Балтийский государственный технический университет  
«ВОЕНМЕХ» им. Д. Ф. Устинова

Кафедра О7 «Информационные системы и программная инженерия»

**Практическая работа №2**по дисциплине «Структуры и организация данных»  
на тему «Нелинейные структуры данных»

вариант №12

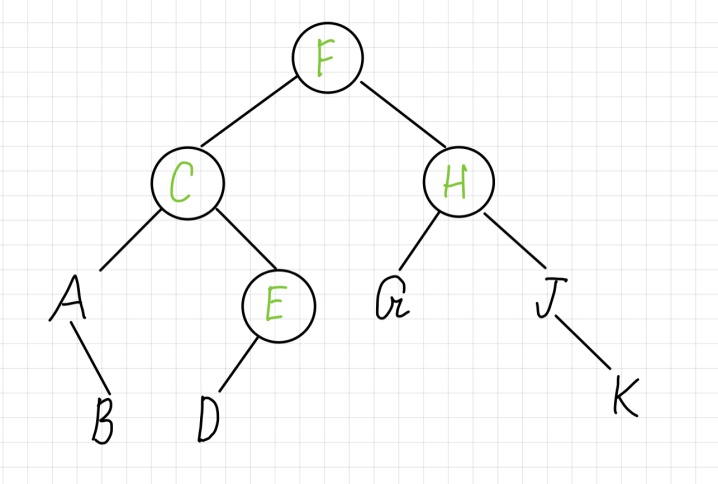
Выполнил: Костров Г. Ю.  
Группа О712Б  
  
Преподаватель:  
Палехова О. А.

Санкт-Петербург  
2022 г.

**Задание 1.**

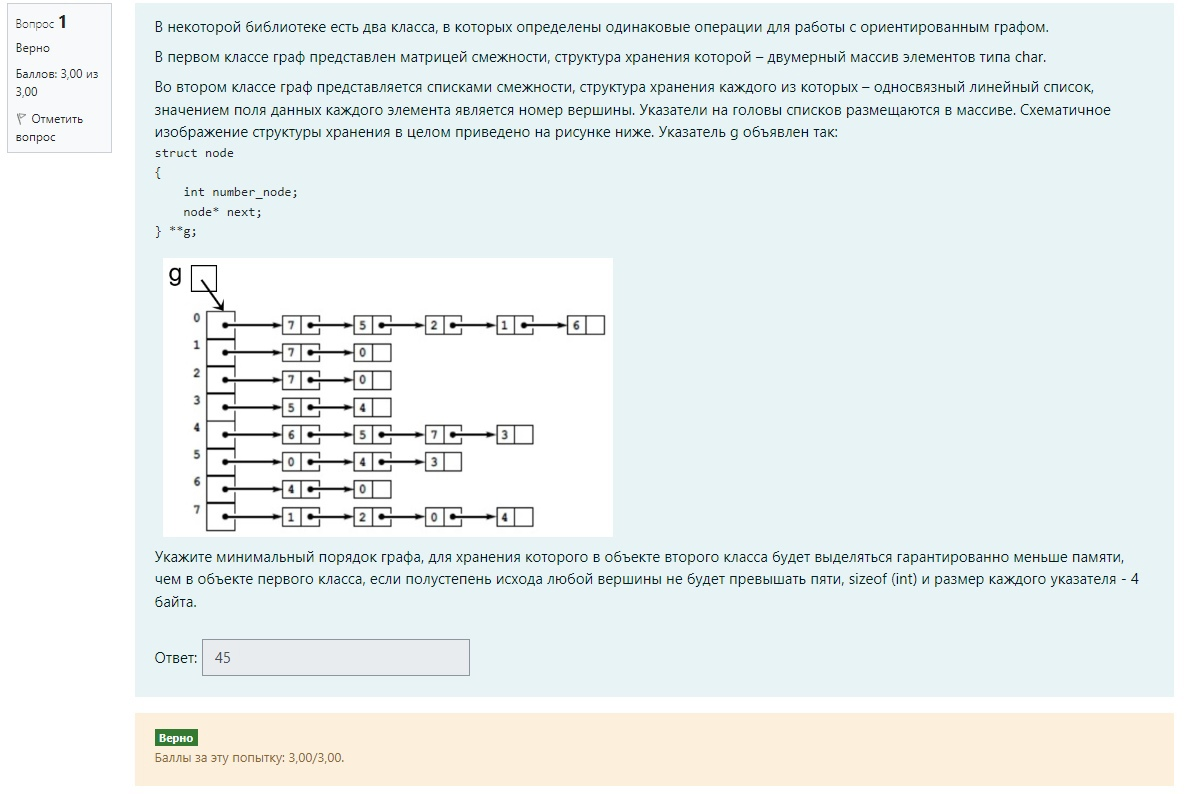
В прямом порядке обхода бинарного дерева была получена последовательность узлов FCABEDHGJK, а в симметричном - ABCDEFGHJK. У каких узлов этого дерева есть левые сыновья.

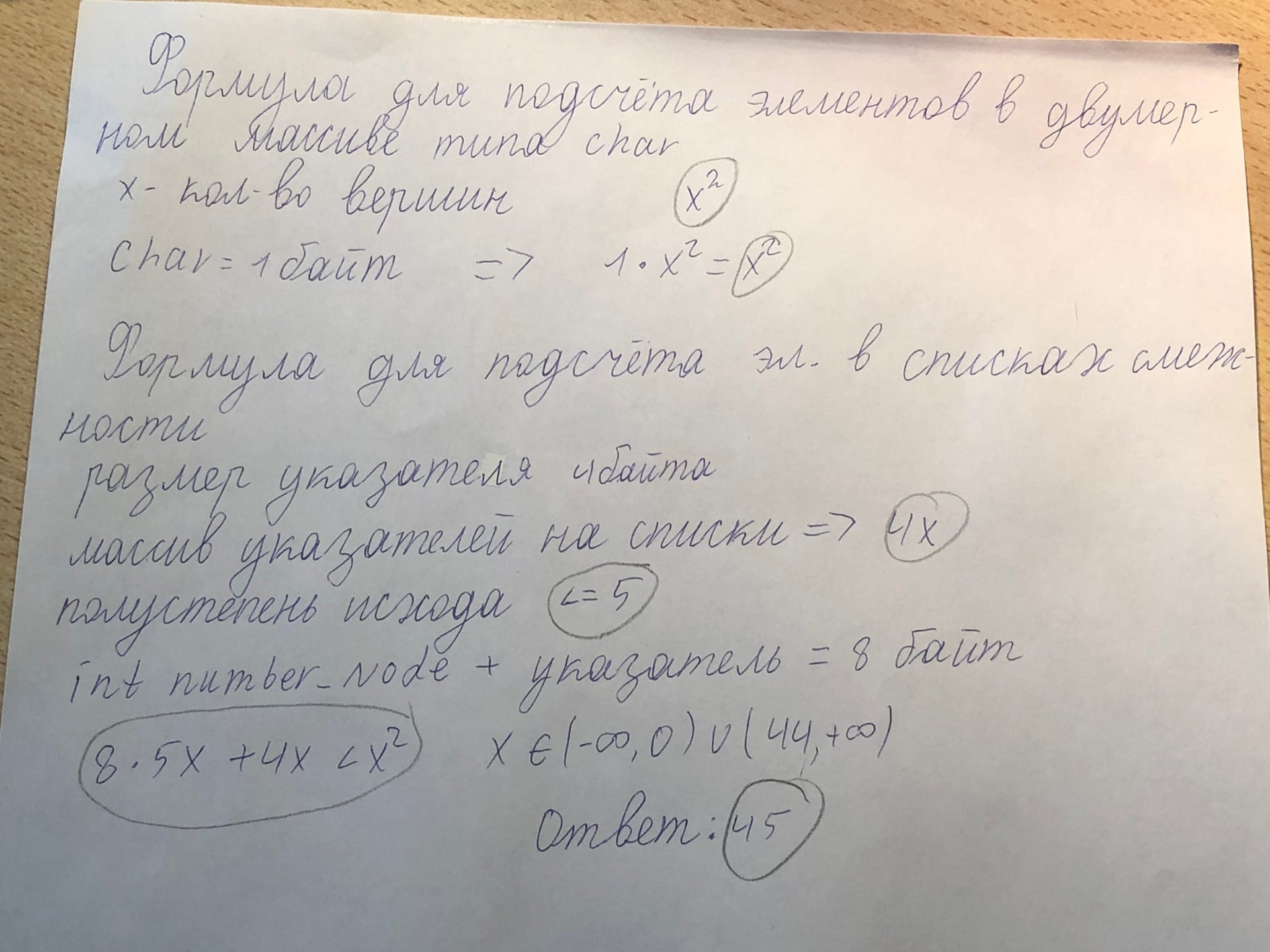
**Полученное дерево**:



**Ответ**: CEFH

**Задание 2.**

Решение задачи:



Ответ: 45

**Задание 3.** Написать программы для решения задачи.

Уровень сложности – **повышенный**.

Граф представляется двумя способами (матрицей смежности или весов и списками смежности), для каждого представления требуется написать отдельную программу решения задачи. Для тестирования программ требуется создать файлы с описанием графа одним способом (только матрицей или только списками), обе программы должны уметь заполнять структуры хранения, считывая файлы, как содержащие матрицы смежности, так и содержащие списки смежных вершин. При выборе структур хранения руководствоваться требованием разумной экономии памяти.

Вариант 12.

В системе двусторонних дорог за проезд каждой дороги взимается некоторая пошлина. Найти путь из города А в город В с минимальной величиной S+P, где S – сумма длин дорог пути, а P – сумма пошлин проезжаемых дорог. А и В вводятся с клавиатуры. На среднем и повышенном уровнях сложности помимо найденного пути вывести отдельно S и P.

Тестовые графы:

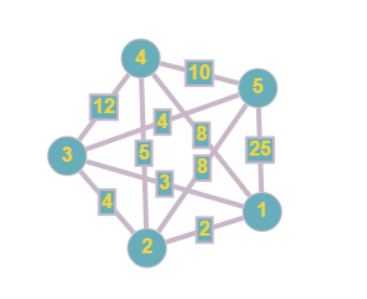
*.*

Рисунок 1.

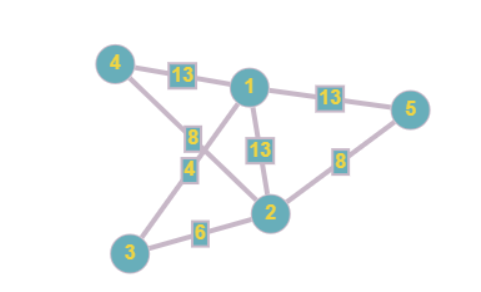


Рисунок 2.

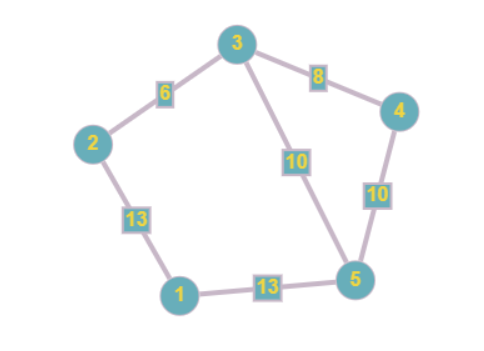
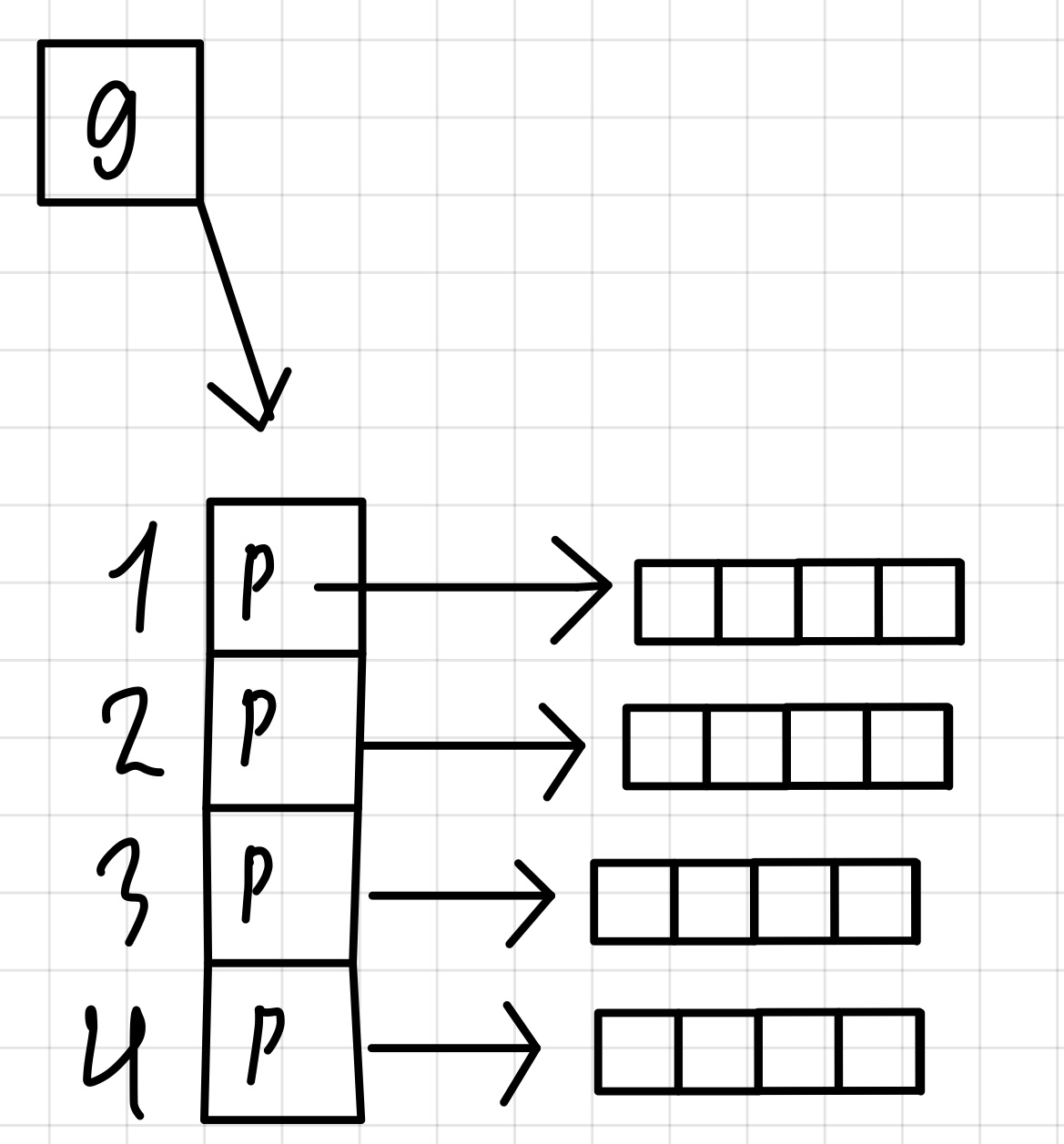


Рисунок 3

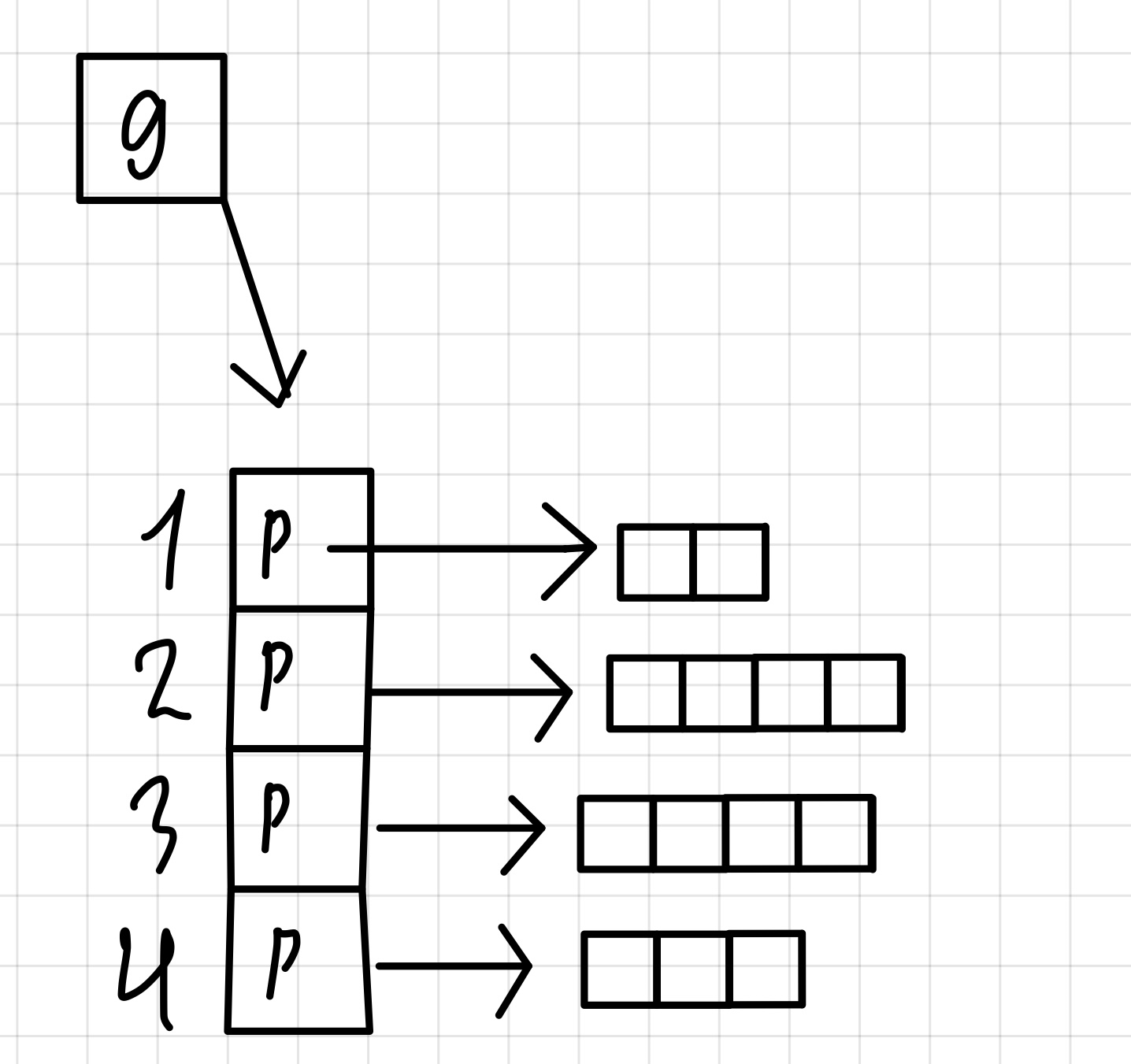
|  |  |  |  |
| --- | --- | --- | --- |
| № | Матрица смежности  Пример: 2 – (длина + пошлина) | Список смежности  Пример: 5;10;15;25  5 – вершина  10 – длина дороги  15 – пошлина  15 – (длина + пошлина) | Ожидаемый результат  (S + P) для каждой вершины(от 2 вершины) |
| 1 | 0 2 3 8 25  2 0 4 5 8  3 4 0 12 4  8 5 12 0 10  25 8 4 10 0 | 2;1;1;2 3;1;2;3 4;4;4;8 5;10;15;25  1;1;1;2 3;2;2;4 4;3;2;5 5;7;1;8  1;1;2;3 2;2;2;4 4;6;6;12 5;2;2;4  1;4;4;8 2;3;2;5 3;6;6;12 5;9;1;10  1;10;15;25 2;7;1;8 3;2;2;4 4;9;1;10 | (S + P) -  2 0 4 5 8 |
| 2 | 0 13 4 13 13  13 0 6 8 8  4 6 0 0 0  13 8 0 0 0  13 8 0 0 0 | 2;11;2;13 3;1;3;4 4;1;12;13 5;1;12;13  1;11;2;13 3;2;4;6 4;3;5;8 5;3;5;8  1;1;3;4 2;2;4;6  1;1;12;13 2;3;5;8  1;1;12;13 2;3;5;8 | (S + P) –  10 0 6 8 8 |
| 3 | 0 13 0 0 13  13 0 6 0 0  0 6 0 8 10  0 0 8 0 10  13 0 10 10 0 | 2;11;2;13 5;1;12;13  1;11;2;13 3;2;4;6  2;2;4;6 4;3;5;8 5;4;6;10  3;3;5;8 5;4;6;10  1;1;12;13 4;4;6;10 3;4;6;10 | (S + P) -  13 0 6 14 16 |

Структура хранения для представления графа матрицей смежности или весов:



Динамическая матрица позволяет грамотно оперировать памятью, выделяющуюся по мере необходимости в результате обработки данных считываемых из файла.

