Autor: Daniel Cok

Dokumentácia k zadaniu 3

Algoritmus

Na začiatku preiterujem cez celé pole a do premennej P si zapíšem počet princezien , aby sa mi s nimi v ďalšom priebehu programu ľahšie pracovalo.

```
314         for( i=0;i<n;i++){
315             for( j=0; j<m;j++){
316                 if(map[i][j] == 'P') P++;
317                 type = map[i][j] - 48;
318                 if(type >=0 && type <=9){
319                     XY_path *new = (XY_path*)malloc(sizeof(XY_path));
320                     new->x = i;
321                     new->y = j;
322                     new->next = ports[type];
323                     ports[type] = new;
324                 }
325             }
326         }
```

Následne prechádzam na hľadanie draka alebo ak som ho už našiel prechádzam na hľadanie princezien.

```
331         HEAP *found = search(n,m,'D',&x,&y,&first, &mergedpath);
```

```
338             found = search(n,m,'P',&x,&y,&first, &mergedpath);
```

Vo funkcii search si cele pole znakov namapujem na pole štruktúr, ktoré obsahuje aktuálnu polohu x, y, polohu z ktorej som sa dostal na aktuálnu polohu x_p, y_p, typ políčka na akom stojím, ci som ho už navštívil a vzdialenosť od začiatku prehľadávania.

```
7   typedef struct HEAP{
8       int x;
9       int y;
10      int x_p;
11      int y_p;
12      char type;
13      int visited;
14      int distance;
15  }HEAP;
```

Následne pri prehľadávaní poľa sa vždy pozriem na všetky 4 strany, zistím či som náhodou už suseda nenavštívil a ak nie tak ho vkladám do haldy, v ktorej ich zoraďujem podľa vzdialenosti od začiatočného bodu hľadania.

```
204     if (x>=1 && tracing[x-1][y]->visited != 1) insert(x, y, tracing[x][y]->distance, tracing[x-1][y]);// upper neighbour
205     if (x<n-1 && tracing[x+1][y]->visited != 1) insert(x, y, tracing[x][y]->distance, tracing[x+1][y]);//lower neighbour
206     if (y>=1 && tracing[x][y-1]->visited != 1) insert(x, y, tracing[x][y]->distance, tracing[x][y-1]);//left neighbour
207     if (y<m -1&& tracing[x][y+1]->visited != 1) insert(x, y, tracing[x][y]->distance, tracing[x][y+1]);//right neighbour
```

Z haldy potom vyberám polohu s najkratšou vzdialenosťou a vkladám jej susedov do haldy. Takto to opakujem, až dokým nenájdem hľadaný cieľ. Po nájdení cieľa si zapíšem jeho pozíciu do premennej journey, v ktorej si pamatam pozicie najdenych hodnot(drak, princezna). Následne, pomocou

backtracku si postavím štruktúru/cestu, či už od začiatku alebo od naposledy nájdenej pozície a pripojím ju už k vytvorenej ceste. Vyprázdnim haldu a prechádzam na hľadanie ďalšieho cieľa.

Autor: Daniel Cok

```
332         add_journey(found);
333         mergedpath = mergepaths(mergedpath,backtrack(first,*x,*y,found));
334         free_heap();
```

Hľadanie princezien opakujem viackrát za sebou až pokým nenájdem všetky čo sa v poli nachádzajú.

```
335     for (i = 0;i<P;i++) {
336         x = &found->x;
337         y = &found->y;
338         found = search(n,m,'P',&x,&y,&first, &mergedpath);
339         add_journey(found);
340         mergedpath = mergepaths(mergedpath,backtrack(first,*x,*y,found));
341         free_heap();
342     }
343     int *path = make_array(mergedpath,&dlzka_cesty);
344     return path;
345 }
```

Keďže pri backtracku si cestu zapisujem do štruktúry, nakoniec štruktúru prekopírujem do jednorozmerného poľa a cestu posielam do hlavnej funkcie.

```
285     while(iter!=NULL){
286         counter+=2;
287         path = realloc(path,counter*sizeof(int));
288         path[counter-2] = iter->y;
289         path[counter-1] = iter->x;
290         iter = iter->next;
291     }
```

Časová zložitosť

bubble_up() = O(log n)
bubble_down() = O(log n)
get_min() = O(1)
free_heap() = O(n)
init_visited() = O(n)
type_of() = O(1)
insert = O(1)
add_neighbours = O(E + V log V)
add_journey() = O(1)
mergepaths() = O(1)
backtrack() = O(n)
search() = O(log n)
make_array() = O(n)