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: В. А. Слесарчук
Преподаватель: Н. К. Макаро
Группа: М8О-301Б
Дата:
Оценка:
Подпись:

Москва, 2025

Лабораторная работа №7

Вариант: 5

Задача:

Требуется разработать программу использующую динамическое программирование, осуществляющую поиск кратчайшего пути из какой-нибудь клетки верхней строки, до любой клетки нижней строки матрицы.

Формат ввода:

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 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

Для решения задачи был использован алгоритм, основанный на динамическом программировании. Алгоритм находит минимальный путь стоимости в матрице, где разрешены переходы только вниз и по диагонали в соседние ячейки.

2 Исходный код

Основная идея:

Алгоритм использует двумерный массив для хранения минимальной стоимости достижения каждой ячейки матрицы. Для каждой ячейки (i, j) вычисляется минимальная стоимость пути из одной из трех возможных предыдущих ячеек: $(i-1, j-1)$, $(i-1, j)$ или $(i-1, j+1)$.

Структуры данных:

- **struct p_node** - структура для хранения информации о пути:
  - `_from` - указатель на предыдущую вершину
  - `cost` - минимальная стоимость достижения данной ячейки
  - `pos` - позиция в строке

- **matrix** - исходная матрица стоимостей
- **path** - матрица структур p_node для восстановления пути

1 Этапы алгоритма

1. **Инициализация** (`path_init`): Устанавливаются начальные стоимости для первой строки матрицы.
2. **Прямой проход** (`forward`): Для каждой ячейки обновляются стоимости трех соседних ячеек в следующей строке.
3. **Восстановление пути** (`backtrack`): Находится ячейка с минимальной стоимостью в последней строке и по ссылкам восстанавливается оптимальный путь до первой строки.

```
#include <iostream>
#include <unordered_map>
#include <string>
#include <vector>
#include <limits.h>
#include <random>
#include <stack>
using namespace std;

struct p_node
{
    p_node* _from = nullptr;
    long long cost = LLONG_MAX;
    int pos = -1;
};


std::ostream& operator<<(std::ostream& os, const p_node &p) {
    os << p.cost;
    return os;
}

std::ostream& operator<<(std::ostream& os, const pair<long long, long long> &p) {
    os << "(" << p.first << "," << p.second<< ")";
    return os;
}

template<typename T>
void matrix_print(vector<vector<T>> &matrix){
    for (size_t i = 0; i < matrix.size(); i++)
    {
```

```
        for (size_t j = 0; j < matrix[0].size(); j++)
        {
            cout << matrix[i][j] << " ";
        }
        cout << endl;
    }
}


void path_init(vector<vector<long long>> &matrix, vector<vector<p_node>> &path){
    for (int j = 0; j < matrix[0].size(); j++)
    {
        p_node first_row_node = {nullptr, matrix[0][j], j};
        path[0][j] = first_row_node;
    }

    for (size_t i = 1; i < matrix.size(); i++)
    {
        for (int j = 0; j < matrix[0].size(); j++)
        {
            p_node empty_node = {nullptr, LLONG_MAX, j};
            path[i][j] = empty_node;
        }
    }
}

void path_update(p_node &from_node, p_node &to_node, long long plus){
    long long new_cost = from_node.cost + plus;
    if (new_cost < to_node.cost){
        to_node.cost = new_cost;
        to_node._from = &from_node;
    }
}

void forward(vector<vector<long long>> &matrix, vector<vector<p_node>> &path){
    int n = matrix.size();
    int m = matrix[0].size();
    for (size_t i = 0; i < n - 1; i++)
    {
        for (size_t j = 0; j < m; j++)
        {
            if (j == 0){
                path_update(path[i][0], path[i+1][0], matrix[i+1][0]);
                path_update(path[i][0], path[i+1][1], matrix[i+1][1]);
            } else if (j == (m - 1)){
                path_update(path[i][m - 1], path[i+1][m - 2], matrix[i+1][m - 2]);
                path_update(path[i][m - 1], path[i+1][m - 1], matrix[i+1][m - 1]);
            } else {
                path_update(path[i][j], path[i+1][j - 1], matrix[i+1][j - 1]);
```

```
81                path_update(path[i][j], path[i+1][j], matrix[i+1][j]);
82                path_update(path[i][j], path[i+1][j + 1], matrix[i+1][j + 1]);
83            }
84        }
85    }
86 }
87
88 stack<pair<long long, long long>> backtrack(vector<vector<long long>> &matrix, vector<
      vector<p_node>> &path){
89
90    p_node min_cost_node;
91
92    for (size_t j = 0; j < path[0].size(); j++)
93    {
94        p_node check = path[path.size() - 1][j];
95        if (check.cost < min_cost_node.cost){
96            min_cost_node = check;
97        }
98    }
99
100   cout << min_cost_node.cost << endl;
101
102   p_node* current = &min_cost_node;
103   int row = path.size();
104   stack<pair<long long, long long>> result;
105
106   while(row != 0){
107       pair<long long, long long> point;
108       point.first = row;
109       point.second = current->pos + 1;
110       result.push(point);
111       row -= 1;
112       current = current->_from;
113   }
114
115   return result;
116 }
117
118
119 int main(){
120    int n, m;
121    cin >> n >> m;
122    vector<vector<long long>> matrix(n, vector<long long>(m, 0));
123    vector<vector<p_node>> path(n, vector<p_node>(m));
124    stack<pair<long long, long long>> result;
125    for (size_t i = 0; i < n; i++)
126    {
127        for (size_t j = 0; j < m; j++)
128        {
```

```
129             long long cost;
130             cin >> cost;
131             matrix[i][j] = cost;
132         }
133     }
134 
135     path_init(matrix, path);
136     forward(matrix, path);
137     result = backtrack(matrix, path);
138 
139     while (!result.empty()) {
140         cout << result.top() << " ";
141         result.pop();
142     }
143     cout << endl;
144 }
```

