TMA | Milan Wikarski

Značenie

T - Théseu

M - Mínotaurus

A - Ariadna

Postava - jedna z T, M, A

Labyrint - ten labyrint, v ktorom sa postavy pohybujú

Postrehy

Labytint, v ktorom sa postavy pohybujú je graf. Jeho vrcholy sú miestnosti a chodby medzi miestnosťami sú hrany grafu. Tento graf budeme nazývať *graf labyrintu*.

Počet vrcholov grafu labyrintu označíme `n`, a platí `n <= 100`.

Počet hrán (chodieb) grafu labyrintu označíme `m`, a platí `m <= 100 * 100`.

Reprezentácia dat

Graf labyrintu

Graf labyrintu budeme reprezentovať pomcou dvoch polí. Pole `n_edges` bude typu `int8` (`1B`) a každému vrcholu v priradí počet jeho susedov. Pole `edges` bude pole pointerov (32-bitových) na pole typu `int8`, kde bude zoznam susedov daného vrcholu.

Na `n_edges` potrebujeme najviac `100B`, na pointery poľa `edges` potrebujeme najviac `400B` a na zoznamy hrán potrebujeme najviac `100 * 100B = 10000B`. Dokopy na graf labyrintu potrebujeme `~10.5kB`, takže nám ostane `> 500kB` na ostatné data.

V našom grafe povolíme aj slučky a každý vrchol `v` bude mať slučku (tj. hranu `{ v, v }`). To je preto, lebo postavy môžu stáť aj na mieste, čo je ekvivalentné s cestou po slučke z `v` do `v`.

Stav

Trojicu `(T, M, A)` vrcholov (tj. čísel z množiny `{ 0, 1, ..., n - 1 }`) nazvyme stav. Každé z čísel sa dá reprezentovať pomocou `int8`, takže celý stav zaberie `3B`.

Hešovanie stavov

Stavy budeme hešovať pomocou amortizovanej hešovacej tabuľky (postupne ju budeme podľa potreby zväčšovať). Každý stav, ktorý sme už objavili zahešujeme a uložíme do tejto tabuľky. Označenie stavu ako objaveného a aj odpoveď na otázku, či sme v danom stave už boli takto stihneme v konštantnom čase.

Algoritmus

1. Úprava vstupných dat

Na vstupe dostaneme názvy miestností - tie sú nám ale zbytočné. Namiesto toho pridelíme každej miestonosti ID - čo bude unikátne číslo z množiny `{ 0, 1, ..., n - 1 }`. Začneme číslovať od 0 a vždy, keď narazíme na novú miestnosť, pridelíme jej číslo o 1 vyššie, ako predchádzajúcej miestnosti. To nám zároveň dovolí reprezentovať labyrint pomocou grafu labyrintu (vrcholy budú čísla z množiny `{ 0, 1, ..., n - 1 }`, takže ich môžeme ukladať do poľa veľkosti `n`).

Vytovríme si hešovaciu tabuľku, ktorá názvu miestnosti pridelí jej ID. Takto zvládneme transformáciu názvy --> ID v čase lineárnom s počtom chodieb. Po vybudovaní grafu labyrintu môžeme pamäť potrebunú na hešovaciu tabuľku uvoľniť (tabuľka zaberie `~10kB` -- na začiatku toľko pamäťe určite budeme mať).

2. Vhodný grafový pohľad na problém

Prvá vec, ktorú si musíme uvedomiť je, že každá postava sa v každom momente musí nachádzať na práve jednom mieste.

Vždy sa teda všetky postavy nachádzajú v nejakom *stave*. Celkovo existuje menej, ako `n^3` stavov (to je menej, ako milión stavov pre zadaný limit počtu izieb).

