|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | |  |  |  | | 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 № 5** | |
| **по дисциплине** | |
| **«**Структуры и алгоритмы обработки данных**»**  **Тема: «Основные алгоритмы работы с графами.»** | |
|  | |
| Выполнил студент группы ИКБО-02-21 | Семянников Н.С. |
| Принял преподаватель | Сорокин А.В. |

|  |  |  |
| --- | --- | --- |
| Лабораторная работа выполнена | «\_\_»\_\_\_\_\_\_\_202\_\_ г. | *(подпись студента)* |
|  |  |  |
| «Зачтено» | «\_\_»\_\_\_\_\_\_\_202\_\_ г. | *(подпись руководителя)* |

Москва 2022

[1. Постановка задачи 3](#_Toc120414743)

[1.1. Цель работы 3](#_Toc120414744)

[1.2. Условие задачи 3](#_Toc120414745)

[1.3. Подход к решению 3](#_Toc120414746)

[1.4. Алгоритмы операций 4](#_Toc120414747)

[1.4.1. Функция генерации нового графа 4](#_Toc120414748)

[1.4.2. Функция вывода графа 4](#_Toc120414749)

[1.4.3. Функция добавления графа 5](#_Toc120414750)

[1.4.4. Функция конвертация списка в матрицу 5](#_Toc120414751)

[1.4.5. Функция нахождения кратчайших путей (метод Флойда) 6](#_Toc120414752)

[1.4.6. Функция вывода матрицы 6](#_Toc120414753)

[2. Код 7](#_Toc120414754)

[3. Результат 11](#_Toc120414755)

[3.1. Генерация неориентированного графа 11](#_Toc120414756)

[3.2. Генерация ориентированного графа 11](#_Toc120414757)

[3.3. Добавление связи в граф 11](#_Toc120414758)

[3.4. Метод Флойда 11](#_Toc120414759)

[3.5. Поиск кратчайшего пути 12](#_Toc120414760)

[3.6. Выполнение работы на введённом графе 12](#_Toc120414761)

[4. Вывод 13](#_Toc120414762)

[5. Список литературы 14](#_Toc120414763)

# Постановка задачи

## Цель работы

Получение практических навыков по выполнению операций над структурой данных граф.

## Условие задачи

Выполнить разработку программы управления графом, в соответствии с вариантом №8, на основе класса Граф. Предусмотреть в качестве данных: количество вершин в графе, структура для хранения графа.

Условие варианта №8(таблица 1):

Таблица 1 - Условие варианта №8

|  |  |  |
| --- | --- | --- |
| №  варианта | Представление графа в памяти | Задачи |
| 8 | Список смежных вершин | Ввод с клавиатуры графа (применение операции вставки ребра в граф).Составить программу нахождения кратчайших путей методом «Флойда». Используя результат алгоритма вывести путь между вводимыми парами вершин. |

## Подход к решению

Списки смежных вершин – связанные списки, содержащие список вершин, смежных заданной вершине.

Для вывода программа проходит по всему списку вершин и выводим все грани к другим вершинам с их весом.

При добавлении нового ребра/вершины с клавиатуры происходит проверка на размер списка, если вставленная вершина имеет номер выше, чем размер списка, то список увеличивается до соответственного размера. Затем происходит вставка значений в связанные вершины.

Для переключения между ориентированный и неориентированным графом была введена специальная переменная, при переключении которой меняется режим добавления рёбер в список.

Для нахождения кратчайших путей был использован метод Флойда, а он в свою очередь требует преобразования списка в матрицу смежных вершин. Поэтому были написаны алгоритмы конвертации списка в матрицу и вывода матрицы смежных вершин.

## Алгоритмы операций

### Функция генерации нового графа

Принимает: граф, размер графа.

Описание: генерирует случайные вершины и связи.

