Министерство науки и высшего образования Российской Федерации  
Федеральное государственное автономное учреждение высшего образования  
«Пермский национальный исследовательский политехнический университет»

Электротехнический факультет  
Кафедра «Информационные технологии и автоматизированные системы»

**О Т Ч Ё Т  
по лабораторной работе**

Дисциплина: «Основы алгоритмизации и программирования»

Выполнил  
студент группы РИС-22-1б  
Бадртдинов Т.З

Проверил  
доцент кафедры ИТАС Полякова О.А.

Пермь 2023

Постановка задачи:

Реализовать алгоритмы для собственного варианта графа, имеющего не менее 6 вершин.

Необходимо реализовать функции для редактирования графа:

- Создание новой вершины.

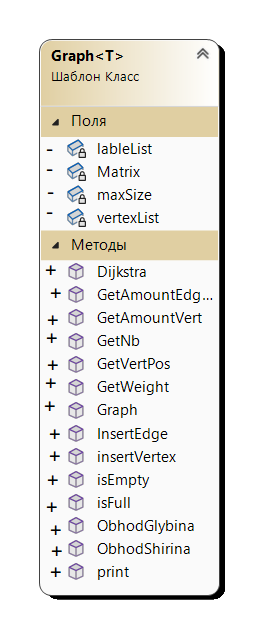
- Удаление вершины.

- Добавление и удаление ребра.

- Редактирование весов ребер.

- Редактирование матрицы смежности или инцидентности

Алгоритм программы



Код программы:

#include <iostream>

#include <vector>

using namespace std;

template <class T>

class Graph

{

private:

vector<T> vertexList;

vector<vector<int>> Matrix;

vector<T> lableList;

int maxSize;

public:

inline Graph(const int& size)

{

this->maxSize = size;

this->Matrix = vector<vector<T>>(size, vector<T>(size));

for (int i = 0; i < this->maxSize; i++)

{

for (int j = 0; j < this->maxSize; j++)

{

this->Matrix[i][j] = 0;

}

}

}

inline bool isFull()

{

return this->vertexList.size() == this->maxSize;

}

inline bool isEmpty()

{

return this->vertexList.size() == 0;

}

inline void insertVertex(const T& v)

{

if (this->isFull())

{

cout << "Невозможно добавить вершину" << endl;

return;

}

this->vertexList.push\_back(v);

}

inline int GetVertPos(const T& v)

{

for (int i = 0; i < this->vertexList.size(); i++)

{

if (this->vertexList[i] == v)

{

return i;

}

}

return -1;

}

inline int GetAmountVert()

{

return this->vertexList.size();

}

inline int GetWeight(const T& v1, const T& v2)

{

if (this->isEmpty())

{

return 0;

}

int v1\_p = this->GetVertPos(v1);

int v2\_p = this->GetVertPos(v2);

if (v1\_p == -1 || v2\_p == -1)

{

cout << "Одного из узлов в графе не существует" << endl;

return 0;

}

return this->Matrix[v1\_p][v2\_p];

}

vector<T> GetNb(const T& v)

{

vector<T> nbList;

int pos = this->GetVertPos(v);

if (pos != -1)

{

for (int i = 0; i < this->vertexList.size(); i++)

{

if (this->Matrix[pos][i] != 0)

{

nbList.push\_back(this->vertexList[i]);

}

}

}

return nbList;

}

void InsertEdge(const T& v1, const T& v2, int weight = 1)

{

if (GetVertPos(v1) != (-1) && this->GetVertPos(v2) != (-1))

{

int vPos1 = GetVertPos(v1);

int vPos2 = GetVertPos(v2);

if (this->Matrix[vPos1][vPos2] != 0 && this->Matrix[vPos2][vPos1] != 0)

{

cout << "Ребро между вершинами уже есть" << endl;

return;

}

else

{

this->Matrix[vPos1][vPos2] = weight;

this->Matrix[vPos2][vPos1] = weight;

}

}

else

{

cout << "Какой-то вершины нет в графе" << endl;

return;

}

}

void print()

{

int vertListSize = this->vertexList.size();

if (!this->isEmpty()) {

cout << "Матрица смежности: " << endl;

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

cout << this->vertexList[i] << " ";

