Курсовая Работа

Титульный лист

ОГЛАВЛЕНИЕ

[**ОГЛАВЛЕНИЕ**](#_Toc58530395) 3

[**ВВЕДЕНИЕ**](#_Toc58530395) 4

[**1 Анализ задачи и методика программы** 5](#_Toc58530387)

[**1.1 Последовательность решения поставленной задачи** 5](#_Toc58530388)

[**1.2 Частные задачи**](#_Toc58530388) 6

[**2 Методы и алгоритмы, используемые в работе** 7](#_Toc58530389)

[**2.1 Метод Cycle и принцип его работы** 7](#_Toc58530390)

[**2.2 Графическое отображение** 7](#_Toc58530390)

[**3 Внешний вид программы** 11](#_Toc58530391)

[**3.1 Первоначальное окно программы** 11](#_Toc58530392)

[**3.2 Задание рёбер** 1](#_Toc58530393)2

[**3.3 Вывод результатов нахождения минимального остова** 14](#_Toc58530394)

[**ЗАКЛЮЧЕНИЕ** 15](#_Toc58530395)

[**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ** 16](#_Toc58530396)

**ВВЕДЕНИЕ**

Целью данной курсовой работы является: составление на языке программирования C# программы с использованием всех изученных функциональных элементов класса, которая производит поиск минимального остова в неориентированном графе, заданном пользователем. Частными задачами являются: написание программы с возможностью первоначального ввода в нее данных, обработку входных данных и графов с минимальным остовом.

# **1 Анализ задачи и методика программы**

## **1.1 Последовательность решения поставленной задачи**

Общей задачей, поставленной для решения в рамках выполнения курсовой работы, является создание графического приложения для работы в операционных системах Windows, в качестве инструмента разработки которого необходимо использовать Microsoft Visual Studio 2013. В соответствии с предметной задачей, разработкой программы, реализующей поиск минимального остова в неориентированном графе, было решено разделить процесс разработки программы на три этапа:

Первый этап – анализ задачи по поиску минимального остова в неориентированном графе;

Второй этап – написание алгоритмов для поиска и получения суммы весов минимального остова в неориентированном графе;

Третий этап – написания алгоритмов для графического представления полученных результатов.

Второй и третий этапы, ввиду своей логической связности, будут описаны в главе 2 настоящей курсовой работы с разделением на отдельные пункты.

**1.2 Частные задачи**

Определенная заданием на курсовую работу, тематика разрабатываемого программного продукта – написание программы, реализующей поиск минимального остова в неориентированном графе, заданном пользователем.

Таким образом, выделим основные направления разработки программы:

- разработка алгоритма поиска минимального остова в неориентированном графе;

- разработка алгоритмов для графического представления результатов;

# **2 Методы и алгоритмы, используемые в работе**

## **2.1 Метод Cycle и принцип его работы**

В данной работе для поиска минимального остова неориентированного графа был использован Алгоритм Краскала: “В начале текущее множество рёбер устанавливается пустым. Затем, пока это возможно, проводится следующая операция: из всех рёбер, добавление которых к уже имеющемуся множеству не вызовет появление в нём цикла, выбирается ребро минимального веса и добавляется к уже имеющемуся множеству. Когда таких рёбер больше нет, алгоритм завершён. Подграф данного графа, содержащий все его вершины и найденное множество рёбер, является его остовным деревом минимального веса.”. Метод Cycle позволяет проверить наличие в данном наборе ребер зацикливание. На вход метод принимает список ребер и количество вершин.

public static int Cycle(List<string> Dict, int num)

{

List<string> D = Dict;

int count = 0;

string versh = "";

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

{

versh += i.ToString();

}

foreach (char t in versh)

{

int v\_last = 0;

int v\_prev = 0;

int v\_now = Int32.Parse(System.Convert.ToString(t));

string seq;

int z = 0;

int n = 0;

seq = System.Convert.ToString(t);

while (z < 5)

{

int kk = 0;

while (true)

{

n += 1;

kk = seq.Length;

foreach (var item in D)

{

if ((Int32.Parse(item.Split()[0]) == v\_now) && (Int32.Parse(item.Split()[1]) != v\_last) && (Int32.Parse(item.Split()[1]) != v\_prev))

{

seq += item.Split()[1];

v\_now = Int32.Parse(item.Split()[1]);

v\_prev = Int32.Parse(item.Split()[0]);

v\_last = Int32.Parse(item.Split()[1]);

break;

}

if ((Int32.Parse(item.Split()[1]) == v\_now) && (Int32.Parse(item.Split()[0]) != v\_last) && (Int32.Parse(item.Split()[0]) != v\_prev))

{

seq += item.Split()[0];

v\_now = Int32.Parse(item.Split()[0]);

v\_prev = Int32.Parse(item.Split()[1]);

v\_last = Int32.Parse(item.Split()[0]);

break;

}

}

if ((kk == seq.Length) || (n == 20)) break;

}

if ((seq[0] == seq[seq.Length - 1]) && (seq.Length != 1))

{

return 1;

}

if (seq.Length >= 3)

{

v\_last = Int32.Parse(System.Convert.ToString(seq[seq.Length - 1]));

v\_now = Int32.Parse(System.Convert.ToString(seq[seq.Length - 2]));

v\_prev = Int32.Parse(System.Convert.ToString(seq[seq.Length - 3]));

}

else

{

v\_last = Int32.Parse(System.Convert.ToString(seq[seq.Length - 1]));

v\_prev = 0;

v\_now = Int32.Parse(System.Convert.ToString(seq[0]));

}

string seq1 = System.Convert.ToString(t);

for (int s = 1; s < seq.Length - 1; s++)

{

seq1 += seq[s];

}

seq = seq1;

z += 1;

}

}

return 0;

}

**2.2 Графическое отображение**

Для графического отображения на форме графа была использована встроенная библиотека Graphics. Для отображения вершин был применен метод DrawEllipse, для ребер – DrawLine, а для значений вершин и весов – DrawString.

gr.DrawEllipse(Pen, X, Y, A, B);

gr.DrawString(Text, Font, SolidBrush, x, y);

gr.DrawLine(Pen, X1, Y1, X2, Y2);

# **3 Внешний вид программы**

## **3.1 Первоначальное окно программы**

При запуске программы мы видим поле, предлагающее заполнить количество вершин в графе (Рисунок 1).

