ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 5
Смежный Граф. Минимальное островное дерево Прима

Оглавление

Описание задачи.

Создайте взвешенный граф, состоящий из [10, 20, 50, 100] вершин.

- Каждая вершина графа связана со случайным количеством вершин, минимум с [3, 4, 10, 20].
- Веса ребер задаются случайным значением от 1 до 20.
- Каждая вершина графа должна быть доступна, т.е. до каждой вершины графа должен обязательно существовать путь до каждой вершины, не обязательно прямой.

Для каждого графа требуется провести серию из 5 - 10 тестов, в зависимости от времени затраченного на выполнение одного теста.

Построить минимальное островное дерево взвешенного связного неориентированного графа с помощью алгоритма Прима.

В рамках каждого теста, необходимо замерить потребовавшееся время на выполнение задания из пункта 3 для каждого набора вершин. По окончанию всех тестов необходимо построить график используя полученные замеры времени, где на ось абсцисс (X) нанести N – количество вершин, а на ось ординат(Y) - значения затраченного времени.

Описание метода/модели.

Под графом в математике понимается абстракция реальной системы объектов безотносительно их природы, обладающих парными связями.

Вершина графа – это некоторая точка, связанная с другими точками

Ребро графа – это линия, соединяющая две точки и олицетворяющая связь между ними

Граф – это множество вершин, соединённых друг с другом произвольным образом множеством ребер

Что описывает граф:

- Взаимоотношение между людьми (Социальные связи)
- Иерархические отношения (Подчиненность людей, подразделений и прочего)
- Пути перемещения в любой местности (Карта метро, сеть дорог)
- Взаимозависимости поставщиков услуг или товаров (Поставщики для сборки одного автомобиля)
- Распределенные системы (Любая микросервисная архитектура)

Ориентированный граф

Связанную с каким либо ребром вершину называются инцидентной, это такая вершина которая каким либо образом принадлежит ребру.

Что описывает ориентированный граф:

- Пути распространения информации между людьми (Социальные связи)
- Пути решения и эскалации проблемы в системах ведения задач
- Пути перемещения в любой местности(Карта метро, сеть дорог)
- Предоставления товаров и услуг различным контрагентам (Поставщики для сборки одного автомобиля)
- Распространение ошибки в сложных системах
- Пути распространения расчета в нейронных сетях

Описание графа

Алгоритм удаления выглядит следующим образом:

- Матрица смежности, это двумерная таблица, для которой столбцы и строки соответствуют вершинам, а значения в таблицы соответствуют ребрам, для невзвешенного графа они могут быть просто 1 если связь есть и идет в нужном направлении и 0 если ее нет, а для взвешенного графа будут стоять конкретные значения.
- Матрица инциндетности, это матрица, в которой строки соответствуют вершинам, а столбцы соответствуют связям, и ячейки ставиться 1 если связь выходит из вершины, -1 если входит и 0 во всех остальных случаях.
- Список смежности, это список списков, содержащий все вершины, а внутренние списки для каждой вершины содержат все смежные ей.
- Список ребер, это список строк в которых хранятся все ребра вершины, а внутренние значение содержит две вершины к которым присоединено это ребро.

Обход графа

Обход графа — это переход от одной его вершины к другой в поисках свойств связей этих вершин. Выделяют два вариант обхода, обход в глубину(или поиск в глубину, DFS) и обход в ширину(или поиск в ширину, BFS)

DFS (Deep first search) следует концепции «погружайся глубже, головой вперед» («go deep, head first»). Идея заключается в том, что мы двигаемся от начальной вершины (точки, места) в определенном направлении (по определенному пути) до тех пор, пока не достигнем конца пути или пункта назначения (искомой вершины). Если мы достигли конца пути, но он не является пунктом назначения, то мы возвращаемся назад (к точке разветвления или расхождения путей) и идем по другому маршруту.

