Ľubomír Dlhý, ID: 102897

POPOLVÁR

Ľubomír Dlhý, ID: 102897

POUŽITÝ ALGORITMUS

V mojej implementácii som využil Dijkstrov algoritmus s binárnou minimálnou haldou. Minimálna halda zefektívňuje algoritmus vďaka tomu, že z haldy vyberá vždy vrchol s minimálnym ohodnotením a teda buduje cestu k destinácii vždy cez najlacnejšie čiastkové cesty. Minimálnu haldu reprezentujem pomocou poľa vrcholov. Implementoval som funkcie na inicializáciu haldy, vloženie vrchola, vybratie minimálneho vrchola. Pri vkladaní sa vrchol vkladá na koniec poľa a výmenami s rodičom sa pridelí vhodné miesto (vymieňa sa pokiaľ je vrchol menší ako rodič).
Pri výbere sa odoberie prvok na indexe 0 (vždy najmenší prvok) a namiesto neho vloží posledný prvok, potom sa vhodnými výmenami s menším dieťaťom nájde miesto tomuto poslednému prvku.

```
typedef struct myHeap {
    int kapacita;
    int pocetPrvkov;
    NODE** array;
}HALDA;
```

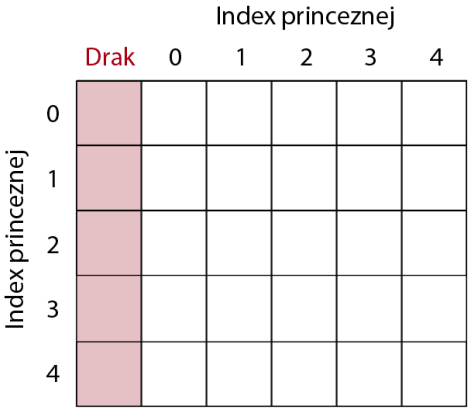
Štruktúra minimálnej haldy

V implementácii si vytváram vlastný graf vrcholov reprezentovaný maticou M x N. Vo vrcholoch si držím informáciu o súradniciach, cenu cesty do daného vrchola, typ políčka (slúži ako hrana) a ukazovateľ na predošlý vrchol (slúži na vybudovanie cesty).

```
typedef struct node {
    unsigned short x;
    unsigned short y;
    unsigned int cost;
    char typ;
    struct node* prev;
}NODE;
```

```
// vytvorenie vlastneho grafu nodov
NODE*** mapaNodov = (NODE***)malloc(n*sizeof(NODE**));
for (int i = 0; i < n;i++) {
    mapaNodov[i] = (NODE**)malloc(m * sizeof(NODE*));
    for (int j = 0; j < m; j++) {
        mapaNodov[i][j] = (NODE*)malloc(sizeof(NODE));
    }
}
```

Dijkstrov algoritmus volám n+1 krát, kde n je počet princezien. Prvé volanie je vždy zo štartovacej pozície k drakovi. Pri tomto volaní je zastavovacia podmienka v dijkstrovom algoritme daná nájdením draka (vybratie draka z minimálnej haldy). Ďalších n volaní je z každej princeznej do každej princeznej a draka. Tu je zastavovacia podmienka daná nájdením každej princeznej a draka (z minimálnej haldy sú vybrané všetky princezné aj drak). Dĺžku, Cenu a Kroky najlacnejších čiastkových ciest si ukladám do štruktúry. Po vykonaní všetkých n dijkstrov dostávam nasledovnú maticu (ukážka matice s n = 5):



Vďaka tomu, že každá princezná nájde najlacnejšiu cestu ku každej princeznej a k drakovi, viem v takejto matici reprezentovať napríklad cestu Princezná 0 → Princezná 3 pomocou Matrix[0][3+1] a cestu z Draka k Princeznej 2 pomocou Matrix[2][0].

Cesty medzi princeznami sú dané vzorcom:
`Matrix[Index Štart. Princeznej][Index Cieľ. Princeznej+1]`

Cesta medzi princeznou a drakom je daná vzorcom:
`Matrix[Index Princeznej][0]`

V takejto matici viem s využitím týchto vzorcov a kombinácií rôznych indexov vybudovať všetky možné cesty medzi princeznami a drakom.

