Университет ИТМО

Факультет программной инженерии и компьютерной техники

Лабораторная работа №4

по «Алгоритмам и структурам данных»

Базовые задачи / Timus

Выполнил:

Студент группы P3234

Баянов Р.Д.

Преподаватели:

Косяков М.С.

Тараканов Д.С.

Санкт-Петербург

2024

Яндекс контест

Задача M. Цивилизация

Ход решения:

Для решения данной задачи я использовал алгоритм Дейкстры. В самом начале подготавливаем данные, то есть создаём вектор строк, для того чтобы иметь какое-то представление карты игры в решении. Подготавливаем все структуры данных для алгоритма Дейкстры: создаём вектор предков *p*, создаём приоритетную очередь *q* и компаратор для неё, вектор посещённых вершин *visited* и вектор соседних вершин для выполнения релаксаций *vertices*. Затем по алгоритму мы будем ходить по нашей карте и аккуратно высчитывать наикратчайший путь для вершинки, в которую мы попали. И не забываем в конце каждого перехода, проводить релаксации, для подсчёта, возможно, нового изменившегося наикратчайшего пути для вершины. Сразу после того, как алгоритм Дейкстры закончит своё выполнение, по вектору предков *p* мы легко можем восстановить путь, проделанный до нашей целевой вершины. Выводим ответ в конце!

Оценка сложности решения:

Время: O ((V + E) log V)

Память: O (N * M)

Код решения:

```
#include <iostream>
#include <vector>
#include <string>
#include <queue>
#include <limits>
#include <algorithm>

enum Moves {
        DOWN = 'S',
        RIGHT = 'E',
        LEFT = 'W',
        UP = 'N'
};

struct CompareSecond {
        bool operator()(const std::pair<int, int>& a, const std::pair<int,
```

```
int>& b) {
17.             return a.second > b.second;
18.       }
19. };
20.
21. int main()
22. {
23.       std::cin.tie(0);
24.       std::ios_base::sync_with_stdio(0);
25.       int INF = std::numeric_limits<int>::max();
26.       int N, M, x1, y1, x2, y2;
27.       std::cin >> N >> M >> x1 >> y1 >> x2 >> y2;
28.       std::string moves = "";
29.       std::string str;
30.       std::vector<std::string> grid(N);
31.       int start = (x1 - 1) * M + (y1 - 1);
32.       int goal = (x2 - 1) * M + (y2 - 1);
33.       std::vector<int> visited(N * M, INF);
34.       visited[start] = 0;
35.       for (int i = 0; i < N; i++) {
36.             std::cin >> grid[i];
37.       }
38.       std::vector<int> p (N * M);
39.       std::priority_queue<std::pair<int, int>, std::vector<std::pair<int,
int>>, CompareSecond> q;
40.       q.push({start, 0});
41.       std::vector<std::pair<int, int>> vertices;
42.       while (!q.empty()) {
43.             std::pair<int, int> v = q.top();
44.             x1 = v.first / M;
45.             y1 = v.first % M;
46.             q.pop();
47.             if (x1 == x2 && y1 == y2) {
48.                   break;
49.             }
50.             if (grid[x1][y1] == '#') {
51.                   continue;
52.             }
53.             if (x1 - 1 >= 0 && grid[x1 - 1][y1] != '#') {
54.                   vertices.push_back({ v.first - M, (grid[x1 - 1][y1] ==
'W') ? 2 : 1 });
55.             }
56.             if (x1 + 1 < N && grid[x1 + 1][y1] != '#') {
57.                   vertices.push_back({ v.first + M, (grid[x1 + 1][y1] ==
'W') ? 2 : 1 });
58.             }
59.             if (y1 - 1 >= 0 && grid[x1][y1 - 1] != '#') {
60.                   vertices.push_back({ v.first - 1, (grid[x1][y1 - 1] ==
'W') ? 2 : 1 });
61.             }
```

```
            if (y1 + 1 < M && grid[x1][y1 + 1] != '#') {
                  vertices.push_back({ v.first + 1, (grid[x1][y1 + 1] ==
'W') ? 2 : 1 });
            }
            for (std::pair<int, int> val : vertices) {
                  int new_len = val.second + v.second;
                  if (visited[val.first] == INF || visited[val.first] >
new_len) {
                        q.push({val.first, new_len});
                        visited[val.first] = new_len;
                        p[val.first] = v.first;
                  }
            }
            vertices.clear();
      }
      if (visited[goal] == INF) {
            std::cout << -1;
            return 0;
      }
      if (visited[goal] == 0) {
            std::cout << visited[goal];
      }
      else {
            std::cout << visited[goal] << std::endl;
      }
      for (int v = goal; v != start; v = p[v]) {
            p.push_back(v);
      }
      p.push_back(start);
      std::reverse(p.begin(), p.end());
      for (int i = 0; i < N * M; i++) {
            if (p[i] == goal) {
                  break;
            }
            int diff = p[i + 1] - p[i];
            if (diff == -1) {
                  moves += LEFT;
            }
            else if (diff == 1) {
                  moves += RIGHT;
                  }
                  else if (diff == M) {
                        moves += DOWN;
                  }
                  else if (diff == -M) {
                        moves += UP;
                  }
            }
            std::cout << moves;
            return 0;
```

```
110.      }
111.
```

