МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное

учреждение высшего образования

«ЧЕРЕПОВЕЦКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

|  |
| --- |
| Институт информационных технологий |
| Кафедра математического и программного обеспечения |
| Структуры и алгоритмы обработки данных |
|  |

ЛАБОРАТОРНАЯ РАБОТА №4

«Динамическое программирование»

|  |  |
| --- | --- |
| Исполнитель: | студент  группы 1ПИб-02-3оп-23  Богданов  Ренат Алексеевич |
| Руководитель: | Журавлёва Юлия Михайловна |
| Оценка: |  |
| Подпись: |  |

2024 год

Задача

Составить программы, решающие следующие задачи. Входные данные находятся в текстовом файле input.txt. В первой строке файла указаны через пробел размеры массива, каждый из которых не превосходит 20. В последующих строках перечислены элементы массива - целые числа. Результатом работы программы должны быть оптимальное значение целевой функции и путь, при котором оно достигается.

Для указания пути вывести массив, отметив элементы пути либо другим цветом, либо некоторым другим образом.

Дан двумерный числовой массив размером N1xN2. Найти такой путь от верхней строки массива к нижней, чтобы сумма чисел по данному пути была минимальной. Из каждой клетки массива допустимо двигаться влево, вправо, вниз, влево-вниз или вправо-вниз.

Описание работы программы

1. **Считывание данных**

Программа считывает данные из файла "input.txt", который содержит:

* Размер матрицы (количество строк и столбцов)
* Сами значения, которые хранятся в матрице

2. **Поиск кратчайшего пути:**

Программа реализует алгоритм Дейкстры для поиска кратчайшего пути от любой ячейки верхнего ряда к любой ячейке нижнего ряда.

Алгоритм Дейкстры работает так:

* Он начинает с начальной ячейки (каждая из верхнего ряда) и постепенно просматривает все соседние ячейки, выбирая ту, которая имеет минимальную стоимость прохода.
* Затем он повторяет этот процесс для выбранной ячейки, двигаясь к цели (нижний ряд) до тех пор, пока не будет найдена кратчайшая траектория.

3. **Визуализация пути**

Программа визуализирует найденный путь, выделяя его ячейки зелёным цветом в консоли.

Код программы

#include <iostream>

#include <fstream>

#include <vector>

#include <limits>

using namespace std;

// Структура для представления вершины графа

struct Node {

int row, col;

int cost;

Node(int r, int c, int co) : row(r), col(c), cost(co) {}

};

// Функция для поиска кратчайшего пути с помощью алгоритма Дейкстры

pair<int, vector<Node>> dijkstra(const vector<vector<int>>& matrix, int startRow, int startCol) {

int rows = matrix.size();

int cols = matrix[0].size();

// Инициализация расстояний до всех вершин

vector<vector<int>> distances(rows, vector<int>(cols, numeric\_limits<int>::max()));

distances[startRow][startCol] = matrix[startRow][startCol];

// Инициализация множества необработанных вершин

vector<vector<bool>> unvisited(rows, vector<bool>(cols, true));

// Инициализация предшественников

vector<vector<Node>> predecessors(rows, vector<Node>(cols, Node(-1, -1, -1)));

// Пока множество необработанных вершин не пусто

while (true) {

// Найти вершину с минимальным расстоянием

int minDist = numeric\_limits<int>::max();

Node current(-1, -1, -1);

for (int i = 0; i < rows; ++i) {

for (int j = 0; j < cols; ++j) {

if (unvisited[i][j] && distances[i][j] < minDist) {

minDist = distances[i][j];

current = Node(i, j, distances[i][j]);

}

}

}

// Если все вершины посещены, выходим

if (current.row == -1) {

break;

}

// Помечаем текущую вершину как посещённую

unvisited[current.row][current.col] = false;

// Проверка соседних вершин (вниз, влево, вправо, влево-вниз, вправо-вниз)

int directions[5][2] = {{1, 0}, {0, -1}, {0, 1}, {1, -1}, {1, 1}};

for (auto& dir : directions) {

int nextRow = current.row + dir[0];

int nextCol = current.col + dir[1];

// Проверка, находится ли соседняя вершина в границах матрицы

if (nextRow >= 0 && nextRow < rows && nextCol >= 0 && nextCol < cols) {

int newCost = current.cost + matrix[nextRow][nextCol];

// Обновление расстояния и предшественника, если найден более короткий путь

if (newCost < distances[nextRow][nextCol]) {

distances[nextRow][nextCol] = newCost;

predecessors[nextRow][nextCol] = current;

}

}

}

}

// Построение пути от конечной вершины до начальной

vector<Node> path;

int minCost = numeric\_limits<int>::max();

// Проверка доступности ячеек нижнего ряда

for (int col = 0; col < cols; ++col) {

if (distances[rows - 1][col] != numeric\_limits<int>::max()) {

if (distances[rows - 1][col] < minCost) {

minCost = distances[rows - 1][col];

path.clear();

Node current = Node(rows - 1, col, distances[rows - 1][col]);

while (current.row != -1 && current.col != -1) {

path.push\_back(current);

current = predecessors[current.row][current.col];

}

}

}

}

// Переворот пути, чтобы получить правильную последовательность

reverse(path.begin(), path.end());

return make\_pair(minCost, path);

}

// Функция для поиска кратчайшего пути из любого элемента верхнего ряда в любой элемент нижнего ряда

pair<int, vector<Node>> find\_shortest\_path(const vector<vector<int>>& matrix) {

int rows = matrix.size();

int cols = matrix[0].size();

