ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ ИМ. ПРОФ. М.А. БОНЧ-БРУЕВИЧА»

(СПбГУТ)

Факультет инфокоммуникационных Сетей и систем (иксс)

кафедра программной инженерии и вычислительной техники (пи и вт)

Дисциплина: «Алгоритмы и структуры данных»

Курсовой проект.

Тема: «Разработка средств для исследования различных алгоритмов на графах»

Выполнил:

Студент группы ИКПИ-05

Принял:

Доцент кафедры ПИиВТ

Молошников Ф.А.

Подпись\_\_\_\_\_\_\_\_\_\_\_\_

Дагаев А.В.

Подпись\_\_\_\_\_\_\_\_\_\_\_\_

«\_\_\_\_\_»\_\_\_\_\_\_\_\_2022

СОДЕРЖАНИЕ

[1. Постановка задачи 4](#_Toc99926638)

[2. Разработка структуры программы 4](#_Toc99926639)

[3. Выбор и обоснование используемых средств 6](#_Toc99926640)

[4. Разработка Модулей (классы) 6](#_Toc99926641)

[4.1. BaseGraph 6](#_Toc99926642)

[4.1.1. vertex 6](#_Toc99926643)

[4.1.2. edge 6](#_Toc99926644)

[4.1.3. Свойства: 6](#_Toc99926645)

[4.1.4. Методы: 6](#_Toc99926646)

[4.2. VisualGraph 7](#_Toc99926647)

[4.2.1. Свойства: 7](#_Toc99926648)

[4.2.2. Методы: 7](#_Toc99926649)

[4.3. Graph 7](#_Toc99926650)

[4.3.1. vertices 7](#_Toc99926651)

[4.3.2. edges 8](#_Toc99926652)

[4.3.3. Свойства: 8](#_Toc99926653)

[4.3.4. Методы: 8](#_Toc99926654)

[4.4. presenthandler 9](#_Toc99926655)

[4.4.1. Свойства: 9](#_Toc99926656)

[4.4.2. Методы: 10](#_Toc99926657)

[4.5. plothandler 10](#_Toc99926658)

[4.5.1. Свойства: 10](#_Toc99926659)

[4.5.2. Методы: 10](#_Toc99926660)

[4.6. StatisticsGraph 10](#_Toc99926661)

[4.6.1. Свойства: 10](#_Toc99926662)

[4.6.2. Методы: 11](#_Toc99926663)

[5. Разработка модулей (функции) 12](#_Toc99926664)

[5.1. Func.h 12](#_Toc99926665)

[5.2. ShortestPathFunc.h 12](#_Toc99926666)

[6. Разработка модулей (скрипты) 13](#_Toc99926667)

[6.1. PlotGraph.bash 13](#_Toc99926668)

[6.2. PlotScript.bash 13](#_Toc99926669)

[7. Разработка модулей (примеры использования) 13](#_Toc99926670)

[7.1. Lab5 main.cpp 13](#_Toc99926671)

[7.2. main\_GenerateGraph\_Demo.cpp 13](#_Toc99926672)

[7.3. main\_Dijkstra\_Demo.cpp 13](#_Toc99926673)

[7.4. main\_BellmanFord\_Demo.cpp 13](#_Toc99926674)

[7.5. main\_VisualGraphDemo.cpp 13](#_Toc99926675)

[7.6. main\_GenerateAdjacency.cpp 13](#_Toc99926676)

[7.7. \_StatisticsGraph\_SimpleDemo.cpp 13](#_Toc99926677)

[8. Инструкция пользователя 13](#_Toc99926678)

[8.1. Подготовка среды (Debian 11) 13](#_Toc99926679)

[8.2. Установка проекта 14](#_Toc99926680)

[8.3. Сборка и запуск 14](#_Toc99926681)

[8.4. Использование Объектов класса Graph 15](#_Toc99926682)

[8.5. Использование объектов класса StatisticsGraph 15](#_Toc99926683)

[8.5.1. Подготовка функции, реализующей алгоритм 15](#_Toc99926684)

[8.5.2. Получение результатов 16](#_Toc99926685)

[9. Примеры использования 18](#_Toc99926686)

[9.1. Возможности визуализации графов 18](#_Toc99926687)

[9.2. Генерация графа с проверкой на связность с помощью обхода в глубину 20](#_Toc99926688)

[9.3. Пошаговая визуализация работы алгоритма Дейкстры. (раскраска вершин) 32](#_Toc99926689)

[10. Описание Программы 42](#_Toc99926690)

[11. Заключение 42](#_Toc99926691)

[12. Список литературы 43](#_Toc99926692)

[13. Приложение 43](#_Toc99926693)

[13.1. CMakeLists.txt 43](#_Toc99926694)

[13.2. BaseGraph.h 43](#_Toc99926695)

[13.3. VisualGraph.h 44](#_Toc99926696)

[13.4. Graph.h 48](#_Toc99926697)

[13.5. presenthandler.h 49](#_Toc99926698)

[13.6. plothandler.h 50](#_Toc99926699)

[13.7. Func.h 51](#_Toc99926700)

[13.8. StatisticsGraph.inl 57](#_Toc99926701)

[13.9. ShortestPathFunc.h 59](#_Toc99926702)

[13.10. PlotScript.bash 65](#_Toc99926703)

[13.11. PlotGraph.bash 67](#_Toc99926704)

[13.12. main\_GenerateGraph\_Demo.cpp 73](#_Toc99926705)

[13.13. main\_StatisticsGraph\_SimpleDemo.cpp 74](#_Toc99926706)

[13.14. main\_VisualGraphDemo.cpp 75](#_Toc99926707)

[13.15. main\_Dijkstra\_Demo.cpp 76](#_Toc99926708)

[13.16. main\_GenerateAdjacency.cpp 77](#_Toc99926709)

[13.17. main\_BellmanFord\_Demo.cpp 78](#_Toc99926710)

[13.18. main.cpp 79](#_Toc99926711)

1. Постановка задачи

Целью работы является разработка средств для исследования времени выполнения различных алгоритмов на графах и наглядного отображения шагов алгоритмов. В результате анализа поставленной цели был выделен следующий список задач:

* 1. Реализовать структуру данных граф в виде класса.
  2. Реализовать визуализацию графа.
  3. Реализовать возможность выбора режима отображения при работе алгоритма над графом.
  4. Реализовать генерацию графа
  5. Реализовать, насколько это возможно, универсальную функцию для сбора информации о времени выполнения алгоритмов над графами.

1. Разработка структуры программы

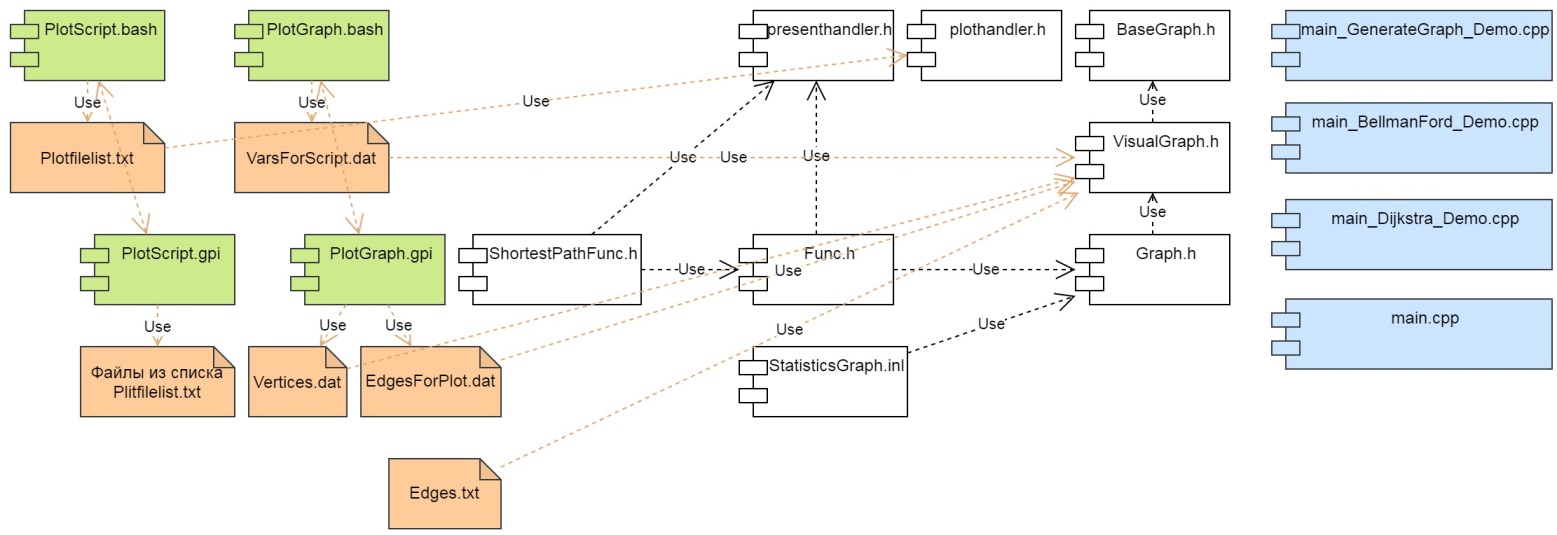


Схема модулей программы

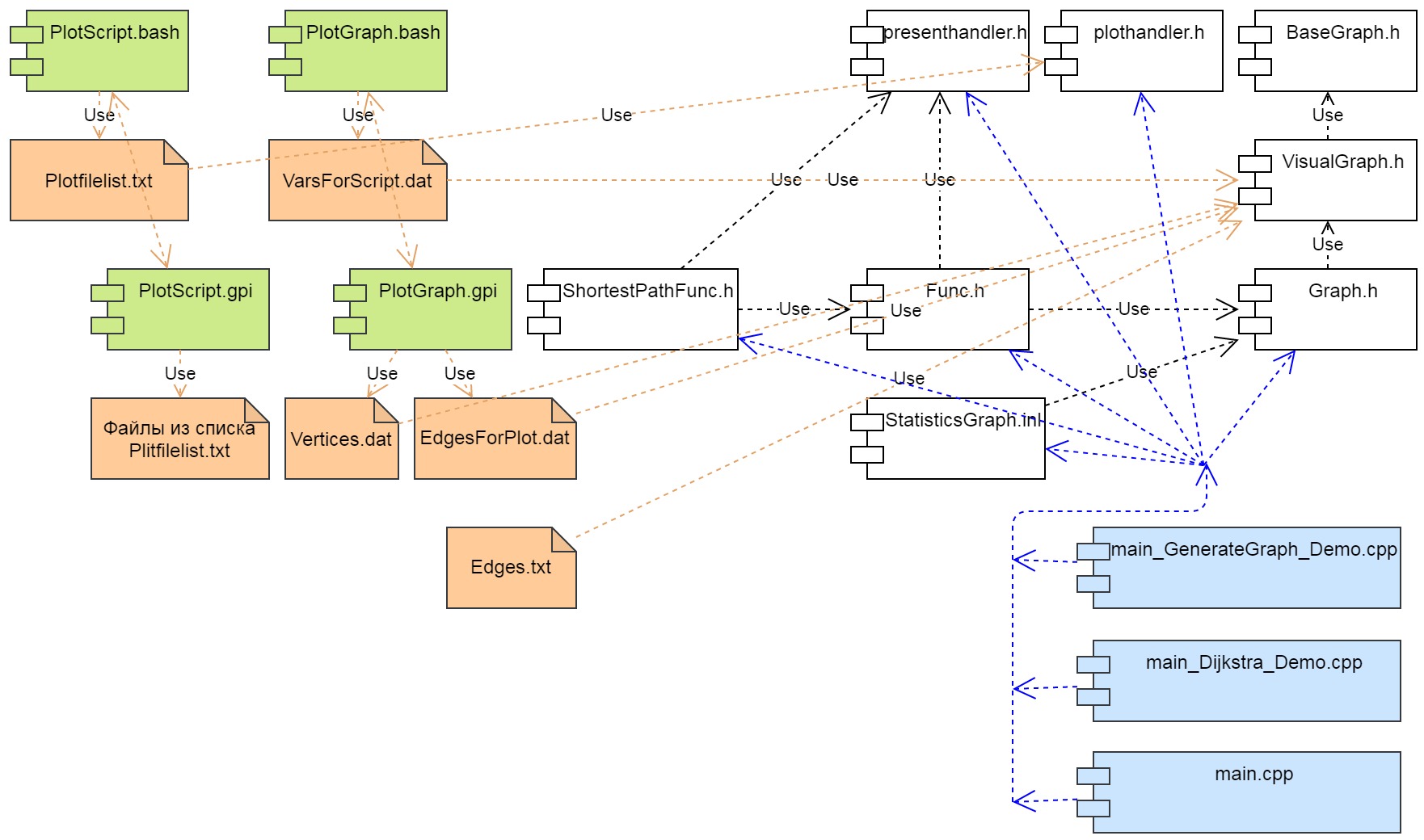


Схема модулей программы

Программа состоит из 3 частей:

1. Основная часть программы. Содержит модули с реализацией графа, функциями для генерации графа и функциями, реализующими некоторые алгоритмы над графами, модуль для сбора информации о времени выполнения алгоритмов. Также к ней относятся 2 модуля, позволяющие задавать режим отображения при работе алгоритмов над графом.
2. Отвечает за визуализацию графа и полученных результатов в виде графиков. Содержит 2 bash скрипта и вспомогательные файлы, генерируемые во время работы программы.
3. Модули, содержащие функцию main. Для демонстрации использования разработанных средств.

Таким образом, программа состоит из следующих модулей:

BaseGraph.h

VisualGraph.h

Graph.h

presenthandler.h

plothandler.h

Func.h

StatisticsGraph.inl

ShortestPathFunc.h

PlotScript.bash

PlotGraph.bash

main\_GenerateGraph\_Demo.cpp

main\_StatisticsGraph\_SimpleDemo.cpp

main\_VisualGraphDemo.cpp

main\_Dijkstra\_Demo.cpp

main\_GenerateAdjacency.cpp

main\_BellmanFord\_Demo.cpp

main.cpp

1. Выбор и обоснование используемых средств

Для вывода графа на экран и для вывода графиков была выбрана программа gnuplot. Данная программа распространяется под свободной лицензией, может выполнять скрипты, читаемые из файлов, кроссплатформенная, дает возможность выбора терминала для отображения данных (qt, x11 и т.д.). Используется в качестве системы вывода изображений в различных математических пакетах: GNU Octave, Maxima, Reduce и других.

Для генерирования и запуска gnuplot-скриптов используются bash-скрипты. Также используется утилита gawk (GNU-реализация языка awk) для операций над текстовыми файлами при работе скриптов. Для преобразования pdf формата в формат png выходных файлов с графами используется программа ImageMagick (Version: 6.9.11-60 Q16 x86\_64 2021-01-25)

1. Разработка Модулей (классы)
   1. BaseGraph

Базовый класс, реализующий структуру данных граф. В данном классе объявленны 2 структуры: vertex и edge. Имеет единственный метод – конструктор, принимающий число вершин в графе.

* + 1. vertex
       1. Свойства:

|  |  |
| --- | --- |
| double x=0; | Координаты вершины на плоскости |
| double y=0; |
| int Weight=0; | Вес вершины |
| int Color=11; | Цвет вершины: 11 белый; 12 зеленый; 13 голубой; 14 красный |

* + 1. edge
       1. Свойства:

|  |  |
| --- | --- |
| int Adjacency=0; | Существование ребра |
| int Weight=0; | Вес ребра |
| int Color=1; | Цвет ребра: 1 черный; 2 зеленый; 3 голубой; 4 красный |

* + 1. Свойства:

|  |  |
| --- | --- |
| std::vector<vertex> Vertices; | Массив вершин |
| std::vector<std::vector<edge>> Edges; | Матрица ребер |
| bool IsOriented=false; | Ориентированность графа |
| bool IsWithVerticesWeights=false; | Имеет ли граф веса на вершинах |

* + 1. Методы:

|  |  |
| --- | --- |
| BaseGraph(int N, bool NIsOriented=false) | Создает граф с N вершинами. |

* 1. VisualGraph
     1. Свойства:

|  |  |
| --- | --- |
| #define RADIUS 40 | Макроимя, используется для генерации положения вершин графа на плоскости. В данном случае, определяет внешний радиус звезды. |

* + 1. Методы:

|  |  |
| --- | --- |
| VisualGraph(int N, bool NIsOriented=false) | Конструктор |
| void SetVarsForScript(bool IsSavePictureToFile, std::string PictureName, const char\* filename) | Записывает информацию о графе(является ли он ориентированным, нужно ли сохранять изображение в файл и т.д.) в файл, который будет прочтен PlotGraph.bash |
| void PrintVertices(const char\* filename="std::cout") | Вывод информации о вершинах в терминал или в файл. |
| void SetVertXYForPlot(int CenterX=0, int CenterY=0, const char\* fname="Vertices.dat") | Генерирует координаты вершин графа. В данном случае, вершины будут расположены в виде звезды. |
| void PrintEdges(const char\* filename="std::cout") | Выводит информацию о ребрах в терминал или в файл. Нужен только для наглядного отображения информации о ребрах. При визуализации графа не используется. |
| void SetEdgesForPlot(const char\* fname="EdgesForPlot.dat") | Записывает информацию о ребрах в виде, пригодном для дальнейшей работы с gnuplot, в файл. |
| void ShowPlot(bool IsSavePictureToFile=false, std::string PictureName="pic", bool IsSetVertXYForPlot=true) | Выполняет все необходимые действия для визуализации графа. |

* 1. Graph

Класс, задача которого заключается в реализации некоторых базовых операции над графом. В связи с этим в классе объявлены 2 класса: vertices и edges. Эти классы получают доступ к массиву вершин и матрице ребер базового класса BaseGraph с помощью соответствующих ссылок, инициализируемых при создании объектов этих классов. В настоящий момент класс Graph не имеет методов, кроме конструктора, а классы vertices и edges имеют методы для сброса цветов вершин и ребер. При развитии проекта в него могли бы быть включены разнообразные методы для удобства обращения с графом. Например: подсчет числа ребер графа, установка всем вершинам(ребрам) максимальных весов, установка матрицы смежности на основе матрицы весов и тому подобное.

* + 1. vertices
       1. Свойства:

|  |  |
| --- | --- |
| std::vector<vertex>& refVertices; | Ссылка на массив вершин. |

* + - 1. Методы:

|  |  |
| --- | --- |
| vertices(std::vector<vertex>& NrefVertices) | Конструктор с параметрами. Инициализирует ссылку переданным ему параметром. |
| void ResetColors() | Сбрасывает цвета вершин (белый цвет) |

* + 1. edges
       1. Свойства:

|  |  |
| --- | --- |
| std::vector<std::vector<edge>>& refEdges; | Ссылка на матрицу ребер |

* + - 1. Методы:

|  |  |
| --- | --- |
| edges(std::vector<std::vector<edge>>& NrefEdges) | Конструктор с параметрами. Инициализирует ссылку переданным ему параметром. |
| void ResetColors() | Сбрасывает цвета ребер(черный цвет) |

* + 1. Свойства:

|  |  |
| --- | --- |
| vertices vertsVertices; | Объекты, оперирующие с вершинами и ребрами графа. |
| edges edgesEdges; |

* + 1. Методы:

|  |  |
| --- | --- |
| Graph(int N, bool NIsOriented=false) | Конструктор, принимающий в качестве параметра число вершин в графе. После создания объект описывает нуль-граф (т.е. граф без ребер). |

Диаграмма классов представлена ниже.

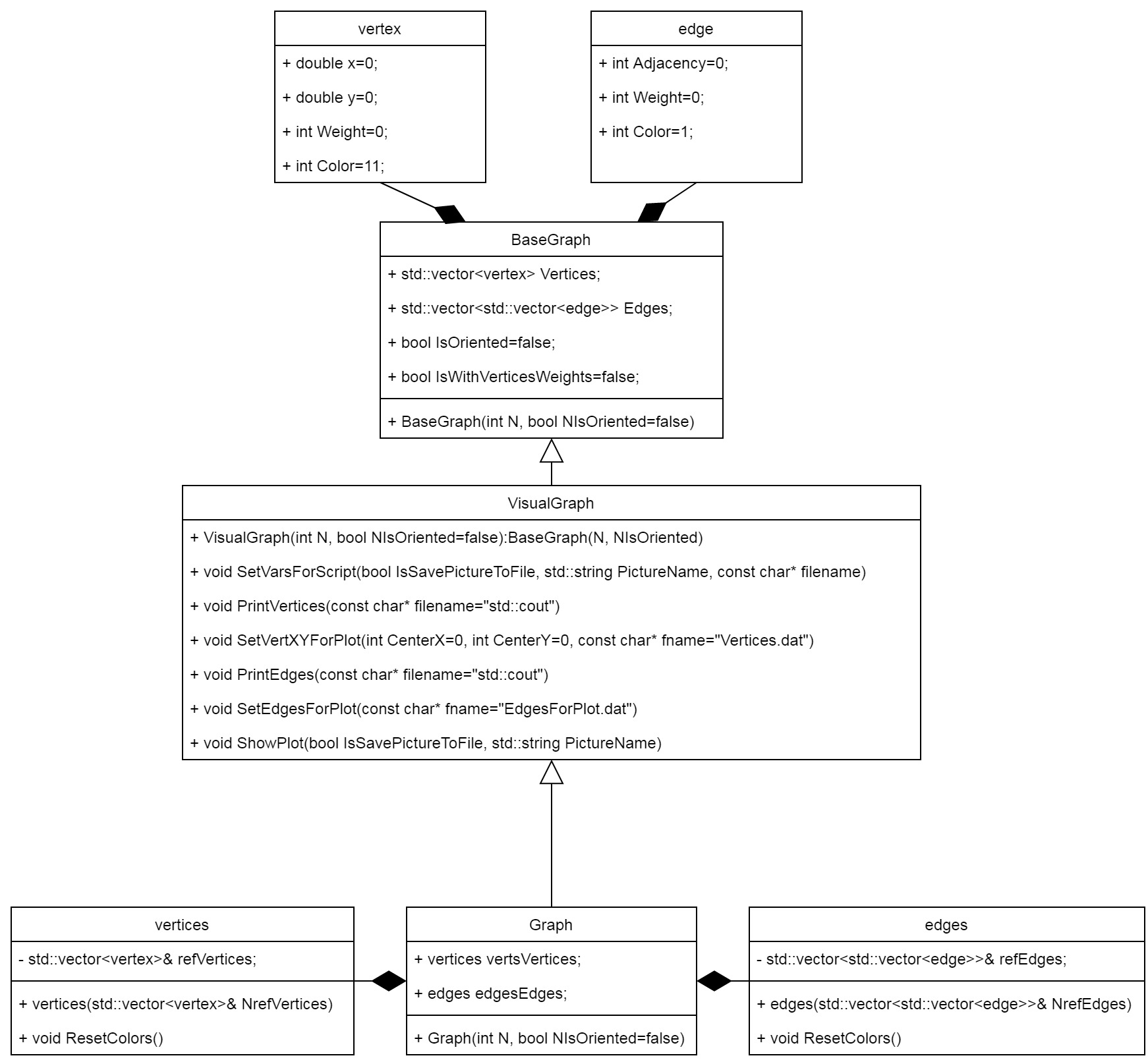


Диаграмма классов

* 1. presenthandler

Класс, отвечающий за режимы отображения при работе различных алгоритмов.

* + 1. Свойства:

|  |  |
| --- | --- |
| int CurrentFileNumber=0; | Счетчик числа созданных изображений. |
| int Mode=0; | Режим отображения. 0-вывод минимальной информации при работе алгоритма; 1-подробный пошаговый вывод при работе алгоритма(визуализация графа в отдельных окнах); 2-подробный пошаговый вывод при работе алгоритма с сохранением различных состояний графа в отдельные изображения. |

* + 1. Методы:

|  |  |
| --- | --- |
| std::string GetFileNumberAndIncrease() | Увеличивает счетчик числа созданных изображений и возвращает его значение. |
| void ResetFileNumber() | Сбрасывает счетчик числа созданных изображений (полезно, если один и тот же экземпляр используется там, где нужно создавать серии изображений с разными именами, т.е. где не нужна сквозная нумерация) |

* 1. plothandler

Класс, облегчающий построение графиков: его экземпляр запоминает имена файлов, в которых будут записаны результаты исследования алгоритмов, а затем создает файл, в который включает все запомненные имена файлов с данными. За счет этого пропадает необходимость редактирования скрипта gnuplot вручную после добавления или исключения графиков, которые нужно вывести на экран.

* + 1. Свойства:

|  |  |
| --- | --- |
| std::vector<std::string> v; | Массив с именами файлов с данными |

* + 1. Методы:

|  |  |
| --- | --- |
| std::string addandreturn(std::string filename) | Добавляет принимаемую строку в массив и возвращает её |
| void tofile(std::string filename="Plotfilelist.txt") | Записывает массив с именами файлов в отдельный файл |

* 1. StatisticsGraph

Класс для сбора информации о времени выполнения различных алгоритмов над графом. Основным его методом является его конструктор с параметрами, представляющий собой шаблонный метод с переменным количеством параметров, принимающий в качестве аргументов в том числе указатель на функцию исследуемого алгоритма и указатель на функцию генерации графа. На функцию исследуемого алгоритма не накладывается никаких ограничений кроме тех, что её первый аргумент должен иметь тип Graph&, а все последующие аргументы должны передаваться в эту функцию по ссылке.

