**Отчёт по лабораторной работе №5**

**Динамическое программирование**

Задание:

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.

2. Дан двумерный числовой массив размером N1xN1, где N1 - нечетное. Найти такой путь из центра массива к одной из сторон, чтобы сумма чисел по данному пути была минимальной. Из каждой клетки массива допустимо двигаться влево, вправо, вниз или вверх, при этом нельзя возвращаться в уже посещенные строки и столбцы.

Код программы:

|  |
| --- |
| Source.cpp |
| #include <algorithm>  #include <iostream>  #include <fstream>  #include <string>  #include <limits>  #include <deque>  struct Cell {  int x;  int y;  int weight;  Cell\* parent;  Cell(int x, int y, int w, Cell\* p) { // Конструктор  this->x = x; this->y = y; weight = w; parent = p;  }  };  // Возвращает не посещены ли повторно уже посещённые столбцы / строки  bool is\_valid(Cell\* cell, int size) {  int\* visited\_x = new int[size]();  int\* visited\_y = new int[size]();  while (cell->parent != nullptr) {  visited\_x[cell->x]++;  visited\_y[cell->y]++;  cell = cell->parent;  }  visited\_x[cell->x]++;  visited\_y[cell->y]++;  int max\_x = \*std::max\_element(visited\_x, visited\_x + size);  int max\_y = \*std::max\_element(visited\_y, visited\_y + size);  return (max\_x <= 2 && max\_y <= 2);  }  Cell\* get\_path(int\*\* matrix, int n) {  int center = n / 2;  int directions[4][2] = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}}; // Вверх Вниз Влево Вправо  int min\_sum = INT\_MAX;  Cell\* best\_path = nullptr;  std::deque<Cell\*> queue;  // Просчёт первых четырёх ячеек  Cell\* c\_center = new Cell(center, center, matrix[center][center], nullptr);  for (int i = 0; i < 4; i++) {  int nx = c\_center->x + directions[i][0];  int ny = c\_center->y + directions[i][1];  Cell\* new\_cell = new Cell(nx, ny, c\_center->weight + matrix[nx][ny], c\_center);  queue.push\_back(new\_cell);  }  // Просчёт последующих ячеек  while (!queue.empty()) {  Cell\* cell = queue.front();  queue.pop\_front();  if (cell->x == 0 || cell->y == 0 || cell->x == n - 1 || cell->y == n - 1) { // Края массива  if (cell->weight < min\_sum) {  min\_sum = cell->weight;  best\_path = cell;  }  continue;  }  if (cell->x == cell->parent->x) { // Выход из посещённой строки  int directions\_x[4][2] = {{-1, 0}, {1, 0}, {0, 0}, {0, 0}};  memcpy(directions, directions\_x, sizeof(directions));  }  if (cell->y == cell->parent->y) { // Выход из посещённого столбца  int directions\_y[4][2] = {{0, -1}, {0, 1}, {0, 0}, {0, 0}};  memcpy(directions, directions\_y, sizeof(directions));  }  // Проход по возможным направлениям  for (int i = 0; i < 2; i++) {  int nx = cell->x + directions[i][0];  int ny = cell->y + directions[i][1];  if (0 <= nx < n && 0 <= ny < n && is\_valid(cell, n)) { // Не край ли и валиден ли путь?  Cell\* new\_cell = new Cell(nx, ny, cell->weight + matrix[nx][ny], cell);  queue.push\_back(new\_cell);  }  }  }  return best\_path;  }  int main() {  std::ifstream file("input.txt");  if (!file.is\_open()) {  std::cout << "input.txt cannot be opened!" << std::endl;  return 255;  }    int size;  file >> size;  if (size % 2 == 0) {  std::cout << "Array size must be odd!" << std::endl;  return 255;  }    int\*\* array = new int\* [size]();  for (int i = 0; i < size; i++)  array[i] = new int[size];  std::string line;  // Преобразование содержимого файла в массив  for (int i = 0; i < size; i++)  for (int j = 0; j < size; j++)  file >> array[i][j];  file.close();  Cell\* path\_end = get\_path(array, size);  std::cout << "Min path: " << path\_end->weight << std::endl;    // Карта пройденного пути  bool\*\* path = new bool\* [size]();  for (int i = 0; i < size; i++)  path[i] = new bool[size]();  while (path\_end->parent != nullptr) {  path[path\_end->x][path\_end->y] = true;  path\_end = path\_end->parent;  }  path[path\_end->x][path\_end->y] = true;  // Вывод пути по карте  for (int i = 0; i < size; i++) {  for (int j = 0; j < size; j++) {  if (path[i][j]) std::cout << "\033[0;31m" << array[i][j] << "\033[0m ";  else std::cout << array[i][j] << " ";  }  std::cout << std::endl;  }  return 0;  } |