Ľubomír Dlhý, ID: 102897

```
void permutation(princezneCesty** prArr,int* arr, int start, int
{
    if (start == end)
    {
        // najprv je priradena cesta Drak -> princezna
        int sum = prArr[arr[0]]->cost[0];
        for (int i = 1; i <= end; i++) {
            // potom su pripocitane cesty Princezna -> Princezna
            sum += prArr[arr[i-1]]->cost[arr[i] + 1];
        }

        if (sum < (*minCost))
        {
            *minCost = sum;
            for (int i = 0; i <= end; i++) {
                minArr[0][i] = arr[i];
            }
        }
```

Na zistenie globálne najlacnejšej cesty musíme nájsť najlacnejšiu zo všetkých možných ciest. Takýchto ciest je N!, kde N je počet princezien. Rôzne cesty riešim pomocou permutácií. Najprv si spravím pole indexov s veľkosťou N, kde N je počet princezien. Pole indexov reprezentuje cestu medzi drakom a princeznami. Napríklad pole = [1,2,0] predstavuje cestu Drak→Princ. 1→Princ. 2→Princ. 0
Pole indexov permutujem a pre každú permutáciu počítam cenu cesty. Cenu porovnávam s minimom a ukladám si minimum. Výstupom tejto funkcie je minimálny cena cesty a pole indexov tejto minimálnej cesty.

Na konci už iba vybudujem cestu z čiastkovej cesty Štart→Drak a najlacnejšej cesty Drak→Princezné zistenej vo funkcii permutation().

ZLOŽITOSŤ

Odhad časovej zložitosti: $N! + N^2 \cdot V + N \cdot V \cdot \log V + N \cdot V + V$

Dijkstrov algoritmus s binárnou min. haldou	$(N+1) \cdot (V+V \cdot \log V)$ $\sim N \cdot V \cdot \log V$	Je spúšťaný N+1 krát, najprv sa resetuje graf O(V) a potom vrcholy pridáva a vyberá z haldy. O(V log V)	N – počet princezien V – počet vrcholov
Inicializacia vlastneho grafu	V	Vlastný graf sa vytvorý v O(V) čase.	
Vytvorenie čiastkových ciest	$(N^2 + N + 1) \cdot V$ $\sim N^2 \cdot V$	Vytvorená cesta má v najhoršom prípade V vrcholov, treba vytvoriť cesty medzi Princeznami (N^2), medzi Princeznami a Drakom (N) a Štartom a Drakom (1)	
Permutácie ciest	$\sim N!$	Je potrebné urobiť permutácie čiastkových ciest,	
Vybudovanie finálnej najlacnejšej cesty	$(N + 1) \cdot V$ $\sim N \cdot V$	Cesta Štart – Drak a najlacnejšia permutácia cesty Drak – Princezné. Cesta môže mať maximálne V vrcholov.	

Odhad pamäťovej zložitosti: $N^2 \cdot V + 3V$

Binárna Halda	$\sim V$	Binárna halda môže v najhoršom prípade obsahovať všetky vrcholy práve raz.	N – počet princezien V – počet vrcholov
Vlastný Graf	$\sim V$	Môj vlastný graf obsahuje všetky vrcholy presne ako pôvodná mapa znakov.	
Pomocné čiastkové cesty	$\sim N^2 \cdot V$	Ukladám si cesty medzi princeznami (N^2), cestu medzi drakom a princeznami (N) a cestu štart - drak (1). V najhoršom prípade cesta obsahuje všetky vrcholy.	
Finálna cesta	$\sim V$	Konečná cesta obsahuje v najhoršom prípade všetky vrcholy.	

