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: Е. С. Пищик
Преподаватель: А. А. Кухтичев
Группа: М8О-206Б

Дата:
Оценка:
Подпись:

Москва, 2021

Лабораторная работа №7

Задача: 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) взымается штраф Aij. 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 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

Требуется написать алгоритм динамического программирования для решения поставленной задачи. Метод динамического программирования заключается в разделении задач на группу независимых подзадач, из которых и формируется общее решение.

Алгоритм. Давайте идти снизу-вверх по матрице и на каждой итерации (i, j) обновлять ответ для данной клетки в матрице-ответе, проверяя возможный путь из клеток (i + 1, j 1), (i + 1, j), (i + 1, j + 1) и обновляя при необходимости ответ. Также в задаче требуется восстановление пути, для реализации данного функционала будем при каждом обновлении ответа для каждой клетки (i, j) запоминать, откуда мы пришли: снизу, снизу-слева или снизу-справа. Для получения оптимального пути из верхней строки матрицы в нижнюю просто пройдемся по верхней строке в матрице-ответе и выберем из всех значений минимум — это и будет оптимальной ценой за путь. Для восстановления ответа будем идти от выбранной клетки до нижней строки, следуя указаниям пути откуда мы пришли. Итого для заполнения матрицы-ответа нам потребуется O(3NM) = O(NM) времени, исходя из алгоритма, для выбора стоимости пути нам потребуется O(M) времени, а для восстановления пути нам нужно O(N) времени. Общее время работы равно O(NM), что в разы быстрее алгоритма полного перебора, который бы работал за время O(NM^3).

2 Исходный код

main.cpp

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

std::istream& operator>>(std::istream& is, std::vector<std::vector<long long>>& M)
{
    long long n = M.size();
    long long m = M[0].size();
    for(long long i = 0; i < n; ++i)
    {
        for(long long j = 0; j < m; ++j)
        {
            is >> M[i][j];
        }
    }
    return is;
}

std::ostream& operator<<(std::ostream& os, std::vector<std::vector<long long>> const&
    M)
{
    long long n = M.size();
    long long m = M[0].size();
    os << "\n";
    for(long long i = 0; i < n; ++i)
    {
        for(long long j = 0; j < m; ++j)
        {
            os << M[i][j] << " ";
        }
        os << "\n";
    }
    return os;
}

std::ostream& operator<<(std::ostream& os, std::vector<std::vector<std::vector<long
    long>>> const& M)
{
    long long n = M.size();
    long long m = M[0].size();
    os << "\n";
    for(long long i = 0; i < n; ++i)
    {
        for(long long j = 0; j < m; ++j)
        {
            os << "Path: " << M[i][j][0] << " i: " << M[i][j][1] << " j: " << M[i][j
```

```
                ][2] << " ";
46          }
47          os << "\n";
48      }
49      return os;
50  }
51
52  void GetMinInitLst(std::initializer_list<long long>& ilist, std::vector<std::vector<
        std::vector<long long>>>& res, std::vector<std::vector<long long>> const& M, long
        long const& i, long long const& j, bool is_left)
53  {
54      std::initializer_list<long long>::iterator min_el = std::min_element(ilist.begin(),
            ilist.end());
55      res[i][j][0] = *min_el + M[i][j];
56      long long ind = min_el - ilist.begin();
57      if(is_left) { ++ind; }
58      res[i][j][1] = i+2;
59      res[i][j][2] = j+ind;
60  }
61
62  void CalcResThree(std::vector<std::vector<std::vector<long long>>>& res, std::vector<
        std::vector<long long>> const& M, long long const& i, long long const& j)
63  {
64      std::initializer_list<long long> ilist = {res[i+1][j-1][0], res[i+1][j][0], res[i
            +1][j+1][0]};
65      GetMinInitLst(ilist, res, M, i, j, false);
66  }
67
68  void CalcResTwoLeft(std::vector<std::vector<std::vector<long long>>>& res, std::vector
        <std::vector<long long>> const& M, long long const& i, long long const& j)
69  {
70      std::initializer_list<long long> ilist = {res[i+1][j][0], res[i+1][j+1][0]};
71      GetMinInitLst(ilist, res, M, i, j, true);
72  }
73
74  void CalcResTwoRight(std::vector<std::vector<std::vector<long long>>>& res, std::
        vector<std::vector<long long>> const& M, long long const& i, long long const& j)
75  {
76      std::initializer_list<long long> ilist = {res[i+1][j-1][0], res[i+1][j][0]};
77      GetMinInitLst(ilist, res, M, i, j, false);
78  }
79
80  void CalcRes(char const& flag, std::vector<std::vector<std::vector<long long>>>& res,
        std::vector<std::vector<long long>> const& M, long long const& i, long long const&
         j)
81  {
82      if(flag == 't') { CalcResThree(res, M, i, j); }
83      else if(flag == 'l') { CalcResTwoLeft(res, M, i, j); }
84      else if(flag == 'r') { CalcResTwoRight(res, M, i, j); }
```