Структура хранения для представления графа списками смежных вершин:



Динамическая матрица отлично для хранения списков смежных вершин, т.к. занимает минимальное количество памяти. Память выделяется динамически после определение размера исходных данных, считываемых из файла. При использовании связных списков мы должны были бы ещё хранить указатель на последующий элемент строки. Это ведёт к увеличению размера необходимой памяти.

Применяемые алгоритмы:

Для каждого представления использован алгоритм дейкстры

Текст программы:

**Файл main.h**

#include "iostream"

#include "fstream"

#include "string"

#include "chrono"

#include "string\_view"

#include "charconv"

#include "iomanip"

#include "algorithm"

#include "..\..\..\helpFunctions\functions.cpp"

using namespace std;

// стркуктура хранения списков смежности

struct Row {

struct Node {

int vertice;

int total;

int tax;

int weight;

};

int countElements; // количетсво элементов в списке

struct Node \* pointerToArrayOfNodes; // указатель на массив "списка смежности"

Row(int countElements) {

this->countElements = countElements;

this->pointerToArrayOfNodes = new Node[countElements];

}

};

**Файл main.cpp**

#include "main.h"

void readFile(int ); // Функция для чтения информации из файла

int fromString\_viewToInt(string\_view, int ); // преобразование строки в int

void fillMatrix(int,int, string\_view, int); // заполнение матрицы смежности вершин

void fillList(int , string\_view , Row \*\*, int , int ); // заполнение списков смежности вершин

void Dijkstra (int , int D[], int P[], int , int weightSum[], int taxSum[]); // алгоритм Дейкстры для матрицы смежности

void Dijkstra2 (int , int D[], int P[], int , Row \*\* , int weightSum[], int taxSum[]); // алгоритм Дейкстры для списков смежности

void path (int P[], int to); // построение пути от начальной вершины до конечной

string fileName = "test1.txt"; // имя файла

int \*\*weightMatrix; // матрица весов

int \*\*taxMatrix; // матрица пошлин

int \*\*mainMatrix; // объединённая матрица (весы + пошлины)

Row \*\* adjacencyList; // указатель на массив указателей на списки смежности

string dataFromFile[800]; // данные из файла

string dataFromFileLists[800]; // данные из файла

int startVertex = 0; // начальный вершина

int endVertex = 0; // конечная вершина

int main() {

// меню

{

int variant = 0;

do {

system("cls");

string menu[] = {

"Choose an option!",

"1. Graph is represented by a matrix",

"2. Graph is represented by a adjacency list"};

printMenu(menu, 3);

exitMenu(3);

variant = getVariant(3);

system("cls");

switch (variant) {

case 1: {

readFile(1);

break;

}

case 2: {

readFile(2);

break;

}

}

} while (variant != 3);

}

return 0;

}

void readFile(int structureType) {

int inputDataType = 1;

// матрица смежности - 1

// списки смежных вершин - 2

int sizeOfMatrix = 0; // размер матрицы

string currentData; // текущие данные вершины из файла

int variant = 0;

// работа с файлом

{

ifstream file;

file.open(fileName);

string line;

if (file)

{

// проверка входящих данных

file >> line;

file.clear();

file.seekg(0, file.beg); // курсор в начало файла

if (line.length() == 0) {

cout << "File is empty" << endl;

}

//Вычисление размера матрицы

int counter = 0;

while (!file.eof()) {

// запись данных в массив

file >> dataFromFile[counter++];

if(file.peek() == '\n'|| file.peek() == EOF) {sizeOfMatrix++;}

}

string temp = dataFromFile[0];

inputDataType = count(temp.cbegin(), temp.cend(), ';');

file.clear();

file.seekg(0, file.beg); // курсор в начало файла

// если нам даны списки смежности, то считываем данные по строчно

int n = 0;

string line;

while (!file.eof()) {

getline(file, dataFromFileLists[n++], '\n');

}

file.close();

} else {

cout << "File not found" << endl;

}

}

// меню для выборам начальных и конечных вершин

{

cout << "Choose the peak from which we start(1-" << sizeOfMatrix << ")" << endl << ">";

startVertex = getVariant(sizeOfMatrix);

system("cls");

cout << "Select end vertex(1-" << sizeOfMatrix << ")" << endl << ">";

endVertex = getVariant(sizeOfMatrix);

system("cls");

}

int weightSum[sizeOfMatrix]; // сумма весов

int taxSum[sizeOfMatrix]; // сумма пошлин

int D[sizeOfMatrix], P[sizeOfMatrix];