Код:

void generateList(vector<pair<int, int>> adjList[], int vectorSize) {

srand(time(NULL));

bool exists = false;

int from, to, count;

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

{

count = mode ? 1 : (rand() % 10 < 2 ? 2 : 1);

from = i;

for (int j = 0; j < count; j++)

{

do

{

exists = false;

to = rand() % vectorSize;

for (int k = 0; k < vectorSize && !exists; k++)

{

for (auto node : adjList[k])

{

if ((k == from && node.first == to) || (k == to && node.first == from))

{

exists = true;

break;

}

}

}

} while (from == to || exists);

adjList = addEdge(adjList, from, to, rand() % 9 + 1, vectorSize);

}

}

}

### Функция вывода графа

Принимает: граф, размер графа.

Описание: выводит список поочерёдно.

Код:

void printList(vector <pair<int, int>> adjList[], int vectorSize)

{

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

cout << i;

for (auto j : adjList[i])

cout << " --> " << j.first << "(" << j.second << ")";

cout << endl;

}

cout << endl;

}

### Функция добавления графа

Принимает: граф, отправная вершина, конечная точка, длина пути, размер графа.

Возвращает: граф.

Описание: добавляет новую вершину, если она выходит за размер графа и связывает её с отправной/конченой.

Код:

vector<pair<int, int>>\* addEdge(vector<pair<int, int>> adj[], int u, int v, int wt, int& vectorSize)

{

if (u >= vectorSize || v >= vectorSize)

{

vector<pair<int, int>>\* adjList = new vector<pair<int, int>>[max(u, v) + 1];

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

{

adjList[i] = adj[i];

}

adj = adjList;

vectorSize = max(u, v) + 1;

}

adj[u].push\_back(make\_pair(v, wt));

if (mode)

{

adj[v].push\_back(make\_pair(u, wt));

}

return adj;

}

### Функция конвертация списка в матрицу

Принимает: граф, размер графа.

Возвращает: матрицу.

Описание: проходит по списку и заполняет матрицу смежных вершин.

Код:

vector<vector<int>> convert(vector<pair<int, int>> adjList[], int vectorSize)

{

vector<vector<int>> matrix(vectorSize, vector<int>(vectorSize, INF));

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

for (auto j : adjList[i])

matrix[i][j.first] = j.second;

}

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

{

matrix[i][i] = 0;

}

return matrix;

}

### Функция нахождения кратчайших путей (метод Флойда)

Принимает: граф, размер графа.

Возвращает: матрицу кратчайших путей.

Описание: находит кратчайший путь от вершины i в вершину j с помощью прохода через вершину k, при этом вершины i и j не проходят через вершины, большие чем k.

Код:

vector<vector<int>> floydWarshall(vector<vector<int>> adjMatrix, int vectorSize)

{

int i, j, k;

vector<vector<int>> dist(vectorSize, vector<int>(vectorSize, -1));

for (i = 0; i < vectorSize; i++)

for (j = 0; j < vectorSize; j++)

dist[i][j] = adjMatrix[i][j];

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

for (j = 0; j < vectorSize; j++) {

for (k = 0; k < vectorSize; k++) {

if (dist[j][k] > (dist[j][i] + dist[i][k]) && (dist[i][k] != INF && dist[j][i] != INF))

dist[j][k] = dist[j][i] + dist[i][k];

}

}

}

return dist;

}

### Функция вывода матрицы

Принимает: граф, размер графа.

Описание: проходит по матрицы, вывода длины путей.

Код:

void printMatrix(vector<vector<int> > adj, int vectorSize)

{

cout << " ";

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

{

cout << i << "\t";

}

cout << endl;

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

cout << i << "| ";

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

if (i != j && adj[i][j] == INF)

{

cout << "#\t";

}

else

cout << adj[i][j] << "\t";

}

cout << endl;

}

cout << endl;

}

# Код

#include <iostream>

#include <vector>

using namespace std;

#define INF 99999

bool mode = true;

// List to matrix convertation

vector<vector<int>> convert(vector<pair<int, int>> adjList[], int vectorSize)

{

vector<vector<int>> matrix(vectorSize, vector<int>(vectorSize, INF));

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

for (auto j : adjList[i])

matrix[i][j.first] = j.second;