Функция для генерации графа должна принимать аргументы следующих типов:

Graph&, int, int, std::default\_random\_engine&, presenthandler&

(Чтобы принимать функции с другим набором аргументов можно поступить так же, как и с функцией, реализующей исследуемый алгоритм над графом)

* + 1. Свойства:

|  |  |
| --- | --- |
| int CurrentSize=0; |  |
| int CurrentM=0; |  |
| int NStart=0, NEnd=0, NStep=0; |  |
| double MRatio=0; |  |
| int WeightMax=0; |  |
| int NumberOfRuns=1; |  |

* + 1. Методы:

|  |  |
| --- | --- |
| template<typename... CallBackParamsTail>  StatisticsGraph(int NNStart, int NNEnd, int NNStep, double NMRatio, int NWeightMax, int NNumberOfRuns, std::default\_random\_engine& generator, presenthandler& PresentHandler, std::string filename, void (\*CallBackGenerate)(Graph&, int, int, std::default\_random\_engine&, presenthandler&), void(\*CallBackFind)(Graph&, CallBackParamsTail&...), void(\*CallBackTailHandler)(int N, CallBackParamsTail&...), CallBackParamsTail& ...callbackparamstail) | * Конструктор с параметрами. * NNStart NNEnd NNStep – начальное, конечное число вершин в графе, шаг изменения числа вершин. * NMRatio – отношение m/m\_max, где m- число ребер, которое должно генерироваться для графа, m\_max – число ребер, соответствующее полносвязному графу. * NWeightMax – максимальный вес ребра при случайной генерации весов ребер. * NNumberOfRuns – число прогонов для данного числа вершин в графе. * generator – генератор случайных чисел. * PresentHandler – объект для управления режимами отображения при выполнении алгоритмов * filename – имя файла, куда будет записан результат: время работы алгоритма при данном числе вершин в графе. (также в файл попадет информация о filename, MRatio, WeightMax, NumberOfRuns) * CallBackGenerate – указатель на функцию, генерирующую граф * CallBackFind – указатель на функцию, представляющую собой исследуемый алгоритм. * CallBackTailHandler – Эта функция, которая будет вызываться каждый раз перед запуском CallBackFind. Её задача – подготовка аргументов, принимаемых функцией CallBackFind, если это необходимо. Для каждой функции, реализующей алгоритм, необходимо написать соответствующую функцию CallBackTailHandler. Её тело можно оставлять пустым. * Параметры шаблона определяются автоматически. |
| void printLabel(std::string filename) | Записывает в файл о filename, MRatio, WeightMax, NumberOfRuns |
| void printValue(int CurrentSize, double value, std::string filename) | Записывает в файл число вершин в графе и время выполнения алгоритма над графом |

1. Разработка модулей (функции)
   1. Func.h

Модуль, содержащий функции для генерации и проверки на связность графа. Для генерации случайных чисел используется заголовочный файл <random> из стандартной библиотеки C++. Все функции, связанные с генерацией случайных чисел принимают в качестве аргумента по ссылке объект типа std::default\_random\_engine. Это позволяет легко управлять последовательностью псевдослучайных чисел.

|  |  |
| --- | --- |
| void GenerateAdjacencyProb(Graph& Graph1, double probability, std::default\_random\_engine& generator, bool IsWithLoops) | Модель Эрдеша-Реньи. Будем соединять  любые две вершины i и j ребром с некоторой вероятностью p∈[0,1] независимо от всех остальных пар вершин. |
| void GenerateAdjacencyMNumberWithLoops(Graph& Graph1, int m, std::default\_random\_engine& generator) | Генерирование случайного графа с заданным числом ребер. (т.е. фиксированное количество ребер распределяется по возможным положениям в матрице смежности). Принцип – классическое определение вероятности. (Т.е. 1-ое ребро будет сгенерировано с вероятностью m/n второе – с вероятностью либо (m-1)/(n-1) либо m/(n-1) и т.д.) |
| void GenerateAdjacencyMNumberNoLoops(Graph& Graph1, int m, std::default\_random\_engine& generator) |
| void GenerateWeights(Graph& Graph1, std::default\_random\_engine& generator, int WeightMin, int WeightMax) | Генерирует веса ребер случайным образом. От минимального до максимального указанных значений |
| bool IsConnectedDFS(Graph& Graph1, int StartVert, presenthandler& PresentHandler); | Проверяет граф на связность. |
| void GenerateGraph(Graph& Graph1, int m, int WeightMax, std::default\_random\_engine& generator, presenthandler& PresentHandler) | Генерация случайного связного графа. Генерируем случайные графы до тех пор, пока один из них не окажется связным. |
| int DFS(Graph& Graph1, int StartVert, presenthandler& PresentHandler) | Алгоритм обхода в глубину. Рекурсивный. Возвращает число вершин, которые удалось посетить. |
| bool IsConnectedDFS(Graph& Graph1, int StartVert, presenthandler& PresentHandler) | Проверяет связность графа с помощью обхода в глубину. |
| template<typename T>  void PrintMatrix(std::vector<std::vector<T>>& Matrix, const char\* MatrixName) | Шаблонная функция для вывода матрицы в терминал. |

* 1. ShortestPathFunc.h

Состоит из функций, реализующих алгоритмы поиска кратчайших путей в графе

|  |  |
| --- | --- |
| void FloydWarshall(Graph& Graph1, std::vector<std::vector<int>>& MPathLength, std::vector<std::vector<int>>& MPath) | Алгоритм Флойда-Уоршелла |
| void FloydWarshallCallBackParamsTailHandler(int N, std::vector<std::vector<int>>& MPathLength, std::vector<std::vector<int>>& MPath) |
| void Dijkstra(Graph& Graph1, int& SourceIndex, std::vector<std::vector<int>>& Pathes, presenthandler& PresentHandler) | Алгоритм Дейкстры |
| void DijkstraCallBackParamsTailHandler(int N, int& SourceIndex, std::vector<std::vector<int>>& Pathes, presenthandler& PresentHandler) |
| void BellmanFord(Graph& Graph1, int& s, std::vector<int>& x, std::vector<std::vector<int>>& D, presenthandler& PresentHandler) | Алгоритм Беллмана-Форда |
| void BellmanFordCallBackParamsTailHandler(int N, int& s, std::vector<int>& x, std::vector<std::vector<int>>& D, presenthandler& PresentHandler) |

1. Разработка модулей (скрипты)
   1. PlotGraph.bash

Скрипт, генерирующий на основе данных файла VarsForScript.dat скрипт для программы gnuplot и запускающий его. Необходимость создания отдельного скрипта bash объясняется недостаточной поддержкой работы с файлами скриптовым языком gnuplot. Для конвертации в формат png полученных векторных изображений скрипт запускает программу ImageMagick.

* 1. PlotScript.bash

Скрипт, генерирующий на основе данных Plotfilelist.txt файла скрипт gnuplot и запускающий его.

1. Разработка модулей (примеры использования)
   1. Lab5 main.cpp

Исследование алгоритмов ФлойдУоршелл Дейкстра БеллманФорд

* 1. main\_GenerateGraph\_Demo.cpp

Демонстрация проверки связности. Поиск в глубину.

* 1. main\_Dijkstra\_Demo.cpp

Демонстрация работы алгоритма Дейкстры (раскраска вершин)

* 1. main\_BellmanFord\_Demo.cpp

Демонстрация работы алгоритма Беллмана-Форда.

* 1. main\_VisualGraphDemo.cpp

Вывод графа в виде слова GRAPH.

* 1. main\_GenerateAdjacency.cpp

Проверка генерации графа

* 1. \_StatisticsGraph\_SimpleDemo.cpp

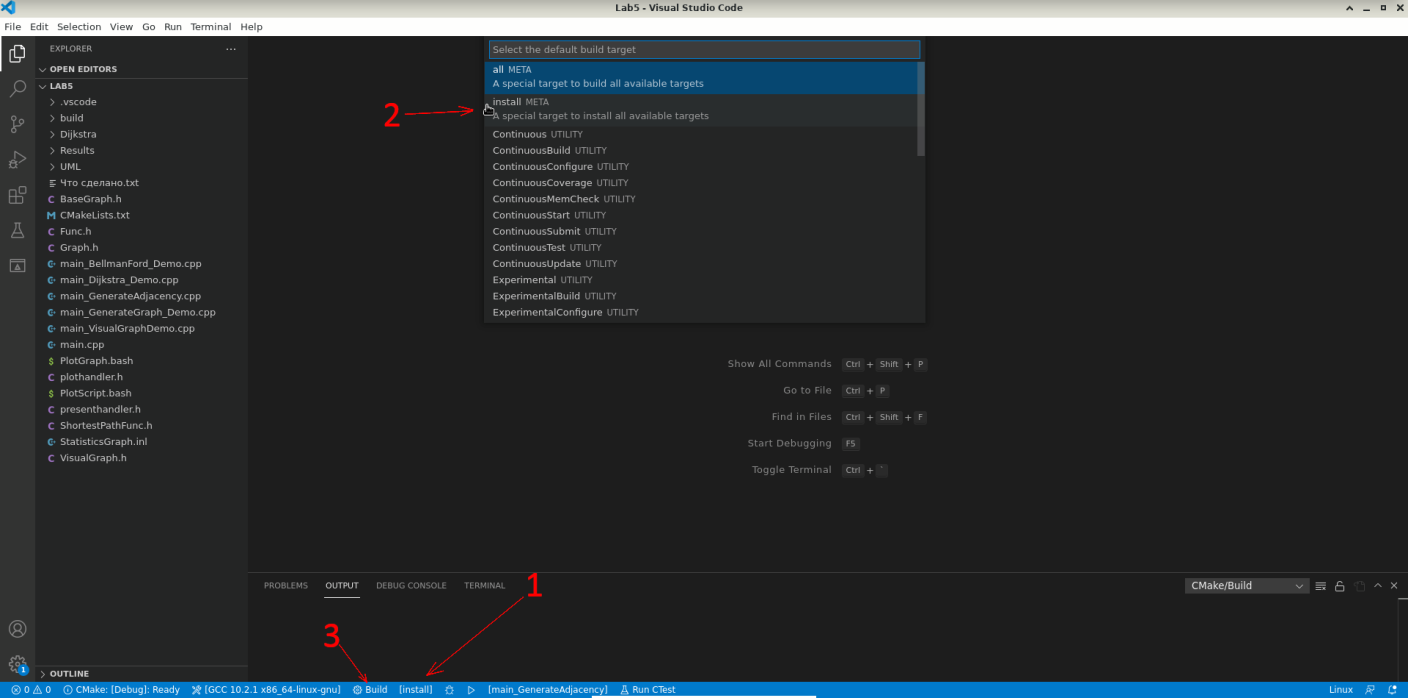
Демонстрация использования StatisticsGraph.

1. Инструкция пользователя

Для начала работы с проектом может быть предложен следующий набор действий:

* 1. Подготовка среды (Debian 11)

1. Установить среду разработки : VS Code
2. Установить расширения для VS Code:
   1. C/C++
   2. C/C++ Include Guard
   3. CMake
   4. CMake Tools
3. Установить gnuplot
4. Установить gawk
5. Установить ImageMagick
   1. Установка проекта
6. Порядок действий показан на рисунке ниже: необходимо выбрать цель *install* и нажать *Build*. После этого в папку build установятся скрипты. **После каждого редактирования скриптов в папке исходного кода необходимо выполнять запуск цели *install* !**

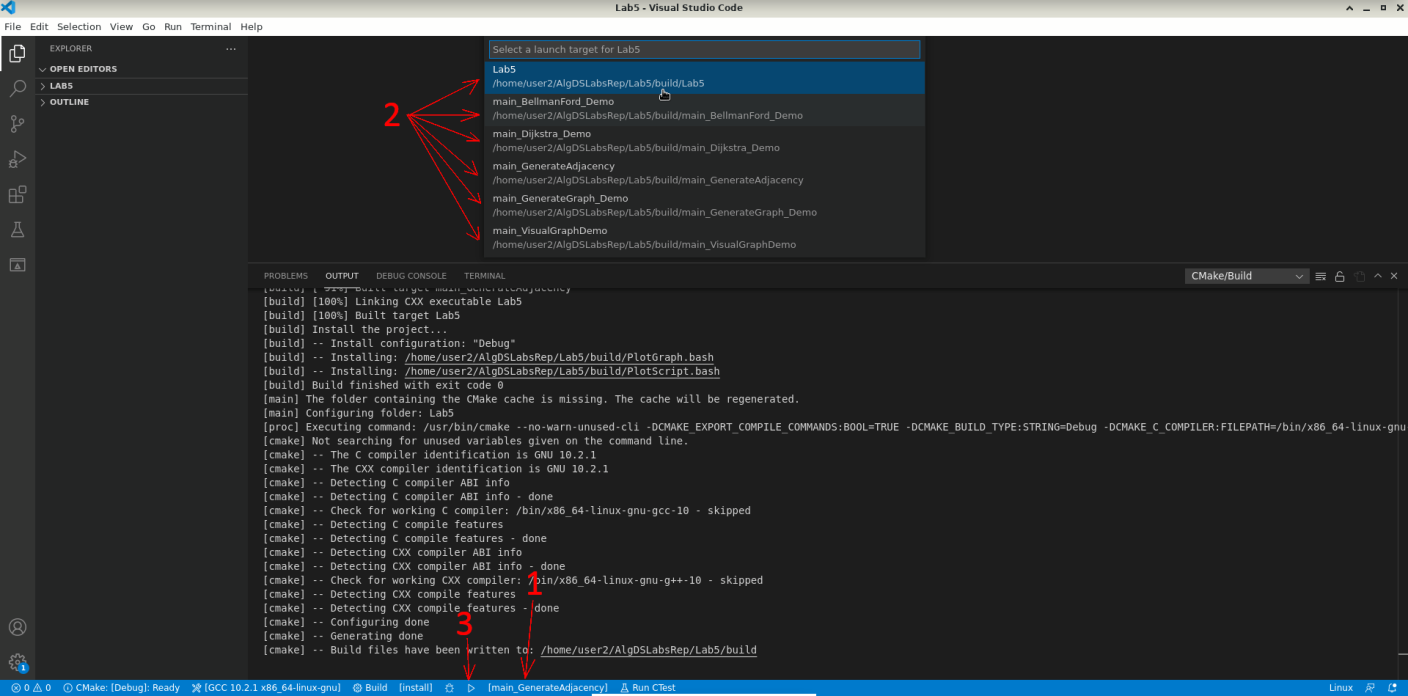


Установка скриптов

* 1. Сборка и запуск

Для установки, сборки и запуска проекта используется система сборки CMake.

1. Выбор цели осуществляется с помощью панели расширения CMake Tools внизу окна:



Сборка и запуск

1. Создание и изменение существующих целей осуществляется с помощью редактирования файла CMakeLists.txt
   1. Использование объектов класса Graph

Подробное описание класса приведено выше. Приведем здесь возможности, важные для его практического использования.

Создание нуль-графа(т.е. графа без ребер) с N вершинами (по умолчанию неориентированный без отображения весов вершин):

Graph Graph1(N);

Доступ к контейнеру (тип vector из стандартной библиотеки шаблонов STL) вершин:

Graph1.Vertices

Например, зададим i-ой вершине красный цвет:

Graph1.Vertices.at(i).Color=14;

Узнаем число вершин в графе:

Graph1.Vertices.size();

Доступ к контейнеру (тип vector<vector> из стандартной библиотеки шаблонов STL) ребер:

Graph1.Edges

Например, соединим ребром i-ую и j-ую вершины:

Graph1.Edges.at(i).at(j).Adjacency=true;

Сбросим цвета всех ребер (аналогично сбрасываются цвета вершин):

Graph1.edgesEdges.ResetColors();

Выведем граф на экран:

Graph1.ShowPlot();

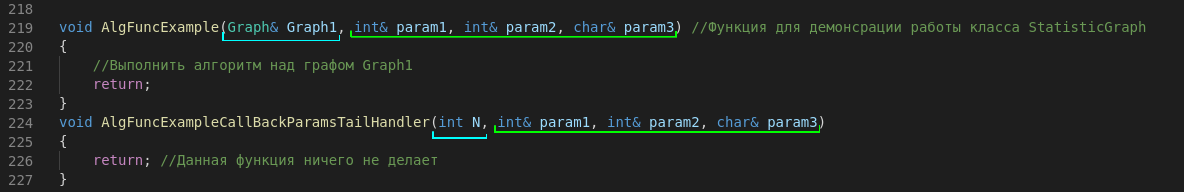
* 1. Использование объектов класса StatisticsGraph

Для исследования алгоритма с помощью класса StatisticsGraph необходимо:

* + 1. Подготовка функции, реализующей алгоритм

Пример (рисунок 6):

Если функция1 (в примере AlgFuncExample) с исследуемым алгоритмом помимо параметра *Graph& Graph1* имеет параметры *param1, param2, param3*, то функция2 (в примере AlgFuncExampleCallBackParamsTailHandler) должна помимо параметра *int N* (числа вершин в графе)иметь параметры *param1, param2, param3*.



Пример подготовки функции с исследуемым алгоритмом. Голубым цветом обозначены параметры, присутствие которых в объявлении функции обязательно. Зеленым – опционально.

Пояснение:

1) Оформить алгоритм в виде функции, первым параметром которой является ссылка на объект класса Graph (т.е. Graph& ) (назовем её здесь функция1). Число остальных параметров может быть любым. Их тип также может быть любым. Важно лишь, чтобы все они передавались по ссылке. Тип возвращаемого значения: void.

2) Написать функцию (назовем её здесь функция2), которая будет подготавливать параметры, которые принимает функция1. При работе метода StatisticsGraph(…) Она будет вызываться каждый раз перед запуском функции1. Первым её параметром должно быть число вершин в графе (int N). Остальные её параметры должны совпадать с параметрами функции1 (кроме первого). Если в такой функции нет необходимости, то её нужно написать и тело оставить пустым.

* + 1. Получение результатов

После того, как функции подготовлены для передачи в StatisticsGraph. Можно перейти к написанию вызова конструктора объекта этого класса, в результате работы которого можно получить готовый результат (т.е. табуляцию времени работы функции1 от числа вершин в графе)

Пример:

#include <iostream>

#include "Graph.h"

#include "StatisticsGraph.inl"

#include "ShortestPathFunc.h"

#include "presenthandler.h"

#include "plothandler.h"

#include "Func.h" //GenerateGraph();

int main(int, char\*\*) {

Graph Graph1(5);

std::default\_random\_engine generator1;

presenthandler PresentHandler0;

plothandler PlotHandler;

int param1;

int param2;

char param3;

StatisticsGraph(5, 10, 1, 0.5, 100, 1, generator1, PresentHandler0, PlotHandler.addandreturn("AlgFuncExampleOut.txt"), GenerateGraph, AlgFuncExample, AlgFuncExampleCallBackParamsTailHandler, param1, param2, param3);

return 0;

}

Пояснение:

Параметры(слева направо):

1. Начальное число вершин в графе
2. Конечное число вершин в графе
3. Шаг изменения числа вершин
4. Параметр, связанный с генерацией графа: m/M, где m – число вершин, которое будет сгенерировано для графа, M – число вершин, маскимальное для графа с таким числов вершин. (Т.е. когда параметр равен 0, будет попытка сгенеровать нуль-граф, когда 1 – будут генерироваться полносвязные графы. Например, если поставить значение этого параметра 0.5 для графа, у которого 4 вершины, то будет генерироваться граф с 3 ребрами (0.5\*M=0.5\*6=3))
5. Максимальный вес ребра (В данном примере веса ребер будут от 0 до 100)
6. Число прогонов для каждого значения числа вершин графа (Чем больше число прогонов, тем более усредненными будут результаты, и тем плавнее будут выглядеть графики) В данном примере 1. Т.е. для каждого числа вершин алгоритм будет выполнен всего 1 раз.
7. Генератор случайных чисел. Передается по ссылке. Если мы хотим создать несколько объектов *StatisticsGraph* в каждом из которых исследуются разные алгоритмы, но при этом хотим, чтобы они работали над одинаковыми последовательностями графов, нужно поступить следующим образом: Создать несколько объектов типа *std::default\_random\_engine* иинициализировать их одним и тем же зерном. В каждый объект *StatisticsGraph* передавать свой генератор случайных чисел.

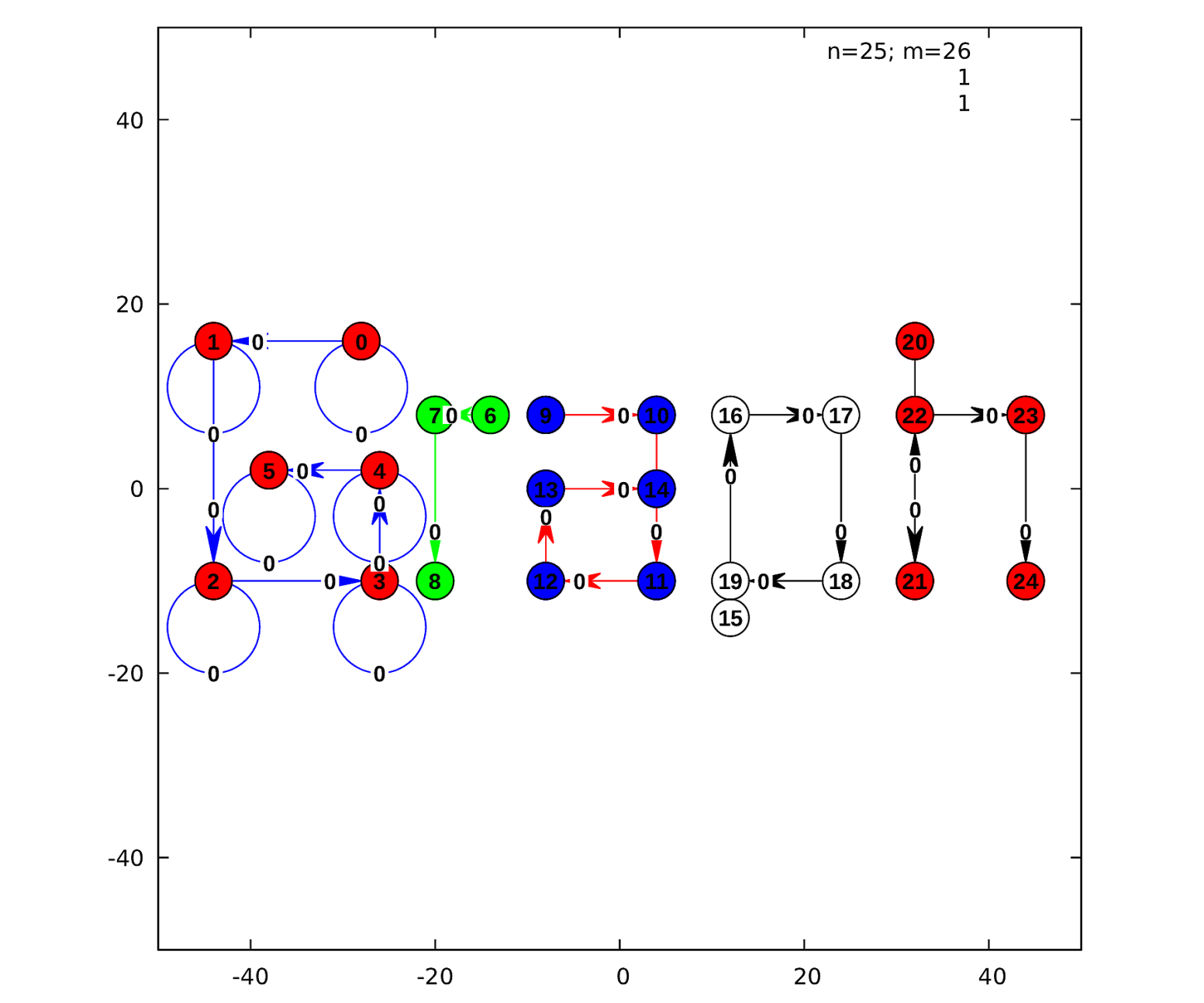
Пример:

// int seed=time(0);

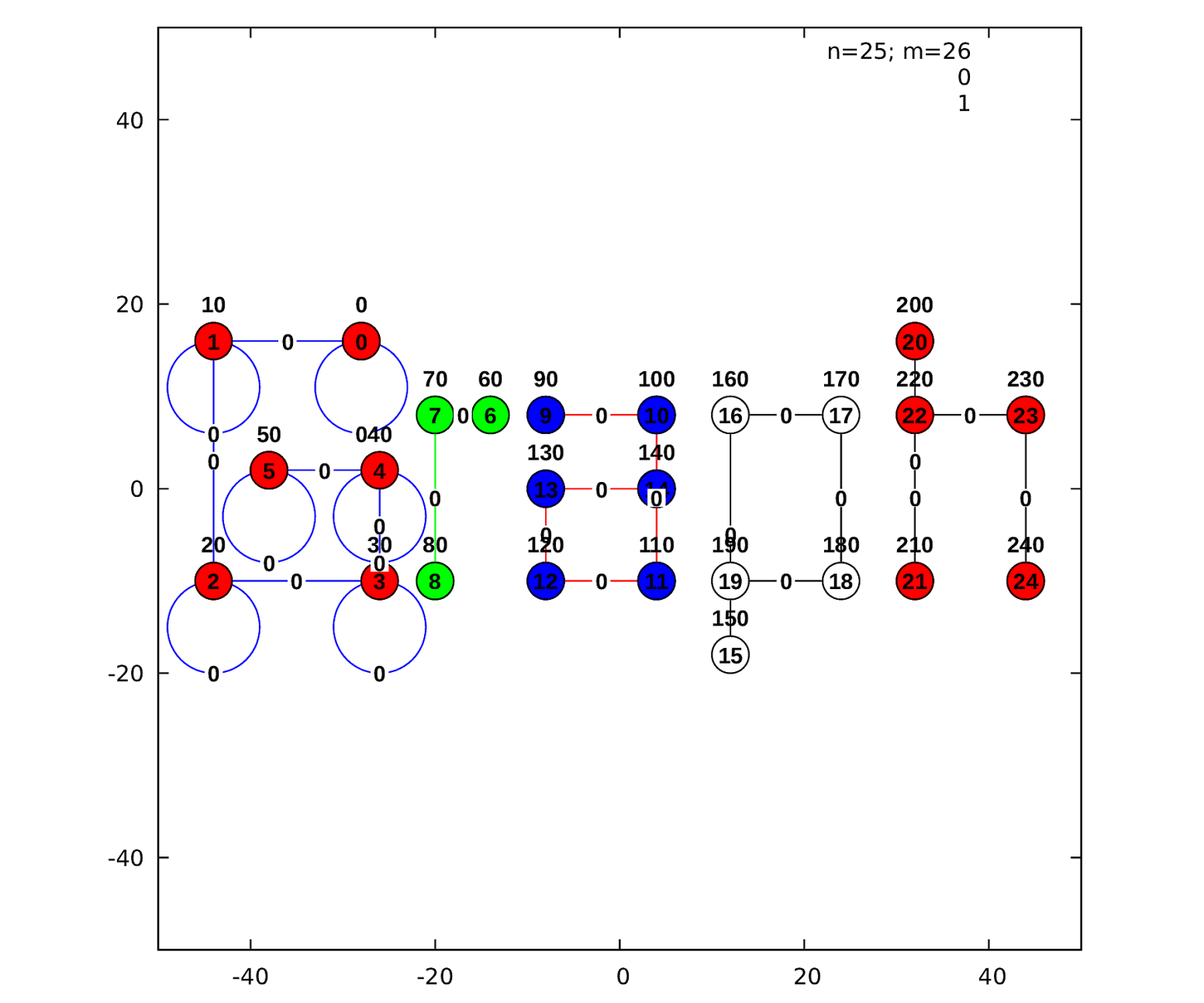
int seed=4; //с seed 4 получается 3 итерации в Adjacency при n=5 m=4

std::default\_random\_engine generator1(seed), generator2(seed), generator3(seed); //будут генерировать одинаковые последовательности.

