Федеральное государственное автономное
образовательное учреждение высшего образования
«Научно-образовательная корпорация ИТМО»

Факультет программной инженерии и компьютерной техники
Направление подготовки 09.03.04 Программная инженерия

Отчёт по лабораторной работе №4

По дисциплине «Алгоритмы и структуры данных» (4 семестр)
Четвёртый блок задач

Студент:
Дениченко Александр P3212
Преподаватели:
Косяков М.С.
Тараканов Д.С.

Санкт-Петербург
2024 г.

1 Задача №M «Цивилизация»

```cpp
#include <iostream>
#include <queue>
#include <list>
#include <vector>

using namespace std;

struct Node {
    int cost;
    int marked;
    int x;
    int y;
    int predecessor;

    Node(int cost = -1, int marked = 0, int x = -1, int y = -1, int pred = -1)
        : cost(cost), marked(marked), x(x), y(y), predecessor(pred) {}

    bool operator>(const Node& other) const {
        return this->cost > other.cost;
    }
};

using NodeQueue = priority_queue<Node, vector<Node>, greater<Node> >;

int main() {
    int N, M, start_x, start_y, finish_x, finish_y;
    cin >> N >> M >> start_x >> start_y >> finish_x >> finish_y;
    start_x--;
    start_y--;
    finish_x--;
    finish_y--;
    int size = N * M;

    vector<vector<pair<int, int> > > map(N, vector<pair<int, int> >(M));

    string tmp;
    for (int i = 0; i < N; i++) {
        cin >> tmp;
        for (int j = 0; j < M; j++) {
            if (tmp[j] == '.') {
                map[i][j].first = 1;
                map[i][j].second = 0;
            } else if (tmp[j] == 'W') {
                map[i][j].first = 2;
                map[i][j].second = 0;
            } else {
                map[i][j].first = 666;
                map[i][j].second = 0;
            }
        }
    }

    NodeQueue pq_sorted;
    vector<Node> visited(size);
```

```
55
56      pq_sorted.push(Node(0, 1, start_x, start_y));
57      visited[start_x * M + start_y] = Node(0, 1, start_x, start_y, -1);
58
59      while (true) {
60          if(pq_sorted.empty()) break;
61          Node curr = pq_sorted.top();
62          pq_sorted.pop();
63
64          int curr_index = curr.x * M + curr.y;
65          if (curr_index == finish_x * M + finish_y) break;
66
67
68          int new_y = curr.y-1;
69          if (new_y >= 0 && new_y < M && map[curr.x][new_y].first != 666 && !map[curr.x][new_y]
70              int new_cost = curr.cost + map[curr.x][new_y].first;
71              int next_index = curr.x * M + new_y;
72              if (visited[next_index].cost == -1 || visited[next_index].cost > new_cost) {
73                  pq_sorted.push(Node(new_cost, 1, curr.x, new_y, curr_index));
74                  visited[next_index] = Node(new_cost, 1, curr.x, new_y, curr_index);
75                  map[curr.x][new_y].second = 1;
76              }
77          }
78
79          new_y = curr.y+1;
80          if (new_y >= 0 && new_y < M && map[curr.x][new_y].first != 666 && !map[curr.x][new_y]
81              int new_cost = curr.cost + map[curr.x][new_y].first;
82              int next_index = curr.x * M + new_y;
83              if (visited[next_index].cost == -1 || visited[next_index].cost > new_cost) {
84                  pq_sorted.push(Node(new_cost, 1, curr.x, new_y, curr_index));
85                  visited[next_index] = Node(new_cost, 1, curr.x, new_y, curr_index);
86                  map[curr.x][new_y].second = 1;
87              }
88          }
89
90          int new_x = curr.x-1;
91          if (new_x >= 0 && new_x < N && map[new_x][curr.y].first != 666 && !map[new_x][curr.y]
92              int new_cost = curr.cost + map[new_x][curr.y].first;
93              int next_index = new_x * M + curr.y;
94              if (visited[next_index].cost == -1 || visited[next_index].cost > new_cost) {
95                  pq_sorted.push(Node(new_cost, 1, new_x, curr.y, curr_index));
96                  visited[next_index] = Node(new_cost, 1, new_x, curr.y, curr_index);
97                  map[new_x][curr.y].second = 1;
98              }
99          }
100
101         new_x = curr.x+1;
102         if (new_x >= 0 && new_x < N && map[new_x][curr.y].first != 666 && !map[new_x][curr.y]
103             int new_cost = curr.cost + map[new_x][curr.y].first;
104             int next_index = new_x * M + curr.y;
105             if (visited[next_index].cost == -1 || visited[next_index].cost > new_cost) {
106                 pq_sorted.push(Node(new_cost, 1, new_x, curr.y, curr_index));
107                 visited[next_index] = Node(new_cost, 1, new_x, curr.y, curr_index);
108                 map[new_x][curr.y].second = 1;
109             }
110         }
```

```
111
112     }
113
114
115     if (visited[finish_x * M + finish_y].cost == -1){
116         cout << visited[finish_x * M + finish_y].cost << endl;
117         return 0;
118     }
119
120     string way;
121     int curr = finish_x * M + finish_y;
122     while (curr != -1 && visited[curr].predecessor != -1) {
123         int pred = visited[curr].predecessor;
124         int diff = curr - pred;
125         if (diff == 1) way = 'E' + way;
126         else if (diff == -1) way = 'W' + way;
127         else if (diff == M) way = 'S' + way;
128         else if (diff == -M) way = 'N' + way;
129         curr = pred;
130     }
131
132     cout << visited[finish_x * M + finish_y].cost << endl;
133     cout << way;
134     return 0;
135 }
```

