

Московский авиационный институт  
(национальный исследовательский университет)

Институт информационных технологий и прикладной  
математики

Кафедра вычислительной математики и программирования

Лабораторная работа №7 по курсу «Дискретный анализ»

Студент: Г. С. Шубин  
Преподаватель: Н. С. Капралов  
Группа: М8О-308Б-19  
Дата:  
Оценка:  
Подпись:

Москва, 2022

# Лабораторная работа №7

**Задача:** Обход матрицы

## Вариант №5

При помощи метода динамического программирования разработать алгоритм решения задачи, определяемой своим вариантом; оценить время выполнения алгоритма и объем затрачиваемой оперативной памяти. Перед выполнением задания необходимо обосновать применимость метода динамического программирования.

Разработать программу на языке C или C++, реализующую построенный алгоритм. Формат входных и выходных данных описан в варианте задания:

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)** взимается штраф  $A_{i,j}$ . Необходимо пройти из какой-нибудь клетки верхней строки до любой клетки нижней, набрав при проходе по клеткам минимальный штраф.

## Формат входных данных

Первая строка входного файла содержит в себе пару чисел  $2 \leq n \leq 1000$  и  $2 \leq m \leq 1000$ , затем следует **n** строк из **m** целых чисел.

## Формат результата

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.

# 1 Описание

Как сказано в [2]: "оптимальная подструктура в динамическом программировании означает, что оптимальное решение подзадач меньшего размера может быть использовано для решения исходной задачи".

"В общем случае мы можем решить задачу, в которой присутствует оптимальная подструктура, проделывая следующие три шага:

1. Разбиение задачи на подзадачи меньшего размера.
2. Нахождение оптимального решения подзадач рекурсивно, проделывая такой же трёхшаговый алгоритм.
3. Использование полученного решения подзадач для конструирования решения исходной задачи.

Подзадачи решаются делением их на подзадачи ещё меньшего размера и т.д., пока не приходят к тривиальному случаю задачи, решаемой за константное время (ответ можно сказать сразу)."[2]

В данной задаче мы будем делать проход снизу вверх, последовательно считая минимальный штраф для попадания в текущую клетку массива для всех клеток на основе уже посчитанных таким образом минимальных штрафов для клеток внизу. Очевидно, что для клеток со строки  $n-1$  ничего считать не нужно.

После этого их первой строки выбирается клетка с наименьшим штрафом, а затем алгоритм проходит вниз по таблице, выбирая наименьшее число из трёх клеток снизу.

## 2 Исходный код

```
1  #include <iostream>
2  #include <vector>
3  #include <algorithm>
4
5  using namespace std;
6
7  int main() {
8
9      int n, m;
10     cin >> n >> m;
11     vector<vector<long long>> A (n, vector<long long>(m));
12
13     for(int i = 0; i < n; ++i) {
14         for (int j = 0; j < m; ++j) {
15             cin >> A[i][j];
16         }
17     }
18
19     for(int i = n - 2; i > (-1); --i) {
20         for (int j = 0; j < m; ++j) {
21             if (j == 0) {
22                 A[i][j] += min(A[i + 1][j], A[i + 1][j + 1]);
23             }
24             else if (j == m - 1) {
25                 A[i][j] += min(A[i + 1][j - 1], A[i + 1][j]);
26             }
27             else {
28                 A[i][j] += min({A[i + 1][j - 1], A[i + 1][j], A[i + 1][j + 1]});
29             }
30         }
31     }
32
33     long long minimum = A[0][0];
34     int ind = 0;
35     for(int i = 1; i < m; ++i) {
36         if (A[0][i] <= minimum) {
37             minimum = A[0][i];
38             ind = i;
39         }
40     }
41
42     cout << minimum << "\n";
43
44     int i, j = ind;
45     for(i = 0; i < n - 1; ++i) {
46         cout << "(" << i + 1 << ", " << j + 1 << " ) ";
```

```

47     if (j == 0) {
48         long long x = min(A[i + 1][j], A[i + 1][j + 1]);
49         if (x == A[i + 1][j + 1]) {
50             j++;
51         }
52     }
53     else if (j == m - 1) {
54         long long x = min(A[i + 1][j - 1], A[i + 1][j]);
55         if (x == A[i + 1][j - 1]) {
56             j--;
57         }
58     }
59     else {
60         long long x = min({A[i + 1][j - 1], A[i + 1][j], A[i + 1][j + 1]});
61         if (x == A[i + 1][j - 1]) {
62             j--;
63         }
64         else if (x == A[i + 1][j + 1]) {
65             j++;
66         }
67     }
68 }
69 cout << "(" << i + 1 << "," << j + 1 << ")\n";
70
71 return 0;
72 }

```

### 3 Консоль

```
gregory@DESKTOP-7L8SUG4:~/DA/7lab$ ./solution
3 3
3 1 2
7 4 5
8 6 3
8
(1,2) (2,2) (3,3)
gregory@DESKTOP-7L8SUG4:~/DA/7lab$ ./solution
3 2
-1 2
3 4
5 6
7
(1,1) (2,1) (3,1)
```

## 4 Тест производительности

Если решать задачу перебором, то мы получим сложность  $O(n^3)$ , тогда как наш алгоритм работает со сложностью  $O(n * m + m + n)$ . Сравним время их работы на матрицах различных размеров.

```
Array size: 5*5  
Dynamic: 7.22e-07  
Naive: 5.591e-06  
Array size: 10*10  
Dynamic: 2.314e-06  
Naive: 8.1853e-05  
Array size: 25*25  
Dynamic: 8.847e-06  
Naive: 3.16346  
Array size: 30*30  
Dynamic: 1.1181e-05  
Naive: 124.613
```

Можно заметить, что динамический алгоритм работает значительно быстрее наивного.

## 5 Выводы

Выполнив седьмую лабораторную работу по курсу «Дискретный анализ», я узнал такой подход к решению задач как динамическое программирование, который позволяет разбивать сложные задачи на более маленькие и простые, что значительно уменьшает время работы.



## Список литературы

- [1] Томас Х. Кормен, Чарльз И. Лейзерсон, Рональд Л. Ривест, Клиффорд Штайн. *Алгоритмы: построение и анализ, 2-е издание*. — Издательский дом «Вильямс», 2007. Перевод с английского: И. В. Красиков, Н. А. Орехова, В. Н. Романов. — 1296 с. (ISBN 5-8459-0857-4 (рус.))
- [2] *Динамическое программирование* — *Википедия*.  
URL: [https://ru.wikipedia.org/wiki/Динамическое\\_программирование](https://ru.wikipedia.org/wiki/Динамическое_программирование) (дата обращения: 25.06.2021).