Алгоритм работы:

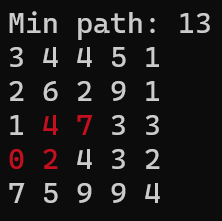
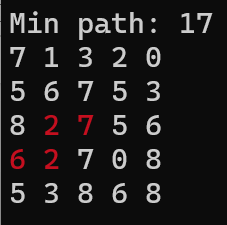
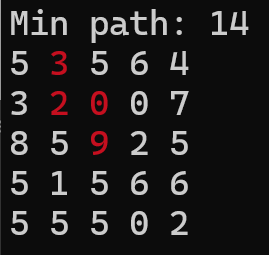
Программа открывает для чтения файл input.txt и считывает из него первую строку, содержимое которой указывает на размер квадратного массива (NxN) если файл не считан или размер массива чётный – выводится сообщение об ошибке. Затем программа считывает в массив целых чисел дальнейшее содержимое файла, разделённое пробелом, после чего файл закрывается. Полученный массив передаётся в функцию, возвращающую минимальную сумму пути от центра до одного из краёв массива в виде структуры.

Функция работает по алгоритму обхода графа в ширину: перед тем как перейти в граф на вершину k+1 обходятся все вершины k. В программе алгоритм реализован следующим образом: создаётся дек с структурами «ячейками» и массив размером NxN, в котором содержится информация посещена ли конкретная «ячейка». «Ячейка» хранит в себе индекс в массиве, её вес и предыдущую ячейку. После создания необходимых структур, функция просчитывает всевозможные пути до вершин графа – концов массива: сначала просчитываются 4-пути из центральной вершины, затем для них просчитываются все возможные пути до края массива: в деку справа добавляются возможные ячейки, в которые можно продвинуться, согласно условию задачи (проверяется сменилась ли ячейка или столбец и выбираются возможны направления), затем для каждой добавленной ячейки в дек добавляются возможные пути и из них, перед этим проверив не посещены ли повторно уже посещённые вершины, просчитав количество их посещений. При этом пройденные ячейки слева дека удаляются, а проходы просчитываются пока не будет достигнута граница массива. По итогу алгоритм построит все возможные пути от центра до края, а в деке останутся «ячейки», находящиеся по краям массива. Затем функция проходит по всем оставшимся ячейкам и находит минимальное значение веса пути и возвращает лучшую ячейку.

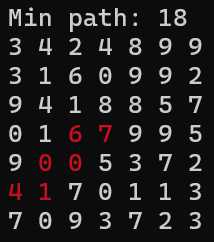
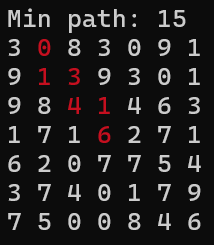
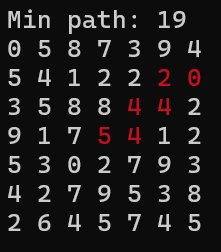
На экран выводится полученная минимальная сумма пути, а затем выводится прочитанный из файла массив. При выводе учитываются индексы пройденного пути из ранее полученной «ячейки» путём создания массива-карты, элементы которого указывают на пройденный путь. Если индекс элемента массива совпадает с индексом элемента карты, то выводимый элемент окрашивается цветом.

Результат работы:

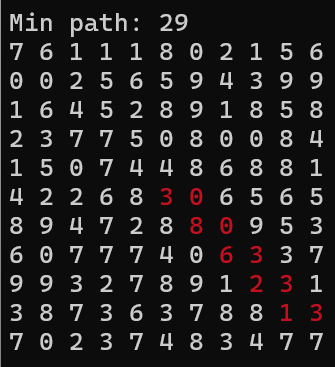
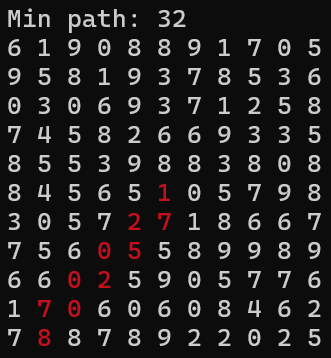
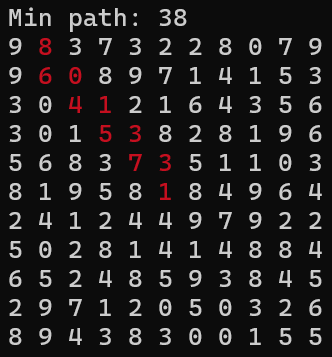
5x5:



7x7:



11x11:



15x15:



19x19:

