že nejvyšší počet kostiček, které lze umístit na stůl, aniž by se dotýkaly, lze vypočíst jako $\lceil \frac{RS}{2} \rceil$, kde $\lceil x \rceil$ označuje horní celou část čísla x, tedy nejmenší celé číslo z splňující $z \geq x$.

Lukáš Folwarczný

27-K-3 Obdélníky — 10 bodů

Cestu k řešení zahájíme dvěma drobnými pozorováními. Obdélníky, které mají některou ze stran nulové délky, nezabírají žádnou plochu a nemusíme se jimi zabývat. Dále si všimneme, že obdélníky, které mají svůj druhý roh v odlišných kvadrantech roviny, se překrývají nulovou plochou. Tudíž můžeme úlohu vyřešit pro každý kvadrant roviny zvlášť. Ve zbytku řešení tedy bez újmy na obecnosti předpokládáme, že

každý ze zadaných obdélníků má jeden svůj roh na souřadnicích $(0, 0)$ a druhý na souřadnicích (x_i, y_i), přičemž platí $x_i, y_i > 0$.

Naší úlohou je nyní spočítat obsah plochy pokryté těmito obdélníky. Obdélníky si setřídíme sestupně podle jejich výšky (to jest podle hodnoty y_i). V průběhu výpočtu si budeme pamatovat hodnotu x_z, která značí, že nám zbývá započítat pouze obsah té části plochy, která leží na části roviny s $x \geq x_z$. Na začátku položíme $x_z = 0$, protože potřebujeme spočítat obsah celé plochy.

Nyní zpracováváme obdélníky tak, jak jsme si je uspořádali, to znamená v sestupném pořadí podle jejich výšky. Nechť právě zpracovávaný obdélník má svůj pravý horní roh v bodě (x_i, y_i). V případě, kdy $x_i \leq x_z$, nemusíme se obdélníkem zabývat, protože leží celý ve zpracované části roviny. V případě, kdy $x_i > x_z$, můžeme k celkovému obsahu bez obav přičíst $(x_i - x_z) \cdot y_i$ a položit $x_z = x_i$, protože je tento obdélník nejvyšším z těch obdélníků, které zasahují do intervalu x-ových souřadnic $[x_z, x_i]$.

Složitost našeho algoritmu je dána složitostí třídění, což při použití standardních třídicích algoritmů dává $\mathcal{O}(N \log N)$. V řešení si krom vstupu pamatujeme pouze konstantní počet proměnných, paměťová složitost je tedy lineární. Pokud jste se s tříděním ještě nesetkali, nahlédněte do příslušné kuchařky.[1]

Lukáš Folwarczný

27-K-4 Školní exkurze — 10 bodů

Než začneme plnit autobus, uděláme si malou analýzu našeho grafu. Ten se skládá z vrcholů o výstupním stupni právě jedna. Pokud se vydáme z jednoho vrcholu směrem po hranách (vždy máme jen jednu možnost, kam jít), určitě skončíme v nějakém cyklu. Graf se tedy skládá z cyklů, ke kterým můžou být přilepeny nějaké „ocasy“. Cykly mohou mít i velikost 1.

Při přidávání dětí (vrcholů) do autobusu, vždy musíme vzít celý cyklus a případně můžeme přidat libovolný počet vrcholů z ocasů, které z tohoto cyklu vedou. Taková úloha nápadně připomíná problém batohu. Máme batoh (autobus) o kapacitě K a chceme jej co nejvíce naplnit věcmi $[a_1, b_1], \ldots, [a_k, b_k]$, kde zápis $[a_i, b_i]$ znamená že máme věc velkou a_i, kterou můžeme nafouknout až do velikosti b_i (to jsou naše cykly s ocasy).

Tento problém již vyřešíme jako klasický problém batohu. Vytvoříme si pole velikosti $K + 1$, kde jedničkou budou označeny kapacity, které umíme vytvořit. Na začátku umíme vytvořit jen kapacitu o velikosti nula. Věci budeme jednu po druhé přidávat. S každou věcí od konce zkontrolujeme každou kapacitu c, zda na ni umíme navázat pomocí věci a_i, tedy zda je na pozici $c - a_i$ jednička. Pokud ano, zapíšeme jedničku pro kapacitu c a začneme věc nafukovat. Tj. přidáváme jedničky až do kapacity $\min(c - a_i + b_i, K))$. A to je celé.

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

Časová složitost algoritmu je $\mathcal{O}(K \cdot N)$, protože pro každou kapacitu autobusu a každý vrchol děláme v batohu maximálně jednu aktualizaci – pro každý vrchol a jednu kapacitu buď přidáváme novou jedničku, a nebo právě jednou nafukujeme.

Karel Tesař

27-K-5 Cyklojízda — 10 bodů

Na první pohled můžeme vidět, že cílem je najít v grafu uzavřený *Eulerovský tah*, který má všechny prefixové součty ohodnocení hran nezáporné. Takže první, co uděláme, je kontrola sudosti všech stupňů a že součet všech hran dohromady je nezáporný. Pokud některá z těchto podmínek není splněna, automaticky odpovídáme, že řešení neexistuje.

Dalším krokem bude nalezení libovolného Eulerovského tahu. V tom zatím není splněna podmínka nezápornosti všech prefixových součtů, to teprve zařídíme. Spočítáme si všechny prefixové součty hran námi nalezeného Eulerovkského tahu a vybereme z nich nejmenší hodnotu. Celý Eulerovský tah pak pootočíme tak, aby pozice této hodnoty byla na začátku a takový Eulerovský tah prohlásíme za řešení.