DFS (Deep first search) следует концепции «погружайся глубже, головой вперед» («go deep, head first»). Идея заключается в том, что мы двигаемся от начальной вершины (точки, места) в определенном направлении (по определенному пути) до тех пор, пока не достигнем конца пути или пункта назначения (искомой вершины). Если мы достигли конца пути, но он не является пунктом назначения, то мы возвращаемся назад (к точке разветвления или расхождения путей) и идем по другому маршруту.

Поиск в глубину

1. Выбираем любую вершину из еще не пройденных, обозначим ее как u.
2. Запускаем процедуру dfs(u)
3. Помечаем вершину u как пройденную
4. Для каждой не пройденной смежной с u вершиной (назовем ее v) запускаем dfs(v)
5. Повторяем шаги 1 и 2, пока все вершины не окажутся пройденными.

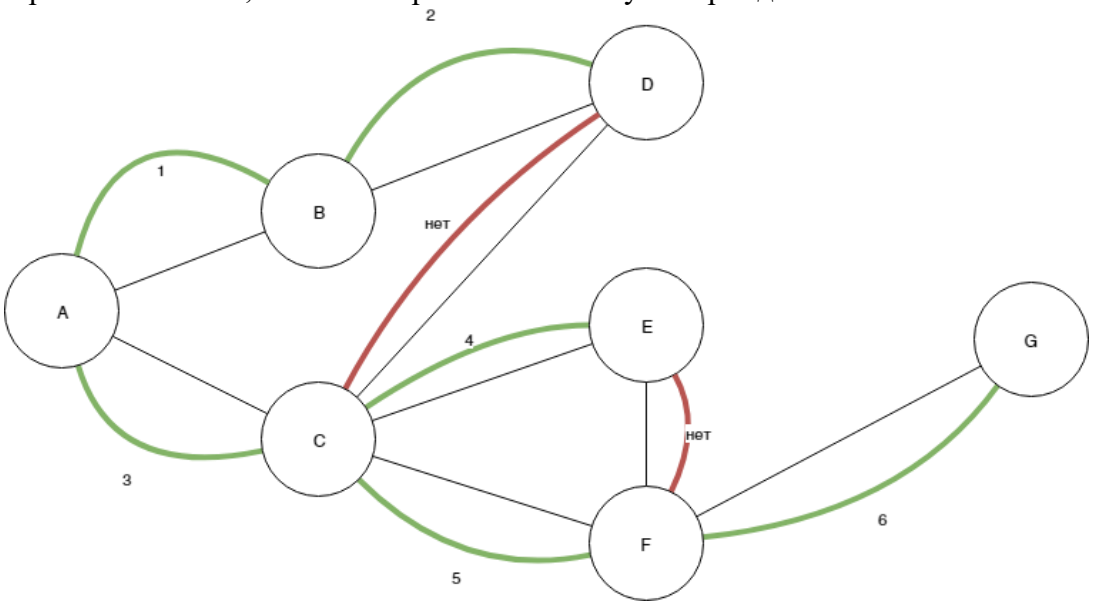

Рис.1 Поиск в глубину.

Сложность алгоритма определяется количеством вершин и количеством ребер в графе, вся процедура вызывается для каждой вершины не более одного раза, а в рамках работы процедуры рассматриваются все ребра, исходящие из вершины.

Поиск в ширину

1. Поместить узел, с которого начинается поиск, в изначально пустую очередь.
2. Извлечь из начала очереди узел u и пометить его как развёрнутый.
3. Если узел u является целевым узлом, то завершить поиск с результатом «успех».
4. В противном случае, в конец очереди добавляются все преемники узла u, которые ещё не развёрнуты и не находятся в очереди.
5. Если очередь пуста, то все узлы связного графа были просмотрены, следовательно, целевой узел недостижим из начального; завершить поиск с результатом «неудача».
6. Вернуться к п. 2.

