Peter Smreček, AIS ID: 103130
Utorok 11:00

Zadanie 3 – Popolvár

Zadanie bolo vypracované v prostredí Visual Studio

Opisované riešenie využíva Djikstrov algoritmus s minimálnou binárnou haldou a permutáciami určenou najvýhodnejšou cestou.

Charakteristika minimálnej binárnej haldy

Minimálna binárna halda je stromová štruktúra s obmedzeniami. Je reprezentovaná vektorom. Hodnoty sú od koreňa, ktorý obsahuje minimálny kľúč, zoradené vzostupne.

V riešení som použil haldu štruktúr, pričom halda bola reprezentovaná ako pole štruktúr. Štruktúry som zoraďoval podľa ich vzdialenosti od začiatočného prvku hľadania v mape.

K halde štruktúr používanej v tomto riešení prislúcha aj mapa haldy, ktorá na základe súradníc vrcholu zistí jeho aktuálny index v halde.

Charakteristika Djikstrovho algoritmu

Djikstrov algoritmus slúži na nájdenie najkratšej cesty v grafe z počiatočného vrcholu do iného vrcholu. V riešení som použil na udržiavanie prehľadu o tom, ktoré vrcholy boli prehľadané a ktoré nie, binárnu haldu vrcholov.

Z vrcholu s minimálnou cestou vždy hľadám cesty do všetkých susedov tohto vrcholu. Ak je cesta k susedovi cez tento vrchol kratšia ako predošlá cesta, prepíšem ju. Po tom, ako prehľadám susedov vrcholu s minimálnou vzdialenosťou od začiatku, vrchol z haldy odstránim a pokračujem novým najmenším vrcholom až pokým halda nie je prázdna.

Opis riešenia

Po zavolaní predpísanej funkcie `zachran_princezne` sa najskôr vynuluje premenná určujúca dĺžku cesty. Alokuje sa pamäť pre binárnu haldu a definujú sa pomocné premenné. V riešení som použil binárne haldy reprezentované jednorozmernými poľami štruktúr, kde každá štruktúra reprezentovala konkrétny vrchol a držala v sebe informácie o danom vrchole, pričom bola halda zoradená podľa vzdialenosti od určitého štartovacieho vrcholu.

```
typedef struct vertex {
    int x, y;
    int parentX, parentY;
    int length;
}VERTEX;
```

Funkcia `transformMapToHeap` prepíše mapu zadanú 2D poľom charov do poľa štruktúr. Premenné štruktúry `x` a `y` reprezentujú súradnice v mape a sú podľa nej aj nastavené. Hodnoty premenných `parentX` a `parentY` sú nastavené pri tejto inicializácii na -1. Hodnota premennej `length` prvku štruktúry je nastavená na konštantu `INF`, ktorá v tomto programe reprezentuje nekonečno. Jej hodnota je maximálna hodnota integera, teda 2147483647. Z tohto poľa sú následne vyradené políčka s nepriechodnými prekážkami. Tiež sa nájdu súradnice draka a všetkých princezien, ako aj počet princezien. Počas prepisu mapy do haldy sa vytvorí aj mapa tejto haldy. Ako mapu haldy používam 2D pole, ktoré na pozícii `x y` drží aktuálny index vrcholu `x y` v halde.

Peter Smreček, AIS ID: 103130
Utorok 11:00

Moje riešenie nutne potrebuje pracovať s viacerými haldami, ktoré sú ale na začiatku rovnaké, líšia sa len začiatočným políčkom, preto som vytvoril funkciu `copyHeap` ktorá skopíruje haldu vrátenú predošlou funkciou do novej haldy. Toto riešenie šetrí čas, pretože už nie je potrebné prechádzať opätovne celou mapou, ale iba priechodnými políčkami.

Funkcia `decrease` zmení hodnotu dĺžky od začiatočného políčka prvku v hlade. Zmení kľúč konkrétneho prvku v halde a zavolá funkciu `fixHeapUp` ktorá tento prvok presunie na správne miesto v halde. Pri tom je aktualizovaná mapa haldy. Funkcia `getParent` vráti index rodiča prvku v halde. Funkcia `exchange` vymení 2 konkrétne štruktúry v halde navzájom. Funkcia `verticesLen` vráti čas potrebný na prechod políčkom. Funkcia `decrease` sa v kontexte riešenia Popolvára používa na nastavenie začiatočného políčka, z ktorého bude spustený Djikstrov algoritmus.

