Zadanie 3 - Popolvár

Autor: Ondrej Harnúšek

Použité dátové štruktúry

Prioritný rad pre Dijkstru - QUEUE_NODE *queue;

- QUEUE_NODE:
    - unsigned int x, y; //súradnice
    - unsigned int g; //generátor = 2 stavy, t.j našiel alebo nenašiel
    - unsigned int d; //drak = 2 stavy, t.j našiel alebo nenašiel
    - unsigned int p; //1. princezná = 2 stavy, t.j našiel alebo nenašiel
    - unsigned int q; //2. princezná = 2 stavy, t.j našiel alebo nenašiel
    - unsigned int r; //3. princezná = 2 stavy, t.j našiel alebo nenašiel
    - unsigned int teleported; //bol teleportovaný = 2 stavy
    - unsigned int priority; //čas cesty do toto vrchola

Matica vrcholov grafu pre Dijkstru - VERTEX tab[2][2][2][2][2][height][width];

- VERTEX:
    - unsigned int x, y; //súradnice
    - unsigned int isVisited; //2 stavy
    - unsigned int time; //čas cesty do vrcholu
    - struct vertex* previous; //vrchol odkiaľ prišiel

Pole odkazov na spájaný zoznam teleportov - TELEPORT *teleports[10];

- TELEPORT:
    - unsigned int x, y; //súradnice
    - struct teleport* next; //ďalší teleport

Opis použitého algoritmu

Stavový priestor je reprezentovaný maticov VERTEX tab[2][2][2][2][2][height][width]) * 2 stavy pre generátor (s/bez) * 2 stavy pre draka (našiel/nenašiel) * 2 stavy pre 1. princeznú (našiel/nenašiel) * 2 stavy pre 2. princeznú (našiel/nenašiel) * 2 stavy pre 3. princeznú (našiel/nenašiel) **stavový priestor sú všetky permutácie = 2^5 = 32 stavov** Dijkstrov algoritmus**v takto zadefinovanom stavovom priestore ho stačí raz spustiť

- \1. inicializácia:
    - prechod tabuľkou: previous=NULL, time=TIME_MAX
    - prechod mapou, nájdenie teleportov a princezien
    - vytvorenie binárnej haldy
- **2. vloženie prvého vrchola do radu: tab[0][0][0][0][0][0][0]**
- \3. Pokiaľ nenájde všetky princezné vyberá z haldy vrcholy a relaxuje ich susedov

    - relaxácia: aktualizácia času suseda, ak čas z vrcholu do suseda je menší ako jeho pôvodný
    - ak vyberie generátor alebo draka nastaví príznak na TRUE
    - ak vyberie princeznú a vyberal už draka nastaví príznak na TRUE
    - ak vyberie teleport, vloží do haldy všetky vrcholy s daným teleportom, s rovnakým časom
    - finálny stav znamená, že sú všetky princezné==TRUE
- **4. Prehľadáva rekurzívne predchádzajúce stavy finálneho stavu a vytvára cestu**

Odhad priestorovej zložitosti

stavový priestor = 32 stavov \1. Prioritný rad pre Dijkstru = 8 bajtov * (32 * výška_mapy * šírka_mapy) \2. Matica všetkých stavov = 8 bajtov * (32 * výška_mapy * šírka_mapy) \3. Pole odkazov na spájaný zoznam teleportov = 8 bajtov * (počet_teleportov) **celková zložitosť = O((64 * výška_mapy * šírka_mapy) + počet_teleportov)**

Odhad časovej zložitosti

- Inicializácia
    - prechod tabuľkou = O(32 * výška_mapy * šírka_mapy)
    - prechod mapou, nájdenie teleportov = O(výška_mapy * šírka_mapy)
- Prioritný rad je implementovaný ako binárna halda s výškou log(n)
    - operácie pop() a push(x) majú preto zložitosť = O(log n)
- Dijkstrov algoritmus v najhoršom prípade vlozí do prioriného radu a vyberie z neho všetky vrcholy grafu.
    - zložitosť O(log n * výška_mapy * šírka_mapy)

celková zložitosť = O((64 * výška_mapy * šírka_mapy) + počet_teleportov)

Zhodnotenie

- Implementácia spĺňa všetky požiadavky a nájde vždy optimálne riešenie=Princezné boli zachránené v najkratšom možnom čase!.
- Časová zložitosť je vďaka prioritnému radu tiež dobrá.
- Trochu horšia je priestorová zložitosť, keďže Dijkstrov algoritmus pracuje naraz nad celým stavovým priestorom.

