**Лабораторная работа №7**

**Поиск расстояний во взвешенном графе**

**Общие сведения.**

Во взвешенном графе в отличие от не взвешенного каждое ребро имеет вес, отличный от нуля. Поэтому в матрице смежности взвешенного графа содержится информация не только о наличии ребра, но и о его весе.

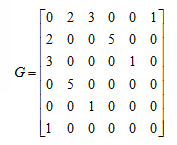
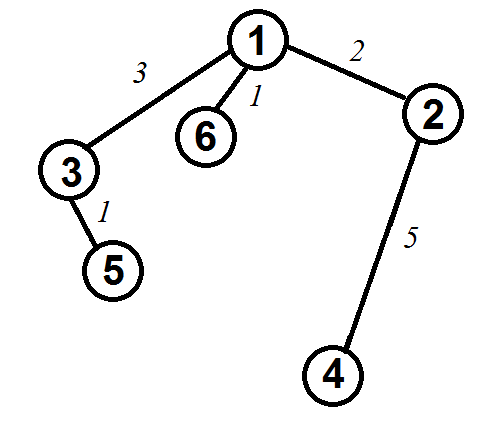


Рисунок 1 – Граф

Поиск расстояний между вершинами в таком графе также возможно построить используя процедуры обхода графа. Отличие от поиска расстояний в не взвешенном графе будет состоять в том, что при обновлении расстояния до вершины при ее посещении оно будет увеличиваться не на 1, а на величину веса ребра.

Таким образом, можно предложить следующую реализацию алгоритма обхода в ширину.

**Вход**: G – матрица смежности графа, v – исходная вершина.

**Выход**: DIST – вектор расстояний до всех вершин от исходной.

**Алгоритм ПОШ**

1.1. для всех i положим DIST [i] = -1 пометим как "не посещенную";

1.2. **ВЫПОЛНЯТЬ** BFSD (v).

1.3 для всех i вывести DIST [i] на экран;

**Алгоритм** BFSD(v):

2.1. Создать пустую очередь Q = {};

2.2. Поместить v в очередь Q.push(v);

2.3. Обновить вектор расстояний DIST [ x ] = 0;

2.4. **ПОКА**  Q != ∅ очередь не пуста **ВЫПОЛНЯТЬ**

2.5. v = Q.front() установить текущую вершину;

2.6. Удалить первый элемент из очереди Q.pop();

2.7. вывести на экран v;

2.8. **ДЛЯ** i = 1 **ДО** size\_G **ВЫПОЛНЯТЬ**

2.9. **ЕСЛИ** G(v,i) > 0 **И** DIST = = -1

2.10. **ТО**

2.11. Поместить i в очередь Q.push(i);

2.12. Обновить вектор расстояний DIST [ i ] = DIST [ v ] + G(v,i);

Реализация состоит из подготовительной части, в которой все вершины помечаются как не поcещенные (п.1.1). Не посещенные вершины помечаются – 1, т.к. значение 0 и 1 могут быть расстояниями. Расстояние 0 – от исходной вершины до самой себя.

В самой процедуре сначала создается пустая очередь (п. 2.1), в которую помещается исходная вершина, из которой начат обход (п.2.2). Расстояние до этой вершины (п.2.3) устанавливается равным 0 (расстояние до самой себя).

Далее итерационно, пока очередь не опустеет, из нее извлекается первый элемент, который становится текущей вершиной (п. 2.5, 2.6). Затем в цикле просматривается **v**-я строка матрицы смежности графа G(v,i). Как только алгоритм встречает смежную с **v** не посещенную вершину (п.2.9), эта вершина помещается в очередь (п.2.11) и для нее обновляется вектор расстояния (п.2.12). Расстояние до новой **i**-й вершины вычисляется как расстояние до текущей **v**-й вершины плюс вес ребра до новой вершины G(v,i).

После просмотра строки матрицы смежности алгоритм делает следующую итерацию цикла 2.4 или заканчивает работу, если очередь пуста.