Funkcia `djikstra` je aplikácia Djikstrovho algoritmu v mojom riešení. Funkcia v cykle, až pokým halda nie je prázdna, extrahuje minimálny prvok z haldy funkciou `extractMin`, ktorá vracia minimálny prvok haldy, tj. prvý prvok haldy. Funkcia `extractMin` vymení prvý a posledný prvok v halde, zmenší celkovú veľkosť haldy a zavolá funkciu `heapify`, ktorá haldu utriedi. Funkcie `getLeft` a `getRight` volané funkciou `heapify` sú funkcie na získanie indexov ľavého a pravého dieťaťa konkrétneho prvku v halde. Na základe dodržiavania haldových vlastností, rekurzívna funkcia `heapify` obnoví správne poradie prvkov v halde. Akonáhle je získaný minimálny prvok z binárnej haldy, v cykle vo funkcii `djikstra` sú nájdené všetky susedné vrcholy daného minimálneho vrcholu. Sú na to použité funkcie `top`, `right`, `bottom` a `left`. Súradnice získané týmito funkciami nemusia byť nutne platné súradnice. Následne je stále v cykle vo funkcii `djikstra` spustená funkcia `vertextNewLenghtDjikstra`, ktorá pomocou funkcie `validXY` zistí, či sú súradnice posunuté do tejto funkcie argumentom platné (nachádzajú sa v mape a v halde, teda nie sú záporné, nie sú už prehľadané a nie sú mimo mapy). V prípade ak sú to súradnice platného vrcholu, tak sa tomuto vrcholu vypočíta nová vzdialenosť tak, že vezme vzdialenosť od začiatku mapy prislúchajúca jeho rodičovi a pripočíta sa hodnota políčka, ktorému sa práve ráta vzdialenosť. Hodnotu, ktorá sa pripočítava k vzdialenosti rodiča vráti funkcia `verticesLen`. Ak je nová vzdialenosť menšia ako aktuálna vzdialenosť prislúchajúca práve navštevovanému vrcholu, tak sa prepíše. Ak bol vrchol nenavštívený, mal pôvodne nekonečnú vzdialenosť od začiatku mapy, teraz sa mu vzdialenosť od začiatku mapy zmenšila na novú vypočítanú hodnotu. Zmenu hodnoty vzdialenosti vrcholu zabezpečuje funkcia `decrease`. Funkcionalitu tejto funkcie som opisoval vyššie. Funkcia `vertextNewLenghtDjikstra` sa spúšťa pre všetkých 4 susedov aktuálneho políčka, no novú hodnotu vzdialenosti od začiatku poľa nastaví len platným vrcholom, ktorým sa takto našla kratšia cesta. Následne cyklus prechádza vo funkcii `djikstra` do ďalšej iterácie a opäť extrahuje minimum z haldy a nastavuje nové vzdialenosti jeho susedom. Cyklus končí ak je halda prázdna.

Potom, čo je nad haldou zavolaný Djikstrou algoritmus tak funkcia `pathToDragonCreator` spätne prechádza haldou od políčka draka až po začiatočné políčko. Ak cesta k drakovi neexistuje, funkcia vracia dĺžku cesty rovnú 0. Ak cesta k drakovi existuje, celá sa zapíše do poľa. Následne je pole obrátené funkciou `reverse` aby spĺňalo špecifické požiadavky tohto zadania.

Je vytvorená nová halda pre draka, taká, že je skopírovaná pôvodná halda vrátená funkciou `transformMapToHeap` do novej haldy. Tejto halde je nastavené ako prvotné políčko

drak pomocou funkcie `decrease`. Nad touto haldou s prvotným políčkom draka je spustený Djikstrov algoritmus ktorý reprezentuje funkcia `djikstra`. Takto je získaná halda obsahujúca vrcholy dostupné z políčka drak a ich vzdialenosti k drakovi.

Ak sa nachádza na mape iba jedna princezná, znamená to triviálne riešenie úlohy a teda že Popolvár najskôr zabije draka a následne zachráni princeznú. Teda z políčka draka sa nájde najkratšia priama cesta medzi drakom a princeznou. Na to slúži funkcia `princessPathCreator` ktorá dostane argumentom haldu draka, mapu haldy draka, pole s výslednou cestou, súradnice princeznej a ešte niekoľko argumentov. Táto funkcia spätným prehľadaním haldy draka nájde najkratšiu cestu medzi drakom a princeznou a zapíše ju do nového poľa. Nové pole otočí funkciou `reverse` a pridá toto nové pole k pôvodnému poľu cesty funkciou `addToArray`. Nakoniec funkcia hľadania cesty k princeznej vráti novú celkovú dĺžku cesty.

```
typedef struct princess {
    int x, y;
    VERTEX *heap;
    int **mapHeap;
    int heapSize;
}PRINCESS;
```