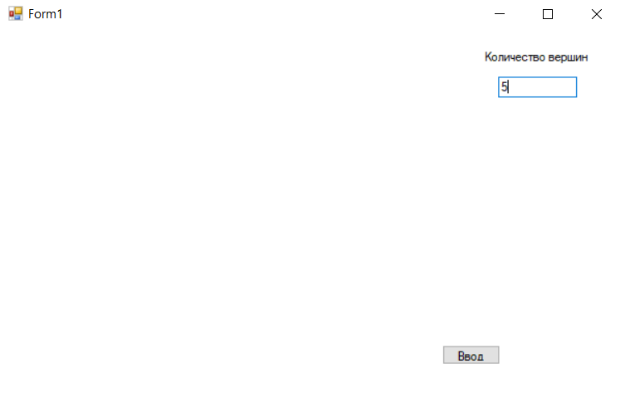


Рисунок 1- Первоначальное окно программы

После нажатия на кнопку, выводятся вершины граф (Рисунок 2).

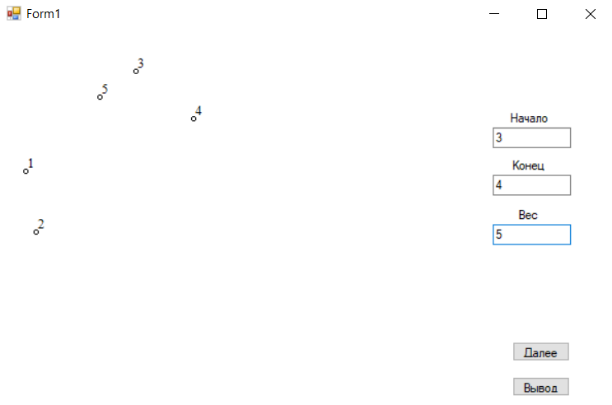


Рисунок 2- Графическое отображение вершин

## **3.2 Задание ребер**

Задание ребер выполняется вводом значений в указанные поля и нажатием кнопки “Далее” (Рисунок 2). После нажатия на кнопку “Далее” ребро будет отображено (Рисунок 3).



Рисунок 3- Задание ребер

После ввода всех ребер графа можно выполнить поиск минимального остова нажатием на кнопку “Вывод”(Рисунок 4).

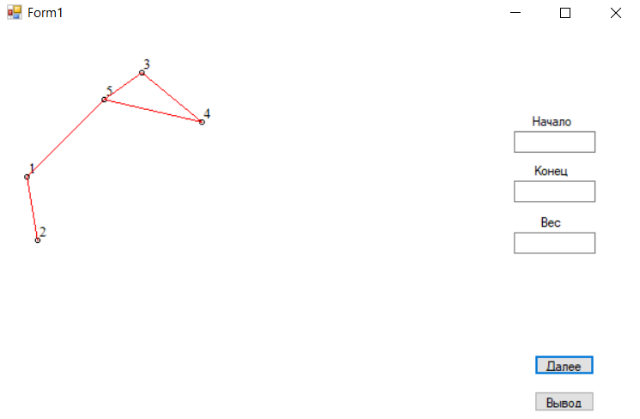


Рисунок 4- Ввод неориентированного графа

## **3.3 Вывод результатов**

Вот так будут выглядеть полученные результаты.

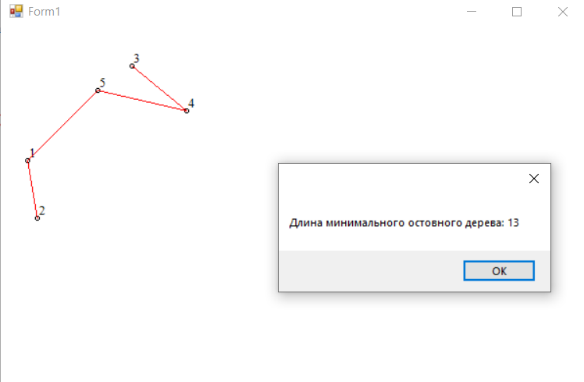


Рисунок 5 – Вывод результатов программы

# **ЗАКЛЮЧЕНИЕ**

Заданием на курсовую работу было определена задача на разработку программного продукта для поиска минимального остова в неориентированном графе.

Разработанная программа ориентирована на рядового пользователя, не обладающего специальными знаниями в области математики. Пользователь может легко решить, поставленную перед ним задачу, укладывающуюся в функционал данной программы. Инструментом разработки выступил пакет Microsoft Visual Studio 2013, представляющий обширные возможности разработки кроссплатформенных приложений [1].

Поставленные на решение курсовой работы задачи и цель были достигнуты. В ходе выполнения задания были продемонстрированы основные структурные элементы классов и умения работы с ними.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Павловская, Т.А. C#. Программирование на языке высокого уровня: учебник / Т.А. Павловская - СПб.: Питер, 2014. — 432 с.