// Если граф представляется матрицев весов

if (structureType == 1) {

//Выделение памяти для массива указателей на массивы строк матрицы

weightMatrix = new int \* [sizeOfMatrix];

taxMatrix = new int \* [sizeOfMatrix];

mainMatrix = new int \* [sizeOfMatrix];

//Обнуление указателй массива

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

mainMatrix[i] = taxMatrix[i] = weightMatrix[i] = nullptr;

}

// заполнение матриц данными из файла

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

// выделение памяти для массива строк матрицы

weightMatrix[i] = new int [sizeOfMatrix];

taxMatrix[i] = new int [sizeOfMatrix];

mainMatrix[i] = new int [sizeOfMatrix];

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

weightMatrix[i][j] = taxMatrix[i][j] = mainMatrix[i][j] = 9999;

}

weightMatrix[i][i] = taxMatrix[i][i] = mainMatrix[i][i] = 0;

if (inputDataType == 1) {

// матрица смежности из файла приходит

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

if(i == j) {continue;}

string\_view dataFromFileElem = dataFromFile[sizeOfMatrix \* i + j];

fillMatrix(i, j, dataFromFileElem, 1);

}

} else {

// приходят из файла списки смежности

int partLength = 0;

string\_view dataFromFileString = dataFromFileLists[i];

while( ( partLength = dataFromFileString.find(' ') ) != dataFromFileString.npos ) {

int temp;

string\_view currentString = dataFromFileString.substr(0, partLength);

fillMatrix(i, i, currentString, 0);

dataFromFileString.remove\_prefix( partLength + 1 );

}

// заполнение последнего элемента

fillMatrix(i, i, dataFromFileString, 0);

}

}

// вывод матрицы

{

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

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

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

}

cout << endl;

}

cout << endl << endl;

}

// алгоритм дейкстры для матрицы смежности вершин

Dijkstra(startVertex - 1, D, P, sizeOfMatrix, weightSum, taxSum);

} else { // Если граф представляется в виде списков смежности

// указатель на массив указателей на списки смежности

Row \*\* adjacencyList = new Row \* [sizeOfMatrix];

// обнуление указателей массива

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

adjacencyList[i] = nullptr;

}

for (int linkedListNumber = 0; linkedListNumber < sizeOfMatrix; linkedListNumber++) {

if (inputDataType == 2) {

// Если из файла нам пришли списки смежности

int i = 0;

int partLength = 0;

int listLenght = 1;

string\_view sizeSTR = dataFromFileLists[linkedListNumber];

while( ( partLength = sizeSTR.find(' ') ) != sizeSTR.npos ) {

sizeSTR.remove\_prefix( partLength + 1 ); listLenght++;

}

string\_view dataFromFileString = dataFromFileLists[linkedListNumber];

adjacencyList[linkedListNumber] = new Row(listLenght);

while( ( partLength = dataFromFileString.find(' ') ) != dataFromFileString.npos ) {

int temp;

string\_view currentString = dataFromFileString.substr(0, partLength);

fillList(linkedListNumber, currentString, adjacencyList, -1, i++);

dataFromFileString.remove\_prefix( partLength + 1 );

}

// заполнение последнего элемента

fillList(linkedListNumber, dataFromFileString, adjacencyList, -1, i);

} else {

// Если из файла нам приходит матрица смежности

int listLenght = 0;

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

string\_view currentString = dataFromFile[sizeOfMatrix \* linkedListNumber + j];

if (0 != fromString\_viewToInt(currentString, 0)) {

listLenght++;

}

}

adjacencyList[linkedListNumber] = new Row(listLenght);

for (int j = 0, position = 0; j < sizeOfMatrix; j++) {

string\_view currentString = dataFromFile[sizeOfMatrix \* linkedListNumber + j];

if (0 != fromString\_viewToInt(currentString, 0)) {

fillList(linkedListNumber, currentString, adjacencyList, position++, j + 1);

}

}

}

}

// вывод списков смежности

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

for(int j = 0; j < adjacencyList[i]->countElements; j++) {

auto currentNode = adjacencyList[i]->pointerToArrayOfNodes[j];

cout << currentNode.vertice << ";";

cout << currentNode.weight << ";";

cout << currentNode.tax << ";";

cout << currentNode.total << " ";

}

cout << endl;

}

cout << endl << endl;

// алгоритм дейкстры для списков смежности вершин

Dijkstra2(startVertex - 1, D, P, sizeOfMatrix, adjacencyList, weightSum, taxSum);

}

// вывод необходимых значений из условия

{

cout << "S(weight) for every vertex: " << endl;

for (int m = 0; m < sizeOfMatrix; m++) {

cout << weightSum[m] << " " ;

}

cout << endl;

cout << "P(tax) for every vertex: " << endl;

for (int m = 0; m < sizeOfMatrix; m++) {

cout << taxSum[m] << " " ;

}

cout << endl;

cout << "Shortest path from " << startVertex << " to " << endVertex <<"(tax + weight)" << ": " << endl;

for (int i =0; i<sizeOfMatrix; i++)

printf("%d ",D[i]);

cout << endl << endl;

}

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

if (D[i]<9999) {

path (P, i);

} else {

printf ("no way from 0 to %d\n", i);

}

}

system("pause");

