Slovenská Technická Univerzita
Fakulta Informatiky a Informačných Technológií

Dátové Štruktúry a Algoritmy

Zadanie 3 - Popolvár

Marek Adamovič

Cvičenie: Pondelok 16:00 – 17:40
Ing. Peter Pištek, PhD.

Rok: 2019/2020

Použité algoritmy a riešenia

Ako funguje hľadanie najkratšej cesty v programe

Pri zavolaní funkcie zachran_princezne sa najskôr vytvorí pomocou mapy graf. Alokujeme pole pre graf a postupne ho napĺňame štruktúrami uzlov. Každý ma jedinečné ID, podľa ktorého vieme spätne zistiť, kde sa nachádzal na mape. Keď mám vytvorené prvky, pridám do nich parametre z mapy, teda informáciu, o aké políčko sa jedná, koľko trvá prechod cez neho. Keď už viem, o aké políčka sa jedná, môžem ich navzájom pospájať hranami (nemám štruktúru pre hrany, len pointre na 4 svetové strany pri každom uzle) s tým, že nepriechodné prekážky neprepájam a ich pointre ostanú nastavené na NULL.

Po vytvorení grafu zavoláme funckiu dijkstra, ktorá priamo vracia výslednú cestu (všetka logika sa deje vnútri funkcie). Na začiatku zavoláme funckiu dijkstra_drakobijec, čo je samotný Dijkstrov algoritmus, popísaný nižšie. Vďaka nemu môžeme skontrolovať, či nie sú niektoré ciele zablokované a prípadne vrátiť NULL vo funckii najdi_v_mape. Hneď na to skontrolujeme, či stihneme zabiť draka predtým, ako sa zobudí, alebo je priďaleko. Cestu k drakovi si uložíme do premennej a ideme zachrániť princezné. Na to zavoláme funckiu uzasna_zachranna_akcia_vsetkych_princezien.

Vo funckii uzasna_zachranna_akcia_vsetkych_princezien si najskôr naplníme 2D pole vzdialeností medzi jednotlivými princeznami a drakom (cieľmi). Následne spravíme permutácie vzdielností odvíjajúcich sa od zvoleného poradia a vrátime minimálnu cestu. Aby sme vedeli zistiť súradnice tejto cesty (zatiaľ máme len poradie indexov, ktoré tvoria najkratšiu cestu), musíme znovu volať dijkstrov algoritmus. Keď zložíme cestu, tak ju spojíme s cestou k drakovi, prevedieme ju do súradníc a vrátime túto finálnu cestu.

```
int *zachran_princezne(char **mapa, int n, int m, int t, int *dlzka_cesty){
    UZOL *graf;
    int *vysledna_cesta, pocet = n * m;
    graf = sprav_graf(mapa, n, m);
    vysledna_cesta = dijkstra(dlzka_cesty, graf, pocet, t);
    if(vysledna_cesta == NULL){
        free(graf);
        *dlzka_cesty = 0;
        return NULL;
    }
    int cas = 0;
    vysledna_cesta = postav_cestu(graf, vysledna_cesta, m, dlzka_cesty, &cas);
    //uvolni graf
    free(graf);

    printf("Najkratsia cesta s casom: %d\n", cas);


    for(int i = 0; i < *dlzka_cesty; i++){
        printf("(%d %d) ", vysledna_cesta[i * 2], vysledna_cesta[i * 2 + 1]);
    }
    printf("\n");



    return vysledna_cesta;
}
```

Dijkstrov algoritmus s minimálnou haldou

Dijkstrov algoritmus je algoritmus, ktorý rieši problém nájdenia najkratšej cesty v grafe. Na začiatku si označíme počiatočný vrchol do skupiny „vyriešených“ a všetky ostatné vrcholy do skupiny „nevyriešených“. Následne postupujeme tak, že vždy sa pozrieme na všetky susedné vrcholy posledného pridaného vyriešeného, ak nám výde pre ne kratšia vzdialenosť z nášho vyriešeného vrcholu, tak si to zaznačíme. Potom, ako pozrieme všetky susedné vrcholy, tak vieme prehlásiť, že nevyriešený vrchol s najmenšou cestou môžeme priradiť do vyriešených a v ďalšej iterácií pokračovať s ním. Tu môže byť problémom časová zložitosť. V každej iterácií musíme nájsť minimum a následne ho odstrániť. Preto použijeme minimáldu haldu, ktorá si udržiava minimum vždy v koreni pomocou vyplavovania/potápania prvkov pri zmene ich hodnôt, resp. vymazaní prvku z hlady, tým pádom ho vieme veľmi ľahko odstrániť.