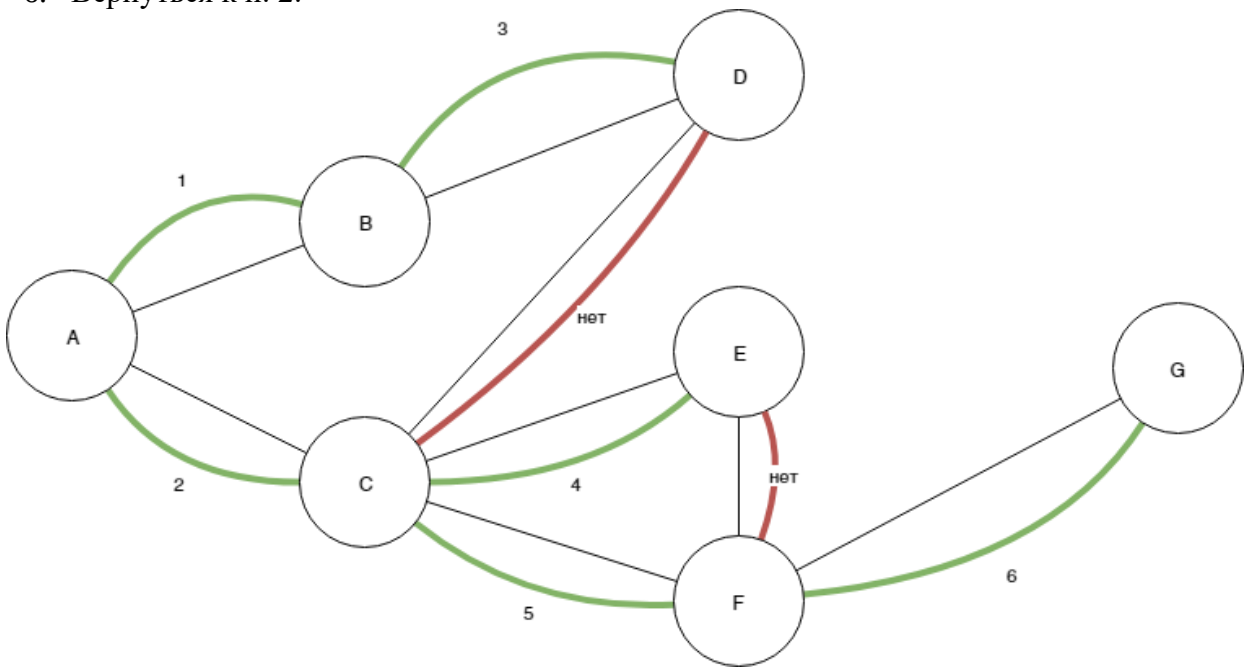

Рис.2 Поиск в ширину.

Сложность алгоритма определяется количеством вершин и количеством ребер в графе, вся процедура вызывается для каждой вершины не более одного раза, а в рамках работы процедуры рассматриваются все ребра, исходящие из вершины.

Поиск в ширину: применение

- Волновой алгоритм поиска пути в лабиринте
- Волновая трассировка печатных плат
- Поиск компонент связности в графе
- Поиск кратчайшего пути между двумя узлами невзвешенного графа
- Поиск в пространстве состояний: нахождение решения задачи с наименьшим числом ходов, если каждое состояние системы можно представить вершиной графа, а переходы из одного состояния в другое — рёбрами графа
- Нахождение кратчайшего цикла в ориентированном невзвешенном графе
- Нахождение всех вершин и рёбер, лежащих на каком-либо кратчайшем пути между двумя вершинами a и b
- Поиск увеличивающего пути в алгоритме Форда-Фалкерсона (алгоритм Эдмондса-Карпа)

Различия двух алгоритмов

Обход графа в ширину и в глубину имеет общие цели, но разные характеристики, вот несколько из них:

- Поиск в ширину и в глубину — это и есть обход графа.
- Поиск в глубину — это поиск по ребрам графа туда-обратно, а поиск в ширину — это плавный «обход по соседям».
- В DFS главное — стек, а в BFS главное — очередь.
- Результат алгоритма поиска в глубину — это некий маршрут, который открывается от стартовой вершины и до искомой. А в алгоритме поиска в ширину маршрут не всегда является результатом.
- DFS является рекурсивным алгоритмом, а BFS — нет.