}

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

{

matrix[i][i] = 0;

}

return matrix;

}

// To add an edge

vector<pair<int, int>>\* addEdge(vector<pair<int, int>> adj[], int u, int v, int wt, int& vectorSize)

{

if (u >= vectorSize || v >= vectorSize)

{

vector<pair<int, int>>\* adjList = new vector<pair<int, int>>[max(u, v) + 1];

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

{

adjList[i] = adj[i];

}

adj = adjList;

vectorSize = max(u, v) + 1;

}

adj[u].push\_back(make\_pair(v, wt));

if (mode)

{

adj[v].push\_back(make\_pair(u, wt));

}

return adj;

}

// Floyd method

vector<vector<int>> floydWarshall(vector<vector<int>> adjMatrix, int vectorSize)

{

int i, j, k;

vector<vector<int>> dist(vectorSize, vector<int>(vectorSize, -1));

for (i = 0; i < vectorSize; i++)

for (j = 0; j < vectorSize; j++)

dist[i][j] = adjMatrix[i][j];

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

for (j = 0; j < vectorSize; j++) {

for (k = 0; k < vectorSize; k++) {

if (dist[j][k] > (dist[j][i] + dist[i][k]) && (dist[i][k] != INF && dist[j][i] != INF))

dist[j][k] = dist[j][i] + dist[i][k];

}

}

}

return dist;

}

// Print adjacency list/matrix representation of graph

void printList(vector <pair<int, int>> adjList[], int vectorSize)

{

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

cout << i;

for (auto j : adjList[i])

cout << " --> " << j.first << "(" << j.second << ")";

cout << endl;

}

cout << endl;

}

void printMatrix(vector<vector<int> > adj, int vectorSize)

{

cout << " ";

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

{

cout << i << "\t";

}

cout << endl;

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

cout << i << "| ";

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

if (i != j && adj[i][j] == INF)

{

cout << "#\t";

}

else

cout << adj[i][j] << "\t";

}

cout << endl;

}

cout << endl;

}

// Generate new list

void generateList(vector<pair<int, int>> adjList[], int vectorSize) {

srand(time(NULL));

bool exists = false;

int from, to, count;

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

{

count = mode ? 1 : (rand() % 10 < 2 ? 2 : 1);

from = i;

for (int j = 0; j < count; j++)

{

do

{

exists = false;

to = rand() % vectorSize;

for (int k = 0; k < vectorSize && !exists; k++)

{

for (auto node : adjList[k])

{

if ((k == from && node.first == to) || (k == to && node.first == from))

{

exists = true;

break;

}

}

}

} while (from == to || exists);

adjList = addEdge(adjList, from, to, rand() % 9 + 1, vectorSize);

}

}

}

int main()

{

setlocale(0, "");

int vectorSize = 0, a, from, to, weight;

vector<pair<int, int>>\* adjList = new vector<pair<int, int>>[vectorSize];

vector<vector<int>> adjMatrix;

vector<vector<int>> minDist;

bool exists = false;

while (true) {

cout << "0-Сменить тип графа\n";

cout << "1-Создать новый список\n";

cout << "2-Добавить новый граф/связь\n";

cout << "3-Алгоритм Флойда\n";

cout << "4-Найти минимальный путь между двумя узлами\n";

cout << "Выбор действия: ";

cin >> a;

switch (a)

{

case 0: {

mode = !mode;

cout << "Новый тип графа: " << (mode ? "неориентированный" : "ориентированный") << endl;

break;

}

case 1: {

cout << "Кол-во узлов: ";

cin >> vectorSize;

adjList = new vector<pair<int, int>>[vectorSize];

generateList(adjList, vectorSize);

printList(adjList, vectorSize);

break;

}

case 2: {

cout << "Из какого узла: ";

cin >> from;

if (from < 0)

{

cout << "Значение неверно" << endl;

break;

}

cout << "В какой узел: ";

cin >> to;

if (to < 0)

{

cout << "Значение неверно" << endl;

break;