Задача N. Свинки-копилки

Ход решения:

Для решения данной задачи будем использовать алгоритм dfs. Чтобы получить ответ нам нужно посчитать все компоненты связности нашего графа. При обработке входных данных будем каждое ребро дублировать, чтобы при попадании в какую-то компоненту связности мы могли из любой её вершины обойти все вершины этой компоненты. Мы знаем, что dfs обойдёт все вершины, поэтому при остановке алгоритма dfs мы можем с уверенностью сказать, что одна компонента связности обнаружена и все её вершины помечены. Затем, мы будем искать другую непомеченную вершину и снова запустим dfs, но уже с этой вершины. Таким образом после каждого dfs увеличиваем счётчик и получаем ответ.

Оценка сложности решения:

Время: O (V + E) = O (N)

Память: O (N^2)

Код решения:

```
1.  #include <iostream>
2.  #include <vector>
3.  #include <queue>
4.
5.  enum Color {
6.        WHITE = 0,
7.        BLACK = 1
8.  };
9.
10. void dfs( std::vector<std::vector<int>>& graph, std::vector<int>& visited,
    int v) {
11.       visited[v] = BLACK;
12.       for (int u : graph[v]) {
13.             if (visited[u] == WHITE) {
```

```
                    dfs(graph, visited, u);
                }
        }
}

int main()
{
        int n;
        std::cin >> n;
        std::vector<std::vector<int>> graph(n + 1);
        std::vector<int> visited(n + 1, WHITE);
        int e;
        for (int v = 1; v < n + 1; ++v) {
                std::cin >> e;
                graph[v].push_back(e);
                graph[e].push_back(v);
        }
        int count = 0;
        for (int i = 1; i < graph.size(); i++) {
                if (visited[i] == WHITE) {
                        count++;
                        dfs(graph, visited, i);
                }
        }
        std::cout << count;
        return 0;
}

```

Задача O. Долой списывание!

Ход решения:

Для решения данной задачи я использовал алгоритм bfs. Нетрудно догадаться, что здесь надо проверить является ли граф двудольным. Таким образом, если граф двудольный мы легко сможешь разбить лкшат на две группы. В этом нам поможет алгоритм bfs. Мы будем аккуратно обходить граф в ширину и раскрашивать вершины в два цвета. И если мы всё-таки встретим ситуацию, когда две соседние вершины раскрашены в один и тот же цвет, то граф не двудольный. Вот и всё!