```
85  }
86
87  std::vector<long long> GetMinVct(std::vector<std::vector<std::vector<long long>>>
        const& res, long long const& n, long long const& m)
88  {
89      std::vector<long long> min_vct;
90      long long cur_min_ind_i = 1;
91      long long cur_min_ind_j = 1;
92      long long min = res[0][0][0];
93      long long min_ind_j = 1;
94      long long min_ind_i = 1;
95      for(long long i = 1; i < m; ++i)
96      {
97          if(res[0][i][0] < min)
98          {
99              min_ind_j = i+1;
100             min = res[0][i][0];
101         }
102     }
103     min_vct.push_back(min);
104     for(long long i = 0; i < n; ++i)
105     {
106         min_vct.push_back(min_ind_i);
107         min_vct.push_back(min_ind_j);
108         cur_min_ind_i = res[min_ind_i-1][min_ind_j-1][1];
109         cur_min_ind_j = res[min_ind_i-1][min_ind_j-1][2];
110         min_ind_i = cur_min_ind_i;
111         min_ind_j = cur_min_ind_j;
112     }
113     return min_vct;
114 }
115
116 std::vector<long long> GetRes(std::vector<std::vector<long long>> const& M, long long
        const& n, long long const& m)
117 {
118     char flag = '0';
119     std::vector<std::vector<std::vector<long long>>> res(n, std::vector<std::vector<
            long long>>(m, std::vector<long long>(3, 0)));
120     for(long long j = 0; j < m; ++j) { res[n-1][j][0] = M[n-1][j]; }
121     for(long long i = n - 2; i >= 0; --i)
122     {
123         for(long long j = 0; j < m; ++j)
124         {
125             if(j == 0) { flag = 'l'; }
126             else if(j == m-1) {flag = 'r'; }
127             else { flag = 't'; }
128             CalcRes(flag, res, M, i, j);
129         }
130     }
```

```
131     return GetMinVct(res, n, m);
132 }
133 
134 int main()
135 {
136     long long n = 0;
137     long long m = 0;
138     std::cin >> n >> m;
139     std::vector<std::vector<long long>> M(n, std::vector<long long>(m, 0));
140     std::cin >> M;
141     std::vector<long long> result = GetRes(M, n, m);
142     std::cout << result[0] << "\n";
143     for(long long i = 1; i <= result.size() - 1; i += 2)
144     {
145         std::cout << "(" << result[i] << "," << result[i+1] << ") ";
146     }
147     std::cout << "\n";
148     return 0;
149 }
```

3 Консоль

```
pe4eniks@pe4eniks-HP-Laptop-14-dk0xxx:~/solution$ cat test.txt
3 3
3 1 2
7 4 5
8 6 3
pe4eniks@pe4eniks-HP-Laptop-14-dk0xxx:~/solution$ ./solution<test.txt
8
(1,2) (2,2) (3,3)
```

4 Тест производительности

Тест состоит из сравнения алгоритма полного перебора и алгоритма ДП на матрице размера 50*50.

```
pe4eniks@pe4eniks-HP-Laptop-14-dk0xxx:~/solution$ ./benchmark
Time for DP: 0us
Time for fullfind algo: 121783us
```

Как можно увидеть, время работы на полном переборе намного больше, чем на алгоритме ДП даже на матрице размера 50*50. При матрице большего размера результат ухудшается в кубической прогрессии.

5 Выводы

Выполнив седьмую лабораторную работу по курсу «Дискретный анализ», я познакомился с алгоритмом динамического программирования, который позволяет разбивать сложные задачи на более простые и значительно быстрее работает алгоритма обычного перебора, что является основным преимуществом данного алгоритма.

Список литературы

[1] Динамическое программирование

[2] Динамическое программирование habr