1. Объект, позволяющий задать режим отображения промежуточных шагов при генерации графа.
2. Установка имени файла, куда будут помещены результаты исследования алгоритма, и добавление его в список файлов, которые будут использоваться при построении графиков.
3. Указатель функции генерации графа.
4. Указатель на функцию, реализующую алгоритм
5. Указатель на вспомогательную функцию, которая должна (при необходимости) подготавливать параметры для функции, реализующей алгоритм
6. Параметры функции, реализующей алгоритм (передаются по ссылке)
7. Примеры использования
   1. Возможности визуализации графов

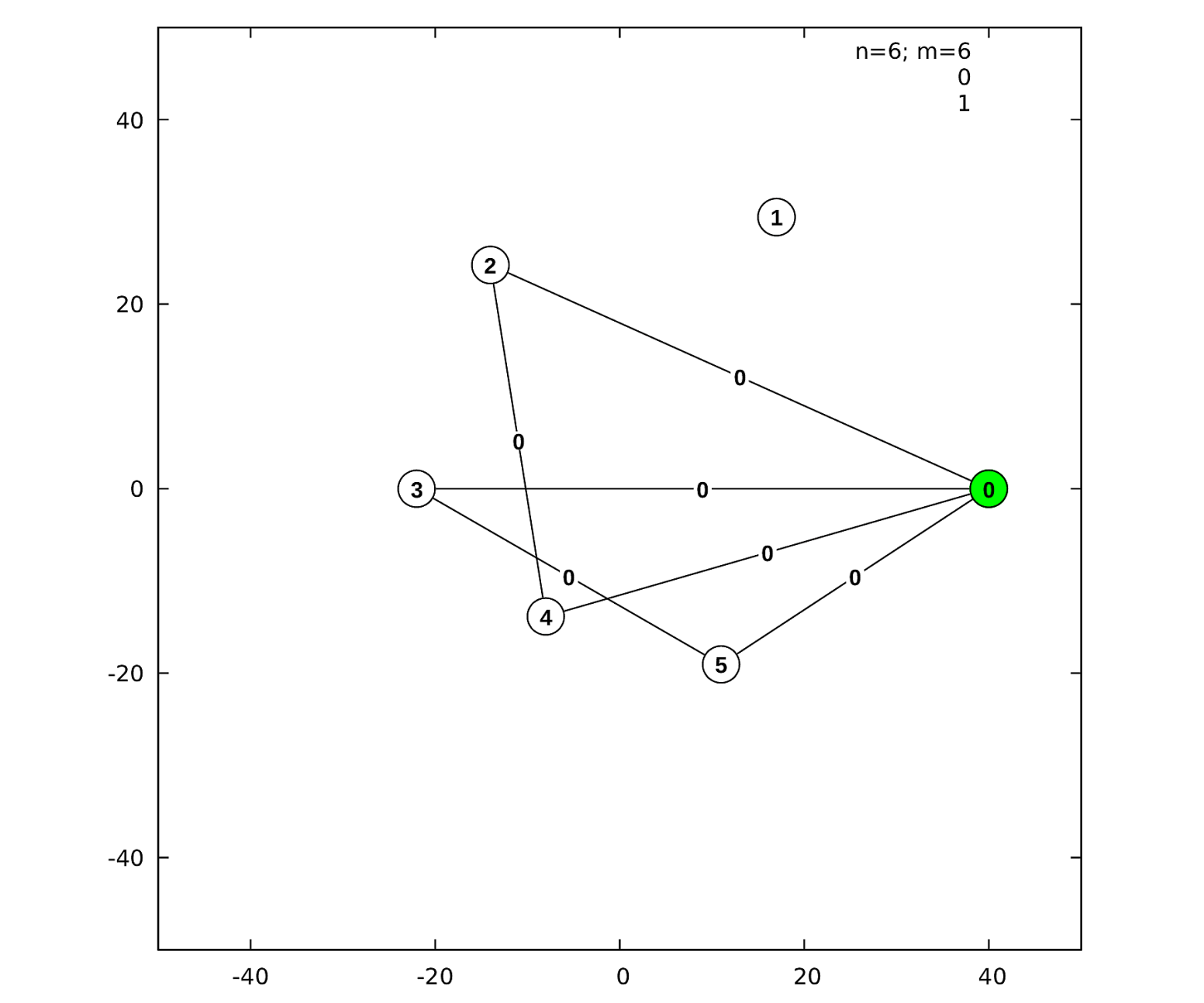


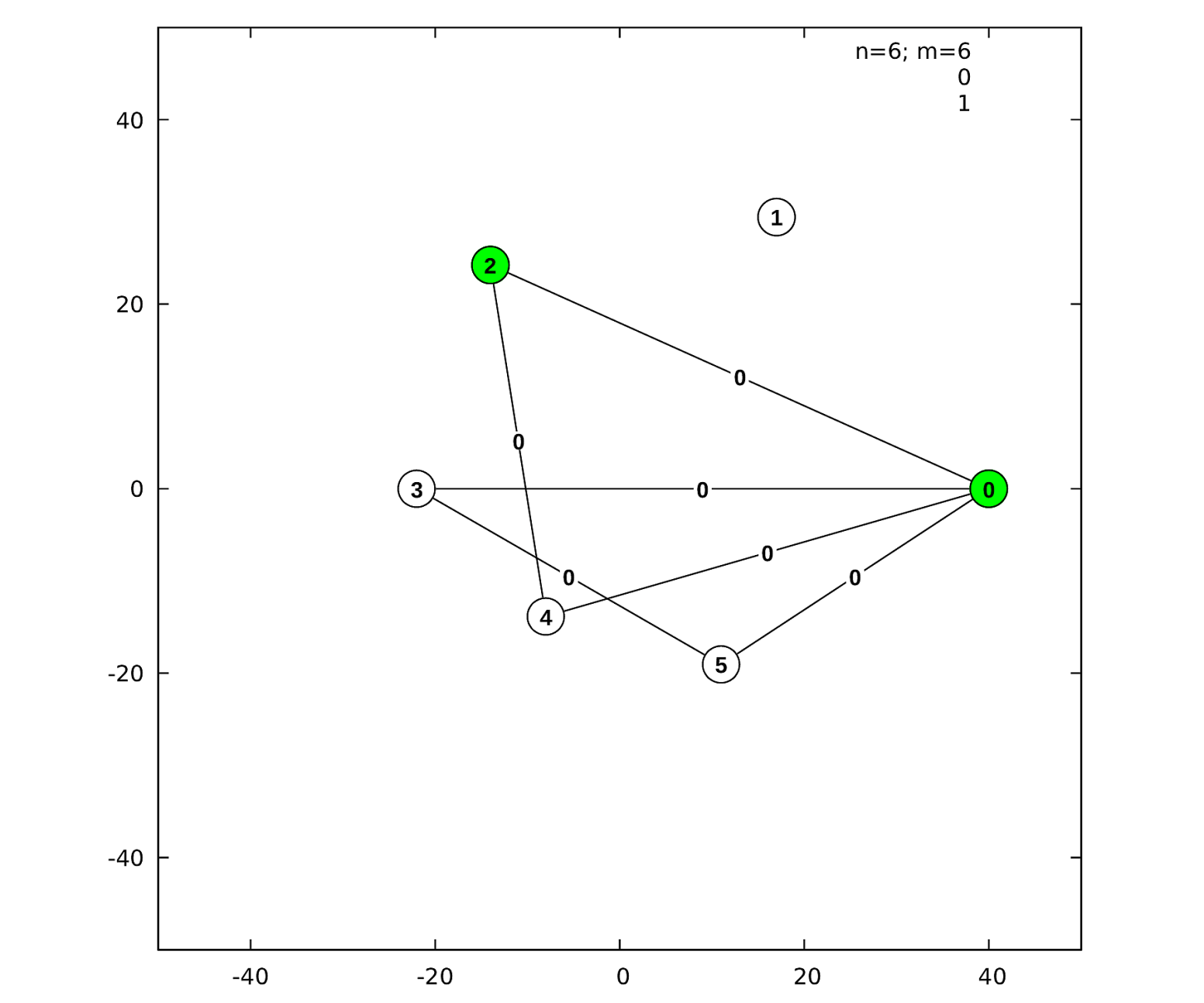
Ориентированный граф с петлями

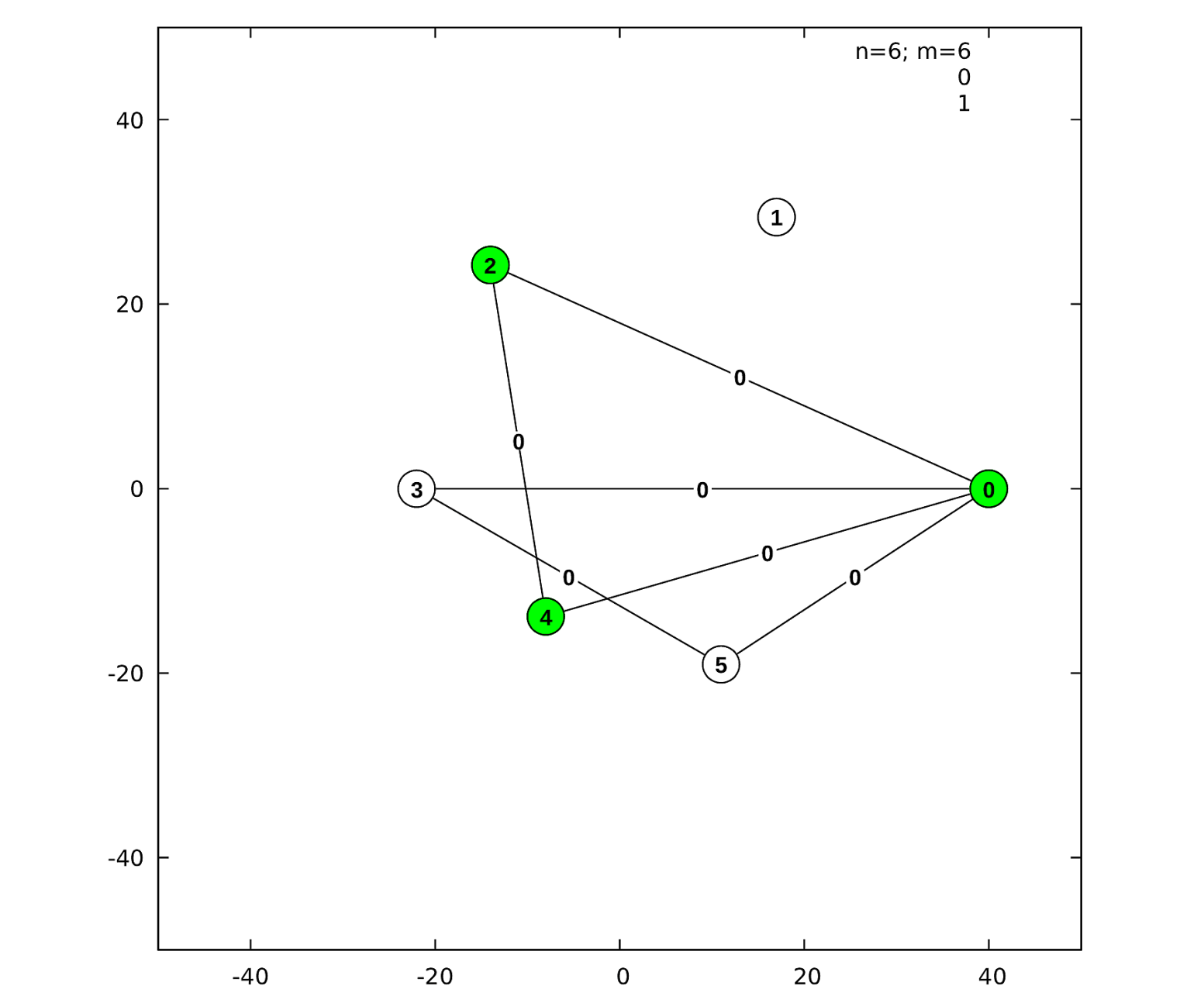


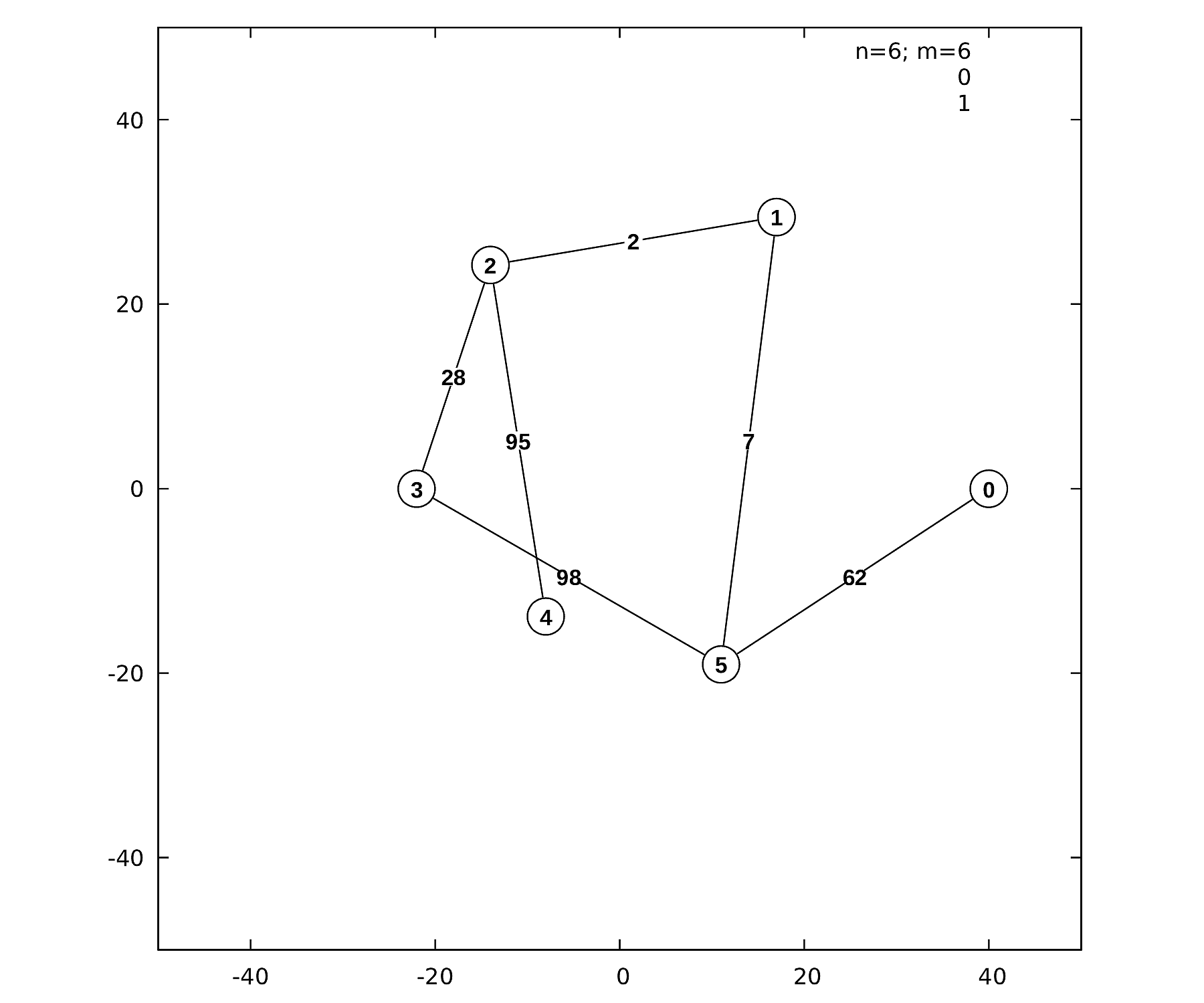
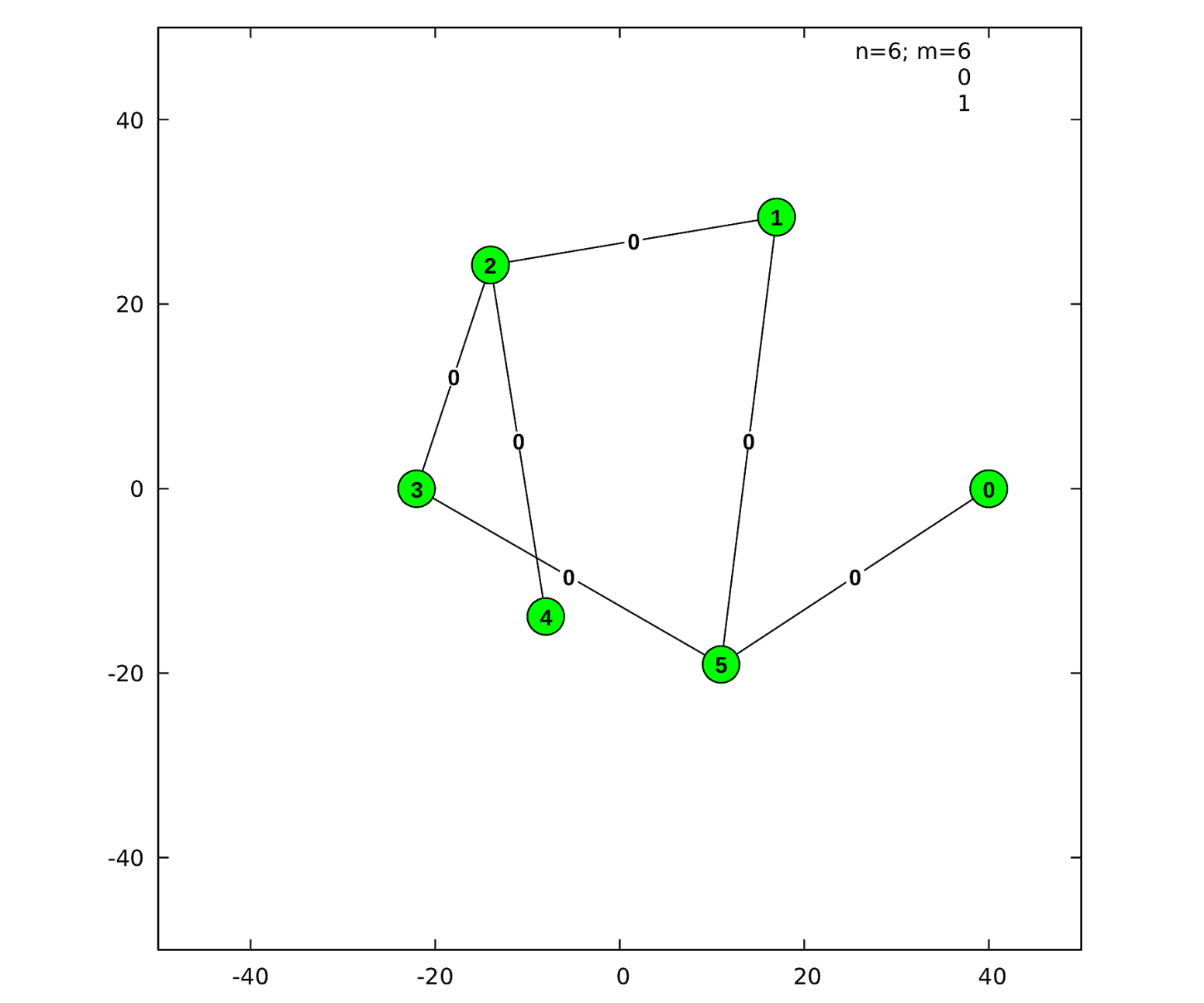
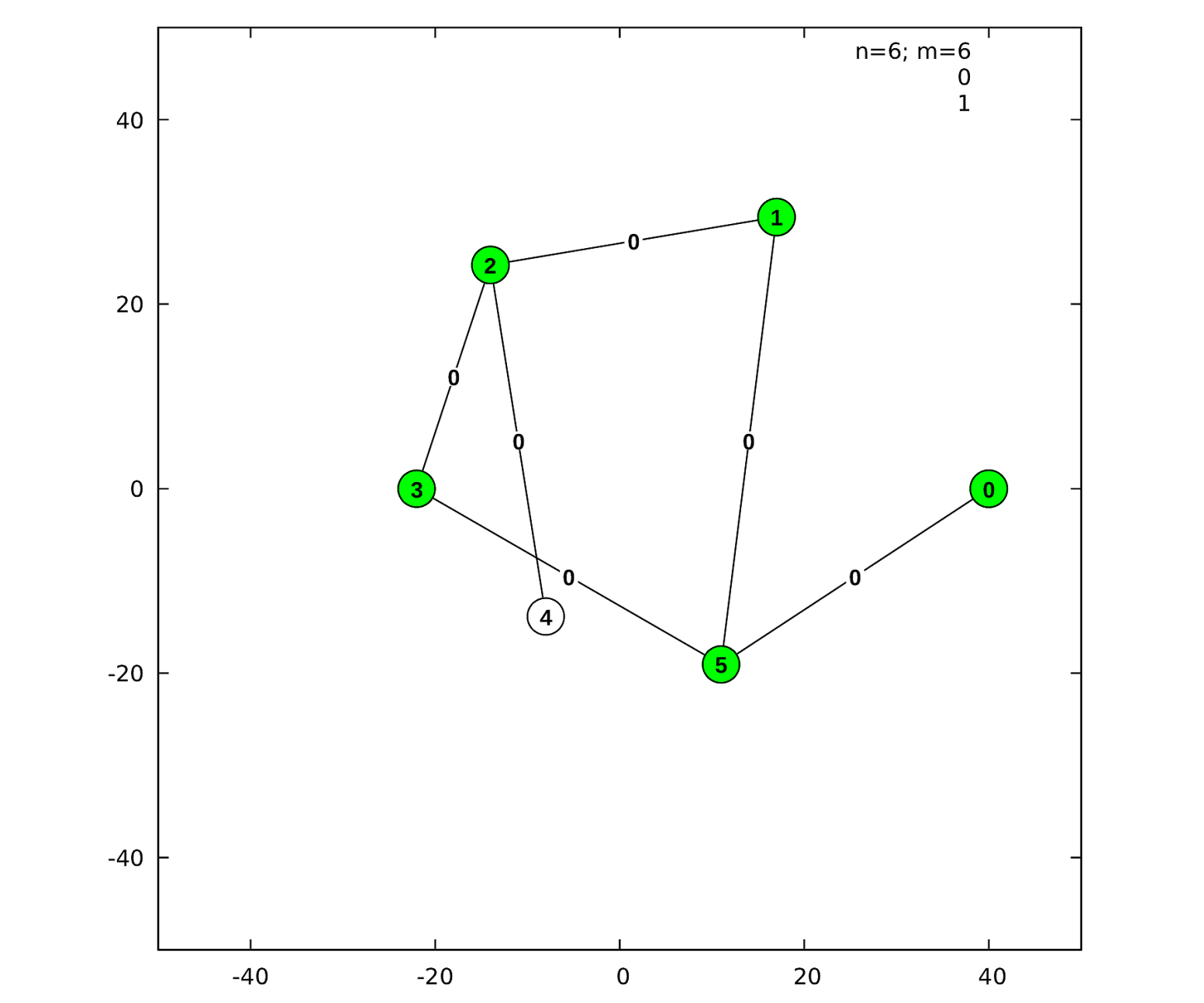
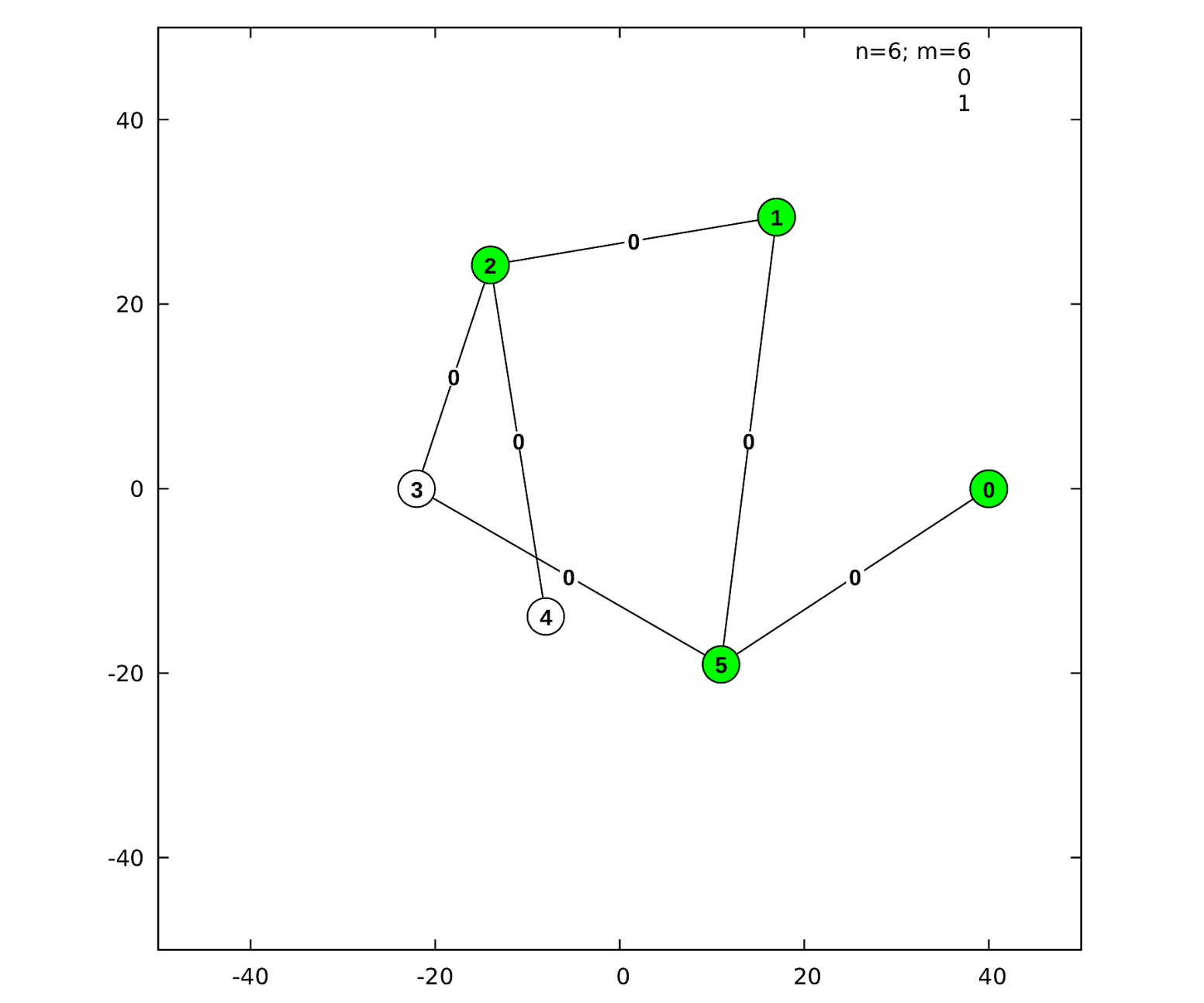
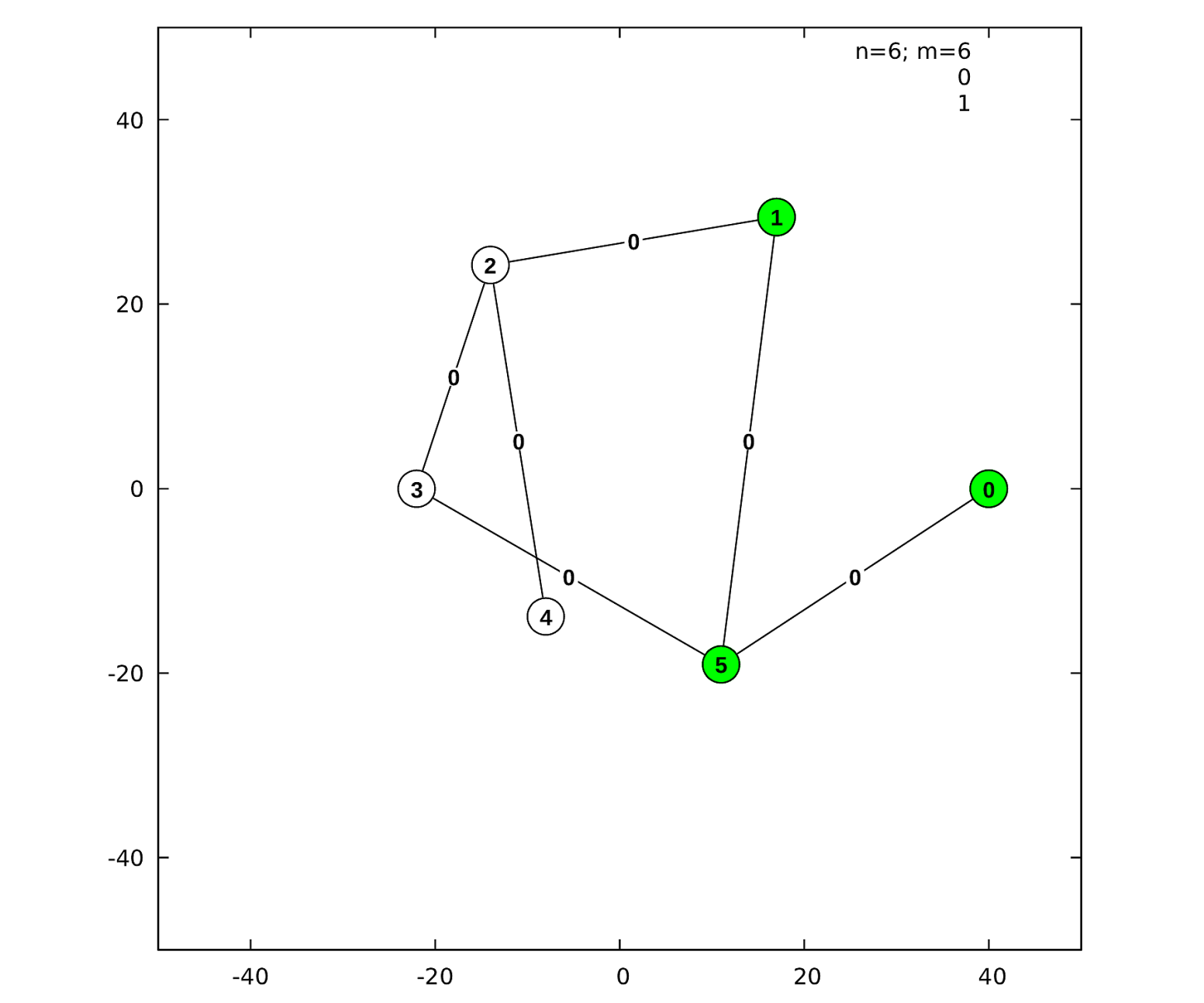
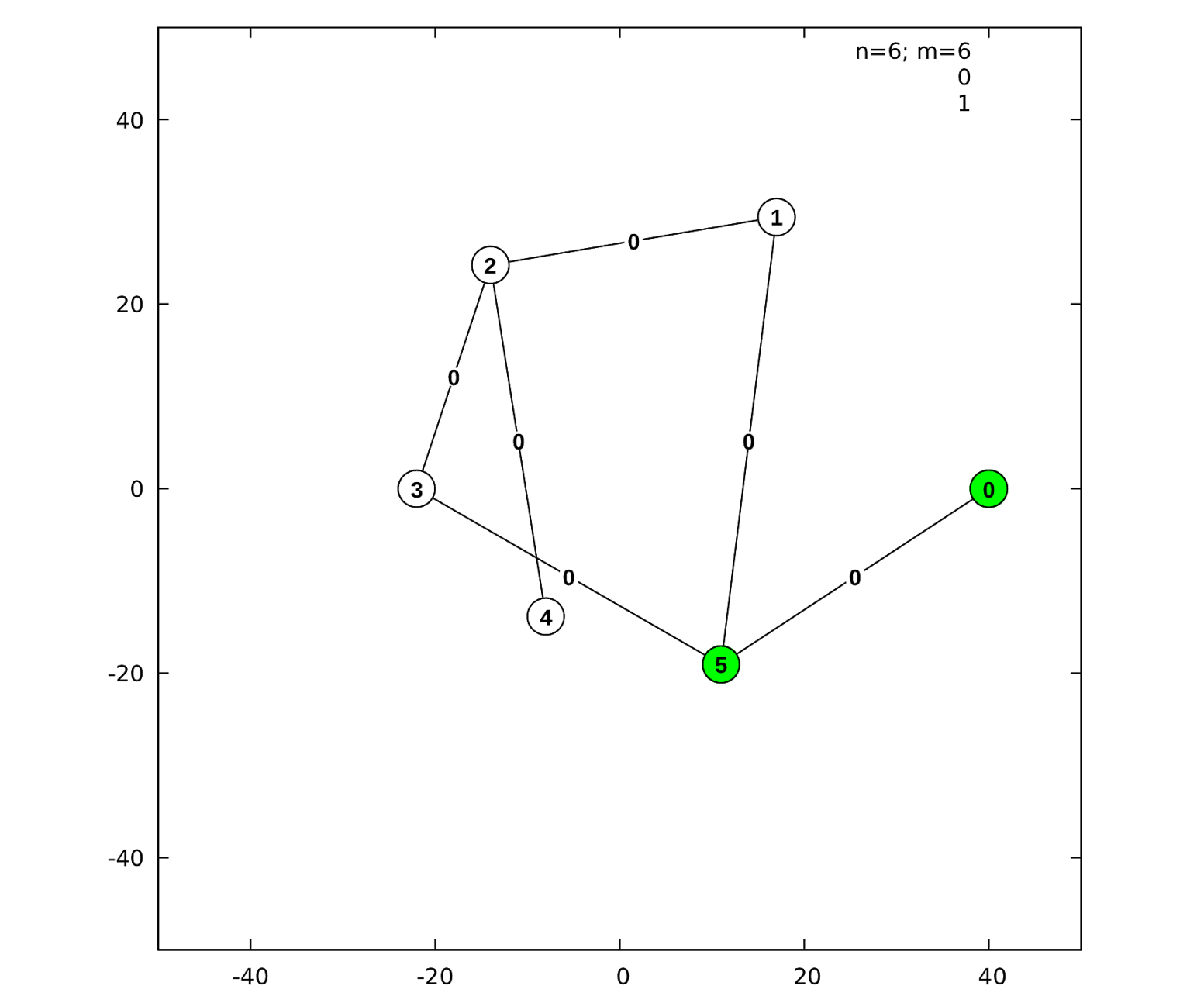
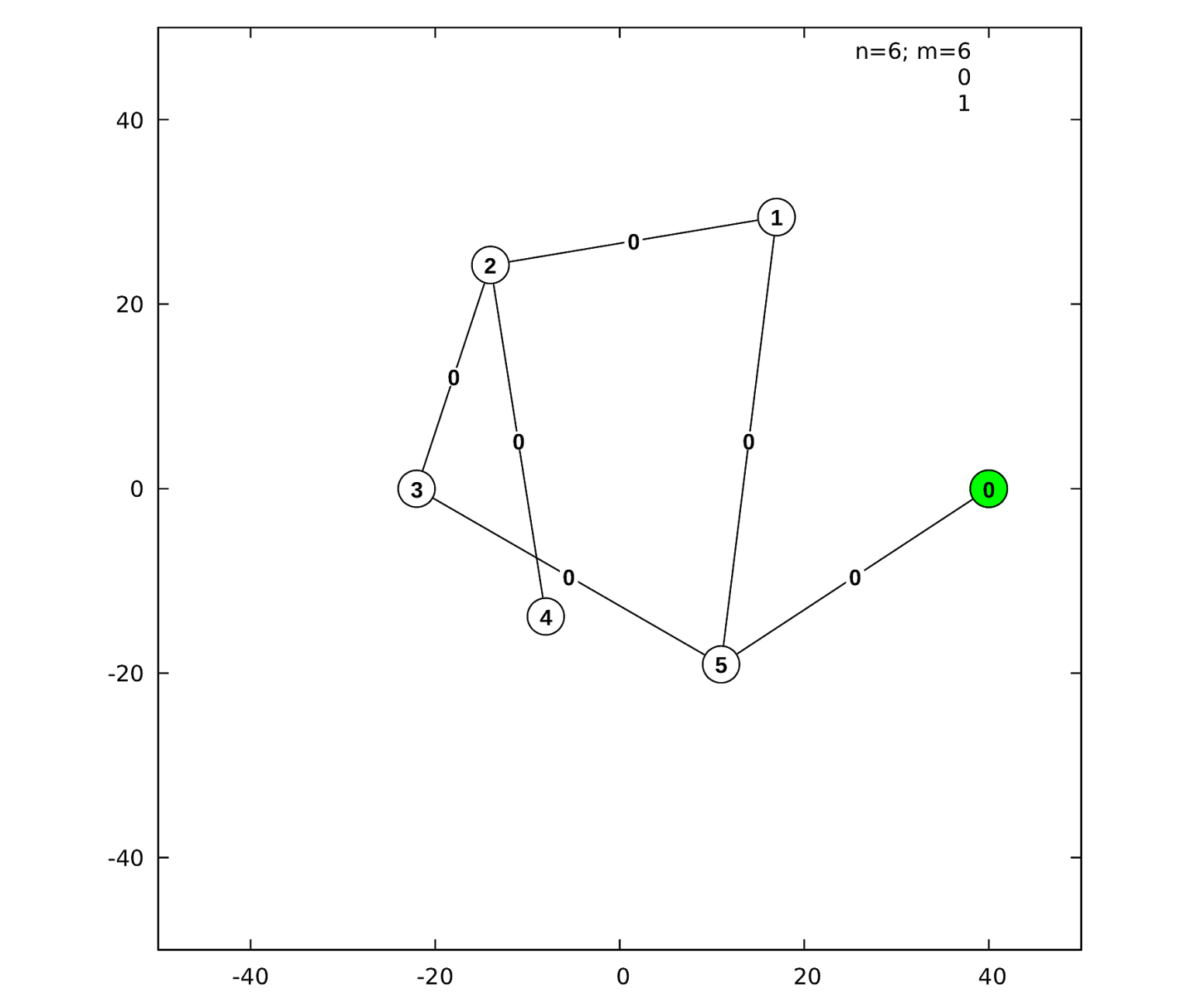
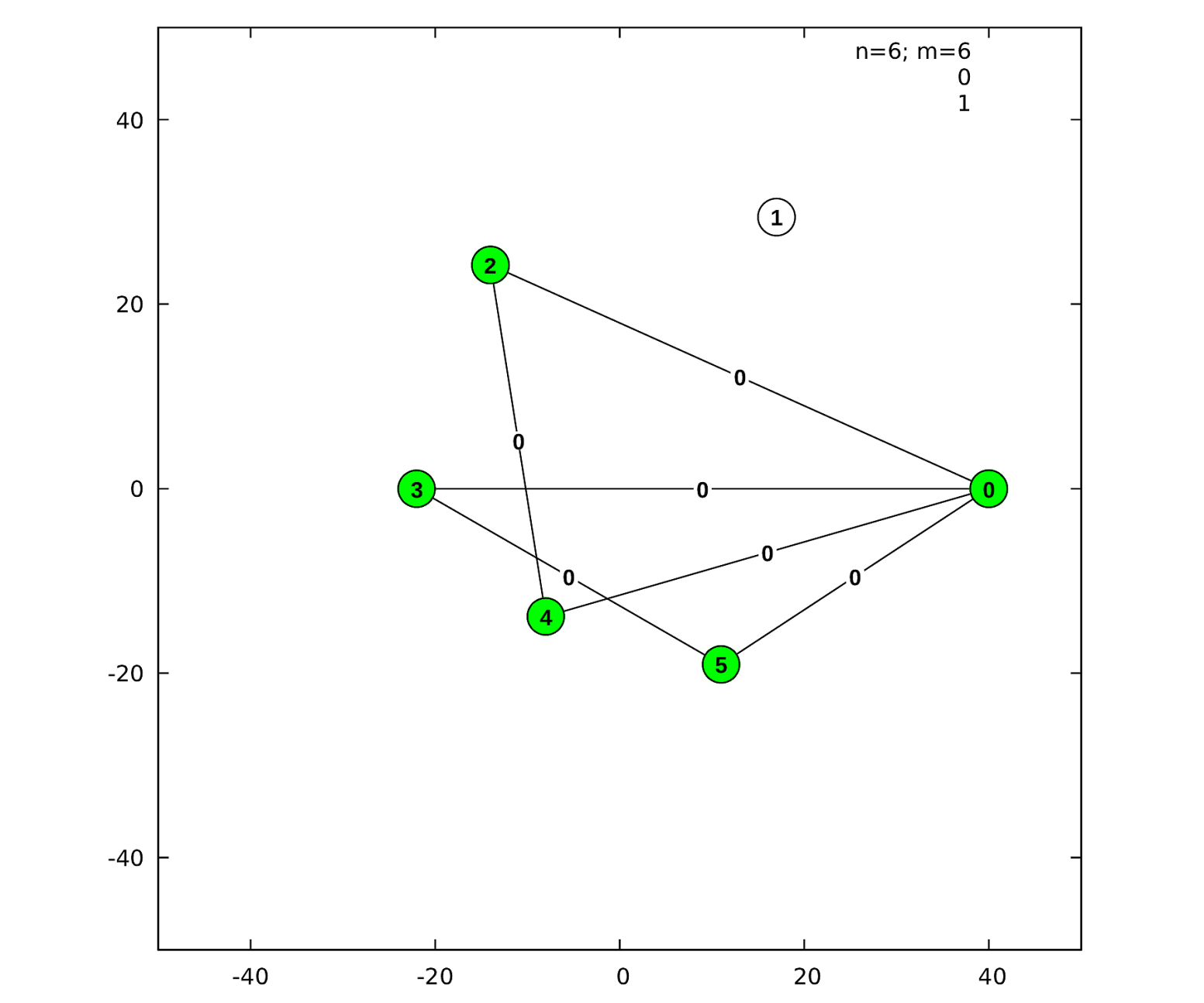
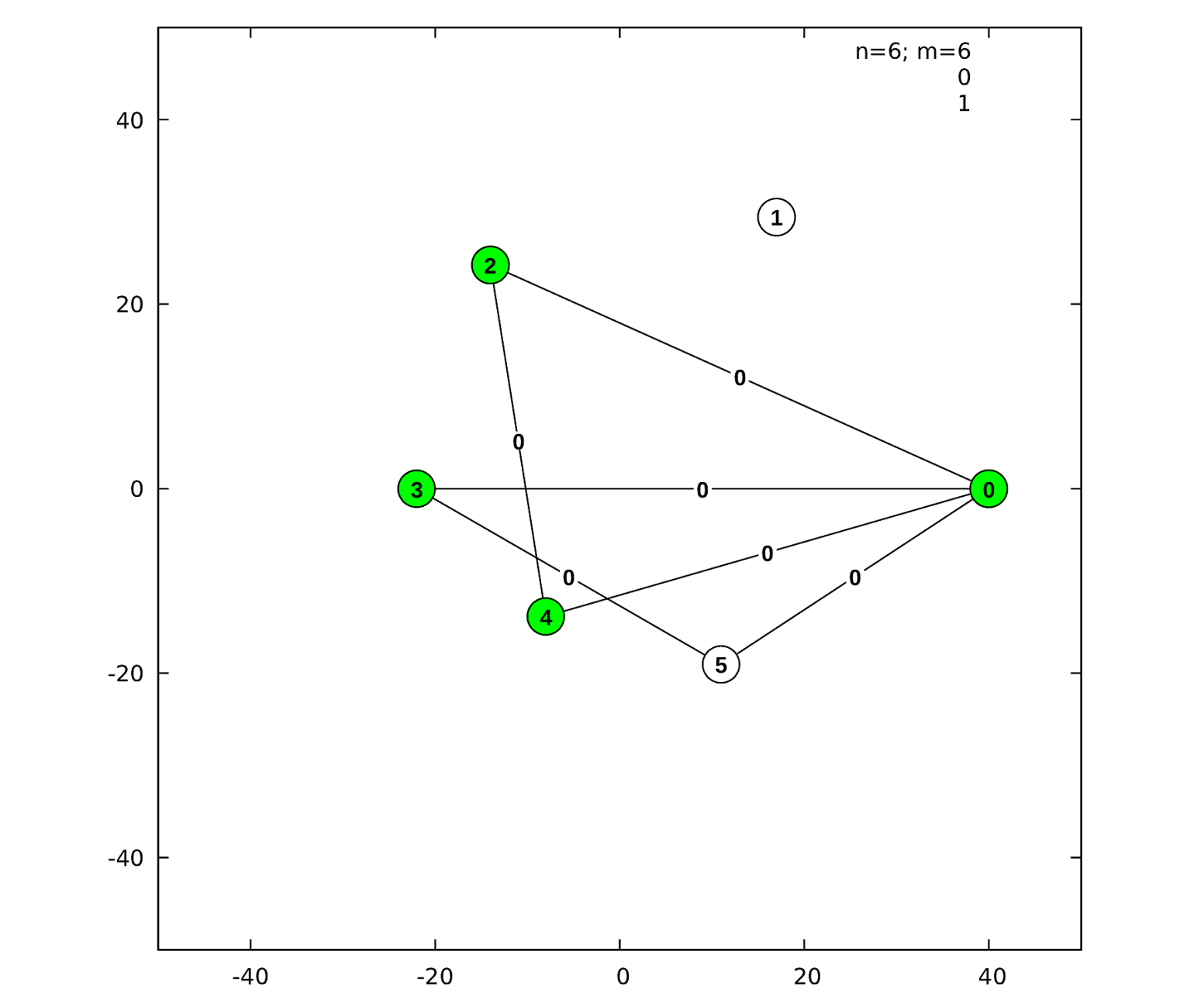
Неориентированный граф с петлями и весами вершин

