A-III-4 Tučný Santa

Hľadanie najkratšej cesty v grafe

Vždy, keď sa vás zadanie úlohy pýta na najmenší počet krokov, na ktorý sa dá nejaký cieľ dosiahnuť, mali by sa vaše myšlienky okamžite vydať nasledujúcim smerom: Podľa situácie v úlohe si zostrojíme vhodný graf a v tomto grafe nájdeme najkratšiu cestu. Graf vznikne zakaždým rovnako: jeho jednotlivé vrcholy zodpovedajú stavom, v ktorých sa môžeme nachádzať, a hrany vychádzajúce z vrcholu zodpovedajú ťahom, ktoré môžeme v danej situácii spraviť.

Ako to bude vyzerať v našom prípade? Stavy, teda vrcholy grafu, budú pochopiteľne jednotlivé pozície, na ktorých sa Santa môže nachádzať. Presnejšie, budeme uvažovať pozície, na ktorých sa môže nachádzať ľavý horný roh Santu – tým je jeho poloha jednoznačne určená. Každý vrchol vieme jednoznačne popísať dvomi číslami: jeho súradnicami. No a keďže $r_1, s_1 \leq 2000$, bude náš graf mať nanajvýš 4 milióny vrcholov.

Z každého vrcholu povedú hrany zodpovedajúce krokom, ktoré v tej chvíli vie Santa spraviť. Keďže máme na výber len 4 smery pohybu, budú z ľubovoľného vrcholu vychádzať najviac 4 hrany.

Na to, aby sme vedeli, aké hrany v našom grafe máme, potrebujeme zistiť, na ktorých pozíciách môže Santa stáť a na ktorých (kvôli prekážkam) stáť nemôže. Týmto sa budeme zapodievať neskôr, nateraz sa tvárme, že to už vieme zistiť, a vráťme sa späť k nášmu grafu.

Keď už sme zostrojili graf, zostáva v ňom nájsť najkratšiu cestu. Na to použijeme algoritmus nazvaný prehľadávanie do šírky (breadth-first search, BFS, niekedy tiež nazývané flood fill). Toto prehľadávanie predstavuje spôsob, akým by daný graf preskúmala voda, šíriaca sa zo začiatočného vrcholu rovnomerne do všetkých smerov: najskôr spracujeme začiatočný vrchol (vzdialenosť doň je 0), potom postupne všetkých jeho susedov (do každého z nich je vzdialenosť 1), potom susedov jeho susedov (vzdialenosť 2), a tak ďalej.

Prehľadávanie do šírky ľahko implementujeme pomocou dátovej štruktúry fronta:

```
prehľadaj(vrchol v):
    označ v ako navštívený
    vzdialenosť do v = 0
    vyrob frontu Q obsahujúcu jediný prvok v
    kým Q nie je prázdna:
        vyber zo začiatku fronty Q prvok x
        pre každú hranu vedúcu z x:
            ak jej koncový vrchol w nie je navštívený:
                označ w ako navštívený
                vzdialenosť do w = 1 + vzdialenosť do v
                zapamätaj si, že do w som sa dostal z x
                vlož w na koniec fronty Q
```

Časová aj pamäťová zložitosť tohto riešenia je lineárna od veľkosti plochy obývačky, teda $O(r_1 s_1)$. Prečo to tak je? V prvom rade si uvedomme, že máme $r_1 s_1$ vrcholov a najviac $4r_1 s_1$ hrán, teda dokopy máme $O(r_1 s_1)$ vrcholov a hrán. Každý vrchol najviac raz vložíme do fronty (v okamihu, keď sa nám doň prvýkrát podarí dostať) a najviac raz ho z nej potom vyberieme. Každú hranu tiež spracujeme najviac raz: vtedy, keď spracúvame vrchol, z ktorého vedie. Celkový počet krokov, ktoré náš algoritmus spraví, je preto priamo úmerný počtu vrcholov a hrán v našom grafe.

V našej implementácii si nepotrebujeme samostatne značiť, ktoré vrcholy už sú navštívené: poznáme to jednoducho tak, že akonáhle má vrchol vzdialenosť inú ako nekonečno, už sme ho niekedy navštívili.

