Московский авиационный институт (национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №7 по курсу «Дискретный анализ»

Студент: А. О. Знай
Преподаватель: С. А. Михайлова
Группа: М8О-301Б-21

Дата: 19.10.2023
Оценка:
Подпись:

Москва, 2023

Лабораторная работа №7

Задача: При помощи метода динамического программирования разработать алгоритм решения задачи, определяемой своим вариантом.

Вариант: 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 входных данных: Первая строка входного файла содержит в себе пару чисел 2 >= n <= 1000 и 2 >= m <= 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.

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 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)."[1] В данной задаче мы будем проходить матрицу, начиная со второй строки, где для каждого элемента будем находить минимальный элемент из трех соседних элементов из строки выше. Таким образом, для расчетов на следующей строки мы будем использовать уже посчитанные оптимальные значения. Пройдя всю матрицу, в последней строке найдем минимальное число - это ответ. Чтобы восстановить путь к ответу, будем подниматься вверх и выбирать из трех соседей сверху минимальный - это тот элемент, из которого мы пришли в текущий.

[1]

Код

```cpp
#include <string>
#include <iostream>
#include <vector>
#include <stack>
#include <memory>
#include <chrono>
using namespace std;


int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(nullptr);
    cout.tie(nullptr);

    int rows, cols;

    cin >> rows;
    cin >> cols;

    vector<vector<long long> > matrix(rows, vector<long long>(cols));

    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            cin >> matrix[i][j];
        }
    }
    vector<int> steps{-1, 0, 1};
    for (int i = 1; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            long long minNeighb = LLONG_MAX;
            for (int k = 0; k < 3; k++) {
                int neighbInd = j + steps[k];
                if (neighbInd >= 0 && neighbInd < cols && matrix[i-1][neighbInd] <
                    minNeighb) {
                    minNeighb = matrix[i-1][neighbInd];
                }
            }
            matrix[i][j] += minNeighb;
        }
    }

    long long minPenalty = matrix[rows-1][0];
    vector<pair<int, int> > coords;
    pair<int, int> minPenaltyCoords(rows, 1);
    for (int i = 1; i < cols; i++) {
        if (matrix[rows-1][i] < minPenalty) {
            minPenalty = matrix[rows-1][i];
            minPenaltyCoords = make_pair(rows, i+1);
```

```
48          }
49      }
50      coords.push_back(minPenaltyCoords);
51
52      pair<int, int> currentCoords = make_pair(minPenaltyCoords.first, minPenaltyCoords.
            second);
53      for (int i = rows - 2; i >= 0; i--) {
54          long long minNeighb = LLONG_MAX;
55          pair<int, int> minNeighbCoords;
56          for (int k = 0; k < 3; k++) {
57              int neighbInd = currentCoords.second - 1 + steps[k];
58              if (neighbInd >= 0 && neighbInd < cols && matrix[i][neighbInd] < minNeighb)
                     {
59                  minNeighb = matrix[i][neighbInd];
60                  minNeighbCoords = make_pair(i+1, neighbInd+1);
61              }
62          }
63          currentCoords = minNeighbCoords;
64          coords.push_back(minNeighbCoords);
65      }
66
67      cout << minPenalty << '\n';
68      for (int i = coords.size() - 1; i >= 0; i--) {
69          if (i != 0) {
70              cout << '(' << coords[i].first << ',' << coords[i].second << ')' << ' ';
71          }
72          else {
73              cout << '(' << coords[i].first << ',' << coords[i].second << ')';
74          }
75      }
76  }
```

2 Консоль

```
artemznaj@MacBook-Air-Artem lab7 % ./a.out
3 3
3 1 2
7 4 5
8 6 3
8
(1,2) (2,2) (3,3)
artemznaj@MacBook-Air-Artem lab7 % ./a.out
10 9
56 68 16 57 94 17 38 31 49
87 18 66 7 33 72 79 38 78
81 21 41 19 72 34 30 68 49
60 21 44 2 89 58 44 72 87
100 30 74 66 39 97 80 85 97
16 71 55 59 33 78 3 25 48
61 80 10 6 100 12 81 44 5
27 64 53 22 56 38 20 34 58
52 39 31 89 50 3 52 44 74
78 55 74 45 71 56 92 78 94
207
(1,3) (2,4) (3,4) (4,4) (5,5) (6,5) (7,6) (8,7) (9,6) (10,6)
```

3 Тест производительности

Сравним время их работы на матрице размером 100*100. Если решать задачу перебором, то мы получим сложность O(n^3), тогда как наш алгоритм работает со сложностью O(n^2).

```
artemznaj@MacBook-Air-Artem lab7 % time ./Lab7 <test0.txt >out.txt
real    0m0.540s
user    0m0.521s
sys     0m0.181s
artemznaj@MacBook-Air-Artem lab7 % time ./bench <test1.txt >out.txt
real    2m1.320s
user    2m1.134s
sys     0m0.031s
```

4 Выводы

Выполнив седьмую лабораторную работу по курсу «Дискретный анализ», я узнал про такой подход к решению задач как динамическое программирование, который позволяет разбивать сложные задачи на более маленькие и простые, этот подход не только ускоряет решение, но и улучшает понимание самой задачи.

Список литературы

[1] Динамическое программирование https://ru.wikipedia.org/wiki/

[2] Лекции Н.К.Макарова, Московский авиационный институт.