Ak sa na mape nachádza princezien viac, je potrebné zistiť, ako sa k týmto princeznám dá dostať po zabití draka v čo najkratšom čase. Keďže pôvodné spustenie funkcie `transformMapToHeap` vrátilo okrem haldy aj počet a súradnice princezien, vytvoril som nové pole štruktúr, kde je každá princezná reprezentovaná samostatnou štruktúrou. Každej princeznej sú zapísané jej súradnice na mape a taktiež je jej priradená jej vlastná binárna halda s mapou. Táto halda je kópiou pôvodne vytvorenej haldy pri prepise mapy do haldy. Následne je funkciou `decrease` nastavený ako prvý prvok haldy konkrétna princezná. Funkcia `djikstra` vytvorí haldu, ktorá zodpovedá vzdialenostiam od súčasnej princeznej.

Následne je vytvorený nový graf formou matice susednosti. Tento graf však obsahuje iba draka a princezné. Hodnotami v matici sú vzdialenosti medzi konkrétnou princeznou a drakom, alebo medzi 2 z princezien. Formálne mám teda graf definovaný maticou susednosti a hľadám v ňom najlacnejšiu Hamiltonovskú cestu – cestu, ktorá obsahuje všetky vrcholy práve raz, neobsahuje cykly a nevetví sa.

Funkcia `findBestConfiguration` naplní maticu susednosti hodnotami váh ciest medzi konkrétnymi dvojicami bodov. Následne sa alokuje pomocné pole ktoré obsahuje indexy princezien. Funkciou `factorial` sa zistí, koľko permutácii poľa s indexami princezien existuje. Alokuje sa 2D pole ktoré obsahuje všetky permutácie. Pokiaľ sú princezné 2, je toto pole veľké 2x2, ak sú 3, tak je to 6x3, ak sú 4, tak je to 24x4 a ak je princezien 5, tak má pole rozmer 120x5. Alokuje sa dynamicky a faktoriál sa ráta funkciou. Následne je globálna premenná `numberOfPermutations` vynulovaná. Táto funkcia funguje ako iteračná premenná pre zápis výsledných permutácii do poľa permutácii. Funkcia `permute` vytvorí rekurzívne všetky možné permutácie daného vstupného poľa princezien a zapíše ich do poľa permutácii. Funkcia `permute` používa na výmenu 2 celých čísel funkciu `exchange2`. Po tom, ako je vytvorené pole permutácii ciest medzi princeznami, je pomocou cyklov vo funkcii `findBestConfiguration` nájdená najkratšia cesta. Hľadanie prebieha tak, že sa sčítajú hodnoty ciest medzi drakom a prvou princeznou zo zoznamu permutácii, s hodnotou vzdialenosti medzi prvou princeznou zo zoznamu a druhou princeznou zo zoznamu permutácii až kým sa nepriráta hodnota vzdialenosti medzi predposlednou a poslednou princeznou zo zoznamu permutácii. Ak je hodnota tohto súčtu menšia ako predošlá v cykle vypočítaná hodnota, tak sa táto postupnosť princezien uloží. Ak nie je najlepšia, tak sa pokračuje ďalšou

iteráciou. V každej iterácii cyklov sa používa iný riadok z 2D poľa permutácii vráteného funkciou `permute`. Po prebehnutí cyklov hľadajúcich minimálnu cestu sa v poli konfigurácii nachádza najlacnejšia cesta medzi drakom a princeznami navzájom.

Po tom, čo funkcia `findBestConfiguration` našla najlepšie poradie navštevovania princezien, funkcia `princessPathCreator` sa použije na generovanie cesty medzi drakom a prvou princeznou zo zoznamu. Následne sa pomocou tejto istej funkcie nájdu cesty aj medzi prvou princeznou zo zoznamu a nasledujúcou až po nájdenie cesty medzi predposlednou a poslednou princeznou zo zoznamu. Všetky tieto cesty sú pripisované do toho istého poľa.

Po vygenerovaní cesty od začiatku mapy k drakovi a od draka k jednej princeznej, prípadne od draka k jednej z princezien a od nej k ďalšie a takto až k poslednej, funkcia vráti pole obsahujúce túto cestu ako aj dĺžku tejto cesty.

Riešenie hraničných situácii

Ak sa na mape nenachádza drak, funkcia vracia NULL a dĺžku cesty 0.