Взвешенный граф

Взвешенным называют такой граф, каждое ребро которого сопоставимо с каким-либо числовым значением называемым весом графа. Вес ребра в графе может отражать какой-либо параметр в системе, которая этим гарфом описывается. Например, если есть набор пунктов назначения соединённых друг с другом дорогами веса на таком графе могут означать длину этих дорог.

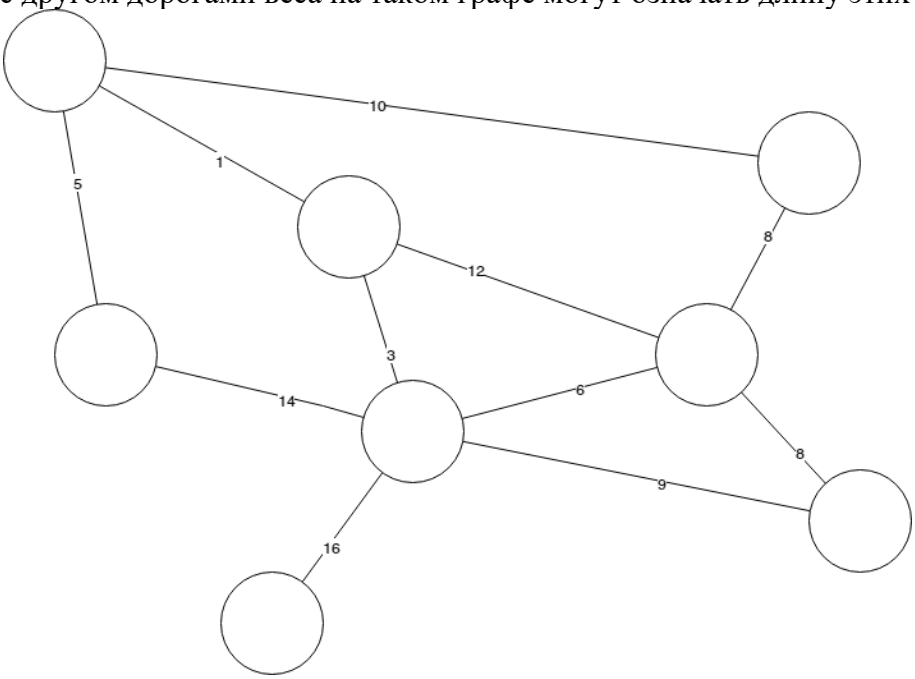

Рис.1. Взвешенный Граф

Островные деревья

Деревом называют такой граф в котором не содержатся циклы, так же, для каждой вершины такого графа существует только один маршрут позволяющий добраться до этой вершины.

Островными деревьями в графе называются такие подмножества ребер графа, которые создают дерево, содержащее все вершины графа.

Для взвешенного графа существует минимальное островное дерево, это такое дерево пути в котором являются минимальными.

