МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»
РТУ МИРЭА

Институт Информационных технологий

Кафедра Математического обеспечения и стандартизации информационных технологий

ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ № 7
по дисциплине
«Структуры и алгоритмы обработки данных»

Тема: «Алгоритмические стратегии или методы разработки алгоритмов. Перебор и методы его сокращения.»

Выполнил студент группы ИКБО-11-22 Бондаренко Е.А.

Принял преподаватель Скворцова Л. А.

Самостоятельная работа выполнена «__»_____202__г. *(подпись студента)*

«Зачтено» «__»_____202__г. *(подпись руководителя)*

Москва 2023

Содержание

Цель работы

Получить навыки применения методов, позволяющих сократить число переборов в задачах, которые могут быть решены только методом перебора всех возможных вариантов решения.

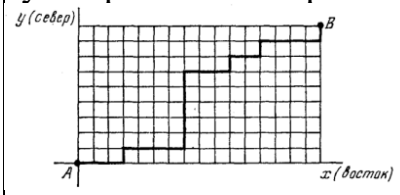
Задача №1

Постановка задачи

Разработка и программная реализация задач с применение метода сокращения числа переборов.

1. Разработать алгоритм решения задачи с применением метода, указанного в варианте и реализовать программу.
2. Оформить отчет, включив в него:
    a. Условие задачи варианта и требования к выполнению задания.
    b. Описание метода, предлагаемого к реализации в варианте.
    c. Оценку количества переборов при решении задачи стратегией «в лоб» - грубой силы.
    d. Привести анализ снижения числа переборов при применении метода

№	Задача	Метод
7	Нам нужно соорудить путь, соединяющий Рис. 13.1. два пункта A и B, из которых второй лежит к северо-востоку от первого. Для простоты допустим, что прокладка пути состоит из ряда шагов, и на каждом шаге мы можем двигаться либо строго на восток, либо строго на север; любой путь из A в B представляет собой ступенчатую ломаную линию, отрезки которой параллельны одной из координатных осей (рис. 13.1). Затраты на сооружение каждого из таких	Динамическое программирование

	отрезков известны. Требуется проложить такой путь из A в B, при котором суммарные затраты минимальны. 	

Метод ветвей и границ – это общий алгоритмический метод для нахождения оптимальных решений различных задач оптимизации, особенно дискретной и комбинаторной оптимизации. Метод является развитием

метода полного перебора, в отличие от последнего — с отсевом подмножеств допустимых решений, заведомо не содержащих оптимальных решений.

Решение

Упражнение №1

Функция для ручного ввода содержимого матрицы

```
void fillMatrixFromInput(vector<vector<size_t>>& matrix)
{
    size_t rows, cols;

    cout << "Enter the number of rows: ";
    cin >> rows;

    cout << "Enter the number of columns: ";
    cin >> cols;

    matrix.resize(rows, vector<size_t>(cols, 0));

    cout << "Enter the elements of the matrix:" << endl;
    for (size_t i = 0; i < rows; ++i) {
        for (size_t j = 0; j < cols; ++j) {
            cout << "Element [" << i << "][" << j << "]: ";
            cin >> matrix[i][j];
        }
    }
}
```

Рисунок 1. Решение задачи 1

Упражнение №2

Разберем функцию нахождения минимального пути **findMinCost.**

Эта функция решает задачу нахождения минимальной стоимости пути в матрице. Давайте рассмотрим шаги функции подробно:

1. **Инициализация базовых значений:**
    - dp[0][0] = cost[0][0]; - Инициализирует значение в верхнем левом углу dp как стоимость начальной ячейки матрицы.
    - path[0][0] = '-'; - Инициализирует соответствующий элемент массива path как '-', что означает, что из начальной ячейки нет движения.
2. **Заполнение первой строки и первого столбца:**

- Используя цикл, вычисляет и заполняет значения в первой строке и первом столбце массивов dp и path. Значения в первой строке dp представляют собой кумулятивную стоимость движения слева направо, а в первом столбце - сверху вниз.