Rekonštrukcia cesty

Vtejto úlohe pribudla ešte jedna zákernosť a tou je vypisovanie samotnej cesty. Táto zákernosť sa dá vyriešiť napríklad tým, že si pre každé políčko zapamätáme, odkiaľ sme sa naň počas prehľadávania prvýkrát dostali. Keďže vieme, kde Santa skončil, môžeme následne pomocou týchto údajov od konca zrekonštruovať jeho cestu.

Predpočítanie

Skôr než začne BFS, potrebujeme zistiť, ktoré pozície Santu sú navštíviteľné. Zo zadania vieme, že sú to také pozície, na ktoré keď sa Santa postaví, prekryje najviac M prekážok.

Do tabuľky T si chceme na políčko $T[y, x]$ zapísať, koľko prekážok je v odĺžniku $[y, x]$ až $[y+r_2-1, x+s_2+1]$. Ak $T[y, x] \leq M$ Santa môže stáť na pozícii y, x.

Jednoduché riešenie je, pre každé políčko prezrieť $r_2 \times s_2$ políčok vľavo dole od neho a spočítať si prekážky. Takéto riešenie dokopy spraví $\Theta(r_1 s_1 r_2 s_2)$ krokov, čo je pre veľké vstupy priveľa.

Pri vymýšľaní rýchlejšieho riešenia sa skúsme najskôr pozrieť na jednorozmerný prípad. (Samo zadanie nám ponúka radu – 6 bodov môžeme získať, ak predpokladáme, že Santa má rozmery $1 \times s_2$.) V jednorozmernom prípade nám stačí spočítať, koľko prekážok je na políčkach $[y, x]$ až $[y, x + s_2 - 1]$. To si vypočítame postupne pre každé x:

```
T[y,0] = počet prekážok od [y,0] po [y,s2-1]
T[y,1] = T[y,0] + (1 ak [y,s2] je 'X') - (1 ak [y,0] je 'X')
...
T[y,i] = T[y,i-1] + (1 ak [y,i+s2-1] je 'X') - (1 ak [y,i-1] je 'X')
..
T[y,s1-s2] = T[y,s1-s2-1] + (1 ak [y,s1-1] je 'X') - (1 ak [y,s1-s2-1] je 'X')
```

Výborná vec na celom tomto je rýchlosť. Výpočet $T[y, 0]$ nám síce trvá s_2 krokov, ale potom už každé zo zvyšných $s_1 - s_2$ políčok vypočítame v konštantnom čase, celkovo teda spravíme pri spracovaní jedného riadku len $O(s_1)$ krokov.

No a ďalej to už je veľmi jednoduché: Stačí si uvedomiť, že dvojrozmerný prípad vyriešime tak, že spravíme ešte raz to isté, ale tentokrát po stĺpcoch (pričom použijeme hodnoty, ktoré sme práve vypočítali prechodom po riadkoch). Toto predpočítanie dokážeme spraviť v čase $O(r_1 s_1)$. To isté platí aj pre BFS. Takže dostávame riešenie so zjavne optimálnou časovou zložitosťou $O(r_1 s_1)$. Pamäťová zložitosť je tiež $O(r_1 s_1)$.

(Kvôli úspore miesta listing programu uvedieme až v elektronickej verzii vzorových riešení.)

Alternatívne riešenie

Namiesto predpočítania vyššie uvedeným spôsobom sa dajú priamo použiť dvojrozmerné prefixové súčty: pre každé y, x predpočítame počet prekážok $P[y, x]$ v obdĺžniku, ktorý má v protiľahlých rohoch políčka $[0, 0]$ a $[y - 1, x - 1]$. Toto vieme vhodným trikom spraviť v $O(r_1 s_1)$. No a z týchto hodnôt už vieme v konštantnom čase o ľubovoľnom obdĺžniku povedať, koľko prekážok obsahuje.

```
predpočítanie:
    P[y,x] = P[y-1,x] + P[y,x-1] - P[y-1,x-1] + (1 ak [y-1,x-1] je 'X')
počet prekážok v obdĺžniku od [y1,x1] po [y2-1,x2-1]:
    počet = P[y2,x2] - P[y2,x1] - P[y1,x2] + P[y1,x1]
```