Testovač

- Otestovanie prioritného radu
    - vloženie 100 pseudo náhodných čísiel naraz
    - postupné vyberanie minima
    - vždy musí vybrať väčšie číslo, ako predchádzajúce
- Otestovanie algoritmu popolvára
    - manuálne vytvorenie textového súboru zo (zaujímavými) mapami a optimálnymi časmi pre Popolvára
    - spustenie na mapách a kontrola, či je cesta optimálna

- navyše, kontrola správneho používania teleporov

Zdrojový kód testov: tl;dr

```
// Pomocné funkcie
int randomize(int seed) {return (unsigned int)((seed * 1103515245 +12345) / 65536) % 139;}
int cmpfunc (const void * a, const void * b){return ( *(int*)a - *(int*)b );}
```

```
/** Otestovanie prioritneho radu*/
int testQ()
{
/** Otestovanie na danej mape*/
    int size=100;
    queue = (QUEUE_NODE*)calloc(sizeof(QUEUE_NODE),size);    //min halda
    queueCount=0;
    int i, temp, arr[size];
    QUEUE_NODE o;

    for(i=0; i<size; i++)
    {
        o.priority = arr[i] = randomize(i);
        pushQ(o);
    }
    qsort(arr, size, sizeof(int), cmpfunc);
    for(i=0; i<size; i++)
    {
        temp = popQ().priority;
        //printf("q = %d\tarr = %d\n",temp , arr[i]);
        if(temp != arr[i])
            return 1;
    }
    free(queue);
    return 0;
}
```

```
int testMap(char **mapa, int n, int m, int t)
{
    int generatorWork = FALSE, teleported = FALSE;
    int i, x, y;
    int timePath=0, *dlzka_cesty;
    int *cesta = zachran_princezne(mapa, n, m, -1, &dlzka_cesty);

    if(cesta==NULL) return 1;
    for(i=0;i<dlzka_cesty;++i)
    {
        x = cesta[i*2];
        y = cesta[i*2+1];
        if(map[y][x]>='0' && map[y][x]<='9')
        {
             if(teleported)
             {
```

```
                teleported = FALSE;
                timePath-=getTime(map[y][x]);
             }
             else teleported = TRUE;
         }
         timePath+=getTime(map[y][x]);
    }
    if(t == timePath) return 0; //Je to optimalna cesta?
    //TLAC
    printf("[%d,%d] optim.time=%d / my.time=%d\n",n, m, t,timePath);
    int j;
    for(i=0;i<dlzka_cesty;++i)
    {
        printf("%d %d\n", cesta[i*2], cesta[i*2+1]);
        map[cesta[i*2+1]][cesta[i*2]] = '.';
    }
    for(i=0; i<height; i++)
    {
        for(j=0; j<width; j++)
            printf("%c ",map[i][j]);

        printf("\n");
    }
    printf("\n");
    return 1;
}
```

```
/** Otestovanie algritmu popolvara*/
int test()
{
    int COUNT = 3;
    int width, height;
    int time,i,j,c,index;
    FILE *fp = fopen("maps.txt","r");

    for(index=0; index<COUNT; index++)
    {
        //nacitaj
        fscanf(fp,"%d %d %d",&height, &width, &time);
        char map[height][width];
        fgetc(fp);
        for(i=0; i<height;i++)
            for(j=0; j<width; j++)
            {
                fscanf(fp,"%c ",&c);
                map[i][j] = c;
            }
        char * mapa[height];
        for(i=0; i<height;i++)
            mapa[i] = map[i];
        //otestuj

        if(testMap(mapa, height, width, time))
```

```
        {
            fclose(fp);
            return 1;
        }
    }
    fclose(fp);
    return 0;
}
```

Príklady testovaných máp

```
5 15 23
C C C C D 1 H H H H H H N N
H N N N 0 N N N N N N H N 0
H N N N N N N N N N N H N P
H N N N N N N N N N N H 1 P
G N N N N N N N N N N N N P

7 7 17
D C C C C H P
N C N N N N C
0 C N 0 0 N C
C C N 0 0 N C
C C N N N N C
C P C C C C P
C C C C C C C

9 6 18
C N N N N N
C 3 N 0 D N
C 2 N N N N
C 1 N 1 P N
C 0 N N N N
G N N 2 P N
N N N N N N
N N N 3 P N
N N N N N N
```