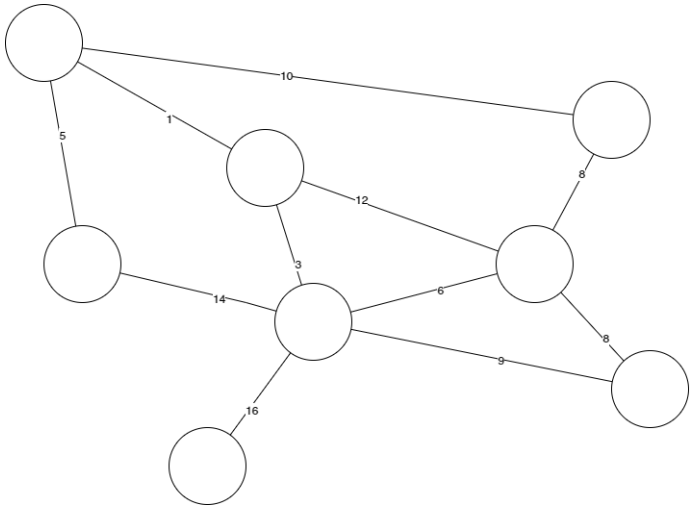

Рис.2.Оригинальный граф

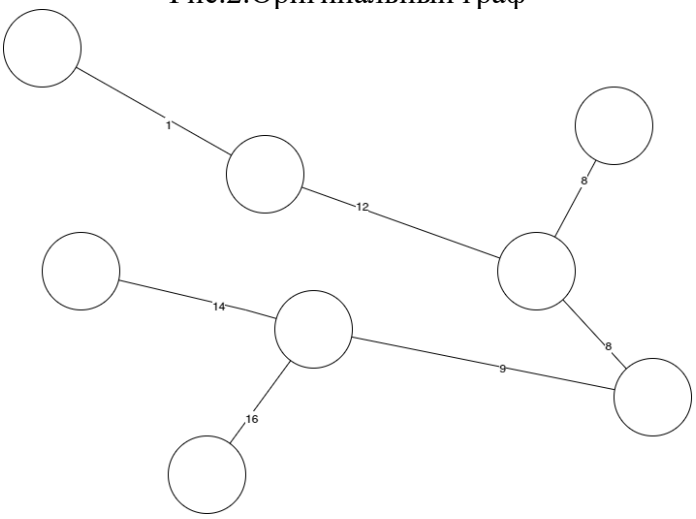

Рис.3.Дерево

Минимальные островные деревья

Для взвешенного графа существует минимальное островное дерево, это такое дерево ребра в котором суммарно имеют наименьший вес из всех существующих деревьев.

Такие деревья позволяют решить задачи соединения множества точек (города, дома, узлы сети) наименьшим объемом каких либо материалов (дорожного полотна, количеством труб, кабеля).

Важно понимать, что для невзвешенного графа любое островное дерево является минимальным, в этом случае его можно найти при помощи алгоритмов обхода в ширину и глубину

Алгоритм Прима

Алгоритм прима для построение минимального островного дерева начинает обход с одной вершины и создает дерево, добавляя по одному ребру, до тех пор пока не будут включены все вершины.

Минимальное-Островное-Дерево-Прима (Граф)

Выбираем произвольную вершину, с которой начинается построение дерева

До тех пор пока остаются вершины не включенные в дерево

Выбираем ребро минимального веса между деревом и вершиной вне дерева

Добавляем выбранное ребро и вершину в итоговое дерево

Такой подход к решению задачи называется жадным алгоритмом, так как оно выбирает лучшее локальное решение игнорируя общую структуру дерева, что чревато издержками на длительность обхода этой структуры.

Жадные алгоритмы могут давать неправильные результаты, для доказательства правильности приведенного алгоритма используется доказательство от обратного, в котором считаем что на некотором шаге произошел неверный выбор, который мог произойти только при равенстве альтернатив, в противном случае был бы выбран другой вариант.

Асимптотическая сложность алгоритма O(n^2)