3 Консоль

Пример компиляции и демонстрация работы программы:

```
yugo@yugo-pc:~/Desktop/Labs/DALabs/report_pattern$ g++ ../lab7/main.cpp -o
main
yugo@yugo-pc:~/Desktop/Labs/DALabs/report_pattern$ ./main
3 3
1 2 3
4 5 6
7 8 9
12
(1,1) (2,1) (3,1)
```

4 Тест производительности

Методика тестирования заключалась в сравнении производительности реализованного алгоритма использующего динамическое программирование (ДП) с алгоритмом использующем перебор. Алгоритм ДП имеет квадратичную сложность O(|n| · |m|), в то время как перебор решает задачу за $O(m \times 3^n)$

```
===>  MATRIX (13,80)   <===
DYNAMIC: 8.4545e-05
FORCE: 0.66722
===>  MATRIX (14,160)   <===
```

```
DYNAMIC: 0.000152359
FORCE: 10.7074
===>  MATRIX (15,320)   <===
DYNAMIC: 0.00123632
FORCE: 79.6216

===>  MATRIX (100,100)   <===
DYNAMIC: 0.00149533
===>  MATRIX (1000,1000)   <===
DYNAMIC: 0.0755794
===>  MATRIX (10000,10000)   <===
DYNAMIC: 6.97176
```

Результаты тестов полностью подтверждают теоретические оценки сложности. Как видно решение использующее динамическое программирование имеет квадратичную сложность $O(|n| \cdot |m|)$.

5 Выводы

В ходе выполнения лабораторной работы была успешно реализована программа для нахождения минимального пути в матрице с использованием динамического программирования.

- Освоены принципы динамического программирования для решения задач поиска оптимального пути в графах и матрицах, включая разбиение задачи на подзадачи и комбинирование их решений.
- Реализован эффективный алгоритм поиска минимального пути стоимости в матрице с возможностью переходов вниз и по диагонали в соседние ячейки.

Разработанное решение демонстрирует эффективность подхода динамического программирования для задач оптимизации пути и может быть применено для решения широкого круга практических задач в областях анализа данных, робототехники, игровых алгоритмов и оптимизационных систем.

Список литературы

[1] Ukkonen E. On-line construction of suffix trees // Algorithmica. — 1995. — Vol. 14, no. 3. — P. 249–260.

[2] Gusfield D. Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology. Cambridge University Press, 1997.

[3] Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К. Алгоритмы: построение и анализ, 3-е изд. — М.: «Вильямс», 2013.