Ak sa Popolvár nedostal na mapu, pretože mapa začína nepriechodnou prekážkou, funkcia vracia NULL a dĺžku cesty 0.

Ak k drakovi neexistuje cesta, funkcia vracia NULL a dĺžku cesty 0.

Ak je časová vzdialenosť políčka drak väčšia ako parameter `t`, funkcia vracia NULL a dĺžku cesty 0.

Ak k nejakej princeznej neexistuje cesta, funkcia vráti NULL a dĺžku cesty 0.

Situácie, že by sa na mape nenachádzal drak alebo by sa na mape nenachádzala princezná nie je nutné ošetriť, pretože zo zadania, ktoré hovorí o drakovi unášajúcom princezné, implicitne vyplýva, že na mape je drak a aspoň 1 princezná.

Testovanie riešenia

Svoje riešenie som otestoval na 16 vlastných vstupoch a na vzorovom vstupe. Upravil som zadaný testovač a pridal doň 16 vlastných testov čítaných zo súboru. Pre prehľadnosť kódu som vytvoril som funkciu `mapaZoSuboru` ktorá prepíše súbor na mapu formou 2D poľa.

Testy 1 až 12 testujú elementárnu funkčnosť zadania. Zabitie draka a záchranu 1 až 5 princezien.

Test 1 je vzorový vstup.

Test 2 testuje funkčnosť na minimálnej vertikálnej mape.

Test 3 testuje funkčnosť na minimálne horizontálnej mape.

Test 4 testuje funkčnosť pre mapu s jednou princeznou.

Test 5 testuje funkčnosť pre mapu s 2 princeznami.

Test 6 testuje funkčnosť pre mapu s 3 princeznami.

Test 7 testuje funkčnosť pre mapu s 4 princeznami.

Test 8 testuje funkčnosť pre mapu s 5 princeznami.

Test 9 testuje funkčnosť na maximálnej mape 100x100 políčok s 1 princeznou.

Test 10 testuje funkčnosť na maximálnej mape 100x100 políčok s 3 princeznami.

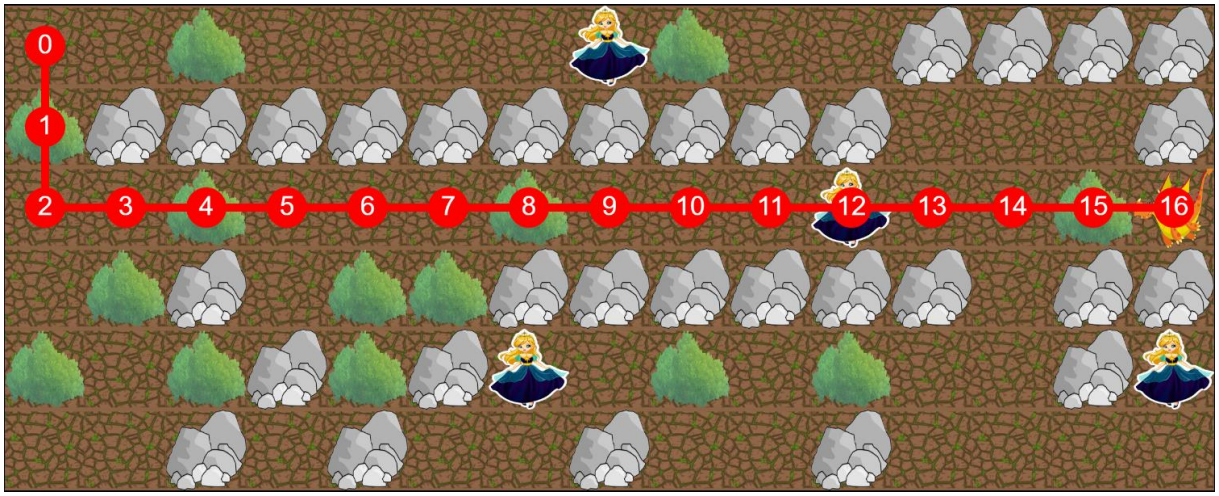
Test 11 testuje funkčnosť na maximálnej mape 100x100 políčok s 5 princeznami.

Test 12 testuje funkčnosť na inej maximálnej mape 100x100 políčok s 5 princeznami.

Testy 13 až 17 testujú hraničné situácie.

Test 13 testuje nájdenie správnej cesty v grafe na malej mape s 4 princeznami. Neprechádza princezné statickým poradím, ale dynamicky medzi nimi nájde najkratšie cesty. Rovnaký princíp je uplatnený aj na vyššie uvedených vzorových mapách, no táto je dostatočne prehľadná na to, aby sa to dalo jasne uvedomiť si to.

