Алгоритм обхода в ширину для матриц

Deeg05

Январь 2023

1 Алгоритм

Предположим, у нас есть несколько складов, располагающихся в определённых координатах на карте, к примеру, в (1, 1), (3, 2), (5, 4). Введём матрицу aDistance, хранящую минимальное расстояние до ячейки. Очевидно, что на склад мы можем доехать за 0 (уже здесь), а все остальные ячейки заполним числом -1 как необработанные:

$$aDistance = \begin{bmatrix} 0 & -1 & -1 & -1 & -1 \\ -1 & -1 & 0 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & 0 \\ -1 & -1 & -1 & -1 & -1 \end{bmatrix}$$

Теперь заводим очередь nQueue, куда изначально заносим координаты складов:

$$nQueue = \begin{bmatrix} (1,1) & (3,2) & (5,4) \end{bmatrix}$$

Забираем из очереди nQueue элемент:

$$nQueue = \begin{bmatrix} (3,2) & (5,4) \end{bmatrix}$$

Это (1, 1), далее обрабатываем смежные ячейки так, что их значение на 1 больше, чем значение изначально взятой ячейки:

$$aDistance = \begin{bmatrix} 0 & 1 & -1 & -1 & -1 \\ 1 & -1 & 0 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & 0 \\ -1 & -1 & -1 & -1 & -1 \end{bmatrix}$$

К тому же, заносим смежные ячейки в очередь nQueue:

$$nQueue = \begin{bmatrix} (3,2) & (5,4) & (1,2) & (2,1)... \end{bmatrix}$$

Повторяем подобные манипуляции до тех пор, пока очередь не будет пуста. В результате мы получим подобную матрицу aDistance:

$$aDistance = \begin{bmatrix} 0 & 1 & 1 & 2 & 3 \\ 1 & 1 & 0 & 1 & 2 \\ 2 & 2 & 1 & 2 & 1 \\ 3 & 3 & 2 & 1 & 0 \\ 4 & 4 & 3 & 2 & 1 \end{bmatrix}$$

2 Восстановление ответа

Чтобы восстановить ответ (кратчайший путь от любого склада до вершины) потребуется хранить массив предков aDistance:

$$aPrevious = \begin{bmatrix} (1,1) & -1 & 1 & -1 & -1 \\ -1 & -1 & (2,2) & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & (5,4) \\ -1 & -1 & -1 & -1 & -1 \end{bmatrix}$$

Требуется модифицировать предыдущий алгоритм так, чтобы при каждом присвоении ячейке матрицы aDistance нового значения в ячейку с такими же координатами матрицы aPrevious записывались координаты предка

$$aPrevious = \begin{bmatrix} (1,1) & (1,1) & (3,2) & (3,1) & ...^{1} \\ (1,1) & (3,2) & (3,2) & (3,2) & ...^{1} \\ (1,2) & (2,2) & (3,2) & (3,2) & (5,4) \\ ...^{1} & ...^{1} & ...^{1} & ...^{1} & (5,4) \\ ...^{1} & ...^{1} & ...^{1} & ...^{1} & ...^{1} \end{bmatrix}$$

Теперь мы можем идти от конечной ячейки по предкам (до того, как предыдущее не будет равно текущему), тем самым восстанавливая кратчайший путь в обратном порядке.

[1] и так далее...

3 Примеры задач с решением

3.1 Задача 1. Блохи

Данная задача на ресурсе "Информатикс"МЦНМО `https://informatics.msk.ru/mod/statements/view.php?chapterid=1668`

3.1.1 Условие

На клеточном поле, размером N x M ($2 \leq N, M \leq 250$) сидит Q ($0 \leq Q \leq 10000$) блох в различных клетках. "Прием пищи"блохами возможен только в кормушке - одна из клеток поля, заранее известная. Блохи перемещаются по полю странным образом, а именно, прыжками, совпадающими с ходом обыкновенного шахматного коня. Длина пути каждой блохи до кормушки определяется как количество прыжков. Определить минимальное значение суммы длин путей блох до кормушки или, если собраться блохам у кормушки невозможно, то сообщить об этом. Сбор невозможен, если хотя бы одна из блох не может попасть к кормушке.

3.1.2 Входные данные

В первой строке входного файла находится 5 чисел, разделенных пробелом: N, M, S, T, Q. N, M - размеры доски (отсчет начинается с 1); S, T - координаты клетки - кормушки (номер строки и столбца соответственно), Q - количество блох на доске. И далее Q строк по два числа - координаты каждой блохи.