**Задание 1**

1. Сгенерируйте (используя генератор случайных чисел) матрицу смежности для неориентированного взвешенного графа *G*. Выведите матрицу на экран.
2. Для сгенерированного графа осуществите процедуру поиска расстояний, реализованную в соответствии с приведенным выше описанием. При реализации алгоритма в качестве очереди используйте класс **queue** из стандартной библиотеки С++.

**3.**\* Сгенерируйте (используя генератор случайных чисел) матрицу смежности для ориентированного взвешенного графа *G*. Выведите матрицу на экран и осуществите процедуру поиска расстояний, реализованную в соответствии с приведенным выше описанием.

**Задание 2\***

1. Модернизируйте программу так, чтобы получить возможность запуска программы с параметрами командной строки (см. описание ниже). В качестве параметра должны указываться тип графа (взвешенный или нет) и наличие ориентации его ребер (есть ориентация или нет).

Примечание: задания, помеченные символом \* выполняются по указанию преподавателя.

**Дополнительная информация.**

При создании консольного приложения в языке программирования C/С++ возможно задание параметров функции **main**(). Для этого объявляются параметры **argс** (argument count) и **argv**[] (argument values):

int main(int argc, char \*argv[])

  Если запустить такую программу через командную строку, то через эти параметры можно передать ей какую-либо информацию. Параметр **argc** имеет тип данных **int**, и содержит количество параметров, передаваемых в функцию **main**. Причем **argc** всегда не меньше 1, даже если мы не передаем никакой информации в программу – в неё передаются путь и имя программы. Через командную строку в программу передаются данные строкового типа. Поэтому параметр **argv**[] – это массив указателей на строки.

Нулевой параметр **argv**[0] – это путь и имя вызываемой программы. Далее следуют параметры, указываемые в командной строке при вызове программы через пробел. Например, вызов

Program.exe P1 P2 P3

передаст в программу:

**argv**[1] = P1

**argv**[2] = P2

**argv**[3] = P3

Обработка параметров обычно начинается с проверки их количества:

#include "stdafx.h"

#include <iostream>

int main(int argc, char \*argv[])

{

if(argc > 1)

{

// Do something

}

return 0;

}

Если аргументы передаются, то с ними совершаются какие-либо действия:

#include "stdafx.h"

#include <iostream>

int main(int argc, char \*argv[])

{

setlocale(LC\_CTYPE, "rus");

std::cout << "В программе аргументов: " << argc << " \n\n";

if(argc > 1)

{

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

std::cout << i << "-й аргумент: " << argv[i] << '\n';

}

else

{

std::cout << "Укажите аргументы. ";

}

std::cout << '\n';

system("pause");

return 0;

}

Часто в программах используются составные параметры, где первая часть указывает на тип (название) параметра, а вторая на его значение. В этом случае для обозначения типа используется какой-либо заранее заданный символ ("-", "--", "\_" и т.д.).

Например, в программу

#include "stdafx.h"

#include <iostream>

int main(int argc, char \*argv[])

{

setlocale(LC\_CTYPE, "rus");

std::cout << "В программе аргументов: " << argc << " \n\n";

if(argc > 1)

{

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

{

if (strcmp(argv[i], "-name1") == 0)

std::cout << "Аргумент " << argv[i] << " имеет значение: " << argv[i + 1] << '\n';

if (strcmp(argv[i], "-name2") == 0)

std::cout << "Аргумент " << argv[i] << " имеет значение: " << argv[i + 1] << '\n';

}

}

else

{

std::cout << "Укажите аргументы. ";

}

std::cout << '\n';

system("pause");

return 0;

}

можно передавать параметры-name1 и -name2 в любом порядке:

Pogram\_with\_parameters.exe -name1 23 -name2 45

или

Pogram\_with\_parameters.exe -name2 45 -name1 23

на результат работы программы это не повлияет.