**Московский авиационный институт (национальный исследовательский университет)**

**Факультет информационных технологий и прикладной математики**

**Кафедра вычислительной математики и программирования**

# Лабораторная работа № 6 по курсу «Дискретный анализ»

Студент: Бачурин П.Д.
Преподаватель: Макаров Н.К.
Даты:
Оценка:
Подпись:

**Москва, 2024**

**Задача**

Задана матрица натуральных чисел *A* размерности *n* × *m*. Из текущей клетки можно перейти в любую из 3-х соседних, стоящих в строке с номером на единицу больше, при этом за каждый проход через клетку *(i, j)* взымается штраф *Ai*,*j*. Необходимо пройти из какой-нибудь клетки верхней строки до любой клетки нижней, набрав при проходе по клеткам минимальный штраф.

Формат ввода

Первая строка входного файла содержит в себе пару чисел $2 \leqslant \mathrm{n} \leqslant 1000$ и $2 \leqslant \mathrm{m} \leqslant 1000$, затем следует n строк из m целых чисел.

Формат вывода

Необходимо вывести в выходной файл на первой строке минимальный штраф, а на второй – последовательность координат из *n* ячеек, через которые пролегает маршрут с минимальным штрафом.

**Алгоритм**

1. Инициализация данных:

   - Считываем размеры матрицы штрафов A, а затем считываем саму матрицу.

2. Алгоритм обхода матрицы A:

   - Для каждой строки A[y] матрицы A, начиная с n-1 строки (в 1 индексации), мы проходим по всем элементам A[y][x] и выбираем наилучший из которого мы могли придти в этот элемент (допустим это элемент с индексом y+1, x_good). Мы запоминаем, что в элемент [y, x] мы пришли из [y+1, x_good](prev[y][x] = [y+1, x_good] ). И в A[y][x] прибавляем A[y+1][x_good], где A[y+1][x_good] — наименьшая стоимость пути до элемента [y+1, x_good].

   - После того, как мы прошлись по всем строкам, то мы смотрим в первую строку и находим минимальную цену пути best_price и запоминаем позицию этого элемента [y_best, x_best]. Это и есть минимальная цена, за которую можно пройти матрицу.

   - Для восстановления пути мы итерационно смотрим откуда мы пришли в элемент [y_best][x_best] и говорим, что [y_best, x_best] = prev[y_best, x_best]. Так мы делаем n раз, для каждой строки.

5. Вывод результата:

   - В процессе восстановления пути, так как мы изначально обходили матрицу снизу вверх, мы можем просто выводить текущий [y_best, x_best], так как путь восстанавливается от конца к началу, то есть сверху вниз.

**Исходный код**

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

const std::vector<std::pair<int, int>> moves = {{-1, 1}, {0, 1}, {1, 1}};

bool can_move(int x, int y, int x_move, int y_move, int n, int m) {
    return x + x_move >= 0 && x + x_move < m && y + y_move >= 0 && y + y_move < n;
}

int main() {
    int n, m;
    std::cin >> n >> m;
    std::vector A (n, std::vector<int64_t>(m));

    for(int i = 0; i < n; ++i) {
        for (int j = 0; j < m; ++j) {
            std::cin >> A[i][j];
        }
    }

    std::vector prev_col(n, std::vector<std::pair<int, int>>(m));

    for(int y = n - 2; y >= 0; --y) {
        for (int x = 0; x < m; ++x) {

            int64_t min_prev_a = std::numeric_limits<int64_t>::max();
            std::pair<int, int> col;

            for (const auto &[x_move, y_move] : moves) {
                if (!can_move(x, y, x_move, y_move, n, m)) {
                    continue;
                }

                if (A[y+y_move][x+x_move] < min_prev_a) {
                    min_prev_a = A[y+y_move][x+x_move];
                    col = {y+y_move, x+x_move};
                }
            }

            prev_col[y][x] = {col.first, col.second};
            A[y][x] += min_prev_a;
        }
    }

    int64_t minimum = A[0][0];
```

```
    int idx = 0;
    for(int i = 1; i < m; ++i) {
        if (A[0][i] <= minimum) {
            minimum = A[0][i];
            idx = i;
        }
    }

    std::cout << minimum << "\n";

    int prev_x = idx, prev_y = 0;
    int counter = 0;
    while(counter < n) {
        std::cout << "(" << prev_y + 1 << "," << prev_x + 1 << ")";
        if (counter != n-1) {
            std::cout << " ";
        }
        int new_prev_y = prev_col[prev_y][prev_x].first, new_prev_x = prev_col[prev_y]
[prev_x].second;
        prev_y = new_prev_y, prev_x = new_prev_x;
        counter++;
    }
}
```

**Тесты**

1.
ввод:
3 3
3 1 2
7 4 5
8 6 3
вывод:
8
(1,2) (2,2) (3,3)

2.
ввод:
3 5
1 2 3 4 5
5 4 3 2 1
8 8 8 8 8
вывод:
13
(1,4) (2,5) (3,4)

3.
ввод:
5 3
1 2 3
4 5 6
6 5 4
3 2 1
1 1 1
вывод:
12
(1,1) (2,1) (3,2) (4,3) (5,2)

4.
ввод:
2 6
1 2 3 4 5 6
1 2 3 4 5 6
вывод:
2
(1,1) (2,1)

## Вывод

В результате выполнения лабораторной работы я изучил основные алгортмы, использующие идею динамического программирования, составил и отладил программу для своего варианта задания. Алгоритм продемонстрировал высокую вычислительную эффективность на тестовых данных, так как временная сложность решения составила O(n·m), где n — количество строк, m — количество столбцов матрицы.

Итоговый результат — минимальный штраф для прохождения из любой клетки верхней строки до любой клетки нижней строки — был успешно вычислен как минимальное значение в первой строке матрицы.

Работа над данной задачей позволила закрепить навыки проектирования и реализации алгоритмов динамического программирования, а также понимание принципов оптимизации.