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: П.А. Земсков

Преподаватель: Н.К. Макаров

Группа: М8О-301Б

Дата: _____

Оценка: _____

Подпись: _____

Москва, 2025

Содержание

1 Лабораторная работа №7

Задача

Требуется разработать программу для поиска образцов в заранее известном тексте с использованием суффиксного массива.
Вариант алгоритма: 5 Обход матрицы

2 Описание

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 A i,j. Необходимо пройти из какой-нибудь клетки верхней строки до любой клетки нижней, набрав при проходе по клеткам минимальный штраф.

3 Исходный код

Общая идея: Мы используем динамическое программирование: для каждой ячейки матрицы вычисляем минимальный штраф пути до неё из верхней строки, рассматривая переходы только из трёх соседних клеток сверху (слева, прямо, справа). При этом храним родителя каждой клетки для восстановления пути. После заполнения dp-таблицы минимальный штраф находится в нижней строке, а сам путь восстанавливается, двигаясь по родителям от клетки с минимальным значением до верхней строки.

Листинг 1: Исходный код

```
#include <iostream>
#include <vector>
using namespace std;
using ll = long long;
const ll INF = 1e18;

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

    int n, m;
    cin >> n >> m;

    vector<vector<ll>> a(n, vector<ll>(m));
    for (int i = 0; i < n; ++i)
    {
        for (int j = 0; j < m; ++j)
        {
            cin >> a[i][j];
        }
    }

    vector<vector<ll>> dp(n, vector<ll>(m, INF));
    vector<vector<int>> parent(n, vector<int>(m, -1));

    for (int j = 0; j < m; ++j)
        dp[0][j] = a[0][j];

    for (int i = 1; i < n; ++i)
    {
        for (int j = 0; j < m; ++j)
        {
            for (int delta_j = -1; delta_j <= 1; ++delta_j)
            {
                int prev_j = j + delta_j;
                if (prev_j < 0 || prev_j >= m)
                    continue;
                ll cur_cost = dp[i - 1][prev_j] + a[i][j];
                if (cur_cost < dp[i][j])
                {
                    dp[i][j] = cur_cost;
                    parent[i][j] = prev_j;
                }
            }
        }
    }

    ll best = INF;
    int col = -1;
```

```
51      for (int j = 0; j < m; ++j)
52      {
53          if (dp[n - 1][j] < best)
54          {
55              best = dp[n - 1][j];
56              col = j;
57          }
58      }
59
60      vector<int> path(n);
61      for (int i = n - 1; i >= 0; --i)
62      {
63          path[i] = col;
64          col = parent[i][col];
65      }
66
67      cout << best << "\n";
68      for (int i = 0; i < n; ++i)
69      {
70          cout << "(" << i + 1 << "," << path[i] + 1 << ")";
71          if (i + 1 != n)
72              cout << "␣";
73      }
74      cout << "\n";
75
76      return 0;
77  }
```

4 Консоль

Пример компиляции и демонстрация работы программы:

```
C:\Users\jocke\Documents\diskran\lab7> g++ --std=c++20 main.cpp -o main
C:\Users\jocke\Documents\diskran\lab7> ./main
3 3
3 1 2
7 4 5
8 6 3
8
(1,2) (2,2) (3,3)
```

5 Тест производительности

Для матриц больших размеров (например, 1000×1000) алгоритм dp срабатывает за доли секунды, так как имеет сложность O(n·m). В отличие от полного перебора всех возможных маршрутов, число которых растёт экспоненциально с высотой матрицы, полный перебор становится невозможным даже для матрицы 10×10, тогда как dp остаётся эффективным и масштабируемым.

```
C:\Users\jocke\Documents\diskran> g++ -std=c++20 lab7.cpp -o main
C:\Users\jocke\Documents\diskran> g++ -std=c++20 benchmark.cpp -o benchmark
C:\Users\jocke\Documents\diskran> .\benchmark
dp time: 260 ms
loop time: 12324 ms
```

Результат:

Для матриц больших размеров (например, 1000×1000) алгоритм dp срабатывает за доли секунды, так как имеет сложность O(n·m).

6 Выводы

Алгоритм корректно находит минимальный штраф и маршрут для любой заданной матрицы, обеспечивая точное решение за линейное время по количеству ячеек. Это демонстрирует эффективность динамического программирования для задач с ограниченными переходами между состояниями и подчёркивает неприменимость полного перебора для больших размеров входных данных.

7 Список литературы

1. Кормен Т. Х., Лейзерсон Ч. И., Ривест Р. Л., Штайн К. *Алгоритмы: построение и анализ. 3-е изд.* – М.: Издательский дом «Вильямс», 2010.

2. Knuth D. E. *The Art of Computer Programming. Vol. 3: Sorting and Searching. 2nd ed.* – Addison-Wesley, 1998.

3. *Динамическое программирование — Алгоритмика.* `https://ru.algorithmica.org/cs/general-dynamic/`.