Оценка сложности решения:

Время: O (V + E) = O (N)

Память: O (N^2)

Код решения:

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

enum Color {
	WHITE,
	BLUE,
	RED
};

#define WHITE -1
#define BLUE 0
#define RED 1

bool is_bipartite(std::vector<std::vector<int>>& graph) {
	std::queue<int> q;
	std::vector<int> visited(graph.size(), WHITE);
	for (int s = 1; s < graph.size(); s++) {
		if (visited[s] == WHITE) {
			visited[s] = BLUE;
			q.push(s);
			while (!q.empty()) {
				int u = q.front();
				q.pop();
				for (int v : graph[u]) {
					if (visited[v] == -1) {
						visited[v] = 1 - visited[u];
						q.push(v);
					}
					else if (visited[u] == visited[v]) {
						return false;
					}
				}
			}
		}
	}
	return true;
}

int main()
{
	int N, M;
	std::cin >> N >> M;
	std::vector<std::vector<int>> graph(N + 1);
	int v, u;
	for (int i = 0; i < M; i++) {
		std::cin >> v >> u;
		graph[u].push_back(v);
		graph[v].push_back(u);
	}
	if (is_bipartite(graph)) {
		std::cout << "YES";
	}
	else {
		std::cout << "NO";
	}
	return 0;
}
```

Задача P. Авиаперелёты

Ход решения:

Для решения данной задачи я использовал dfs и бин поиск. Грубо говоря, мы найдём в матрице максимум, который потом будет служить нам верхней границей для диапазона бин поиска. Мы создадим функцию проверки графа на то, что с каким-то определённым размером бака самолёта, мы сможем облететь все города. В этой функции проверки будем проходиться двумя dfs по графу в прямом направлении и в обратном. Если всё же при данный размер бака нам подходит, то самолёт облетит все города и все вершины графа станут помеченными и мы точно будем знать подходит нам такой бак или нет. И так далее пока бин поиск не обнаружит минимальный такой бак. Готово!

Оценка сложности решения:

Время: O (N^2 log E)

Память: O(V), где V – кол-во вершин

Код решения:

```
#include <iostream>
#include <vector>
#include <limits.h>

struct vertex {
	int id;
	int weight;
};


void dfs1(std::vector<std::vector<vertex>>& graph, int val, int v, std::vec-
tor<int>& visited) {
	visited[v] = 1;
	for (int i = 0; i < graph.size(); ++i) {
		if (!visited[i] && graph[i][v].weight <= val) {
			dfs1(graph, val, i, visited);
		}
	}
```

```
18. }
19.
20. void dfs2(std::vector<std::vector<vertex>>& graph, int val, int v, std::vec-
    tor<int>& visited) {
21.       visited[v] = 1;
22.       for (int i = 0; i < graph.size(); ++i) {
23.             if (!visited[i] && graph[v][i].weight <= val) {
24.                   dfs2(graph, val, i, visited);
25.             }
26.       }
27. }
28.
29. bool check(std::vector<std::vector<vertex>>& graph, int val) {
30.       int size = graph.size();
31.       std::vector<int> visited (size, 0);
32.       dfs1(graph, val, 0, visited);
33.       for (int i = 0; i < size; ++i) {
34.             if (!visited[i]) {
35.                   return false;
36.             }
37.       }
38.       std::fill(visited.begin(), visited.end(), 0);
39.       dfs2(graph, val, 0, visited);
40.       for (int i = 0; i < size; ++i) {
41.             if (!visited[i]) {
42.                   return false;
43.             }
44.       }
45.       return true;
46. }
47.
48. int bin_search(int max, std::vector<std::vector<vertex>>& graph) {
49.       int l = 0;
50.       int r = max;
51.       while (r - l > 1) {
52.             int m = (l + r) / 2;
53.             if (!check(graph, m)) {
54.                   l = m;
55.             }
56.             else {
57.                   r = m;
58.             }
59.       }
60.       return r;
61. }
62.
63. int main()
64. {
65.       int n;
66.       std::cin >> n;
```