}

// заполнение элементов списка смежности

void fillList(int linkedListNumber, string\_view currentString, Row \*\* adjacencyList, int type, int position) {

if (type == -1) {

for( int m = 0; m < 3; m++) {

int partLength = currentString.find(';');

string\_view tempStr = currentString.substr(0, partLength);

int temp = fromString\_viewToInt(tempStr, tempStr.length() - 1);

if (m == 0 ) {

adjacencyList[linkedListNumber]->pointerToArrayOfNodes[position].vertice = temp;

} else if(m == 1) {

adjacencyList[linkedListNumber]->pointerToArrayOfNodes[position].weight = temp;

} else {

adjacencyList[linkedListNumber]->pointerToArrayOfNodes[position].tax = temp;

}

currentString.remove\_prefix( partLength + 1 );

}

adjacencyList[linkedListNumber]->pointerToArrayOfNodes[position].total = adjacencyList[linkedListNumber]->pointerToArrayOfNodes[position].weight + adjacencyList[linkedListNumber]->pointerToArrayOfNodes[position].tax;

} else {

for( int m = 0; m < 2; m++) {

int partLength = currentString.find(';');

int temp;

if (m == 1){

temp = fromString\_viewToInt(currentString, currentString.length() - 1);

}

temp = fromString\_viewToInt(currentString, partLength);

adjacencyList[linkedListNumber]->pointerToArrayOfNodes[type].vertice = position;

if(m == 0) {

adjacencyList[linkedListNumber]->pointerToArrayOfNodes[type].weight = temp;

} else if (m == 1) {

adjacencyList[linkedListNumber]->pointerToArrayOfNodes[type].tax = temp;

}

// cout << mainMatrix[i][j] << " " << weightMatrix[i][j] << " " << taxMatrix[i][j] << endl;

currentString.remove\_prefix( partLength + 1 );

}

adjacencyList[linkedListNumber]->pointerToArrayOfNodes[type].total = adjacencyList[linkedListNumber]->pointerToArrayOfNodes[type].weight + adjacencyList[linkedListNumber]->pointerToArrayOfNodes[type].tax;

}

}

// заполнение матрицы

void fillMatrix(int i, int j, string\_view data, int type) {

int partLength;

// матрица смежности из файла приходит

if (type == 1) {

for( int m = 0; m < 2; m++) {

partLength = data.find(';');

string\_view currentString = data.substr(0, partLength);

int temp = fromString\_viewToInt(currentString, currentString.length() - 1);

if (m == 0 ) {

weightMatrix[i][j] = temp;

} else if(m == 1) {

taxMatrix[i][j] = temp;

}

data.remove\_prefix( partLength + 1 );

}

mainMatrix[i][j] = weightMatrix[i][j] + taxMatrix[i][j];

} else {

// списки смежности

int j = 0, counter = 0;

while( ( partLength = data.find(';') ) != data.npos ) {

string\_view currentString = data.substr(0, partLength);

int temp = fromString\_viewToInt(currentString, currentString.length() -1 );

if (counter == 0 ) {

j = temp;

} else if(counter == 1) {

weightMatrix[i][j - 1] = temp;

}

data.remove\_prefix( partLength + 1 );

counter++;

}

taxMatrix[i][j - 1] = fromString\_viewToInt(data, data.length() -1 );

mainMatrix[i][j - 1] = weightMatrix[i][j - 1] + taxMatrix[i][j - 1];

}

}

// превращение элемента строки в число

int fromString\_viewToInt(string\_view inputStr, int position) {

int result;

string\_view currentStr = inputStr.substr(0, position + 1); // получаем одно число из строки

auto res = from\_chars(currentStr.data(), currentStr.data() + currentStr.size(), result); // преобразуем число в int

return result;

}

// поиск минимального пути в матрице смежности

void Dijkstra (int start, int D[], int P[], int sizeOfMatrix, int weightSum[], int taxSum[]) {

int i, v, w, S[sizeOfMatrix]={0};

S[start] = 1;

for (i=0; i<sizeOfMatrix; i++) {

weightSum[i] = weightMatrix[start][i];

taxSum[i] = taxMatrix[start][i];

D[i] = mainMatrix[start][i]; /\* инициализация D \*/

P[i] = start; /\* инициализация P \*/

}

taxSum[start] = weightSum[start] = D[start] = 9999; /\* начальное значение для поиска минимума \*/

P[start] = -1; /\* стартовой вершине никакая другая не предшествует \*/

for (i=sizeOfMatrix - 3; i<sizeOfMatrix; i++) { /\* количество потенциальных промежуточных узлов на 2 меньше степени графа \*/

for (w=start,v=0; v<sizeOfMatrix; v++) {

if (!S[v] && D[v]<D[w]) { // выбор из множества S такой вершины w, что значение D[w] минимально;

w=v;

}

}

S[w]=1; /\* добавить w к множеству S; \*/

for (v=0; v<sizeOfMatrix; v++) { /\* для каждой вершины v из множества S \*/

if ( D[w] + mainMatrix[w][v] < D[v]) {

weightSum[v] = weightSum[w] + weightMatrix[w][v];

taxSum[v] = taxSum[w] + taxMatrix[w][v];