// Находим кратчайший путь из каждого элемента верхнего ряда

pair<int, vector<Node>> shortestPath;

shortestPath.first = numeric\_limits<int>::max();

for (int col = 0; col < cols; ++col) {

auto result = dijkstra(matrix, 0, col);

if (result.first < shortestPath.first) {

shortestPath = result;

}

}

return shortestPath;

}

int main() {

setlocale(LC\_ALL, "RU");

ifstream file("input.txt");

if (!file.is\_open()) {

cerr << "Ошибка открытия файла input.txt\n";

return 1;

}

int rows, cols;

file >> rows >> cols;

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

// Запись данных в матрицу

for (int i = 0; i < rows; ++i) {

for (int j = 0; j < cols; ++j) {

file >> matrix[i][j];

}

}

// Вывод матрицы

cout << "Матрица:" << endl;

for (int i = 0; i < rows; ++i) {

for (int j = 0; j < cols; ++j) {

cout << matrix[i][j] << " ";

}

cout << endl;

}

file.close();

// Находим кратчайший путь

auto result = find\_shortest\_path(matrix);

// Вывод результата

cout << "Минимальная стоимость пути: " << result.first << endl;

cout << "Путь: " << endl;

for (auto node : result.second) {

cout << "(" << node.row << ", " << node.col << ")" << " ";

}

cout << endl;

// Визуализация пути

cout << "Визуализация пути: " << endl;

for (int i = 0; i < rows; ++i) {

for (int j = 0; j < cols; ++j) {

// Проверка, находится ли текущая ячейка на пути

bool isOnPath = false;

for (auto node : result.second) {

if (node.row == i && node.col == j) {

isOnPath = true;

break;

}

}

if (isOnPath) {

// Если ячейка на пути, выводим её в зелёном цвете

cout << "\033[32m" << matrix[i][j] << "\033[0m" << " ";

}

else {

// Иначе выводим в обычном цвете

cout << matrix[i][j] << " ";

}

}

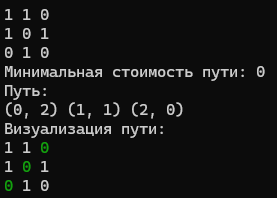
cout << endl;

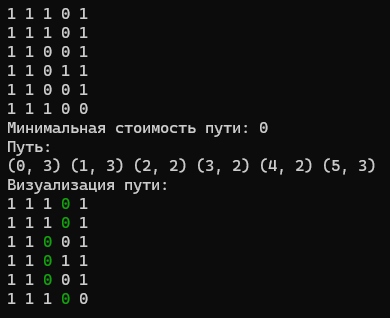
}

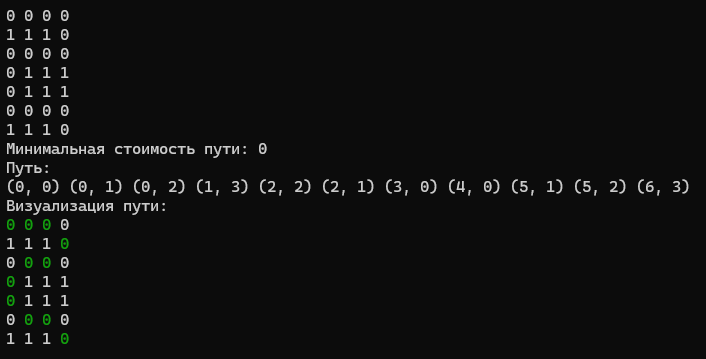
return 0;

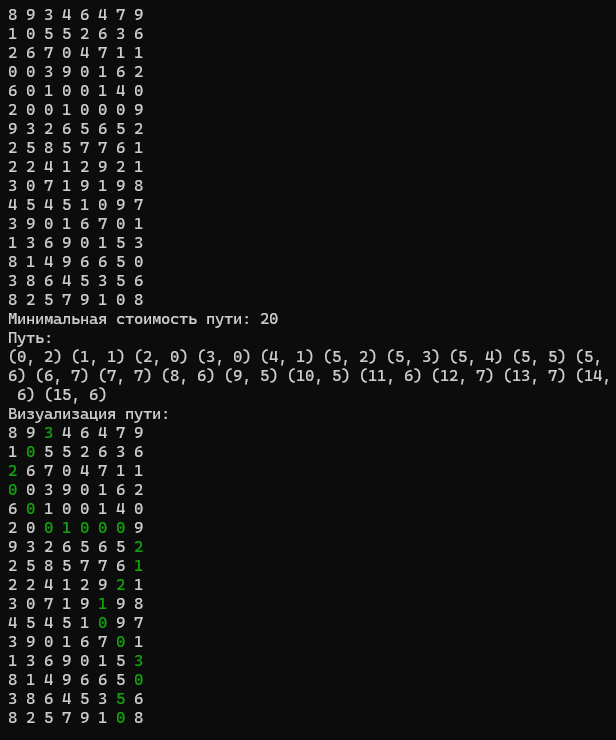
}

Примеры работы программы









Вывод

В ходе выполнения лабораторной работы были освоены принципы работы алгоритма Дейкстры для поиска кратчайшего пути в графе, представленном в виде матрицы. Была реализована программа на языке C++, которая считывает данные из файла, находит кратчайший путь из любого элемента верхнего ряда в любой элемент нижнего ряда и визуализирует полученный результат с использованием ANSI-кодов.