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

cout << " " << this->Matrix[i][j] << " ";

}

cout << endl;

}

}

else

{

cout << "Граф пуст" << endl;

}

}

int GetAmountEdges()

{

int amount = 0;

if (!this->isEmpty())

{

for (int i = 0; i < this->vertexList.size(); i++)

{

for (int j = 0; j < this->vertexList.size(); j++)

{

if (this->Matrix[i][j] != 0)

{

amount++;

}

}

}

}

return amount / 2;

}

void ObhodShirina(T& startV, bool\* visitedV)

{

if (visitedV[this->GetVertPos(startV)] == false)

{

this->vertexList.push\_back(startV);

cout << "Вершина " << startV << " обработана" << endl;

visitedV[this->GetVertPos(startV)] = true;

}

vector<T> neighb = this->GetNb(startV);

this->vertexList.pop\_back();

for (int i = 0; i < neighb.size(); i++)

{

if (!visitedV[this->GetVertPos(neighb[i])])

{

this->vertexList.push\_back(neighb[i]);

visitedV[this->GetVertPos(neighb[i])] = true;

cout << "Вершина " << neighb[i] << " обработана" << endl;

}

}

if (this->vertexList.empty())

{

return;

}

ObhodShirina(vertexList.front(), visitedV);

}

void ObhodGlybina(T& startV, bool\* visitedV)

{

cout << "Вершина " << startV << " посещена" << endl;

visitedV[this->GetVertPos(startV)] = true;

vector<T> neighb = this->GetNb(startV);

for (int i = 0; i < neighb.size(); i++)

{

if (!visitedV[this->GetVertPos(neighb[i])])

{

this->ObhodGlybina(neighb[i], visitedV);

}

}

}

/\*void FillLabels(T& v)

{

for (int i = 0)

}\*/

/\*bool AllVisited(T& v)

{

if ()

}\*/

void Dijkstra(T& startV)

{

const int n = 10000;

int size = vertexList.size();

int\* min\_d = new int[size];

int\* count = new int[size];

int tmp, minindex, min;

int begin\_index = 0;

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

{

min\_d[i] = n;

count[i] = 1;

}

min\_d[begin\_index] = 0;

do {

minindex = n;

min = n;

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

{

if ((count[i] == 1) && (min\_d[i] < min))

{

min = min\_d[i];

minindex = i;

}

}

if (minindex != n)

{

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

{

if (Matrix[minindex][i] > 0)

{

tmp = min + Matrix[minindex][i];

if (tmp < min\_d[i])

{

min\_d[i] = tmp;

}

}

}

count[minindex] = 0;

}

} while (minindex < n);

cout << endl << "Кратчайшие расстояния до вершин: " << endl << endl;

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

{

cout << "До " << (i + 1) << " вершины: ";

cout << " " << min\_d[i] << endl << endl;

}

delete[] min\_d;

delete[] count;

}

};

int main()

{

setlocale(LC\_ALL, "Russian");

bool\* visitedVerts = new bool[20];

fill(visitedVerts, visitedVerts + 20, false);

int amountVerts, amountEdges, vertex, sourceVertex, targetVertex;

int edgeWeight;

cout << "Введите кол-во вершин графа: ";

cin >> amountVerts;

Graph<int> graf(amountVerts);

cout << endl;

cout << "Введите кол-во ребер графа: ";

cin >> amountEdges;

cout << endl;

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

{

cout << "Вершина: ";

cin >> vertex;

graf.insertVertex(vertex);

cout << endl;

}

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

{

cout << "Исходная вершина: "; cin >> sourceVertex; cout << endl;

cout << "Конечная вершина: "; cin >> targetVertex; cout << endl;

cout << "Вес ребра: "; cin >> edgeWeight; cout << endl;

graf.InsertEdge(sourceVertex, targetVertex, edgeWeight);

cout << endl;

}

graf.print();

cout << "Введите вершину, с которой начать обход(Обход в глубину): "; cin >> vertex; cout << endl;

graf.ObhodGlybina(vertex, visitedVerts);

cout << "Алгоритм Дейкстры. С какой вершины начать "; cin >> vertex; cout << endl;

graf.Dijkstra(vertex);

}

Результат работы