3.1.3 Выходные данные

Содержит одно число - минимальное значение суммы длин путей или -1, если сбор невозможен. ususbsubsectionРешение

3.1.4 Решение на языке C++

```
#include <iostream>
#include <vector>
#include <queue>
int N, M, S, T, Q;
// Для удобства работы с очередью будемиспользовать
// специальную структуру (вместо std::pair<int, int>)
struct cell {
        int x, y;
};
int main() {
        std::cin >> N >> M >> S >> T >> Q;
        cell start = {S - 1, T - 1};
        // Суммарное количество прыжков, требуемое блохами для сбо-
ра
        int sum = 0;
        // Заводим очередь q
        std::queue<cell> q;
```

```
        // Заводим матрицу кратчайших расстояний
        std::vector<std::vector<int>> d(N, std::vector<int>(M, -1));
        d[start.x][start.y] = 0;
        q.push(start);
        // Смежные ячейки
        cell mov[8] = {{-2, -1},
                                {-2, 1},
                                {-1, 2},
                                {1, 2},
                                {2, 1},
                                {2, -1},
                                {-1, -2},
                                {1, -2}
        };
        while (!q.empty()) {
                // "Вынимаем"элемент из очереди
                cell pr = q.front();
                q.pop();
                for (cell ce : mov) {
                        // Конечные координаты
                        int _x = pr.x + ce.x, _y = pr.y + ce.y;
                        // Проверяем ячейку, в которую ходим
                        if (    _x >= 0
                                        && _x < N
                                        && _y >= 0
                                        && _y < M
                                        && d[_x][_y] == -1) {
                                d[_x][_y] = d[pr.x][pr.y] + 1;
                                q.push({_x, _y});
                        }
                }
        }
        // Для каждой блохи
        for (size_t i = 0; i < Q; i++) {
                int x, y;
                std::cin >> x >> y;
                cell e = {x - 1, y - 1};
                // Добавляем кратчайшее расстояние до конкретной
блохи в переменную sum, если можно дойти, иначе выводим -1 и заверша-
ем программу
                if (d[x - 1][y - 1] != -1)
                        sum += d[x - 1][y - 1];
                else {
                        std::cout << -1 << std::endl;
                        return 0;
                }
        }
        std::cout << sum << std::endl;
        return 0;
}
```

3.2 Задача 2. Табличка

3.2.1 Условие

Дана таблица, состоящая из N строк и M столбцов. В каждой клетке таблицы записано одно из чисел: 0 или 1. Расстоянием между клетками (x1, y1) и (x2, y2) назовем сумму |x1-x2|+|y1-y2|. Вам необходимо построить таблицу, в клетке (i, j) которой будет записано минимальное расстояние между клеткой (i, j) начальной таблицы и клеткой, в которой записана 1. Гарантируется, что хотя бы одна 1 в таблице есть.

3.2.2 Входные данные

В первой строке вводятся два натуральных числа N и M, не превосходящих 500. Далее идут N строк по M чисел - элементы таблицы.

3.2.3 Выходные данные

Требуется вывести N строк по M чисел - элементы искомой таблицы.

3.2.4 Решение на языке C++

```
#include <iostream>
#include <vector>
#include <queue>
#define forr(i,s,f) for (int i = s; i < f; i++)
#define pb push_back
struct cell {
        int x, y;
        cell(int x, int y) {
                this->x = x;
                this->y = y;
        }
};
int main() {
        int n, m;
        std::cin >> n >> m;
        std::vector<std::vector<int>> t(n, std::vector<int>(m));
        std::vector<std::vector<int>> d(n, std::vector<int>(m, -1));
        std::queue<cell> q;
        cell mov[4] = {{0, 1}, {0, -1}, {1, 0}, {-1, 0}};
        forr(i, 0, n) {
                forr(j, 0, m) {
                        std::cin >> t[i][j];
                        if (t[i][j] == 1) {
                                q.push(cell(j, i));
                                d[i][j] = 0;
                        }
                }
        }
        while (!q.empty()) {
```

```
                cell pr = q.front();
                q.pop();
                for (cell ce : mov) {
                        int _x = pr.x + ce.x, _y = pr.y + ce.y;
                        if (_x >= 0 && _x < m
                                        && _y >= 0
                                        && _y < n
                                        && d[_y][_x] == -1) {
                                d[_y][_x] = d[pr.y][pr.x] + 1;
                                q.push(cell(_x, _y));
                        }
                }
        }
        forr(i, 0, n) {
                forr(j, 0, m) {
                        std::cout << d[i][j] << "␣";
                }
                std::cout << std::endl;
        }
}
```