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: Л. В. Ивенкова
Преподаватель: Н. С. Капралов
Группа: М8О-208Б-19

Дата:
Оценка:
Подпись:

Москва, 2021

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 $\mathbf{n} \times \mathbf{m}$. 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 $A_{i,j}$. 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 $\mathbf{2} \geq \mathbf{n} \geq \mathbf{1000}$ и $\mathbf{2} \geq \mathbf{m} \geq \mathbf{1000}$, 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 [1]: "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 подхадачи 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)."[1]

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 алгорит проходит вниз по теблице, выбирая наименьшее число из трёх клеток снизу.

2 Исходный код

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

using namespace std;

int main() {

    int n, m;
    cin >> n >> m;
    vector<vector<long long>> A (n, vector<long long>(m));

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

    for(int i = n - 2; i > (-1); --i) {
        for (int j = 0; j < m; ++j) {
            if (j == 0) {
                A[i][j] += min(A[i + 1][j], A[i + 1][j + 1]);
            }
            else if (j == m - 1) {
                A[i][j] += min(A[i + 1][j - 1], A[i + 1][j]);
            }
            else {
                A[i][j] += min({A[i + 1][j - 1], A[i + 1][j], A[i + 1][j + 1]});
            }
        }
    }

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

    cout << minimum << "\n";

    int i, j = ind;
    for(i = 0; i < n - 1; ++i) {
        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

```
parsifal@DESKTOP-3G70RV4:~/DA/Lab7$ g++ main.cpp -o Lab7
parsifal@DESKTOP-3G70RV4:~/DA/Lab7$ ./Lab7
3 3
3 1 2
7 4 5
8 6 3
8
(1,2) (2,2) (3,3)
parsifal@DESKTOP-3G70RV4:~/DA/Lab7$ ./Lab7
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 олгоритм работает со сложностью $O(n^2)$ ($O(n*m+m+n)$. Сравним рвемя их работы на матрице размером 100*100

```
parsifal@DESKTOP-3G70RV4:~/DA/Lab7$ time ./Lab7 <test0.txt >out.txt

real    0m0.540s
user    0m0.521s
sys     0m0.181s
```

```
parsifal@DESKTOP-3G70RV4:~/DA/Lab7$ time ./bench <test1.txt >out.txt

real    2m1.320s
user    2m1.134s
sys     0m0.031s
```

5 Выводы

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

Список литературы

[1] *Динамическое программирование — Википедия.*
URL: `https://ru.wikipedia.org/wiki/Динамическое_программирование` (дата обращения: 25.06.2021).