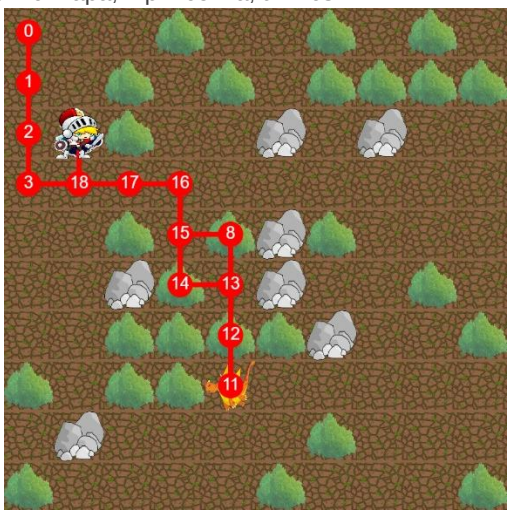
Ľubomír Dlhý, ID: 102897

TESTOVANIE

Na uľahčenie testovania som vytvoril funkciu generateMap(), ktorá po zadaní výšky, šírky, počtu princezien, pravdepodobností výskytu cestičiek, hustých cestičiek a prekážok vytvorí mapu.

V rámci testovania som skúšal rôzne situácie – rôzne počty princezien, rôzne veľkosti máp, špeciálne situácie ako napríklad zablokovaný drak, zablokovaná princezná alebo princezná na štartovacej pozícii.

test1.txt – 10x10 mapa, 1 princezná, t = 20s

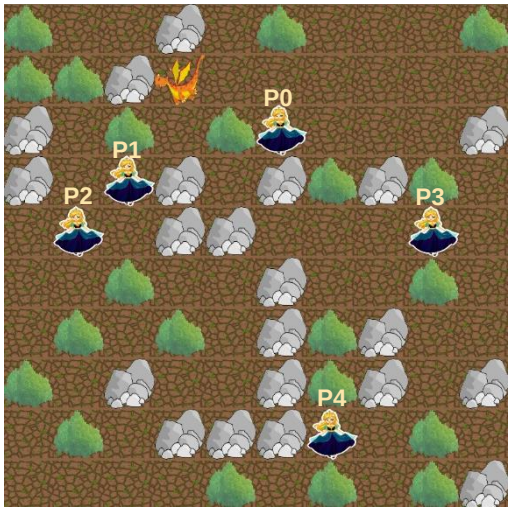


Výstup:

```
0 0
0 1
0 2
0 3
1 3
2 3
3 3
3 4
4 4
4 5
4 6
4 7
4 6
4 5
3 5
3 4
3 3
2 3
1 3
1 2
24
```

Popolvár zabije draka v čase 14s, ďalších 10s trvá záchrana princeznej. Existujú rôzne cesty s rovnakým časom, ale neexistuje lacnejšia cesta.

test2.txt – 10x10 mapa, 5 princezien, t = 20s



Výstup:

```
0 0
0 1
0 2
1 2
1 3
1 4
1 5
0 5
1 5
1 4
1 3
1 2
1 1
2 1
3 1
3 0
4 0
5 0
6 0
6 1
6 2
6 3
7 3
8 3
8 4
8 5
9 5
9 6
8 6
7 6
6 6
5 6
5 7
5 8
5 9
44
```

Drak je zabitý v čase 10s, potom sa zachránia princezné v poradí : P1, P2, P0, P3, P4.

test3.txt a test3.1.txt – 5x15 mapa, 3 princezné – t = 18s a t = 17s


```
Zadajte cislo testu:
0. ukonci program
1. nacitanie mapy zo suboru
2. preddefinovana mapa
3. vlastna mapa
4. generator mapy
1
Nestihol si zabit draka!
0
```

K drakovi sa popolvár vie dostať za 18s. Pri t=17s sa vypíše chybová hláška, pri t=18s sa nájde najlacnejšia cesta.

Ľubomír Dlhý, ID: 102897

test4.txt – 5x15 mapa, 3 princezné, zablokovaný drak, t = 20s

```
Zadajte cislo testu:
0. ukonci program
1. nacitanie mapy zo suboru
2. preddefinovana mapa
3. vlastna mapa
4. generator mapy
1
Neplatna mapa! Nepodarilo sa zabit Draka!
0
```

Drak je zablokovaný nepriechodnými prekážkami, program vypíše chybovú hlášku.