* 1. Генерация графа с проверкой на связность с помощью обхода в глубину

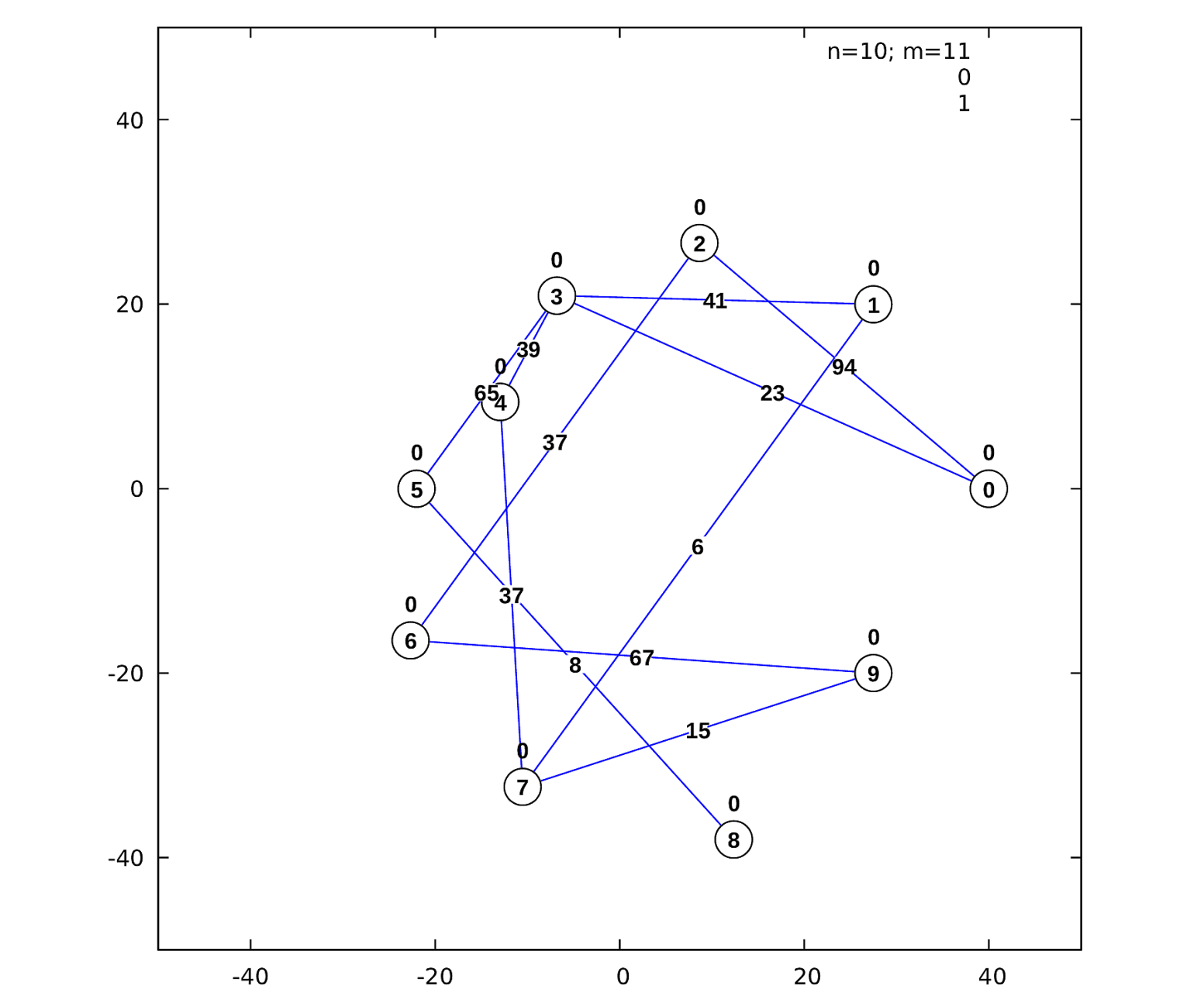


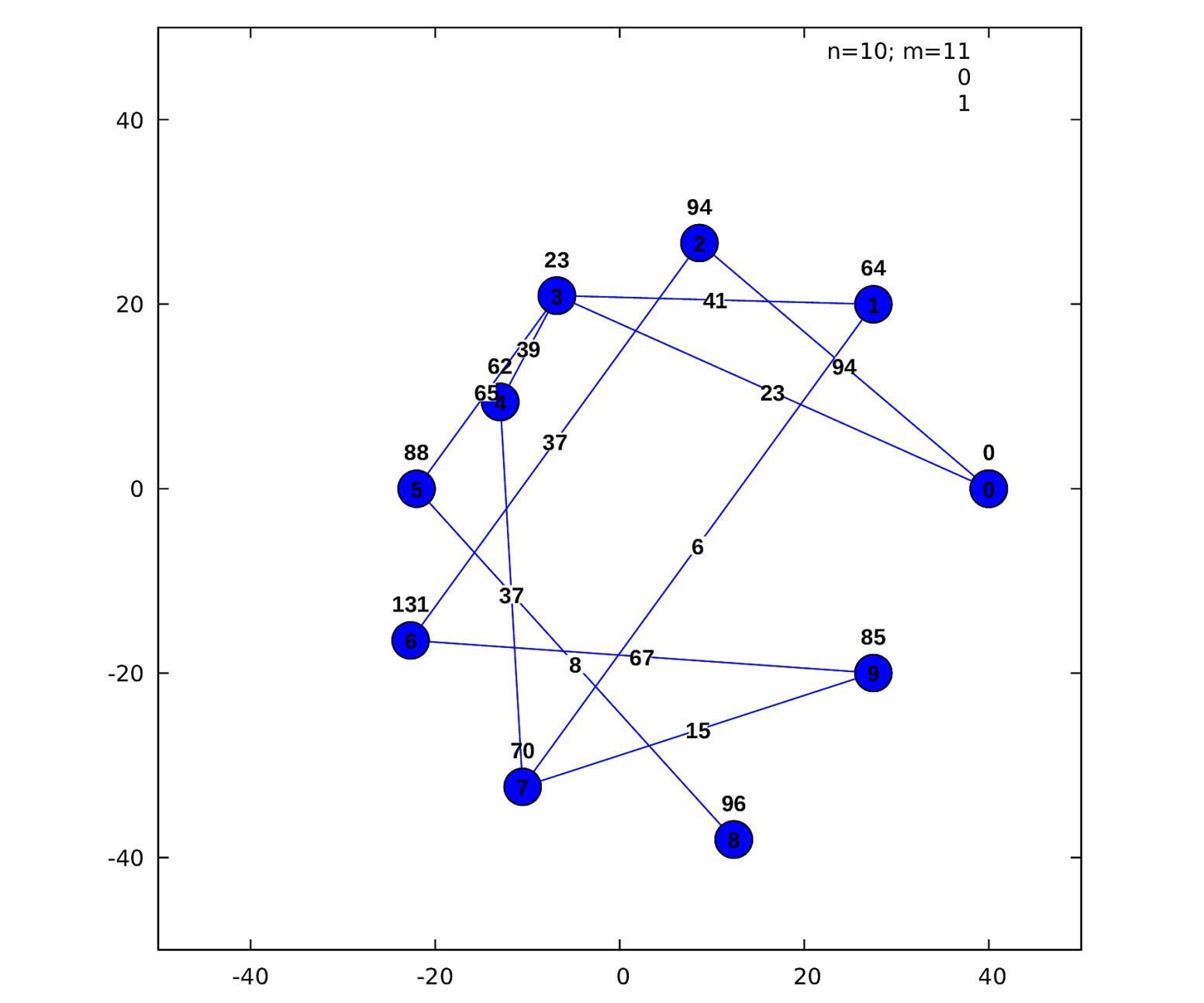
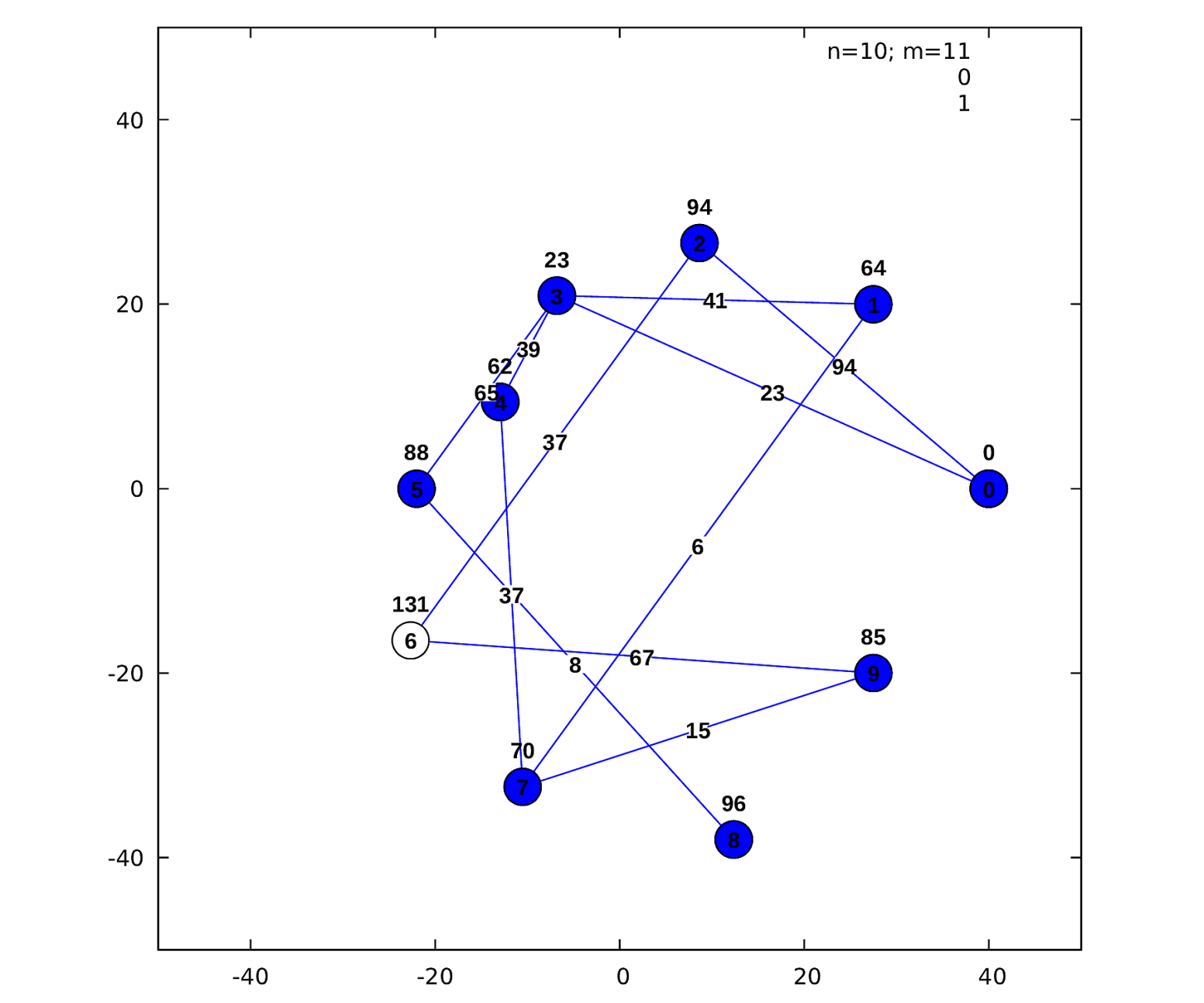
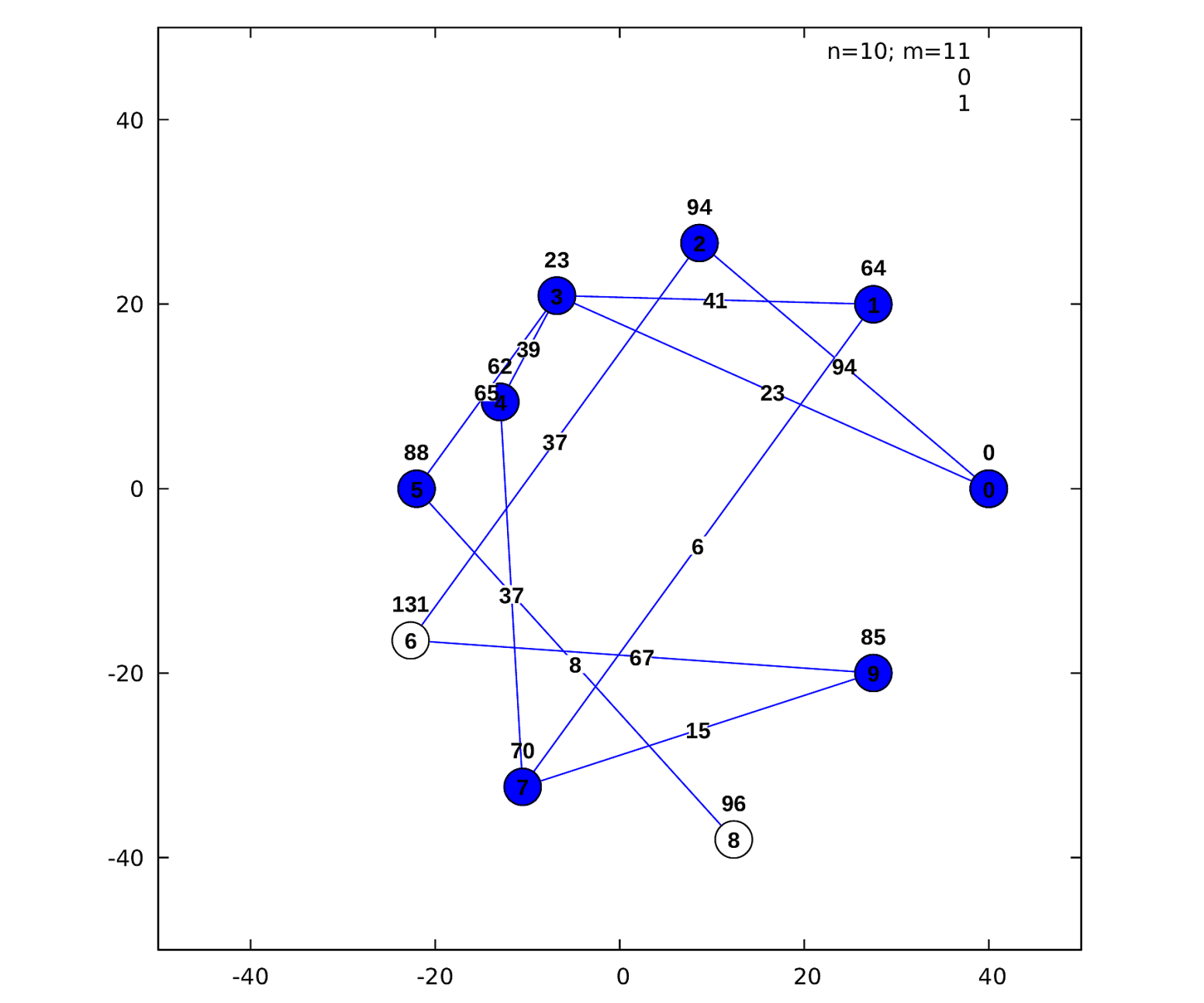
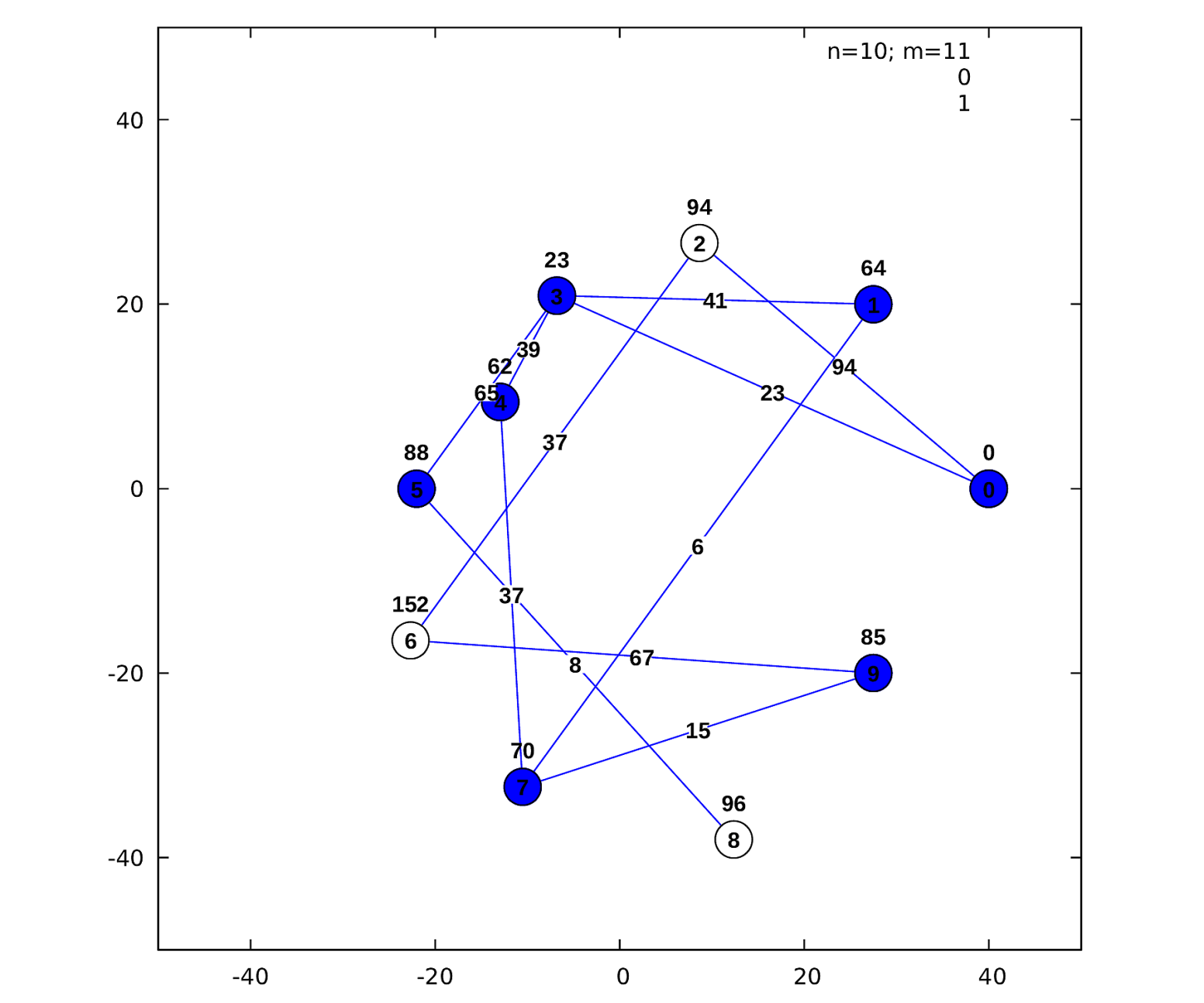
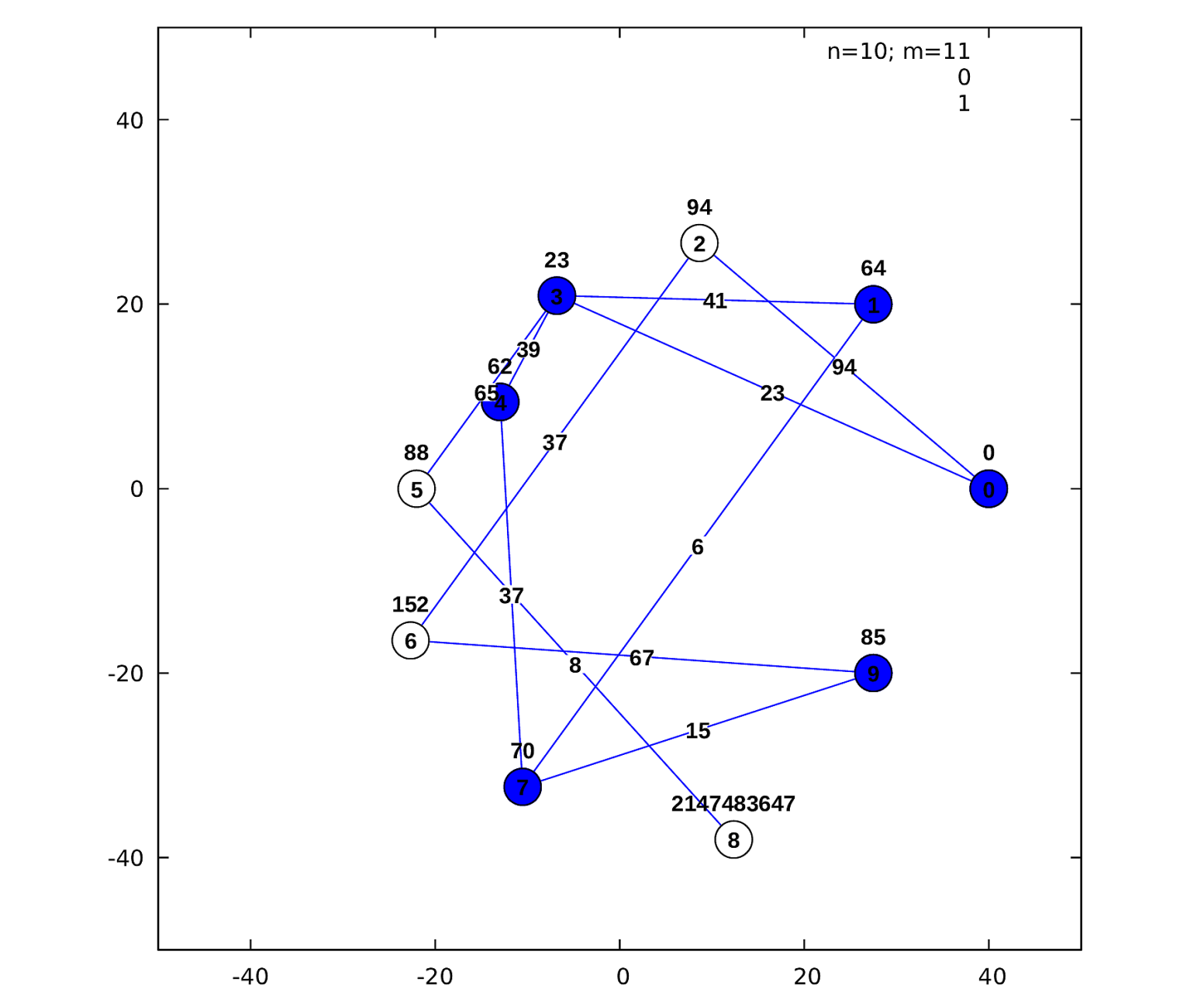
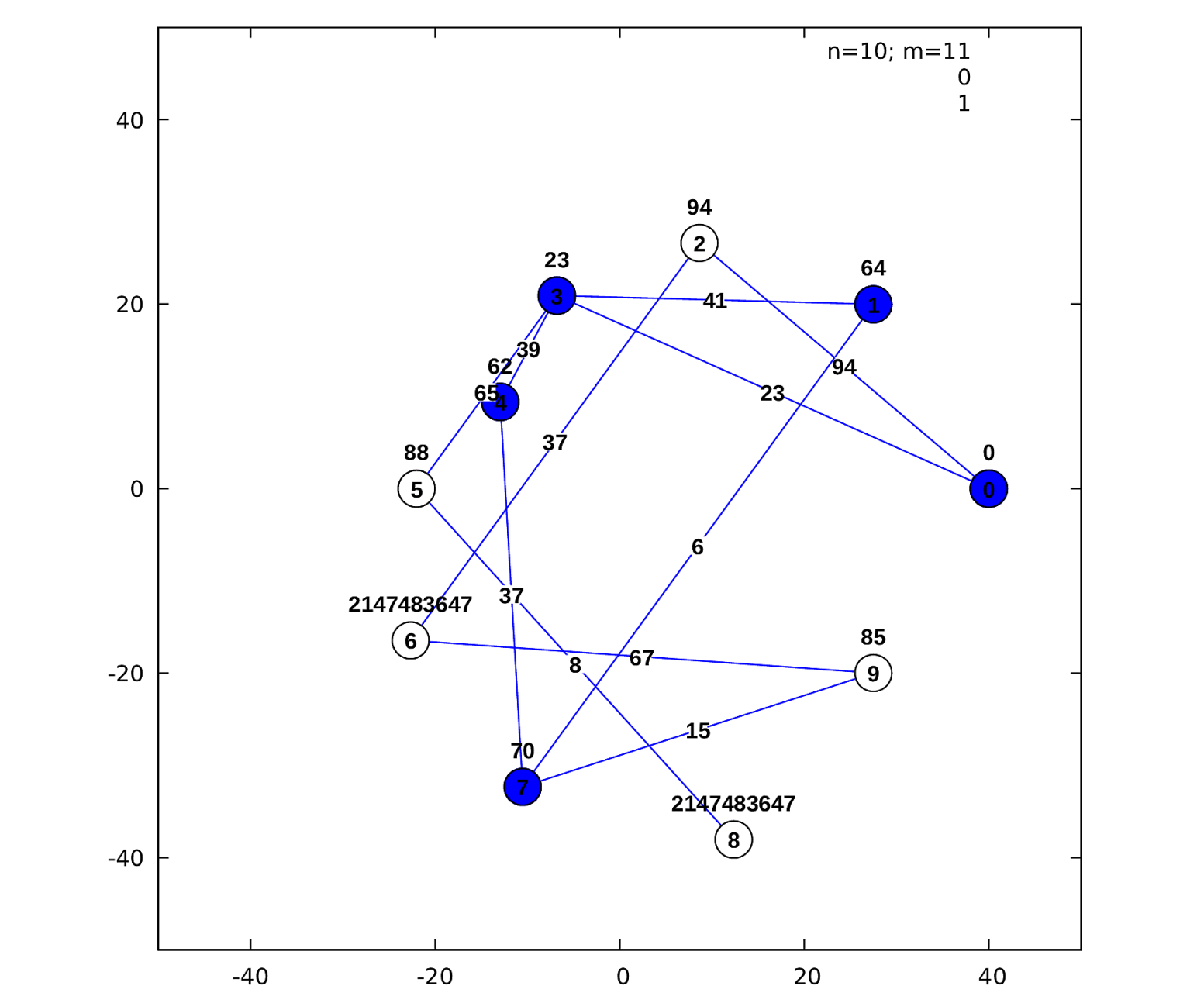
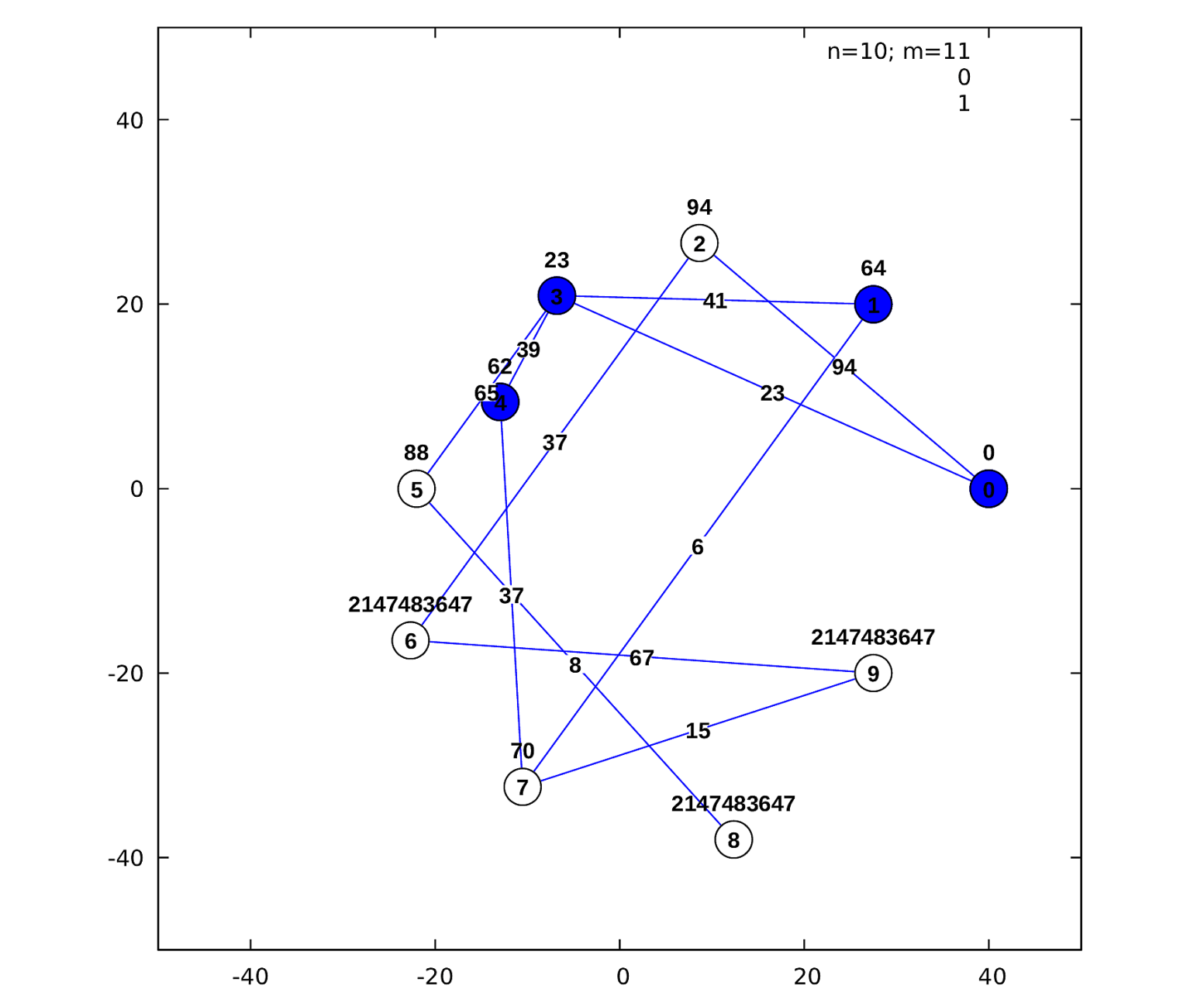
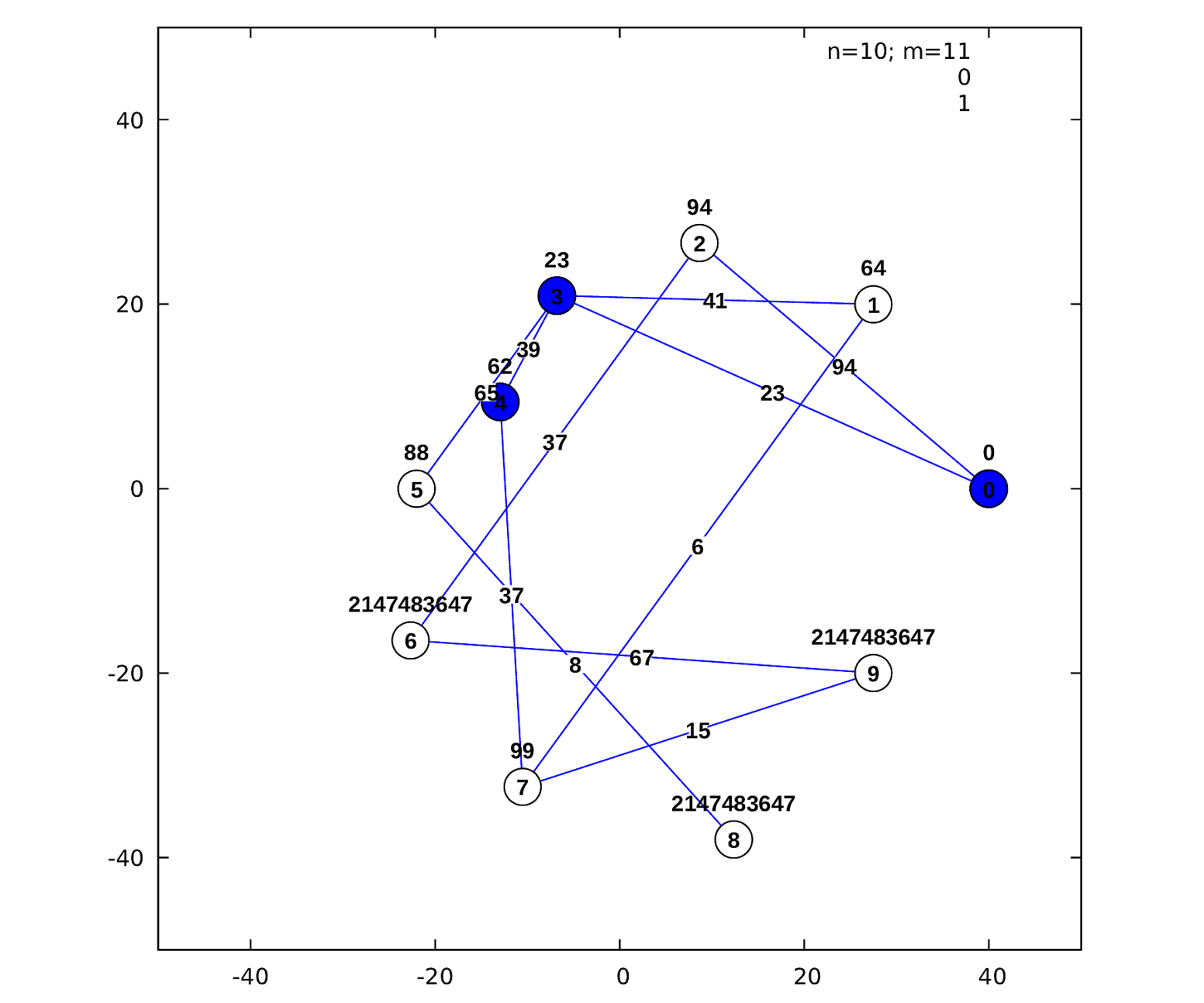
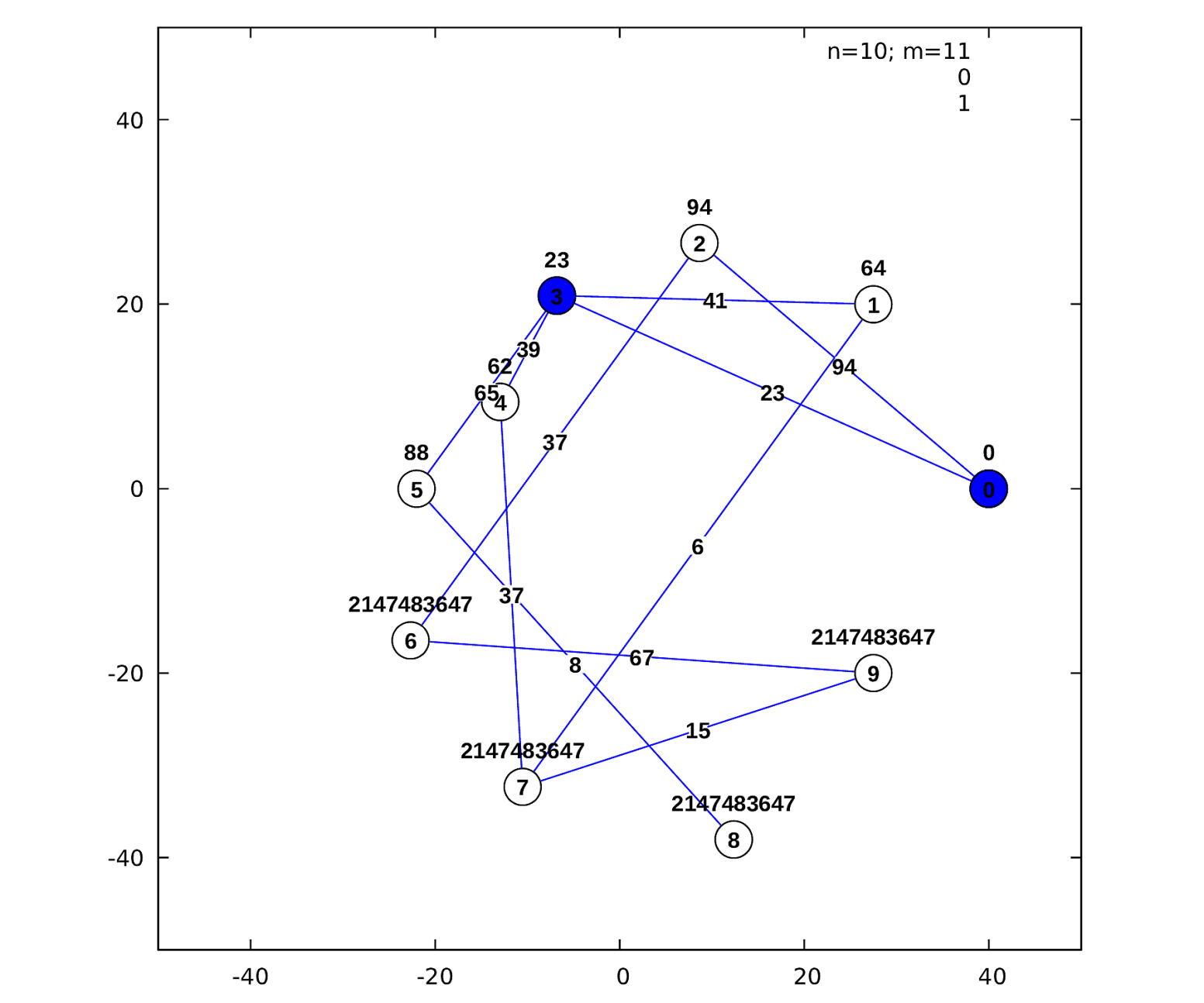
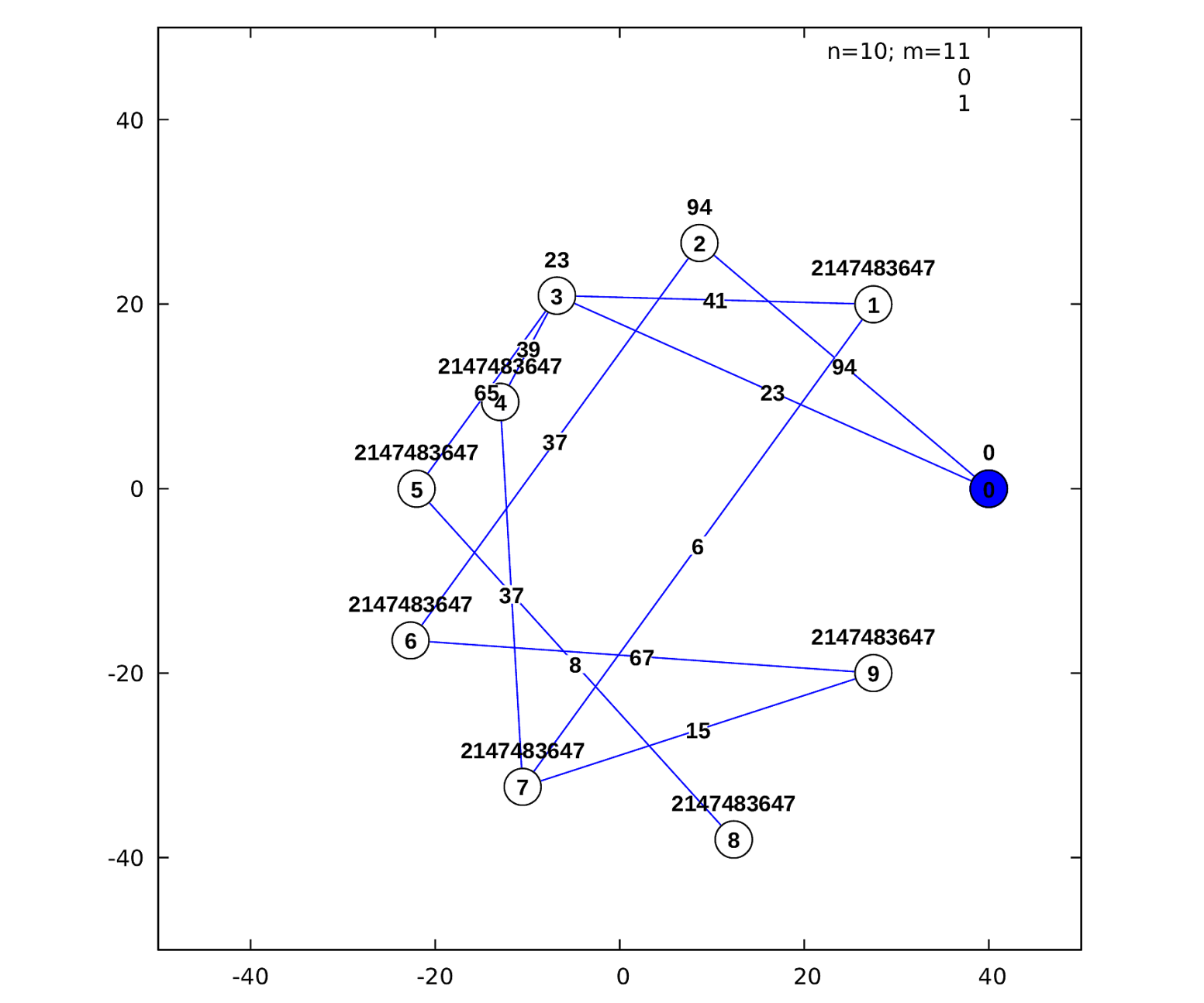