Poradie záchrany princezien na mape 13:



Obrázok 1 Test 13, zabitie draka

Obrázok 2 Test 13, záchrana 1. princeznej

Peter Smreček, AIS ID: 103130
Utorok 11:00

Obrázok 3 Test 13, záchrana 2. princeznej

Obrázok 4 Test 13, záchrana 3. princeznej

Obrázok 5 Test 13, záchrana 4. princeznej

Princezné na mape sú zachraňované v poradí 3, 4, 2, 1. Riešenie zohľadňuje hľadanie najkratších ciest aj medzi princeznami.

Test 14 testuje funkčnosť merania času. Mapa má hranične málo času na zabitie draka, program preto vypíše hlásenie a vráti NULL.

Test 15 testuje funkčnosť riešenia, ak je jedna z princezien nedostupná. Princezná sa nachádza v časti mapy, ktorá nie je dostupná, teda princezná nie je priamo obkolesená nepriechodnými prekážkami, ale časť mapy na ktorej sa nachádza je obkolesená nepriechodnými prekážkami. V tomto prípade program vypíše hlásenie a vráti NULL.

Test 16 testuje funkčnosť na mape, na ktorej nie je dostupný drak. Drak sa nachádza v časti mapy, ktorá nie je dostupná, teda drak nie je priamo obkolesený nepriechodnými prekážkami, ale časť mapy na ktorej sa nachádza je obkolesená nepriechodnými prekážkami. V tomto prípade program vypíše hlásenie a vráti NULL.

Obrázok 6 Test 17

Test 17 testuje funkčnosť na mape, na ktorej sa nachádza 1 drak a 5 princezien, kde drak je na vstupnom políčku. Je to obdoba testov 2 a 3.

Všetky testy prebehli očakávane, teda ak bol dostatok času a existovali cesty k drakovi a princeznám, vypísala sa cesta, inak sa vypísalo chybové hlásenie. Poradie princezien na záchranu je vždy také, aby sa našla najkratšia možná cesta. Ak nebol dostatok času alebo neexistovala cesta k drakovi alebo princeznej, vypísalo sa podľa očakávania chybové hlásenie. Keďže som riešenie otestoval na rôznych hraničných vstupoch, pokladám ho za správne.

Odhad zložitosti

Pre použitie binárnej haldy sa riešenie zefektívnilo oproti riešeniam bez binárnej haldy. Odhad realizujem pre najhorší možný prípad, teda 5 princezien a ŽIADNA nepriechodná prekážka v grafe, teda všetky vrcholy sú dosiahnuteľné z počiatočného bodu.

Odhad časovej zložitosti:

N je počet vrcholov v grafe, M je počet hrán v grafe, P je počet princezien v grafe.

Vytvorenie binárnej – 1-krát = O(N)

Skopírovanie binárnej haldy – 7-krát = 7 * O(N)

Extrahovanie minima funkciou `extractMin` – N-krát = N * O(log N)

Relaxácia sa vykonáva funkciou `decrease` – 7 * M-krát = 7 * M * O(log N)

Výpočet permutácii – 1-krát = O(P!)

Nájdenie ideálnej permutácie – 1-krát = O(P!)

Časová zložitosť môjho riešenia je teda O((8 * N) + (7 * N * M * log N) + (2 * P!)), čo je po zanedbaní menej významných členov O(N * M * log N).

Počet princezien je 5 a výpočet permutácii ako aj nájdenie najkratšej možnej permutácie je zanedbateľný v kontexte použitia Djikstrovho algoritmu. Samotný Djikstrov algoritmus s minimálnou binárnou haldou má časovú zložitosť O(M * log N). Ten je v najhoršom prípade spúšťaný práve 7-krát. Z počiatočného políčka, z políčka draka a z políčka každej z 5

princezien. Nosnú časť pamäťovej záťaže preto tvorí Djikstrov algoritmus a nie tvorba háld, či hľadanie najlepšej možnosti v grafe o 5 vrcholoch.

Odhad priestorovej zložitosti:

Vo vyššie popísanom prípade existuje 7 binárnych háld, ktoré obsahujú každá všetky vrcholy. Preto je odhad pamäťovej zložitosti rovný 7*N*N.

Záver

Problém bol vyriešený Djikstrovým algoritmom s minimálnou binárnou haldou, pričom hľadanie najlacnejšej cesty medzi princeznami bolo realizované permutáciami. Toto riešenie je dostatočne efektívne a ošetruje všetky hraničné situácie.