}

cout << "Вес грани: ";

cin >> weight;

if (weight <= 0 || weight >= 20)

{

cout << "Значение неверно" << endl;

break;

}

for (int j = 0; j < vectorSize && !exists; j++)

{

for (auto node : adjList[j])

{

if ((j == from && node.first == to) || (j == to && node.first == from))

{

exists = true;

break;

}

}

}

if (exists)

{

cout << "Такая связь уже есть" << endl;

break;

}

adjList = addEdge(adjList, from, to, weight, vectorSize);

printList(adjList, vectorSize);

break;

}

case 3: {

adjMatrix = convert(adjList, vectorSize);

printMatrix(adjMatrix, vectorSize);

minDist = floydWarshall(adjMatrix, vectorSize);

printMatrix(minDist, vectorSize);

break;

}

case 4: {

if (minDist.size() == 0)

{

cout << "Необходимо использовать метод Флойда\n";

break;

}

cout << "Отправной узел: ";

cin >> from;

if (from >= vectorSize || from < 0)

{

cout << "Значение неверно\n";

break;

}

cout << "Итоговый узел: ";

cin >> to;

if (to >= vectorSize || to < 0)

{

cout << "Значение неверно\n";

break;

}

cout << "Минимальный путь: " << minDist[from][to] << endl;

break;

}

default:

break;

}

}

return 0;

}

# Результат

## Генерация неориентированного графа

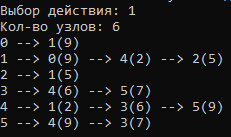


Рисунок 1 - Неориентированный граф

## Генерация ориентированного графа

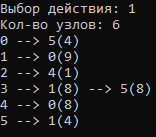


Рисунок 2- Ориентированный граф

## Добавление связи в граф

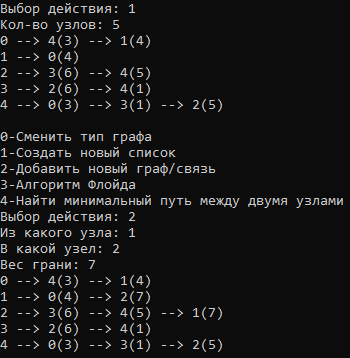


Рисунок 3 - Добавление связи

## Метод Флойда

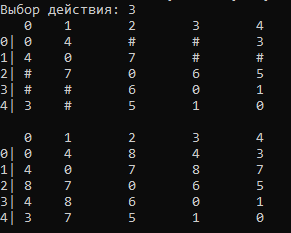
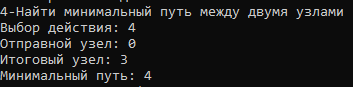


Рисунок 4 - Метод Флойда

## Поиск кратчайшего пути



## Выполнение работы на введённом графе

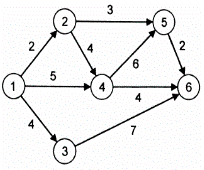


Рисунок 5 - Граф

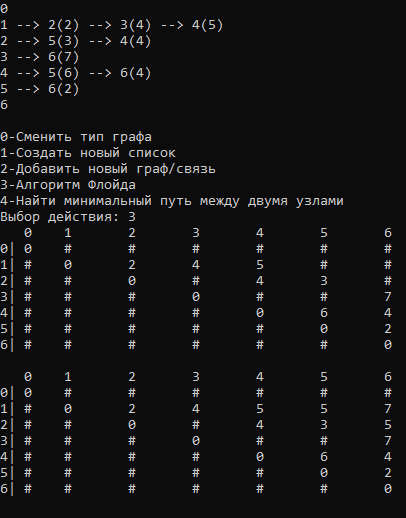


Рисунок 6 - Результат работы

# Вывод

В ходе выполнения практической работы были приобретены навыки по работе с ориентированными и неориентированными графами.

Была разработана и протестирована программа для задачи варианта №8.

# Список литературы

1. Система вопросов и ответов о программировании: <https://stackoverflow.com/>
2. Веб ресурс: <https://ru.wikipedia.org/wiki/Ориентированный_граф>