* 1. Пошаговая визуализация работы алгоритма Дейкстры. (раскраска вершин)





1. Описание Программы
2. Язык: C++
3. Среда: VS Code 1.63.2
4. ОС: Debian 11
5. Оболочка: xfce 4
6. Библиотеки: -
7. Фреймворк: -
8. Зависимости:
   1. gnuplot 5.4 patchlevel 1
   2. bash
   3. gawk 5.1.0, API: 3.0
   4. ImageMagick 6.9.11-60 Q16 x86\_64 2021-01-25
9. Заключение
10. В ходе проделанной работы были изучены вариативные шаблоны в языке C++, скриптовый язык gnuplot, команды и утилиты bash (heredoc, cat, read), произведено знакомство с языком awk.
11. Была разработана иерархия классов, реализующая структуру данных граф. Были написаны скрипты для отображения графов и графиков, а также для сохранения изображений графов в файлы. Были написаны классы, призванные упростить создание различных режимов отображения шагов работы алгоритма. Был написан класс для исследования алгоритмов, основой которого является метод – вариативный шаблон функции.
12. Цель можно считать достигнутой, однако нужно понимать, что время работы алгоритмов может сильно зависеть как от реализации самого алгоритма, так и от реализации структур данных, над которыми он работает. Также важен способ генерации случайного графа. Различные модели вероятностных графов дают разные по характеру графы (например, отличающиеся распределением степеней вершин).
13. Список литературы
14. Введение в теорию случайных графов Гаврилова А. А.
15. https://en.cppreference.com/
16. http://www.gnuplot.info/
17. Приложение
    1. CMakeLists.txt

cmake\_minimum\_required(VERSION 3.0.0)

project(Lab5 VERSION 0.1.0)

include(CTest)

enable\_testing()

add\_executable(Lab5 main.cpp) #Исследование алгоритмов ФлойдУоршелл Дейкстра БеллманФорд

add\_executable(main\_GenerateGraph\_Demo main\_GenerateGraph\_Demo.cpp) #Демонстрация проверки связности. Поиск в глубину.

add\_executable(main\_Dijkstra\_Demo main\_Dijkstra\_Demo.cpp)

add\_executable(main\_BellmanFord\_Demo main\_BellmanFord\_Demo.cpp)

add\_executable(main\_VisualGraphDemo main\_VisualGraphDemo.cpp) #Вывод графа в виде слова GRAPH

add\_executable(main\_GenerateAdjacency main\_GenerateAdjacency.cpp) #Проверка генерации графа

add\_executable(main\_StatisticsGraph\_SimpleDemo main\_StatisticsGraph\_SimpleDemo.cpp) #Демонстрация использования StatisticsGraph

install(PROGRAMS PlotGraph.bash PlotScript.bash DESTINATION ${CMAKE\_CURRENT\_BINARY\_DIR})

set(CPACK\_PROJECT\_NAME ${PROJECT\_NAME})

set(CPACK\_PROJECT\_VERSION ${PROJECT\_VERSION})

include(CPack)

* 1. BaseGraph.h

#ifndef BASEGRAPH\_H

#define BASEGRAPH\_H

#include <vector>

//смотри UML диаграмму

class BaseGraph

{

public:

struct vertex

{

double x=0; //Оставить или нет координаты - вопрос. На сколько они являются характеристикой самого графа, а не его изображения.

double y=0;

int Weight=0;

int Color=11; //Определяет стиль в gnuplot. 11 белый; 12 зеленый; 13 голубой; 14 красный

};

struct edge

{

int Adjacency=0; //существование ребра

int Weight=0; //вес ребра

int Color=1; //1 черный 2 зеленый; 3 голубой; 4 красный

};

std::vector<vertex> Vertices;

std::vector<std::vector<edge>> Edges;

bool IsOriented=false; //Ну тоже свойства графа, как-никак. Т.е. не отображения, а самого графа.

bool IsWithVerticesWeights=false;

BaseGraph(int N, bool NIsOriented=false):Vertices(N), Edges(N), IsOriented(NIsOriented)

{

for(auto& i: Edges)

{

i.resize(N);

}

}

};

#endif /\* BASEGRAPH\_H \*/

* 1. VisualGraph.h

#ifndef VISUALGRAPH\_H

#define VISUALGRAPH\_H

#include <iostream>

#include <iomanip>

#include <string>

#include <cstring>

#include <fstream>

#include <sstream>

#include <math.h>

#include "BaseGraph.h"

#define RADIUS 40

class VisualGraph: public BaseGraph

{

public:

VisualGraph(int N, bool NIsOriented=false):BaseGraph(N, NIsOriented)

{

}

//void SetVarsForScript(edges& Edges, bool IsOriented, bool IsWithVerticesWeights, bool IsSavePictureToFile, std::string PictureName,const char\* filename) //Эти величины будет читать bash скрипт. Edges, чтобы мы могли вывести число вершин и ребер

void SetVarsForScript(bool IsSavePictureToFile, std::string PictureName, const char\* filename) //Эти величины будет читать bash скрипт. Edges, чтобы мы могли вывести число вершин и ребер

{

int m=0; //число ребер в графе

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

for(int j=i; j<Edges.at(i).size(); j++)

{

m+=Edges.at(i).at(j).Adjacency;

}

std::ofstream fd(filename);

fd<<"#nmtitle IsOriented IsSavePictureToFile PictureName IsWithVerticesWeights"<<std::endl;

fd<<"\"n="<<Edges.size()<<"; m="<<m<< "\"\t"<<IsOriented<<"\t"<<IsSavePictureToFile<<"\t"<<PictureName<<"\t"<<IsWithVerticesWeights<<std::endl;

fd.close();

}

void PrintVertices(const char\* filename="std::cout")

{

if(!strcmp(filename, "std::cout")) //если строки равны

{

std::cout<<"#i; x; y; color;weight"<<std::endl;

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

{

std::cout<<i<<" "<<Vertices.at(i).x<<" "<<Vertices.at(i).y<<" "<<Vertices.at(i).Color<<" "<<Vertices.at(i).Weight<<std::endl;

}

}

else

{

std::ofstream fd(filename);

fd<<"#i; x; y; color;weight"<<std::endl;

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

{

fd<<i<<" "<<Vertices.at(i).x<<" "<<Vertices.at(i).y<<" "<<Vertices.at(i).Color<<" "<<Vertices.at(i).Weight<<std::endl;

}

fd.close();

}

}

void SetVertXYForPlot(int CenterX=0, int CenterY=0, const char\* fname="Vertices.dat")

{

double r=RADIUS;

double alpha=0;

double dalpha=2\*M\_PI/Vertices.size();

double x, y;

int CurrVertIndex=0;

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

{

//(i%2==0) ? r/=1.5 : r\*=1.5;

// Vector.at(i).VertIndex=i;

Vertices.at(i).x=cos(alpha)\*r;

Vertices.at(i).y=sin(alpha)\*r;

if(i%8<4) r-=RADIUS\*0.15; //в виде звезды

else r+=RADIUS\*0.15;

alpha+=dalpha;

}

}

void PrintEdges(const char\* filename="std::cout") // можно ввести "std::cout" и тогда информация будет выведена в консоль. Для рисования графиков не используется

{

if(!strcmp(filename, "std::cout")) //если строки равны

{

std::cout<<"[Adgacency Weight Color]"<<std::endl;

for(auto& i: Edges)

{

for(auto& j: i)

{

std::stringstream ss;

ss<<"["<<j.Adjacency<<" "<<j.Weight<<" "<<j.Color<<"]";

// std::cout.setf(std::ios::left);

// std::cout.left;

// std::cout<<std::setfill('=')<<std::setw(20) << std::left << ss.str();

std::cout<<std::setw(10) << std::left << ss.str();

}

std::cout<<std::endl;

}

}

else

{

std::ofstream fd(filename);

fd<<"[Adgacency Weight Color]"<<std::endl;

for(auto& i: Edges)

{

for(auto& j: i)

fd<<"["<<j.Adjacency<<" "<<j.Weight<<" "<<j.Color<<"] ";

fd<<std::endl;

}

fd.close();

}

}

void SetEdgesForPlot(const char\* fname="EdgesForPlot.dat")

{

std::ofstream fd(fname);

fd<<"#i;j;Weight;Color"<<std::endl;

if(IsOriented)

{

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

{

for(int j=0; j<Edges.at(i).size(); j++) //Граф у нас ориентированный! Тут заполняется вектор с ребрами

{

// std::cout<<i<<"\t"<<j<<"\t"<<Edges.at(i).at(j).Weight<<"\t"<<Edges.at(i).at(j).Color<<std::endl;

if(Edges.at(i).at(j).Adjacency==1)

{

fd<<i<<"\t"<<j<<"\t"<<Edges.at(i).at(j).Weight<<"\t"<<Edges.at(i).at(j).Color<<std::endl;

}

}

}

}

else

{

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

{

for(int j=i; j<Edges.at(i).size(); j++) //просматриваем только верхний угл. Граф у нас неориентированный! Тут заполняется вектор с ребрами

{

// std::cout<<i<<"\t"<<j<<"\t"<<Edges.at(i).at(j).Weight<<"\t"<<Edges.at(i).at(j).Color<<std::endl;

if(Edges.at(i).at(j).Adjacency==1)

{

fd<<i<<"\t"<<j<<"\t"<<Edges.at(i).at(j).Weight<<"\t"<<Edges.at(i).at(j).Color<<std::endl;

}

}

}

}

fd.close();

}

void ShowPlot(bool IsSavePictureToFile=false, std::string PictureName="pic", bool IsSetVertXYForPlot=true) //PictureName без расширения

{

if(IsSetVertXYForPlot) SetVertXYForPlot();

PrintVertices("Vertices.dat");

SetEdgesForPlot(); //Вывод в файл для рисования графа

PrintEdges("Edges.txt"); //вывод информации о ребрах для наглядности. Не используется для рисования графа

SetVarsForScript(IsSavePictureToFile, PictureName, "VarsForScript.dat"); //число вершин и ребер в файл

system("./PlotGraph.bash");

}

};

#endif /\* VISUALGRAPH\_H \*/

* 1. Graph.h

#ifndef GRAPH\_H

#define GRAPH\_H

#include "VisualGraph.h"

class Graph: public VisualGraph

{

public:

class vertices

{

private:

std::vector<vertex>& refVertices; //ссылка на массив вершин, находящийся в родительском объекте. Решил сделать закрытой, т.к. обращение к одной сущности по двум именам не есть гуд.

public:

vertices(std::vector<vertex>& NrefVertices): refVertices(NrefVertices)

{

}

void ResetColors()

{

for(auto& i: refVertices) i.Color=11;

}

};

class edges

{

private:

std::vector<std::vector<edge>>& refEdges;

public:

edges(std::vector<std::vector<edge>>& NrefEdges): refEdges(NrefEdges)

{

}

void ResetColors()

{

for(auto& i: refEdges)

for(auto& j: i) j.Color=1;

}

};

vertices vertsVertices; //обертки

edges edgesEdges;

Graph(int N, bool NIsOriented=false): VisualGraph(N, NIsOriented), vertsVertices(BaseGraph::Vertices), edgesEdges(BaseGraph::Edges) //инициализируем ссылки на массивы в базовом объекте

{

}

};

#endif /\* GRAPH\_H \*/

* 1. presenthandler.h

#ifndef PRESENTHANDLER\_H\_INCLUDED

#define PRESENTHANDLER\_H\_INCLUDED

#include <string>

class presenthandler //Класс для управления наглядным представлением шагов, совершаемых внутри функций и прочих.

{

private:

int CurrentFileNumber=0; //Количество уже сгенерированных файлов.

public:

int Mode=0; //0 -не влиять на выполнение вызывающей функции; 1 -выводить шаги в командную строку; 2 -выводить шаги в файл(ы)

std::string GetFileNumberAndIncrease()

{

CurrentFileNumber++;

return std::to\_string(CurrentFileNumber);

}

void ResetFileNumber()

{

CurrentFileNumber=0;

}

};

#endif //PRESENTHANDLER\_H\_INCLUDED

* 1. plothandler.h

#ifndef PLOTHANDLER\_H

#define PLOTHANDLER\_H

//Имена файлов, из которых берутся данные для графиков

#include <string>

#include <vector>

#include <fstream>

class plothandler

{

private:

std::vector<std::string> v;

public:

std::string addandreturn(std::string filename)

{

v.push\_back(filename);

return filename;

}

void tofile(std::string filename="Plotfilelist.txt")

{

std::ofstream fd(filename);

fd<<"filename=\t\"";

for(auto& i: v)

{

fd<<i<<" ";

}

fd<<"\"";

fd.close();

}

};

#endif /\* PLOTHANDLER\_H \*/

* 1. Func.h

#ifndef FUNC\_H\_INCLUDED

#define FUNC\_H\_INCLUDED

#include <iostream>

#include <fstream>

#include <sstream>

#include <string>

#include <random>

#include "presenthandler.h"

#include "Graph.h"

void GenerateAdjacencyProb(Graph& Graph1, double probability, std::default\_random\_engine& generator, bool IsWithLoops) //генерируем матрицу смежности. probability -вероятность появления ребра

{

//настраиваем генератор

std::discrete\_distribution<int> distribution {1-probability, probability}; //единички генерируются с вероятностью probability, а нолики с вероятностью 1-probability

// std::vector<std::vector<int>>::iterator iteri; //два итератора, для итерации по строкам и ячейкам(столбцам)

// std::vector<int>::iterator iterj;

for(int i=0; i<Graph1.Edges.size(); i++)

{

for(int j=0; j<Graph1.Edges.at(i).size();j++)

{

if(IsWithLoops)

{

if(j>=i)

{

//int dice\_roll = distribution(generator);

Graph1.Edges.at(i).at(j).Adjacency=distribution(generator); //генерируем ребро.