D[v] = D[w] + mainMatrix[w][v]; /\* D[v] = min (D[v], D[w]+C[w][v];\*/

P[v] = w; /\* вершине v на кратчайшем пути предшествует вершина w \*/

}

}

}

P[start] = -1;

taxSum[start] = weightSum[start] = D[start] = 0;

}

// поиск минимального пути в списках смежности

void Dijkstra2 (int start, int D[], int P[], int sizeOfMatrix, Row \*\* adjacencyList, int weightSum[], int taxSum[]) {

// int D[sizeOfMatrix], P[sizeOfMatrix];

int i, v, w, S[sizeOfMatrix]={0};

S[start] = 1;

for (i=0; i < sizeOfMatrix; i++) {

if (i == start) {

D[i] = weightSum[i] = taxSum[i]= 0;

} else {

D[i] = weightSum[i] = taxSum[i] = 9999;

}

P[i] = start; /\* инициализация P \*/

}

for (i = 0; i < adjacencyList[start]->countElements; i++) {

auto currentNode = adjacencyList[start]->pointerToArrayOfNodes[i];

int vertex = currentNode.vertice - 1;

D[vertex] = currentNode.total;

weightSum[vertex] = currentNode.weight;

taxSum[vertex] = currentNode.tax;

}

D[start] = weightSum[start] = taxSum[start] = 9999; /\* начальное значение для поиска минимума \*/

P[start] = -1; /\* стартовой вершине никакая другая не предшествует \*/

for (i = sizeOfMatrix - 3; i < sizeOfMatrix; i++) { /\* количество потенциальных промежуточных узлов на 2 меньше степени графа \*/

for (w = start,v = 0; v < sizeOfMatrix; v++) {

if (!S[v] && D[v]<D[w]) { /\* выбор из множества S такой вершины w, что значение D[w] минимально; \*/

w=v;

}

}

S[w]=1; /\* добавить w к множеству S; \*/

for (v = 0; v < adjacencyList[w]->countElements; v++) {

auto currentNode = adjacencyList[w]->pointerToArrayOfNodes[v];

int vertice = currentNode.vertice - 1; // фактическое расположение элемента в массиве меньше на один

if ( D[w] + currentNode.total < D[vertice ]) {

weightSum[vertice] = weightSum[w] + currentNode.weight;

taxSum[vertice] = taxSum[w] + currentNode.tax;

D[vertice] = D[w] + currentNode.total; /\* D[v] = min (D[v], D[w]+C[w][v];\*/

P[vertice] = w; /\* вершине v на кратчайшем пути предшествует вершина w \*/

}

}

}

D[start] = weightSum[start] = taxSum[start] = 0;

P[start] = -1;

}

void path (int P[], int to)

{

printf ("%d", to + 1); /\* конечная точка \*/

while ( P[to] > -1 ) /\* пока не дошли до стартовой вершины, которой ничего не предшествует \*/

{

to = P[to]; /\* "раскручиваем" путь \*/

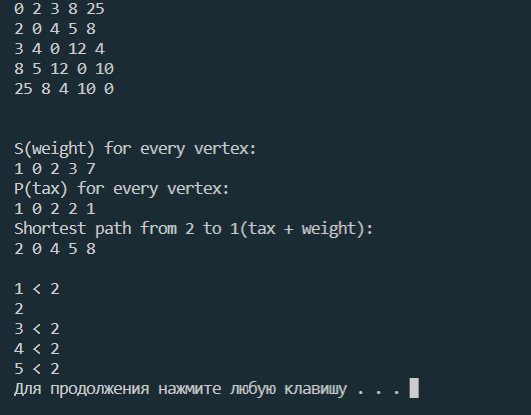
printf (" < %d", to + 1);

}

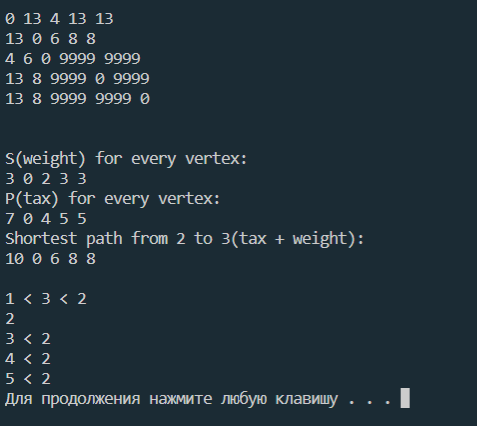
printf ("\n");

}

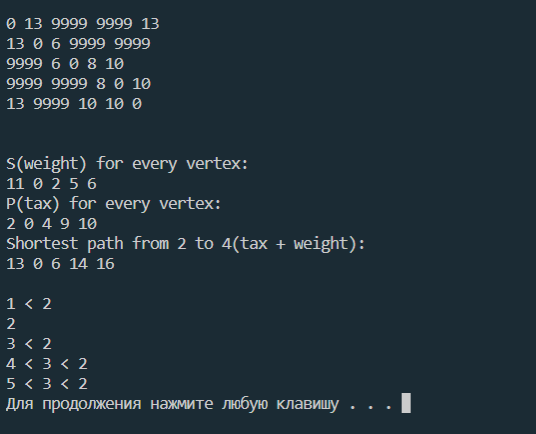
**Результаты работы программы:**



Граф 1



Граф 2



Граф 3