A-III-5 Ministerstvo

Cieľom je samozrejme nájsť riešenie, ktoré bude efektívnejšie ako priamočiara simulácia každého príkazu. Štruktúra ministerstva, popísaná na vstupe, predstavuje strom. A keďže na všeobecnom strome sa predpísané operácie robia ťažko, prvým krokom riešenia bude prevedenie úlohy na trochu inú – na operácie nad intervalmi a prvkami v poli. Každému vrcholu stromu teda priradíme jeden prvok poľa a spravíme to tak, aby každému oddeleniu zodpovedal súvislý úsek poľa. Každý príkaz sa potom bude týkať buď jedného prvku poľa, alebo nejakého intervalu.

Vhodné priradenie pozícií v poli vrcholom strimu vieme nájsť ľahko: stačí napr. ľubovoľným spôsobom prehľadať strom do hĺbky a každému vrcholu priradiť index rovný poradiu, v ktorom je objavený. (Vyhovujúcich očíslovaní je veľa, toto je len jedno z nich, ktoré je ľahko a systematicky zostrojiteľné.)

Počas prehľadávania si zároveň vieme pre každý vrchol zapamätať, kde začína a končí interval, ktorý obsahuje jeho podriadených. (Pri prehľadávaní do hĺbky interval zodpovedajúci vrcholu v začína ním samotným. Koniec intervalu zistíme vtedy, keď prehľadávanie opúšťa vrchol v.)

Túto časť riešenia vieme teda celú ľahko spraviť v čase $O(n)$.

V tejto chvíli teda už môžeme na strom zabudnúť, máme už len obyčajné pole. Aby sme vedeli vykonávať príkazy zo zadania, potrebujeme s ním efektívne robiť nasledovné operácie:

- Zvýš hodnotu prvku a o k
- Zisti súčet v intervale $[a, b]$.
- Zvýš hodnotu všetkých prvkov v intervale $[a, b]$ o k.
- Zisti minimum v intervale $[a, b]$.
- Zmeň hodnotu všetkých prvkov v intervale $[a, b]$ na k.

Intervalové stromy

Začnime efektívnou implementáciou prvých dvoch operácii. (Za to vieme získať 9 bodov. Stačí navyše doplniť, že vždy, keď budeme potrebovať zvýšiť plat celému oddeleniu, spravíme to hrubou silou – prejdeme všetkých jeho zamestnancov a každému zvýšime plat.)

Nad poľom si postavíme binárny strom. Hodnota každého vrchola bude súčet hodnôt jeho synov, a teda zároveň súčet v celom intervale poľa, ktorý je pokrytý týmto vrcholom.

Zmenu hodnoty implementujeme jednoducho: zmeníme hodnotu v poli a potom postupne ideme dohora po strome a upravujeme hodnotu všetkých predkov daného vrchola. Časová zložitosť je úmerná hĺbke tohoto stromu, čiže $O(\lg n)$. Zistenie súčtu v úseku bude trochu komplikovanejšie, ale rovnako rýchle. Potrebujeme sčítať vrcholy stromu, ktoré inteval prekrýva celé a neprekrýva ich otcov. Takýchto vrcholov bude maximálne $O(\lg n)$ – uvedomte si, že z každej úrovne sčítame maximálne dva vrcholy. Nájdeme ich jednoduchou rekurzívnou procedúrou, ktorá prezerá strom zhora nadol:

```
zisti_sucet(vrchol x, sčitovaný interval I):
  Ak je interval I disjuktný s intervalom, ktorý pokrýva podstrom vo vrchole x:
    vráť 0
  Ak je interval I nadmnožinou intervalu, ktorý pokrýva podstrom vo vrchole x:
    vráť súčet vo vrchole x
  vráť zisti_sucet(ľavý syn x, I) + zisti_sucet(pravý syn x, I)
```

Tento prechod má tiež časovú zložitosť $O(\lg n)$. Analogicky, pomocou rekurzie, vieme implementovať aj prvú operáciu: začneme v koreni stromu, zídeme dole na správne políčko poľa, to upravíme a pri vynáraní sa z rekurzie prepočítavame hodnoty v jeho predkoch. Pri tomto riešení to ešte je jedno, ale práve takáto rekurzívna implementácia bude potrebná pri modifikáciách popísaných v nasledujúcej časti riešenia.