}

else

{

Graph1.Edges.at(i).at(j).Adjacency=Graph1.Edges.at(j).at(i).Adjacency; //граф неориентированный

}

}

else

{

if(j==i)

{

Graph1.Edges.at(i).at(j).Adjacency=0;

}

else

{

if(j>i)

{

//int dice\_roll = distribution(generator);

Graph1.Edges.at(i).at(j).Adjacency=distribution(generator); //генерируем ребро.

}

else

{

Graph1.Edges.at(i).at(j).Adjacency=Graph1.Edges.at(j).at(i).Adjacency; //граф неориентированный

}

}

}

}

}

}

void GenerateAdjacencyMNumberWithLoops(Graph& Graph1, int m, std::default\_random\_engine& generator) //генерируем матрицу смежности. m -число ребер в случайном графе. generator нужно передавать, если мы хотим, чтобы была возможность генерации одинаковой последовательности при многократном вызове функции

{

//настраиваем генератор

// std::default\_random\_engine generator(seed);

int n=(Graph1.Edges.size()+1)\*Graph1.Edges.size()/2; //общее число случаев в классическом определении вероятности (пользуемся формулой арфиметической прогрессии для нахождения числа ячеек)

//проверка на одз

if(m>n)

{

std::string str;

std::stringstream ss;

ss << "m=" <<m<<" is too big! (max m=n="<<n<<")"<<std::endl;

throw(ss.str());

}

std::vector<std::vector<int>>::iterator iteri; //два итератора, для итерации по строкам и ячейкам(столбцам)

std::vector<int>::iterator iterj;

for(int i=0; i<Graph1.Edges.size(); i++)

{

for(int j=0; j<Graph1.Edges.at(i).size();j++)

{

if(j>=i)

{

std::discrete\_distribution<int> distribution {1-(double)m/n, (double)m/n}; //

//int dice\_roll = distribution(generator);

Graph1.Edges.at(i).at(j).Adjacency=distribution(generator); //генерируем ребро.

if(Graph1.Edges.at(i).at(j).Adjacency!=0) //пользуемся классическим определением вероятности

{

m--;

n--;

}

else

{

n--;

}

}

else

{

Graph1.Edges.at(i).at(j).Adjacency=Graph1.Edges.at(j).at(i).Adjacency; //граф неориентированный

}

}

}

}

void GenerateAdjacencyMNumberNoLoops(Graph& Graph1, int m, std::default\_random\_engine& generator) //генерируем матрицу смежности. m -число ребер в случайном графе. Т.е. на главной диагонали выставляются 0.

{

//настраиваем генератор

// std::default\_random\_engine generator(seed);

int n=(Graph1.Edges.size()-1+0)\*Graph1.Edges.size()/2; //общее число случаев в классическом определении вероятности (пользуемся формулой арфиметической прогрессии для нахождения числа ячеек)

//проверка на одз

if(m>n)

{

std::string str;

std::stringstream ss;

ss << "m=" <<m<<" is too big! (max m=n="<<n<<")"<<std::endl;

throw(ss.str());

}

// std::vector<std::vector<int>>::iterator iteri; //два итератора, для итерации по строкам и ячейкам(столбцам)

// std::vector<int>::iterator iterj;

for(int i=0; i<Graph1.Edges.size(); i++)

{

for(int j=0; j<Graph1.Edges.at(i).size();j++)

{

if(j==i)

{

Graph1.Edges.at(i).at(j).Adjacency=0;

}

else

{

if(j>i)

{

std::discrete\_distribution<int> distribution {1-(double)m/n, (double)m/n}; //

//int dice\_roll = distribution(generator);

Graph1.Edges.at(i).at(j).Adjacency=distribution(generator); //генерируем ребро.

if(Graph1.Edges.at(i).at(j).Adjacency!=0) //пользуемся классическим определением вероятности

{

m--;

n--;

}

else

{

n--;

}

}

else

{

Graph1.Edges.at(i).at(j).Adjacency=Graph1.Edges.at(j).at(i).Adjacency; //граф неориентированный

}

}

}

}

}

void GenerateWeights(Graph& Graph1, std::default\_random\_engine& generator, int WeightMin, int WeightMax)

{

std::uniform\_int\_distribution<int> distribution(WeightMin, WeightMax);

for(int i=0; i<Graph1.Edges.size(); i++)

{

for(int j=0; j<Graph1.Edges.at(i).size();j++)

{

if(Graph1.Edges.at(i).at(j).Adjacency==1)

{

if(j>=i)

{

Graph1.Edges.at(i).at(j).Weight=distribution(generator);

}

else

{

Graph1.Edges.at(i).at(j).Weight=Graph1.Edges.at(j).at(i).Weight;

}

}

else

{

Graph1.Edges.at(i).at(j).Weight=0;

}

}

}

}

bool IsConnectedDFS(Graph& Graph1, int StartVert, presenthandler& PresentHandler); //Проверка связности графа с помощью обхода в глубину

void GenerateGraph(Graph& Graph1, int m, int WeightMax, std::default\_random\_engine& generator, presenthandler& PresentHandler)

{

int iteration=0;

do

{

std::cout<<"Generating Adjacency: iteration: "<<iteration<<std::endl;

try {GenerateAdjacencyMNumberNoLoops(Graph1, m, generator);}

catch(std::string str)

{

std::cout<<"exeption:"<<str<<std::endl;

getchar();

// return -1;

}

iteration++;

}while(!IsConnectedDFS(Graph1, 0, PresentHandler));

GenerateWeights(Graph1, generator, 1, 100); //все норм. Если seed принял какое-либо значение, то им определяется количество попыток, которые будут сделаны, чтобы получить Adjacency однозначно, поэтому при данном seed Weight будет также однозначным

}

int DFS(Graph& Graph1, int StartVert, presenthandler& PresentHandler) //IsStepByStep 0-не по шагам; 1-по шагам; 2-вывод в файл

{

/\* if(StartVert>=Vertices.Vector.size() || StartVert<0)

{

std::stringstream ss;

ss<<"StartVert cannot be equal to "<< StartVert <<std::endl;

throw(ss.str());

}

\*/

Graph1.Vertices.at(StartVert).Color=12;

int ColoredVertNumber=1;

for(int j=0; j<Graph1.Vertices.size(); j++)

{

if((Graph1.Edges.at(StartVert).at(j).Adjacency==1) && (Graph1.Vertices.at(j).Color!=12))

{

if(PresentHandler.Mode>=1)

{

Graph1.ShowPlot(PresentHandler.Mode-1, std::string("Pic").append(PresentHandler.GetFileNumberAndIncrease()));

if(PresentHandler.Mode==1) getchar();

}

ColoredVertNumber+=DFS(Graph1, j, PresentHandler);

}

}

return ColoredVertNumber;

}

bool IsConnectedDFS(Graph& Graph1, int StartVert, presenthandler& PresentHandler) //Проверка связности графа с помощью обхода в глубину

{

// int dtemp=DFS(Vertices, Edges, StartVert);

// std::cout<<"DFS="<<dtemp<<" Vertices.Vector.size()="<<Vertices.Vector.size()<<std::endl;

if(DFS(Graph1, StartVert, PresentHandler)==Graph1.Vertices.size())

{

if(PresentHandler.Mode>=1)

{

Graph1.ShowPlot(PresentHandler.Mode-1, std::string("Pic").append(PresentHandler.GetFileNumberAndIncrease()));

if(PresentHandler.Mode==1) getchar();

}

Graph1.vertsVertices.ResetColors();

return true;

}

else

{

if(PresentHandler.Mode>=1)

{

Graph1.ShowPlot(PresentHandler.Mode-1, std::string("Pic").append(PresentHandler.GetFileNumberAndIncrease()));

if(PresentHandler.Mode==1) getchar();

}

Graph1.vertsVertices.ResetColors();

return false;

}

}

template<typename T>

void PrintMatrix(std::vector<std::vector<T>>& Matrix, const char\* MatrixName)

{

typename std::vector<std::vector<T>>::iterator iteri; //два итератора, для итерации по строкам и ячейкам(столбцам) .Без typename вообще не компилируется, хотя казалось бы.

typename std::vector<T>::iterator iterj;

for(iteri=Matrix.begin(); iteri!=Matrix.end(); iteri++)

{

std::cout<< MatrixName<<"["<<std::distance(Matrix.begin(), iteri) <<"]=";

for(iterj=iteri->begin(); iterj!=iteri->end(); iterj++)

{

std::cout<< (\*iterj)<<"\t";

}

std::cout<<std::endl;

}

std::cout<<"----------"<<std::endl;

}

/\*template<typename T>

void PrintMatrixToFile(std::vector<std::vector<T>>& Matrix, const char\* MatrixName, const char\* filename)

{

std::ofstream fd(filename);

typename std::vector<std::vector<T>>::iterator iteri; //два итератора, для итерации по строкам и ячейкам(столбцам) .Без typename вообще не компилируется, хотя казалось бы.

typename std::vector<T>::iterator iterj;

for(iteri=Matrix.begin(); iteri!=Matrix.end(); iteri++)

{

// fd<< MatrixName<<"["<<std::distance(Matrix.begin(), iteri) <<"]=";

for(iterj=iteri->begin(); iterj!=iteri->end(); iterj++)

{

fd<< (\*iterj)<<"\t";

}

fd<<std::endl;

}

fd.close();

}\*/

#endif //FUNC\_H\_INCLUDED

* 1. StatisticsGraph.inl

#ifndef STATISTICSGRAPH\_INL\_INCLUDED

#define STATISTICSGRAPH\_INL\_INCLUDED

#include <limits> //для определения максимального числа в типе

#include <time.h> //для измерения времени

#include <fstream> //для файлов

#include <iomanip> // std::setprecision

#include <string>

#include <random>

#include "Graph.h"

#include "presenthandler.h"

class StatisticsGraph

{

private:

int CurrentSize=0; //текущее число вершин в графе

int CurrentM=0;

int NStart=0, NEnd=0, NStep=0;

double MRatio=0; //Отношение числа ребер к максимально возможному для данного графа (будет округляться до целого числа m)

int WeightMax=0; //максимальный вес ребер графа

int NumberOfRuns=1; //число прогонов для каждого размера(числа вершин) графа

public:

//в callback будут Vertices Edges SourceVert DestVert параметры для результатов работы MPath MPathLenght для ФлойдаУоршелла Lenght для Дейкстры и т.д. PresentHandler

template<typename... CallBackParamsTail> //CallBackGenerate m WeightMax

StatisticsGraph(int NNStart, int NNEnd, int NNStep, double NMRatio, int NWeightMax, int NNumberOfRuns, std::default\_random\_engine& generator, presenthandler& PresentHandler, std::string filename, void (\*CallBackGenerate)(Graph&, int, int, std::default\_random\_engine&, presenthandler&), void(\*CallBackFind)(Graph&, CallBackParamsTail&...), void(\*CallBackTailHandler)(int N, CallBackParamsTail&...), CallBackParamsTail& ...callbackparamstail): NStart(NNStart), NEnd(NNEnd), NStep(NNStep), MRatio(NMRatio), WeightMax(NWeightMax), NumberOfRuns(NNumberOfRuns)

{

printLabel(filename);

for(CurrentSize=NStart; CurrentSize<=NEnd; CurrentSize+=NStep) //Цикл табуляции по числу вершин N

{

double timeSeconds=0;

for(int k=0; k<NumberOfRuns; k++) //Число прогонов для данного числа вершин N

{

Graph Graph1(CurrentSize);

CurrentM= (double)(Graph1.Edges.size()-1+0)\*Graph1.Edges.size()/2 \* MRatio; //максимально возможное количество ребер на интересующий нас процент

std::cout<<"Graph generating started"<<std::endl;

CallBackGenerate(Graph1, CurrentM, WeightMax, generator, PresentHandler); //генерируем подходящий граф

std::cout<<"Graph generating ended"<<std::endl;

if(PresentHandler.Mode>=1) Graph1.ShowPlot(false, "Pic");

CallBackTailHandler(CurrentSize, callbackparamstail...); //Управляемся с "лишними" параметрами функции поиска. В частности, изменяем размеры их массивов.

clock\_t timeStart=clock(); //число тиков с начала выполнения программы

CallBackFind(Graph1, callbackparamstail...);

clock\_t timeEnd=clock();

timeSeconds+=double(timeEnd-timeStart)/CLOCKS\_PER\_SEC; //время работы алгоритма в секундах

std::cout<<"CurrentSize="<< CurrentSize <<"; Iteration "<<k<<" is complete"<<std::endl;

}

timeSeconds/=NumberOfRuns;

printValue(CurrentSize, timeSeconds, filename);

}

}

void printLabel(std::string filename) //пишем подпись оси размера массива в файл

{

std::ofstream fd(filename, std::ios\_base::out|std::ios\_base::trunc); //для записи очистив

fd<< "#filename="<< filename<<"; MRatio="<<MRatio<<"; WeightMax="<<WeightMax<<"; NumberOfRuns="<<NumberOfRuns<<std::endl;

fd.close();

}

void printValue(int CurrentSize, double value, std::string filename) //пишем время выполнения в файл

{

std::ofstream fd(filename, std::ios\_base::app); //для записи в конец

fd<< CurrentSize<< "\t"<<std::fixed <<std::setprecision(5)<<value<<std::endl;

// fd<<std::endl;

fd.close();

}

};

#endif //STATISTICSGRAPH\_INL\_INCLUDED

* 1. ShortestPathFunc.h

#ifndef SHORTESTPATHFUNC\_H\_INCLUDED

#define SHORTESTPATHFUNC\_H\_INCLUDED

#include <vector>

#include <limits>

#include "Func.h"

#include "presenthandler.h"

//matrix - матрица смежности

void FloydWarshall(Graph& Graph1, std::vector<std::vector<int>>& MPathLength, std::vector<std::vector<int>>& MPath)

{

for(int i=0; i< Graph1.Edges.size(); i++) //заполняем матрицу стартовыми значениями.

{

for(int j=0; j< Graph1.Edges.size(); j++)

{

MPath.at(i).at(j)=j; //граф у нас связный, так что будем надеяться, что какой-либо путь алгоритм найдет.

if(Graph1.Edges.at(i).at(j).Adjacency==0) //можно обойтись Adjacency, но алг. будет более громоздким

if(i==j) MPathLength.at(i).at(j)=0; //тут есть подводный. У нас граф вроде бы без петель, но мы подразумеваем, что петли все-таки есть, и их вес равен нулю.

else MPathLength.at(i).at(j)=std::numeric\_limits<int>::max();

else MPathLength.at(i).at(j)=Graph1.Edges.at(i).at(j).Weight;

}

}

//Пробегаемся по всем вершинам и ищем более короткий путь

//через вершину k

for(int k = 0; k < Graph1.Edges.size(); k++) {

for(int i = 0; i < Graph1.Edges.size(); i++) {

for(int j = 0; j < Graph1.Edges.size(); j++) {

//Новое значение ребра равно минимальному между старым

//и суммой ребер i <-> k + k <-> j (если через k пройти быстрее)

if(MPathLength.at(i).at(j)>(long long int)MPathLength.at(i).at(k)+(long long int)MPathLength.at(k).at(j))

{

MPathLength.at(i).at(j)=MPathLength.at(i).at(k)+MPathLength.at(k).at(j);

MPath.at(i).at(j)=MPath.at(i).at(k);

// PrintMatrix(MPath, "MPath");

// getchar();

}

// matrix[i][j] = min(matrix[i][j], matrix[i][k] + matrix[k][j]);

}

}

}

}

void FloydWarshallCallBackParamsTailHandler(int N, std::vector<std::vector<int>>& MPathLength, std::vector<std::vector<int>>& MPath) //чтобы в объекте класса статистики функция работала как нужно.

{

MPathLength.resize(N);

for(auto& i: MPathLength) i.resize(N);

MPath.resize(N);

for(auto& i: MPath) i.resize(N);

}

//Дейкстра==================================

void Dijkstra(Graph& Graph1, int& SourceIndex, std::vector<std::vector<int>>& Pathes, presenthandler& PresentHandler)

{

int MAX\_WEIGHT=std::numeric\_limits<int>::max();

int NULL\_INDEX=std::numeric\_limits<int>::max();

int minWeightIndex; //Индекс вершины с минимальным весом

int minWeight; //Вес вершины с минимальным весом

//Подготовка цветов и весов вершин.

Graph1.IsWithVerticesWeights=true;

Graph1.vertsVertices.ResetColors();

for(auto& i: Graph1.Vertices)

{

i.Weight=MAX\_WEIGHT;

}

Graph1.Vertices.at(SourceIndex).Weight=0;

//Нахождение весов всех вершин. Т.е. минимальных путей от начальной вершины до всех остальных вершин.

do

{

/\* Поиск вершины с минимальным весом\*/

minWeightIndex = NULL\_INDEX;

minWeight = MAX\_WEIGHT;

for (int i = 0; i<Graph1.Vertices.size(); i++)

{

if ((Graph1.Vertices.at(i).Color == 11) && (Graph1.Vertices.at(i).Weight<minWeight)) //Если вершину ещё не обошли и вес меньше min

{ // Переприсваиваем значения

minWeight = Graph1.Vertices.at(i).Weight;

minWeightIndex = i;

}

}

/\*Обход соседей(вершин) из выбранной вершины\*/

// Добавляем найденный минимальный вес к текущему весу вершины и сравниваем с текущим минимальным весом вершины

if (minWeightIndex != NULL\_INDEX) //Если удалось найти непомеченную вершину

{

for (int i = 0; i<Graph1.Vertices.size(); i++)

{

if (Graph1.Edges.at(minWeightIndex).at(i).Weight > 0)

{

int tempWeight = minWeight + Graph1.Edges.at(minWeightIndex).at(i).Weight;

if (tempWeight < Graph1.Vertices.at(i).Weight)

{

Graph1.Vertices.at(i).Weight = tempWeight;

}

}

}

Graph1.Vertices.at(minWeightIndex).Color=13;

}

if(PresentHandler.Mode>=1 && minWeightIndex!=NULL\_INDEX) //чтобы не выводились 2 повторяющиеся картинки в конце

{

Graph1.ShowPlot(PresentHandler.Mode-1, std::string("DijkstraPic").append(PresentHandler.GetFileNumberAndIncrease()));

if(PresentHandler.Mode==1) getchar();

}

} while (minWeightIndex != NULL\_INDEX);

//Нахождение путей от исходной вершины до всех остальных вершин

Pathes.clear();

Pathes.resize(Graph1.Vertices.size());

for(int DestIndex=0; DestIndex<Graph1.Vertices.size(); DestIndex++) //ищем путь до каждой вершины из исходной

{

// Восстановление пути

std::vector<int> tempPath; //вектор, в котором путь от исходной вершины до другой вершины хранится задом наперед.

int CurrentDestIndex=DestIndex; //CurrentDestIndex -величина, в которой мы будем хранить нашу текущую вершину при нахождении пути.

tempPath.push\_back(CurrentDestIndex); // начальный элемент - конечная вершина

// int k = 1; // индекс предыдущей вершины

int DestWeight = Graph1.Vertices.at(CurrentDestIndex).Weight; // вес конечной вершины

while (CurrentDestIndex != SourceIndex) // пока не дошли до начальной вершины

{

for (int i = 0; i<Graph1.Vertices.size(); i++) // просматриваем все вершины

if (Graph1.Edges.at(i).at(CurrentDestIndex).Weight != 0) // если связь есть

{

int tempWeight = DestWeight - Graph1.Edges.at(i).at(CurrentDestIndex).Weight; // определяем вес пути из предыдущей вершины

if (tempWeight == Graph1.Vertices.at(i).Weight) // если вес совпал с рассчитанным

{ // значит из этой вершины и был переход

DestWeight = tempWeight; // сохраняем новый вес

CurrentDestIndex = i; // сохраняем предыдущую вершину

tempPath.push\_back(i); // и записываем ее в массив

}

}

}

//переворачиваем и записываем tempPath

while(tempPath.size()!=0)

{

Pathes.at(DestIndex).push\_back(tempPath.back());

tempPath.pop\_back();

}

}

if(PresentHandler.Mode>=1)

{

PrintMatrix(Pathes, "Pathes");

}

}

void DijkstraCallBackParamsTailHandler(int N, int& SourceIndex, std::vector<std::vector<int>>& Pathes, presenthandler& PresentHandler)

{

return; //Данная функция ничего не делает

}

void BellmanFord(Graph& Graph1, int& s, std::vector<int>& x, std::vector<std::vector<int>>& D, presenthandler& PresentHandler) //s- индекс исходной вершины, X -массив со значениями кратчайших путей от исходной вершины до вершины i. D -маршрутизация. каждая строка - маршрут до соответствующей вершины. 0 ячейка - число вершин в маршруте, остальные - индексы вершин в маршруте.

{

x.clear(); //Очищаем и устанавливаем размер выходных параметров.

x.resize(Graph1.Vertices.size());

D.clear();

D.resize(Graph1.Vertices.size());

for(int i=0; i<Graph1.Edges.size(); i++) //В матрице весов веса, соответствующие отсутствию ребер делаем большими

{

for(int j=0; j<Graph1.Edges.size(); j++)

{

if(Graph1.Edges.at(i).at(j).Adjacency==0)

{

if(i==j) Graph1.Edges.at(i).at(j).Weight=0;

else Graph1.Edges.at(i).at(j).Weight=std::numeric\_limits<int>::max();

}

}

}

for(int i=0; i<Graph1.Vertices.size(); i++)

{

x.at(i)=Graph1.Edges.at(s).at(i).Weight;

}

for(int i=0; i<Graph1.Vertices.size(); i++) //D[i][0]=2; D[i][1]=s; D[i][2]=i; Кроме D[s][0]=1;

{

D.at(i).push\_back(2);

D.at(i).push\_back(s);

D.at(i).push\_back(i);

}

D.at(s).at(0)=1;

D.at(s).pop\_back();

for(int k=0; k<Graph1.Vertices.size()-2; k++) //k -число пересадок на нашем пути. При каждой итерации вычилсяется минимальный путь с не более, чем k пересадками на этом пути. Максимальное число пересадок n-2; (в исходном алгоритме почему-то на одно выполнение больше. Или есть ещё какой-то нюанс, или это просто ошибка.)

{

for(int i=0; i<Graph1.Vertices.size(); i++) //Пытаемся улучшить путь из s в i.

{

for(int j=0; j<Graph1.Vertices.size(); j++) //Ищем такую пересадку, чтобы путь был минимальным

{

if(x.at(i)>(long long int)x.at(j)+(long long int)Graph1.Edges.at(j).at(i).Weight)

{

x.at(i)=x.at(j)+Graph1.Edges.at(j).at(i).Weight;

D.at(i)=D.at(j);

D.at(i).at(0)=D.at(j).at(0)+1;

D.at(i).push\_back(i); //D[i,D[i,0]]:=i;

}

}

}

}

if(PresentHandler.Mode>=1)

{

PrintMatrix(D, "BellmanFord D");

std::cout <<"x=";

for(auto& i: x) std::cout<<i<<"\t";

std::cout<<std::endl;

}

}

void BellmanFordCallBackParamsTailHandler(int N, int& s, std::vector<int>& x, std::vector<std::vector<int>>& D, presenthandler& PresentHandler)

{

return; //Данная функция ничего не делает

}

void AlgFuncExample(Graph& Graph1, int& param1, int& param2, char& param3) //Функция для демонсрации работы класса StatisticGraph

{

//Выполнить алгоритм над графом Graph1

return;

}

void AlgFuncExampleCallBackParamsTailHandler(int N, int& param1, int& param2, char& param3)

{

//Настраиваем значения param1, param2 и param3 к последующей передачи в функцию AlgFuncExample

//В данном случае в этом нет необходимости, поэтому тело функции оставляем пустым

return; //Данная функция ничего не делает

}

#endif //SHORTESTPATHFUNC\_H\_INCLUDED

* 1. PlotScript.bash

#! /usr/bin/bash

filelistVar=$(gawk -F '\t' '{printf "%s", $2}' Plotfilelist.txt)

cat > PlotScript.gpi << HEREDOC1

#! /usr/bin/gnuplot -persist

# Будем этим скриптом строить графики

#Тут неизменяемые настройки

title\_f(f,a,u,v,b)= sprintf('%s=%.12f\*x\*log2(%.12fx+%.12f)+%.12f', f, a, u,v, b)

set grid

set datafile separator '\t'

#set terminal png size 1024, 768

#set output "LabPlot.png"

#set terminal svg enhanced size 3,2

#set output 'file.svg'

#Этот способ рабочий. Потом нужно преобразовать к .PNG с помощью команды. and then use imagemagick to convert the vector format to a .png: convert -verbose -density 300 -trim file.pdf -quality 100 -flatten -sharpen 0x1.0 file.png

#set terminal pdfcairo size 6, 5 #этот неплохо работает. Размер в дюймах

#set output 'file.pdf'

#для графиков данных

set linestyle 1 lc rgb "#0000FF" lw 2.5

set linestyle 2 lc rgb "#00FF00" lw 2.5

set linestyle 3 lc rgb "#FF0000" lw 2.5

#для линий треда

set linestyle 11 lc rgb "#000080" lw 1.25 dashtype 2

set linestyle 12 lc rgb "#008000" lw 1.25 dashtype 2

set linestyle 13 lc rgb "#800000" lw 1.25 dashtype 2

#Тут изменяемые настройки

#Положение легенды

#set key left

set key below

set title "Lab5" font "Helvetica Bold, 14"

set xlabel "N - number of graph vertices"

set ylabel "Time - average executing time, seconds"

set yrange [0:\*]

#set xrange[-pi:pi]

#plot sin(x) title "sinux" lc rgb "red", cos(x) title "cosinus" lc rgb "green"

#plot "HeapSortOut.txt" w lp lc 3 pt 7 ps 30 smooth bezier

#plot "HeapSortOut.txt" w lp lc 2 pt 1 ps 2 smooth acsplines

# The equation log(x) -логарифм натуральный

#f1(x) = a1\*x\*log(u1\*x+v1)/log(2)+b1

#f1(x)=(a1\*x\*log(u1\*x))/log(2)+b1

# The fit

#fit f1(x) "HeapSortOut.txt" u 1:2 via a1,b1, u1

#f2(x) = a2\*x\*log(u2\*x+v2)/log(2)+b2

#fit f2(x) "QuickSortHoareOut.txt" u 1:2 via a2,b2, u2, v2

#f3(x) = a3\*x\*log(u3\*x+v3)/log(2)+b3

#fit f3(x) "ShellSortOut.txt" u 1:2 via a3,b3, u3, v3

#listsize=5

array filelist1[5]

filelist=$filelistVar

i=1

do for [file in filelist] {

#filelist(t)="FloydWarshellOut.txt"

#filelist(t)=gawk 'FNR=t{printf "%s", }' filelist.txt

filelist1[i]=sprintf('%s', file);

#plot file using 1:2 smooth csplines with lines linestyle i

i=i+1

}

#plot \ "FloydWarshellOut.txt" using 1:2 notitle smooth csplines with lines linestyle 1, \

# 1 / 0 title "FloydWarshell" with lines linestyle 1

# f1(x) title title\_f("f1", a1,u1, 0,b1) with lines linestyle 11,\

# \

# "QuickSortHoareOut.txt" using 1:2 notitle smooth csplines with lines linestyle 2, \

# 1 / 0 title "QuickSortHoare" with lines linestyle 2, \

# f2(x) title title\_f("f2",a2, u2, v2,b2) with lines linestyle 12, \

# \

# "ShellSortOut.txt" using 1:2 notitle smooth csplines with lines linestyle 3, \

# 1 / 0 title "ShellSort" with lines linestyle 3, \

# f3(x) title title\_f("f3", a3,u3, v3, b3) with lines linestyle 13

plot for [j=1:i-1] filelist1[j] using 1:2 smooth csplines with lines linestyle j title filelist1[j]

#Справка

# \ позволяет записать одну строку как несколько

# -persist позволяет держать окно открытым после завершения скрипта

#

#

#

HEREDOC1

chmod +x PlotScript.gpi

./PlotScript.gpi

* 1. PlotGraph.bash

#! /usr/bin/bash

#Скрипт генерирует скрипт для gnuplot. Это необходимо, т.к. gnuplot не достаточно хорошо умеет работать с переменными.