Jelikož součet všech hran je nezáporný a my začali v místě, kde průběžný součet předtím byl minimální, máme zaručeně nezáporný součet na každé pozici tahu. Tím jsme hotovi.

Časová složitost je rovna složitosti hledání Eulerovského tahu, která je $\mathcal{O}(n + m)$. Závěrem jen stručně naznačíme, jak se takový Eulerovský tah hledá.

Graf si budeme reprezentovat seznamem hran, přičemž každý vrchol bude mít svůj seznam odkazů na hrany, s kterými je incidentní. Graf budeme prohledávat do hloubky z libovolného vrcholu. Do vrcholů vstupujeme opakovaně, do hran nikoliv. Během průchodu si každý vrchol pamatuje pozici, kde jsme v jeho seznamu hran skončili, a když se v něm ocitneme příště, začínáme právě tam (ale přeskakujeme již navštívené hrany).

Samotný Eulerovský tah nám dá pořadí opouštění vrcholů v rámci průchodu. Tedy hrany kreslíme právě, když se v průchodu z vrcholu vracíme. Během prozkoumávání, nekreslíme nic. Proč toto funguje? Díky sudosti stupňů – první vrchol, ve kterém se dostaneme do slepé uličky, bude startovní vrchol a poslední vrchol, který prohledávání opustí, bude také startovní vrchol. Začátek a konec tahu tedy sedí.

Dále všechny vrcholy, které už jsme někdy v průchodu opustili, již nikdy znova nemůžeme najít (myšleno v jiné větvi průchodu), protože jinak bychom v nich předtím našli další hranu a tedy je neopustili. To znamená, že další vrchol, ve kterém narazíme na slepou uličku je vrchol, ve kterém jsme naposled přestali kreslit (opět díky sudosti stupňů, je to jediný vrchol s lichým stupněm nenavštívených hran, který jsme zatím nikdy neopustili). Tím máme zaručeno, že se nám kreslené části na sebe hezky napojují.

Více detailů můžete najít ve vzorovém kódu.

Karel Tesař

27-K-6 Obchodníci — 10 bodů

Poslední úloha šla řešit technikou, které se často říká „dynamic programming over subsets“. Ta vznikla kombinací dvou technik - dynamického programování, a reprezentace nějaké množiny objektů pomocí jednoho celého čísla. Reprezentovat množinu celým číslem X můžeme jednoduše - i-tý bit X bude 0 tehdy, když číslo i do množiny nepatří, a 1 tehdy, když číslo i do množiny patří. Bity číslujeme od 0.

Povšimněme si, že s touto reprezentací se velice dobře pracuje - veškeré operace s množinou dokážeme převést na nějakou kombinaci bitových operací. Nám bude stačit sjednocení množin X a Y (X or Y, $X|Y$), průnik množin (X and Y, $X\&Y$), vytvoření množiny obsahující pouze prvek k (1 shl (bitový posun vlevo) k, $1 << k$ v C) a vytvoření množiny co obsahuje všechny prvky z rozsahu $0 \ldots k-1$ ($(1 << k)-1$ - proč?). Povšimněte si také, že využitím těchto operací můžeme elegantně projít všechny podmnožiny množiny $\{0, 1, \ldots, n\}$ jedním cyklem, který půjde od 0 do $1 << n$ nevčetně, a na aktuální číslo se bude dívat jako na množinu X.

Nyní zpět k úloze. U úloh na dynamické programování často pomáhá napsat nějakou rekurzivní funkci, kterou pak vhodně upravíme, aby jednou spočítané hodnoty nepočítala neustále znova a znova. Představme si, že postupně procházíme nabídky obchodníků, a u každého přemýšlíme, jestli zrovna u něj něco koupíme. Jaké máme možnosti? Buď obchodníka zcela vynecháme, nebo vezmeme nějaký ze zatím nenakoupených dárků, zaplatíme za něj jeho cenu i s dopravou, a buď ještě něco přikoupíme (už bez dopravy), nebo se přesuneme o řádek níže k nabídce dalšího obchodníka.

Přesně takhle bude úlohu řešit ona rekurzivní funkce f. Bude mít tři parametry, aktuálního obchodníka, jehož nabídku zkoušíme, aktuální množinu nakoupených dárků, a jestli už jsme u aktuálního obchodníka platili za dopravu. Jak se tedy $f(o, X, z)$ bude chovat? Zeptá se nejdříve na hodnotu $f(o+1, X, 0)$, ta odpovídá nejmenší ceně, pokud vynecháme obchodníka s číslem o. Poté se podívá na každý nenakoupený dárek d (nulové bity v čísle X), a uváží, jaká je nejmenší cena, jak nakoupit zbytek dárků za předpokladu, že dárek d vezmeme u obchodníka o. Jak jí spočítá? Vezme cenu dárku d, pokud je z nulové, přičte k ní cenu za dopravu u obchodníka o, a vezme nejmenší cenu, jak nakoupit zbytek - o to se postará rekurze, pokud se zeptáme na $f(o, X|(1 << d), 1)$.

To je řešení, které je příliš pomalé, ale již je velice blízko zisku plného počtu bodů. Stačí přidat jediné - kdykoliv spočítáme hodnotu $f(o, X, z)$, poznamenáme si ji do pomocného pole, a příště ji již počítat nemusíme. Více viz zdrojový kód.

Touto drobnou úpravou dostaneme řešení s časovou složitostí $\mathcal{O}(nm2^m)$, protože pro každou trojici parametrů (kterých je $n2^m \cdot 2$) počítáme hodnotu funkce f nejvýše jednou, a v každém výpočtu f máme cyklus přes m dárků. Paměťová složitost je $\mathcal{O}(n2^m)$.

Ondřej Hübsch