```
67.       vertex v;
68.       std::vector<std::vector<vertex>> graph(n);
69.       int oil;
70.       int max = INT_MIN;
71.       for (int i = 0; i < n; i++) {
72.             for (int j = 0; j < n; j++) {
73.                   std::cin >> oil;
74.                   max = std::max(oil, max);
75.                   vertex v;
76.                   v.id = j;
77.                   v.weight = oil;
78.                   graph[i].push_back(v);
79.             }
80.       }
81.       std::cout << bin_search(max, graph);
82.       return 0;
83. }
84.
```

Тимус

Задача 1162. Currency Exchange

Ход решения:

Для каждого ребра в графе будем проверять можно ли увеличить кол-во денег в вершине, в которую ведёт это ребро, обменивая валюту в вершине, из которой ведёт это ребро. Если да, то сума денег в вершине обновляется. Затем проверка на то, что у нас есть цикл в графе, который позволяет увеличить свою сумму денег. И выводим ответ на вопрос!

Оценка сложности решения:

Время: O (V * E)

Память: O (V + E)

Код решения:

```
1. #include <iostream>
```

```cpp
#include <list>
#include <vector>
using namespace std;

struct edge {
    int from, to;
    double rate, commission;
};

int main() {
    int N, M, S, A, B;
    double V, RAB, CAB, RBA, CBA;
    cin >> N >> M >> S >> V;
    list<edge> edges;
    S--;
    for (int i = 0; i < M; i++) {
        cin >> A >> B >> RAB >> CAB >> RBA >> CBA;
        A--; B--;
        edges.push_back({ A, B, RAB, CAB });
        edges.push_back({ B, A, RBA, CBA });
    }

    std::vector<double> currency(N, 0);
    currency[S] = V;

    for (int i = 0; i < N - 1; i++)
        for (edge x : edges)
            if ((currency[x.from] - x.commission) * x.rate > currency[x.to])
                currency[x.to] = (currency[x.from] - x.commission) * x.rate;

    for (edge x : edges)
        if ((currency[x.from] - x.commission) * x.rate > currency[x.to]) {
            cout << "YES";
            return 0;
        }
    cout << "NO";
    return 0;
}

```

Задача 1450. Российские газопроводы

Ход решения:

Для решения этой задачи создадим вектор d, который будет хранить расстояния от начальной вершины до всех остальных вершин в графе. Затем в цикле, выполняется алгоритм Беллмана-Форда. На каждой итерации будем проверять для каждого ребра в графе можно ли улучшить расстояние до вершины, в которую ведёт это ребро ,используя текущее расстояние до вершины, из которой ведёт это ребро и вес этого ребра. В конце проверяем найдено ли решение. Если расстояние до конечной вершины равно 1, то решение не найдено, в противном случае выводим ответ. Всё!

Оценка сложности решения:

Время: O (N * M)

Память: O (N + M)

Код решения:

```
#include <iostream>
#include <vector>

using namespace std;


int main() {

    ios::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);

    int n, m, start, finish;
    cin >> n >> m;

    vector<vector<int> > edges(m, vector<int>(3));
    for (int i = 0; i < m; ++i) {
        cin >> edges[i][0] >> edges[i][1] >> edges[i][2];
        edges[i][0]--;
        edges[i][1]--;
    }
    cin >> start >> finish;
    vector<int> d(n, -1);

    d[--start] = 0;
    for (int i = 0; i < n - 1; ++i){
```

```
27.         for (int j = 0; j < m; ++j){
28.             if (d[edges[j][0]] > -1) {
29.                 d[edges[j][1]] = max(d[edges[j][1]], d[edges[j][0]] +
    edges[j][2]);
30.             }
31.         }
32.     }
33.     if(d[--finish] == 1) {
34.         cout << "No solution";
35.         return 0;
36.     }
37.     cout << d[finish];
38.
39.     return 0;
40. }
```