Описание: В начале было сложно понять, что и как эту задачу решать. Но после просмотра нескольких лекций на тему графы, выбор пал на алгоритм Дейкстры. Создаются структуры для хранения узлов сетки, каждый узел инициализируется начальной стоимостью, состоянием и координатами. Далее извлекаем узел с минимальной стоимостью из очереди с приоритетами и рассматривает его соседей клеткой. Если переход в соседний узел уменьшает стоимость пути, узел добавляется в очередь с обновленной стоимостью. В конце можно легко восстановить путь от целевой точки.
Сложность: Сложность алгоритма Дейкстры составляет $O((V+E)\log V)$, где V — количество вершин (узлов) в графе, а E — количество ребер (связей между узлами). В контексте сетки размером $N \times M$: $V = N \times M$; E приблизительно равно $4 \times V$ (поскольку каждый узел связан с четырьмя соседями). Таким образом, сложность алгоритма для сетки составляет $O((N \times M + 4 \times N \times M)\log(N \times M))$ или $O(N \times M \log(N \times M))$.

2 Задача №O «Долой списывание!»

```
1  #include <iostream>
2  #include <vector>
3  #include <stack>
4  using namespace std;
5
6  vector<vector<int> > graph;
7  int colors[101] = {0};
8
9  bool dfs(int start){
10     stack<pair<int, int> > s;
11     pair<int, int> tmp;
12     tmp.first = start;
13     tmp.second = 1;
14     s.push(tmp);
15
16     while (!s.empty()) {
17         pair<int, int> top = s.top(); s.pop();
```

```
        int v = top.first;
        int color = top.second;

        if (colors[v] == 0) {
            colors[v] = color;
        }

        for (int next : graph[v]) {
            if (colors[next] == 0) {
                tmp.first = next;
                if(color == 1){
                    tmp.second = 2;
                }else{
                    tmp.second = 1;
                }
                s.push(tmp);
            } else if (colors[next] == color) {
                return false;
            }
        }
    }
    return true;
}

int main() {
    int N, M, x1, x2;
    cin >> N >> M;
    graph.resize(N+1);

    for (int i = 0; i < M; i++){
        cin >> x1 >> x2;
        graph[x1].push_back(x2);
        graph[x2].push_back(x1);
    }

    for (int i = 0; i < N; i++){
        if (colors[i] == 0 && !dfs(i)) {
            cout << "NO";
            return 0;
        }
    }

    cout << "YES";
    return 0;
}
```

Описание: Задача относительно проста, быстро додумался что нужно простро проверить граф на двудольность, вспомнит Полякова с его дискреткой и раскроской графа (все цвета можно взять у человека для раскраски, допустим из уха и из носа будет достаточно для этой задачи). Был использован DFS при помощи явного стека (увидел на википедии что-то подобное и решил попробовать использовать суть). Каждый узел представлен парой, где first — это идентификатор узла, а second — цвет для назначения (либо 1, либо 2). Проходимся по каждой вершине и если вершина еще не окрашена (colors[v] == 0), то ей присваивается цвет и она помещается в стек. Пока стек не пуст, программа проверяет каждую вершину v, проверяет её цвет и пытается покрасить смежные вершины в альтернативный цвет. Если она встречает смежную вершину, уже окрашенную в тот же цвет, что и v, граф не является двудольным.

Сложность: Операция DFS имеет временную сложность $O(V + E)$, где V — количество вершин, а E — количество ребер.