Test5.txt – 5x15 mapa, 3 princezné, zablokovaná princezná, t = 20s

```
Zadajte cislo testu:
0. ukonci program
1. nacitanie mapy zo suboru
2. preddefinovana mapa
3. vlastna mapa
4. generator mapy
1
Neplatna mapa! Nepodarilo sa zachranit princeznu!
0
```

Princezná je zablokovaná nepriechodnými prekážkami, program vypíše chybovú hlášku.

Test6.txt – 2x10 mapa, 5 princezien, t = 20s

Popolvár prejde cez princeznú 0, aby zabil draka. Potom sa vráti, aby zachránil princezné v poradí P0, P1, P2, P3, P4.

Výstup:

```
0 0
1 0
1 1
1 0
2 0
3 0
4 0
5 0
6 0
7 0
8 0
9 0
12
```

Test7.txt – 2x10 mapa, 5 princezien, t = 20s

Najprv sa zabije drak, potom sa zachránia princezné P1 a P0 a naspäť sa cez ne vráti, aby zachránil P2, P3 a P4.

Výstup:

```
0 0
1 0
1 1
2 1
3 1
4 1
5 1
4 1
4 0
3 0
2 0
1 0
2 0
3 0
4 0
5 0
6 0
7 0
8 0
9 0
20
```

Test8.txt – 2x10 mapa, 5 princezien, t = 20s

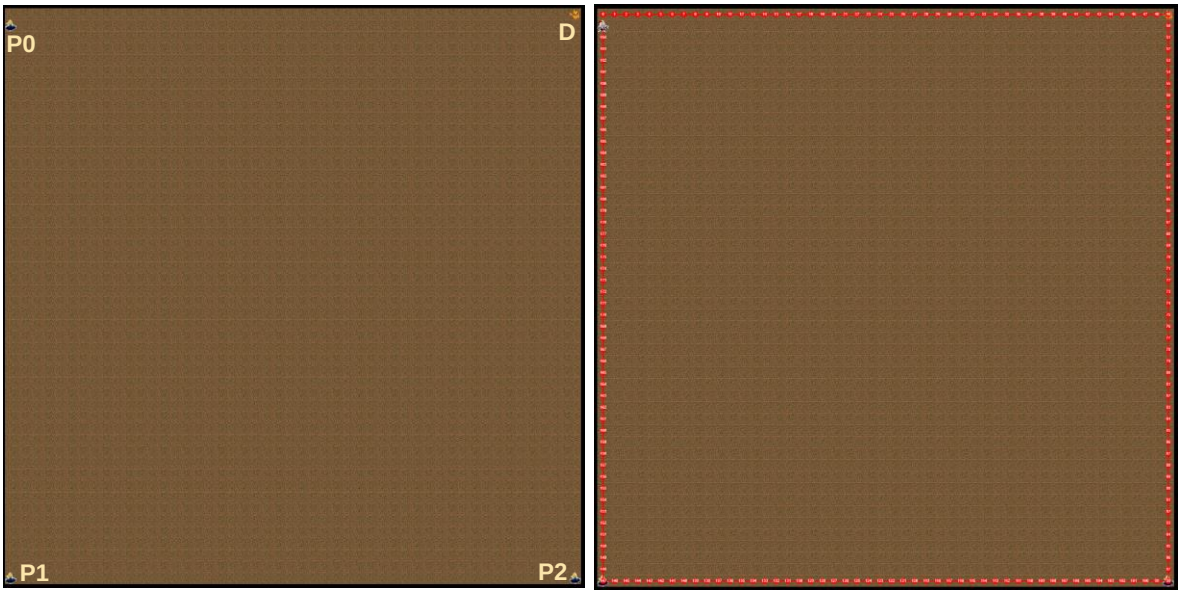
Popolvár sa dostane k drakovi, zabije ho a zachráni princezné v poradí P4, P3, P2, P1, P0.