V mojej verzii dijkstrovho algoritmu sa bude hľadať n-1 krát minimum, kde n je počet prvkov v grafe (lineárna zložitosť O(n)). Minimum v halde udržiavam pomocou vyplavovania/potápania prvkov medzi úrovňami, tým pádom nemusím porovnať všetky prvky (teda časová zložitosť haldy je logaritmická O(log n)). Samotný algoritmus sa zavolá 2q krát (jedenkrát pre vzdielnosti a druhýkrát pre spätné vytiahnutie súradníc z minimálnej cesty), kde q je počet cieľov v mape (teda počet princezien + drak).

```
231         //nevybavene uzly - vyberiem ten s najmensou vzdialenostou
232
233         vybavene[vybavene_pocet] = nevybavene[0];
234         nevybavene[0]->vybaveny = 1;
235         nevybavene[0] = nevybavene[nevybavene_pocet - 1];         //do korena dam posledny list
236         nevybavene[0]->index = 0;
237         heap_reverse_bubble(nevybavene, nevybavene_pocet - 1);
238         aktualny = *vybavene[vybavene_pocet];
239         //nezabudnem znizit/zvysit pocty
240         nevybavene_pocet--;
241         vybavene_pocet++;
```

Testovanie

Testovanie som rozdelil na 2 časti, moje ručne vytvorené mapy a následne náhodné mapy vytvorené mojim generátorom (s prípadnými menšími zmenami). Snažil som sa zachytiť čo najviac hraničných situácií pri rôznych testovacích podmienkach. Testovacie scenáre sú aj v krátkosti popísané v mojom maine v komentároch. Mimo mapy, ktorú sme dostali k zadaniu, som využíval načítanie zo súborov. Popíšem najdôležitejšie z testov, ktoré boli zamerané na krajné podmienky.

Test č.2

Mapa 39x39 obsahujúca úzku cestičku obklopenú prekážkami. Cieľom testovania bolo zistiť, či popolvár nezasahuje pohybom do nepriechodných prekážok.

Test č.3

Úzka mapa s výškou 1, ktorá sa zameriavala na to, či popolvár zvládne záchranu aj v „stiesnených“ podmienkach. Taktiež som umiestnil 4 princezné na začiatok a poslednú na koniec, za draka, aby som skontroloval, či vyberá najrýchlejšie poradie záchrany.

Test č.4

Mapa o veľkost 1x2, s jednou princeznou a jedným drakom. Cieľom bolo zistiť, či aj pri najmenšej možnej mape mi popolvár niekam neujde.

Test č.5

Ďalšia úzka mapa, ktorá sleduje, ako sa popolvár vysporiada s drakom na štartovacom políčku (0, 0).

Test č.6

„Zúbkovaná“ mapa tvorená širokými prekážkami, cez ktoré sa dá prejsť len na jednom mieste. Cieľom bolo zistiť, či popolvár nájde toto miesto, cez ktoré vie prejsť do nižšej „úrovne“.

Test č.7

Prvá náhodná mapa o veľkosti 100x100 s piatimi princeznami a jedným drakom. Testuje správnosť riešenia pri veľkých vstupoch.

Test č.9

Tento test testoval správanie programu pri situácií, keď je práve jeden cieľ nedostupný za nepriechodnými prekážkami.

Test č.11

Obrovská mapa 100x100 s drakom na začiatku a všetkými princeznami okolo neho. Tu testujem, či napriek veľkému bludisku popolvár nezablúdi a sústredí sa len na ciele.

Test č.13-15

Testy testujúce rôzne počty princezien, ktoré ešte neboli otestované.

Test č.16

Mapa, na ktorej sa neviem pohnúť zo štartu, teda (0, 0) je obklopený nepriechodnými prekážkami.

Test č.17

Malá mapa, ktorá má na začiatku (0, 0) nepriechodnú prekážku. Kontrolujem, či popolvár vráti NULL, keď sa zjaví v skale.