Lenivé vykonávanie operácii

Teraz si ukážeme ako efektívne vykonať zvyšné operácie. Začnime treťou operáciou. Na tej si ukážeme správny princíp. Zvyšné dve operácie sa potom dajú doriešiť analogicky.

Hlavnou myšlienkou bude „nerob nič predčasne“. Predstavme si, že chceme zvýšiť o k všetky hodnoty v intervale, ktorý zodpovedá nejakému vrcholu v nášho binárneho stromu. Namiesto toho, aby sme prechádzali celý podstrom a zvyšovali každý jeden prvok v ňom, jednoducho si vo vrchole v upravíme jeho súčet (to vieme spraviť hneď) a poznačíme si: „všetky prvky pod týmto vrcholom treba o k zvýšiť“. Jeho potomkov zatiaľ necháme tak, budú mať dočasne nesprávne hodnoty. To nám ale nevadí – totiž ak niekedy v budúcnosti budeme chcieť pristupovať k potomkom vrcholu v, musíme pri tom prejsť cez v. No a práve až v tomto okamihu príde k vykonaniu zapamätaného príkazu. Presnejšie, vždy, keď chceme z nejakého vrcholu u ísť v strome hlbšie, najskôr sa pozrieme, či v u nemáme „odloženú“ nejakú operáciu. Ak áno, najskôr dotyčnú operáciu presunieme z u do oboch jeho synov, a až potom sa pohneme o krok dodola (a tam celý postup opakujeme, ak treba).

Vo všeobecnosti bude zvyšovanie intervalu o x vyzerať podobne ako zisťovanie súčtu. Len do všetkých troch operácií (zvyšovanie prvku, zvyšovanie intervalu, aj zisťovanie súčtu intervalu) teda musíme doplniť časť, pri

ktorej skontrolujeme a prípadne vyriešime odložené operácie. Taktiež potrebujeme vedieť „skladať“ zvýšenia: keď už máme v nejakom vrchole zapamätané zvýšenie o a a príde zvýšenie o b, jednoducho zmeníme pamätanú hodnotu na $a + b$.

Ako vykonať posledné dve operácie – zisťovanie minima a úpravu intervalu na konkrétnu hodnotu? Zisťovanie minima bude jednoduché: v každom vrchole si budeme udržiavať aj minimum príslušného intervalu. Úprava intervalu na konkrétnu hodnotu prebehne podobne ako zvýšenie celého intervalu o k. Vykonáme ju lenivo len vo vrcholoch, ktoré sú celé pokryté intervalom, ktorý meníme. Keď robíme túto operáciu a náhodou sa vo vrchole predtým vyskytla operácia pridania, tak túto staršiu operáciu zahodíme (lebo nová operácia všetky hodnoty aj tak prepíše). A naopak, ak sme mali odloženú zmenu na hodnotu a a príde nám zvýšenie o b, môžeme si napr. zmeniť zapamätanú operáciu na „zmeň hodnotu na $a + b$“.

Vďaka lenivému spôsobu vykonávania dokážeme každú operáciu vykonať v čase $O(\lg n)$.

(Kvôli úspore miesta listing programu uvedieme až v elektronickej verzii vzorových riešení.)

DVADSIATY SIEDMY ROČNÍK OLYMPIÁDY V INFORMATIKE

Autori úloh: Michal Anderle, Vladimír Boža, Michal Forišek, Peter Fulla, Ján Hozza
Recenzent: Michal Forišek
Slovenská komisia Olympiády v informatike
Vydal: IUVENTA – Slovenský inštitút mládeže, Bratislava 2012