#Тут будут переменные, которые влияют на генерацию.

# -F разделитель полей

nmtitle=$(gawk -F "\t" 'FNR==2{printf "%s", $1}' VarsForScript.dat)

IsOriented=$(gawk -F "\t" 'FNR==2{printf "%s", $2}' VarsForScript.dat)

IsSavePictureToFile=$(gawk -F "\t" 'FNR==2{printf "%s", $3}' VarsForScript.dat)

PictureName=$(gawk -F "\t" 'FNR==2{printf "%s", $4}' VarsForScript.dat)

IsWithVerticesWeights=$(gawk -F "\t" 'FNR==2{printf "%s", $5}' VarsForScript.dat)

#gawk передать переменную gnuplot не удастся, наверное. Проще сделать так.

VerticesRadius=2

VerticesWeightsOffset=$VerticesRadius+2

if [[ $IsOriented -eq 0 ]]

then

#Граф неориентированный

WeightPosVar=0.5

else

WeightPosVar=0.7

fi

#Чтобы можно было отключать показ весов вершин.

loadVerticesWeightsVar="loadVerticesWeights= \"1 / 0\""

if [[ $IsWithVerticesWeights -ne 0 ]]

then

#Если нужно с весами

read -r -d '' loadVerticesWeightsVar<<HEREDOCWEIGHTS1

loadVerticesWeights = sprintf('< gawk '' {printf "%%f\t%%f\t%%s\n", \$2, \$3 + $VerticesWeightsOffset, \$5} '' %s', flePnts);

HEREDOCWEIGHTS1

else

#Отправляем читать gawk заведомо пустой столбец 10

read -r -d '' loadVerticesWeightsVar<<HEREDOCWEIGHTS2

loadVerticesWeights = sprintf('< gawk '' {printf "%%f\t%%f\t%%s\n", \$2, \$3 + $VerticesWeightsOffset, \$10} '' %s', flePnts);

HEREDOCWEIGHTS2

fi

SavePictureVar=""

if [[ $IsSavePictureToFile -ne 0 ]]

then

read -r -d '' SavePictureVar<<ADDTEXT0

set terminal pdfcairo size 6, 5 #этот неплохо работает. Размер в дюймах

set output "$PictureName.pdf"

ADDTEXT0

fi

if [[ $IsOriented -eq 0 ]]

then

#Граф неориентированный

#HereDoc помещаем в перемнную. По read: -r Не позволять экранировать символы обратным слешем -d delimiter разделитель между вводимыми величинами

read -r -d '' plotvar<<ADDTEXT1

#loadEdges=x первой веришны; y первой вершины; Color; x второй веришны; y второй вершины; Color

loadEdges = sprintf('< gawk '' FNR==NR{x[\$1]=\$2;y[\$1]=\$3;next;} {printf "%%f\t%%f\t%%d\n%%f\t%%f\t%%d\n\n", x[\$1], y[\$1], \$4, x[\$2], y[\$2], \$4;} '' %s %s', flePnts, fleEdges);

plot \

loadEdgesLoops using 1:2:(EdgesLoopRadius):3 with circles lc var notitle, \

loadEdges using 1:2:3 with lines lc var notitle, \

flePnts using 2:3:($VerticesRadius):4 with circles fill solid lc var notitle, \

flePnts using 2:3:($VerticesRadius) with circles lc rgb "black" notitle, \

flePnts using 2:3:1 with labels tc rgb "black" font "Arial Bold" notitle, \

loadWeightsNoLoops using 1:2 with points pt 5 lc rgb "white" notitle, \

loadWeightsNoLoops using 1:2:3 with labels tc rgb "black" center font "Arial Bold" notitle, \

loadWeightsLoops using 1:2 with points pt 5 lc rgb "white" notitle, \

loadWeightsLoops using 1:2:3 with labels tc rgb "black" center font "Arial Bold" notitle, \

loadVerticesWeights using 1:2:3 with labels tc rgb "black" center font "Arial Bold" notitle, \

1 / 0 title $nmtitle with points lc rgb "white", \

1 / 0 title "$IsOriented" with points lc rgb "white", \

1 / 0 title "$IsSavePictureToFile" with points lc rgb "white"

ADDTEXT1

else

#Граф ориентированный

read -r -d '' plotvar<<ADDTEXT2

#loadEdges=x первой веришны; y первой вершины; Color; dx(векторы требуют такой формат); dy; Color x[\$2]-x[\$1] -это координата векторов без учета радиуса вершины графа. +0.01 нужно, чтобы в случае, когда вершини лежат друг на друге, не было деления на 0.

loadEdges = sprintf('< gawk '' FNR==NR{x[\$1]=\$2;y[\$1]=\$3;next;} {printf "%%f\t%%f\t%%f\t%%f\t%%d\n\n", x[\$1], y[\$1], (x[\$2]-x[\$1])\*(1-$VerticesRadius/sqrt((x[\$2]-x[\$1])\*(x[\$2]-x[\$1])+(y[\$2]-y[\$1])\*(y[\$2]-y[\$1])+0.01)), (y[\$2]-y[\$1])\*(1-$VerticesRadius/sqrt((x[\$2]-x[\$1])\*(x[\$2]-x[\$1])+(y[\$2]-y[\$1])\*(y[\$2]-y[\$1])+0.01)), \$4;} '' %s %s', flePnts, fleEdges);

plot loadEdgesLoops using 1:2:(EdgesLoopRadius):3 with circles lc var notitle, \

loadEdges using 1:2:3:4:5 with vectors arrowstyle var notitle, \

flePnts using 2:3:($VerticesRadius):4 with circles fill solid lc var notitle, \

flePnts using 2:3:($VerticesRadius) with circles lc rgb "black" notitle, \

flePnts using 2:3:1 with labels tc rgb "black" font "Arial Bold" notitle, \

loadWeightsNoLoops using 1:2 with points pt 5 lc rgb "white" notitle, \

loadWeightsNoLoops using 1:2:3 with labels tc rgb "black" center font "Arial Bold" notitle, \

loadWeightsLoops using 1:2 with points pt 5 lc rgb "white" notitle, \

loadWeightsLoops using 1:2:3 with labels tc rgb "black" center font "Arial Bold" notitle, \

loadVerticesWeights using 1:2:3 with labels tc rgb "black" center font "Arial Bold" notitle, \

1 / 0 title $nmtitle with points lc rgb "white", \

1 / 0 title "$IsOriented" with points lc rgb "white", \

1 / 0 title "$IsSavePictureToFile" with points lc rgb "white"

ADDTEXT2

fi

cat > PlotGraph.gpi <<ADDTEXT3

#! /usr/bin/gnuplot -persist

#Константы

LOOPOFFSET='5'

LOOPWEIGHTOFFSET='10'

EdgesLoopRadius=5;

set style line 1 lc rgb '#000000' lw 1 #Для ребер

set style line 2 lc rgb "green" lw 1

set style line 3 lc rgb "blue" lw 1

set style line 4 lc rgb "red" lw 1

set style arrow 1 head filled size 4,12,36 ls 1

set style arrow 2 head filled size 4,12,36 ls 2

set style arrow 3 head filled size 4,12,36 ls 3

set style arrow 4 head filled size 4,12,36 ls 4

set style line 11 lc rgb "white" #Для узлов

set style line 12 lc rgb "green"

set style line 13 lc rgb "blue"

set style line 14 lc rgb "red"

WeightOffset='0' #на петли не виляет

#Имена файлов с индексами и координатами точек; с информацией о ребрах; с легендой

flePnts = 'Vertices.dat'

fleEdges = 'EdgesForPlot.dat'

#GraphInfo= 'GraphInfo.dat'

#Отображаемая область xrange

x1=50

y1=50

set xr [-x1:x1]

set yr [-y1:y1]

#Возможность читать стили из файла. (Чтобы определять, какими цветами будут окрашены узлы извне. Сами цвета устанавливаются тут)

set style increment user

#Устанавливаем одинаковый масштаб по осям

set size square

#вот тут сложно. sprintf здесь это команда gnuplot. '' здесь кодируют одниночную кавычку. '< gawk '' FNR==NR{x[\$1]=\$2;y[\$1]=\$3;next;} {printf "%%f\t%%f\n%%f\t%%f\n\n", x[\$1], y[\$1], x[\$2], y[\$2];} '' %s %s'

#-здесь это форматная строка sprintf, которая как и положено заключена в одинарные кавычки. %s %s это, собственно, спецификаторы формата в этой строке (больше спецификаторов формата эта строка не имеет)

#flePnt fleEdges - это аргументы, подставляемые вместо спецификаторов формата.

#остальные команды относится сугубо к утилите gawk. (при построении графика вся вот эта длинная строчка вызывается в теримнале).

#FNR==NR сравнение числа обработанных строк из текущего файла с числом строк, обработанных за все время работы. Вместе с командой next позволяет выполнять действие x[\$1]=\$2;y[\$1]=\$3; только для первого файла в списке файлов

#Т.е. x[\$1]=\$2;y[\$1]=\$3; заполняет массивы x и y значениями 2 и 3 столбцов файла pnts.dat Запись x[\$1] означает, что индексы берутся из 1 столбца файла.После чего выполняется действие {printf "%%f\t%%f\n%%f\t%%f\n\n", x[\$1], y[\$1], x[\$2], y[\$2];} уже для второго файла

#Во втором файле у нас в 1 и 2 строках номера вершин, связянных друг с другом. т.е. в выходной строке у нас координаты начала и конца линий. Причем отдельные линии раделяются между собой пустой строкой

#printf в данном случае также команда gawk, а не bush или gnuplot

loadEdgesLoops = sprintf('< gawk '' FNR==NR{x[\$1]=\$2;y[\$1]=\$3;next;} {if(\$1==\$2) printf "%%f\t%%f\t%%d\n", x[\$1], y[\$1]-%s, \$4;} '' %s %s', LOOPOFFSET, flePnts, fleEdges);

$loadVerticesWeightsVar

# gawk ' FNR==NR{x[\$1]=\$2;y[\$1]=\$3;next;} {printf "%%f\t%%f\n%%f\t%%f\n\n", x[\$1], y[\$1], x[\$2], y[\$2];} ' pnts.dat edges.dat

loadWeightsNoLoops = sprintf('< gawk '' FNR==NR{x[\$1]=\$2;y[\$1]=\$3;next;} {if(\$1!=\$2 && \$2!="") printf "%%f\t%%f\t%%s\n", x[\$1]+(x[\$2]-x[\$1])\*$WeightPosVar, y[\$1]+(y[\$2]-y[\$1])\*$WeightPosVar - %s, \$3} '' %s %s',WeightOffset, flePnts, fleEdges);

loadWeightsLoops = sprintf('< gawk '' FNR==NR{x[\$1]=\$2;y[\$1]=\$3;next;} {if(\$1==\$2) printf "%%f\t%%f\t%%s\n", (x[\$1]+x[\$2])/2, (y[\$1]+y[\$2])/2 - %s, \$3} '' %s %s',LOOPWEIGHTOFFSET, flePnts, fleEdges);

#loadGraphInfo = sprintf('< gawk '' {n=\$1;m=\$2;} {printf "n=%%d\tm=%%d", n, m} '' %s', GraphInfo);

#Тут номера столбцов не везде совпадают с номерами столбцов в исходном файле. Т.к. gawk формирует строку, которая воспринимается как файл. И у неё свои столбцы

#Будем ли выводить в файл. (полезно для серии изображений)

$SavePictureVar

#Немного через , но сойдет.

$plotvar

#plot \

# loadEdges using 1:2 with lines lc rgb "black" lw 2 notitle, \ #рисуем

# flePnts using 2:3:(0.6) with circles fill solid lc rgb "black" notitle, \ #рисуем кружки(узлы) (0.6) размер кружков

# flePnts using 2:3:1 with labels tc rgb "white" font "Arial Bold" notitle, \ #рисуем текст в кружочках(узлах)

# loadWeights using 1:2:3 with labels tc rgb "red" center font "Arial Bold" notitle

#Справка

# \ позволяет записать одну строку как несколько

# -persist позволяет держать окно открытым после завершения скрипта

# gawk -утилита для работы с файлами

#

#

ADDTEXT3

chmod +x PlotGraph.gpi

#Избавляемся от надоедливых варнингов, перенаправляя вывод в этот файл.

#./PlotGraph.gpi 2>/dev/null

./PlotGraph.gpi

#Конвертируем картинку

if [[ $IsSavePictureToFile -eq 1 ]]

then

convert -verbose -density 300 -trim $PictureName.pdf -quality 100 -flatten -sharpen 0x1.0 $PictureName.png

fi

* 1. main\_GenerateGraph\_Demo.cpp

#include <iostream>

//#include <filesystem> //для создания директорий

#include "presenthandler.h"

#include "Func.h"

#include "ShortestPathFunc.h"

#include "Graph.h"

#include "StatisticsGraph.inl"

#include "plothandler.h"

int main(int, char\*\*) {

// int seed=time(0);

int seed=4; //с seed 4 получается 3 итерации в Adjacency при n=5 m=4

std::default\_random\_engine generator1(seed);

int N=6;

presenthandler PresentHandler; //для отображения шагов

PresentHandler.Mode=2;

plothandler PlotHandler;

Graph Graph1(N, false);

Graph1.IsWithVerticesWeights=false;

//GenerateAdjacencyProb(Graph1, 0.5, generator1, false);

//GenerateAdjacencyMNumberNoLoops(Graph1, 4, generator1);

// try{ GenerateAdjacencyMNumberNoLoops(Graph1, 4, generator1); }

// catch(std::string str)

// {

// std::cout<<"exeption:"<<str<<std::endl;

// return -1;

// }

/\* int iteration=0;

do

{

std::cout<<"Generating Adjacency: iteration: "<<iteration<<std::endl;

try {GenerateAdjacencyMNumberNoLoops(Graph1, 4, generator1);}

catch(std::string str)

{

std::cout<<"exeption:"<<str<<std::endl;

return -1;

}

iteration++;

}while(!IsConnectedDFS(Graph1, 0, PresentHandler));

GenerateWeights(Graph1, generator1, 1, 100); //все норм. Если seed принял какое-либо значение, то им определяется количество попыток, которые будут сделаны, чтобы получить Adjacency однозначно, поэтому при данном seed Weight будет также однозначным

\*/

GenerateGraph(Graph1, 6, 100, generator1, PresentHandler);

Graph1.ShowPlot(1, "Pic");

/\* std::vector<std::vector<int>> MPath(N), MPathLength(N);

for(auto& i: MPath) i.resize(N);

for(auto& i: MPathLength) i.resize(N);

FloydWarshall(Graph1, MPathLength, MPath);

Graph1.PrintEdges();

PrintMatrix(MPathLength, "MPathLength");

PrintMatrix(MPath, "MPath");

\*/

}

* 1. main\_StatisticsGraph\_SimpleDemo.cpp

#include <iostream>

#include "Graph.h"

#include "StatisticsGraph.inl"

#include "ShortestPathFunc.h"

#include "presenthandler.h"

#include "plothandler.h"

#include "Func.h" //GenerateGraph();

int main(int, char\*\*) {

Graph Graph1(5);

std::default\_random\_engine generator1;

presenthandler PresentHandler0;

plothandler PlotHandler;

int param1;

int param2;

char param3;

StatisticsGraph(5, 10, 1, 0.5, 100, 1, generator1, PresentHandler0, PlotHandler.addandreturn("AlgFuncExampleOut.txt"), GenerateGraph, AlgFuncExample, AlgFuncExampleCallBackParamsTailHandler, param1, param2, param3);

return 0;

}

* 1. main\_VisualGraphDemo.cpp

#include <iostream>

#include "Graph.h"

int main(int, char\*\*) {

Graph Graph1(25);

// Graph1.IsOriented=true;

Graph1.IsWithVerticesWeights=true;

// Graph1.Edges.at(0).at(0).Adjacency=true;

int x[]={11,3,3,12,12,6, 18, 15, 15, 21, 27, 27, 21, 21, 27, 31,31,37,37,31, 41,41,41,47,47};

int y[]={17,17,30,30,24,24, 21,21,30, 21,21,30,30,25,25, 34, 21,21,30,30, 17,30,21,21,30};

for(int i=0; i<Graph1.Vertices.size(); i++)

{

Graph1.Vertices.at(i).x=(x[i]-25)\*2; Graph1.Vertices.at(i).y=-(y[i]-25)\*2;

Graph1.Vertices.at(i).Weight=i\*10;

if(i<=5)

{

Graph1.Vertices.at(i).Color=14;

}

if(i>=6 && i<=8)

{

Graph1.Vertices.at(i).Color=12;

}

if(i>=9 && i<=14)

{

Graph1.Vertices.at(i).Color=13;

}

if(i>=20 && i<=24)

{

Graph1.Vertices.at(i).Color=14;

}

}

for(int i=0; i<5; i++) //G

{

Graph1.Edges.at(i).at(i+1).Adjacency=true;

Graph1.Edges.at(i).at(i+1).Color=3;

Graph1.Edges.at(i).at(i).Adjacency=true;

Graph1.Edges.at(i).at(i).Color=3;

}

Graph1.Edges.at(5).at(5).Adjacency=true;

Graph1.Edges.at(5).at(5).Color=3;

for(int i=6; i<8; i++) //r

{

Graph1.Edges.at(i).at(i+1).Adjacency=true;

Graph1.Edges.at(i).at(i+1).Color=2;

}

for(int i=9; i<14; i++) //a

{

Graph1.Edges.at(i).at(i+1).Adjacency=true;

Graph1.Edges.at(i).at(i+1).Color=4;

}

for(int i=15; i<19; i++) //p

{

Graph1.Edges.at(i).at(i+1).Adjacency=true;

}

for(int i=20; i<24; i++) //h

{

Graph1.Edges.at(i).at(i+1).Adjacency=true;

}

Graph1.ShowPlot(1, "GRAPH", false);

return 0;

}

* 1. main\_Dijkstra\_Demo.cpp

#include <iostream>

//#include <fstream>

//#include <sstream>

//#include <string>

//#include <iomanip> //setprecision

//#include <random>

//#include <filesystem> //для создания директорий

#include "presenthandler.h"

#include "Func.h"

#include "ShortestPathFunc.h"

//#include "BaseGraph.h"

//#include "VisualGraph.h"

#include "Graph.h"

#include "StatisticsGraph.inl"

#include "plothandler.h"

int main(int, char\*\*) {

// int seed=time(0);

int seed=4; //с seed 4 получается 3 итерации в Adjacency при n=5 m=4

std::default\_random\_engine generator1(seed);

presenthandler PresentHandler; //для отображения шагов

PresentHandler.Mode=1;

int N=10;

Graph Graph1(N);

Graph1.IsWithVerticesWeights=true;

GenerateGraph(Graph1, 11, 100, generator1, PresentHandler);

Graph1.ShowPlot(0, "Pic");

PresentHandler.Mode=1;

std::vector<std::vector<int>> Pathes;

int SourceIndex=0;

Dijkstra(Graph1, SourceIndex, Pathes, PresentHandler);

// Graph1.ShowPlot(false, "");

PrintMatrix(Pathes, "Pathes");

}

* 1. main\_GenerateAdjacency.cpp

#include <iostream>

#include "Graph.h"

#include "Func.h"

int main(int, char\*\*) {

std::default\_random\_engine generator;

Graph Graph1(10);

// Graph1.IsOriented=true;

for(int k=0; k<10000; k++)

{

GenerateAdjacencyProb(Graph1, 0.3, generator, false);

for(auto& i:Graph1.Edges) //Увеличиваем вес на 1 и сбрасываем матрицу смежности

{

for(auto& j:i)

{

if(j.Adjacency==1)

{

j.Weight++;

j.Adjacency=0;

}

}

}

}

for(int i=0; i<Graph1.Edges.size(); i++) //Увеличиваем вес на 1 и сбрасываем матрицу смежности

{

for(int j=0; j<Graph1.Edges.at(i).size(); j++)

{

if(i!=j) Graph1.Edges.at(i).at(j).Adjacency=1;

}

}

Graph1.ShowPlot();

Graph1.PrintEdges();

return 0;

}

* 1. main\_BellmanFord\_Demo.cpp

#include <iostream>

//#include <filesystem> //для создания директорий

#include "presenthandler.h"

#include "Func.h"

#include "ShortestPathFunc.h"

#include "Graph.h"

#include "StatisticsGraph.inl"

#include "plothandler.h"

int main(int, char\*\*) {

// int seed=time(0);

int seed=4; //с seed 4 получается 3 итерации в Adjacency при n=5 m=4

std::default\_random\_engine generator1(seed);

presenthandler PresentHandler; //для отображения шагов

PresentHandler.Mode=0;

int N=5;

Graph Graph1(N);

Graph1.IsWithVerticesWeights=true;

GenerateGraph(Graph1, 6, 100, generator1, PresentHandler);

Graph1.ShowPlot(0, "Pic");

PresentHandler.Mode=1;

std::vector<int> x;

std::vector<std::vector<int>> D;

int s=0;

PresentHandler.Mode=1;

BellmanFord(Graph1, s, x, D, PresentHandler);

Graph1.ShowPlot();

/\* PrintMatrix(D, "BellmanFord D");

std::cout <<"x=";

for(auto& i: x) std::cout<<i<<"\t";

std::cout<<std::endl; \*/

}

* 1. main.cpp

#include <iostream>

//#include <fstream>

//#include <sstream>

//#include <string>

//#include <iomanip> //setprecision

//#include <random>

//#include <filesystem> //для создания директорий

#include "presenthandler.h"

#include "Func.h"

#include "ShortestPathFunc.h"

//#include "BaseGraph.h"

//#include "VisualGraph.h"

#include "Graph.h"

#include "StatisticsGraph.inl"

#include "plothandler.h"

int main(int, char\*\*) {

// int seed=time(0);

int seed=4; //с seed 4 получается 3 итерации в Adjacency при n=5 m=4

std::default\_random\_engine generator1(seed), generator2(seed), generator3(seed); //будут генерировать одинаковые последовательности.

int N=5;

presenthandler PresentHandler0, PresentHandler1; //для отображения шагов

PresentHandler0.Mode=0;

PresentHandler1.Mode=0;

plothandler PlotHandler;

std::vector<std::vector<int>> MPath, MPathLength;

StatisticsGraph(600, 700, 100, 0.5, 100, 1, generator1, PresentHandler0, PlotHandler.addandreturn("FloydWarshellOut.txt"), GenerateGraph, FloydWarshall, FloydWarshallCallBackParamsTailHandler, MPathLength, MPath);

int SourceIndex=0;

std::vector<std::vector<int>> Pathes;

// StatisticsGraph(5, 50, 5, 0.5, 100, 10, generator2, PresentHandler0, PlotHandler.addandreturn("DijkstraOut.txt"), GenerateGraph, Dijkstra, DijkstraCallBackParamsTailHandler, SourceIndex, Pathes, PresentHandler1);

int s=0;

std::vector<int> x;

std::vector<std::vector<int>> D;

// StatisticsGraph(100,600, 100, 0.5, 100, 10, generator3, PresentHandler0, PlotHandler.addandreturn("BellmanFordOut.txt"), GenerateGraph, BellmanFord, BellmanFordCallBackParamsTailHandler, s, x, D, PresentHandler1);

// PlotHandler.tofile("Plotfilelist.txt");

system("./PlotScript.bash");

}