3. **Заполнение dp массива и path:**
   - Использует два вложенных цикла для заполнения оставшихся элементов массивов dp и path. Выбирает минимальную стоимость пути из верхней ячейки (dp[i-1][j]) и из левой ячейки (dp[i][j-1]), добавляет стоимость текущей ячейки (cost[i][j]) и записывает результат в текущую ячейку dp. При этом также записывает направление движения в массив path ('U' - вверх, 'R' - направо).
4. **Восстановление оптимального пути:**
   - Начиная с нижнего правого угла (точки B), восстанавливает оптимальный путь, перемещаясь вверх ('U') или направо ('R') в зависимости от записанных значений в массиве path. Этот путь выводится на экран.
5. **Вывод минимальных затрат для достижения точки B:**
   - Возвращает значение в нижнем правом углу массива dp, которое представляет минимальные затраты для достижения конечной точки B.

```cpp
size_t findMinCost(vector<vector<size_t>>& cost, size_t m, size_t n, vector<vector<size_t>>& dp, vector<vector<char>>& path)
{
    // Инициализация базовых значений
    dp[0][0] = cost[0][0];
    path[0][0] = '-';
    // Заполнение первой строки и первого столбца
    for (size_t i = 1; i < m; ++i) {
        dp[i][0] = dp[i - 1][0] + cost[i][0];
        path[i][0] = 'U'; // Движение вверх
    }
    for (size_t j = 1; j < n; ++j) {
        dp[0][j] = dp[0][j - 1] + cost[0][j];
        path[0][j] = 'R'; // Движение направо
    }

    // Заполнение dp массива и path
    for (size_t i = 1; i < m; ++i) {
        for (size_t j = 1; j < n; ++j) {
            if (dp[i - 1][j] < dp[i][j - 1]) {
                dp[i][j] = cost[i][j] + dp[i - 1][j];
                path[i][j] = 'U'; // Движение вверх
            } else {
                dp[i][j] = cost[i][j] + dp[i][j - 1];
                path[i][j] = 'R'; // Движение направо
            }
        }
    }
    // Восстановление оптимального пути
    size_t i = m - 1, j = n - 1;
    cout << "Optimal Path: ";
    while (i > 0 || j > 0) {
        cout << path[i][j] << " ";
        if (path[i][j] == 'U') {
            --i;
        } else {
            --j;
        }
    }
    cout << endl;
    // Вывод минимальных затрат для достижения точки B
    return dp[m - 1][n - 1];
}
```

Рисунок 2. Решение задачи 2

Функция main

```cpp
int main()
{
    vector<vector<size_t>> cost;

    // Заполнение матрицы с клавиатуры
    fillMatrixFromInput(cost);

    // Вывод содержимого матрицы
    cout << "Matrix contents:" << endl;
    for (size_t i = 0; i < cost.size(); ++i) {
        for (size_t j = 0; j < cost[i].size(); ++j) {
            cout << cost[i][j] << "\t";
        }
        cout << endl;
    }

    // Размеры массива
    size_t m = cost.size();
    size_t n = cost[0].size();

    // Инициализация dp и path массивов
    vector<vector<size_t>> dp(m + 1, vector<size_t>(n + 1, 0));
    vector<vector<char>> path(m + 1, vector<char>(n + 1, 0));

    // Нахождение минимальных затрат и оптимального пути
    size_t minCost = findMinCost(cost, m, n, dp, path);

    // Вывод результата
    cout << "Minimal cost of the path: " << minCost << endl;

    return 0;
}
```

Рисунок 3. Главная функция программы

Тестирование

```
Enter the number of rows: 3
Enter the number of columns: 3
Enter the elements of the matrix:
Element [0][0]: 4
Element [0][1]: 3
Element [0][2]: 6
Element [1][0]: 7
Element [1][1]: 1
Element [1][2]: 2
Element [2][0]: 6
Element [2][1]: 7
Element [2][2]: 3
Matrix contents:
4       3       6
7       1       2
6       7       3
Optimal Path: U R U R
Minimal cost of the path: 13
```

Рисунок 3. Результат выполнения задач

Вывод:

Получил навыки применения методов, позволяющих сократить число переборов в задачах, которые могут быть решены только методом перебора всех возможных вариантов решения.