Zo stavov sa dá prechádzať od iných stavov. Existujú 2 význačné stavy:

- začiatočný stav (takto postavy začínajú)
- konečný stav (postavy chceme dostať do tohoto stavu)

Definujme si *graf stavov*. Ten bude ako vrcholy mať trojice `(T, M, A)` a hrana z vrcholu `v` bude viesť do vrcholu `u`, ako sa zo stavu `v` vieme dostať do stavu `u = (T', M', A')`. Tzn. postavy sú v *grafe labyrintu* na daných vrcholoch `T, M, A` a existujú hrany `{T, T'}, {M, M', A, A'}` a zároveň po presune postáv sa zachová podmienka o tom, že nemôžu byť 2 postavy vo vrcholoch, ktoré majú spoločnú hranu.

Problém sme takto zmenili na hľadanie minimálnej cesty v *grafe stavov* -- hľadáme minimálnu cestu zo začiatočného stavu do konečného stavu. Každá hrana bude mať váhu 1 (pretože cesta trvá vždy 1 minútu). Výsledok teda bude počet hrán na minimálnej ceste v *grafe stavov* zo začiatočného stavu do konečného stavu. Ak takáto cesta neexistuje, potom je výsledkom -1.

Keďže hľadáme vzdialenosť v grafe definovanú ako počet hrán, môžeme použiť algoritmus `BFS`.

3. BFS vs DFS a pamäť

Použitie algoritmu BFS by bolo následovné:

```
1. Q <- prázdna fronta
2. pridaj (T0, M0, A0) do fronty
3. dokým fronta nie je prázdna:
4.   T, M, A <- vyber skúmaný stav z fronty
5.   pre všetky T' také, že T tam môže ísť:
6.     pre všetky M' také, že M tam môže ísť:
7.       pre všetky A' také, že A tam môže ísť:
8.         Ak stav (T', M', A') je cieľový, skonči a vráť vzdialenosť medzi začiatočným stavom a stavom (T', 
9.         Ak stav (T', M', A') môže nastať a takýto stav sme ešte neobjavili:
10.           Označ stav (T', M', A') za objavený
11.          Pridaj stav (T', M', A') do fronty
12. Skonči a vráť -1
```

Tu ale prichádza problém s pamäťou -- vo fronte môžeme mať až milión stavov (čo sa určite do pamäte nezmestí). Okrem toho si ešte budeme musieť pamätať vzdialenosť medzi začiatočným stavom a každým stavom, na čo potrebujeme aspoň `int16` (`2B`).

Namiesto toho môžeme použiť algoritmus DFS. Ten ale potrebujeme upraviť tak, aby skúmal graf iba do istej hĺbky `h`. Začneme s `h=1` a spustíme DFS. Ak sme riešenie našli, vypíšeme ho a skončíme. Ak nie, položíme `h <- h + 1` a pokračujeme. Ak už nemôžeme ísť hlbšie, ale ešte sme nedosiahli maximálnu hĺbku, skončíme a vrátime -1.

Potrebujeme ešte sledovať, v ktorých stavoch sme už boli. Na to môžeme použiť hešovaciu tabuľku a stav hešovať (napr. pomocou skalárneho šúčinu alebo budeme hešovať číslo `10000 * T + 100 * M + A`, ktoré bude pre každý stav určite unikátne). Vždy, keď zvýšime `h` o jedna (začneme DFS), hešovaciu tabuľku celú premažeme.

Zásobník rekurzie

BFS sme vybrali, pretože nepoužíva frontu. Určite ale používa zásobník rekurzie, ktorý vyžaduje nejakú pamäť. Pamäť, ktorú vyžaduje je ale zhora obmedzená optimálnym riešením, takže sa nemusíme báť, že by nám pamäť nestačila.

Dekompozícia programu

1. Začneme transformáciou inputu -- názvy miestností zmeníme na ID miestností

2. Napíšeme si funkciu `BFS`, ktorá aku argument vezme *graf labyrintu* `G` a maximálu hĺbku `h`. Tá si alokuje pamäť pre hešovaciu tabuľku, prehľadá graf, pamäť uvoľní a nakoniec vráti hodnot >= 0, ak našla riešenie, -1, ak riešenie neexistuje, -2, ak riešenie nenašla, ale mali by sme hľadať ďalej

3. Potom budeme opakovať volanie funkcie `BFS`, dokým vracia -2. Ak vráti niečo iné, vypíšeme to a skončíme