Výstup:

```
0 0
1 0
1 1
2 1
3 1
4 1
5 1
6 1
7 1
8 1
9 1
9 0
8 0
7 0
6 0
5 0
4 0
3 0
2 0
1 0
20
```

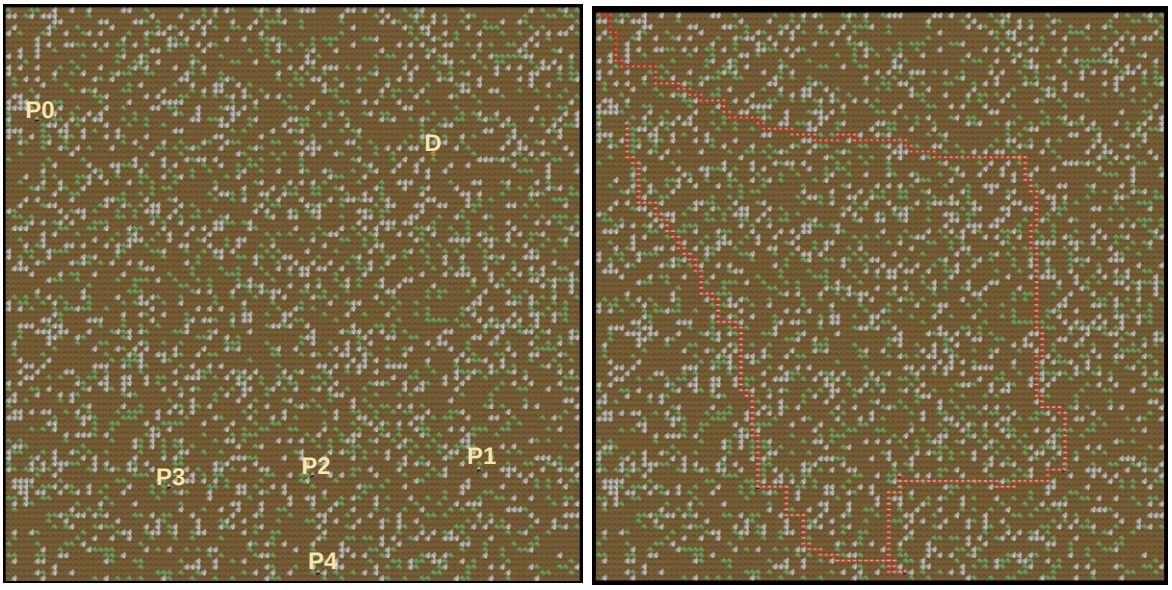
Ľubomír Dlhý, ID: 102897

Test9.txt – 50x50 mapa, 3 princezné, t = 100s



V tomto teste ukazujem, že program funguje aj pre väčšie mapy, draka a princezné som umiestnil do rohov. Popolvár najprv zabije draka v pravom hornom rohu a potom zachráni princezné v poradí P2, P1, P0.

Test10.txt – 100x100 mapa, 5 princezien, t = 200s



Tento test na mape 100x100 dokazuje, že program nemá problém aj s veľkými mapami.

Ľubomír Dlhý, ID: 102897

ZHODNOTENIE

Úlohou zadania bolo vytvoriť program, ktorý v 2-dimenzionálnej mape nájde najlacnejšiu cestu, v ktorej sa najprv zabije drak a potom zachránia všetky princezné. Zadaná úloha je NP-problém, tzn. je vyriešiteľná v polynomiálnom čase. V mojej implementácii som využil permutácie na zistenie globálne najlacnejšej cesty. Čiastkové najlacnejšie cesty, čiže cesty medzi štartom a drakom, drakom a princeznami a navzájom medzi princeznami zisťujem pomocou Dijkstrovho algoritmu s minimálnou binárnou haldou.

Program som otestoval na rôznych scenároch, kde som ukázal správnosť algoritmu a korektné správanie v špeciálnych situáciách ako napríklad zablokovaná princezná. Taktiež som ukázal bezchybný priebeh permutácií.