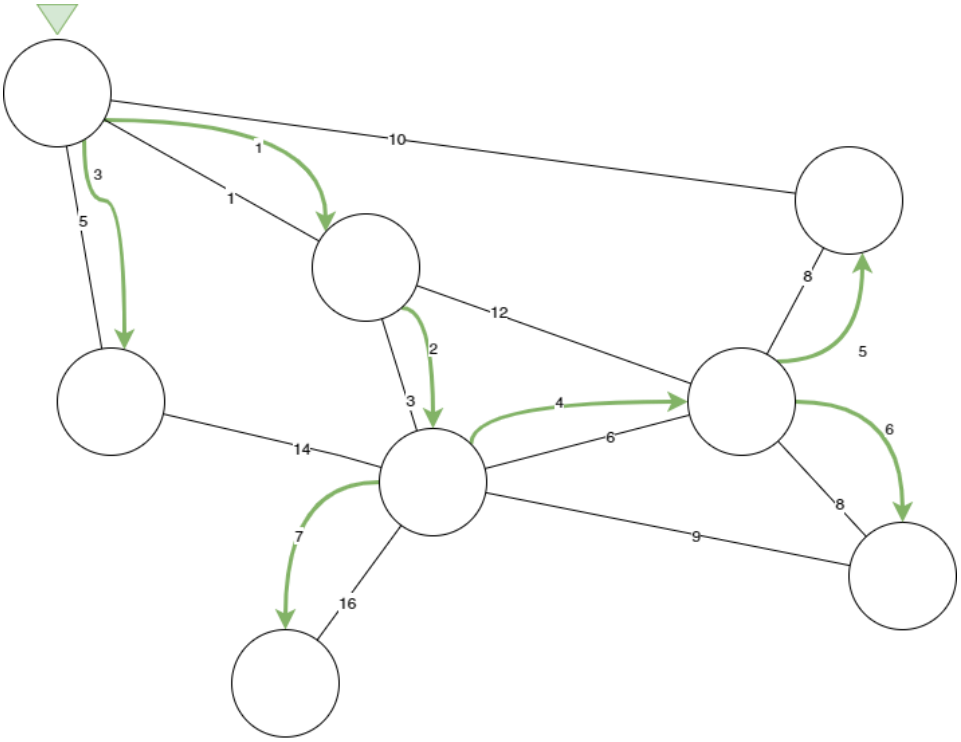

Рис.4. Алгоритм Прима

Начиная от узла помеченного зеленым треугольником, мы начинаем собирать наше минимальное островное дерево.

На каждом шаге мы обходим все ребра, ища то единственное ребро, которое имеет минимальный вес и соединяет две вершины, при этом одна уже входит в наше дерево, а вторая в дерево еще не входит.

После добавления очередного такого ребра мы начинаем поиск заново.

Можно слегка оптимизировать его, и рассматривать на каждом шаге ребра только тех вершин, что уже входят в островное древо.

Выполнение задачи.

Программа для получения тестовых данных была написана на C#.

Код:

```
using System.Diagnostics;
using System.Text;
const int AMMOUNT = 10;//кол-во тстовС
int[] EDGE_COUNT = { 3, 4, 10, 20 };//вероятные рёбра графа
int[] VERT_COUNT = { 10, 20, 50, 100 };//все вершины графа
StreamWriter f = new StreamWriter("test1.txt");
Stopwatch stopwatch1 = new Stopwatch();
//тест
for (int k = 0; k < VERT_COUNT.Length; k++)
{
    f.WriteLine("Вершин: " + VERT_COUNT[k]);
    for (int i = 0; i < AMMOUNT; i++)
    {

        var graph = RandomGraphGen(VERT_COUNT[k], k);
        graph.PrintMatrix();
        stopwatch1.Start();
        var tree = graph.Prim(VERT_COUNT[k]);
        stopwatch1.Stop();
        tree.PrintMatrix();
        f.WriteLine(stopwatch1.ElapsedMilliseconds.ToString());
    }
}
f.Close();

/// <summary>
/// Метод, возвращающий рандомно-сгенерированный граф
/// </summary>
/// <param name="vert">Кол-во вершин</param>
/// <param name="test_number">Номер теста</param>
/// <returns>Рандомно-сгенерированный граф</returns>
/// <exception cref="Exception"></exception>
Graph RandomGraphGen(int vert, int test_number)
{
    var graph = new Graph(false, vert);//создание экземпляра графа
    Random rnd = new Random();
    int j_temp = 0;//подсчет последнего j-того в строке
    for (int i = 0; i < vert; i++)//прохождение по каждой вершине, где на каждой вершине будет
создано n-ное связей ребер и каждая новая вершина будет сдвигаться вправо относительно
прошлой, чтобы у всех элементов был доступ к друг другу
    {
        int edge = rnd.Next(EDGE_COUNT[test_number], vert+1);//Выбор кол-ва рёбер у графа в
диапазоне от минимального кол-ва ребер до кол-ва вершин
        int count_edge = 0; //подсчет ребёр
        for (int j = j_temp; j < vert && count_edge < edge; j++)//прохождение по строке, где
новая вершина стартует с того же места, где была последняя прошлая
        {
            if (graph.adj_matrix[i, j] == 0 && j != i)//если пустая и не равна самой себе
            {
                graph.AddEdge(i+1,j+1, rnd.Next(1, 21));
                count_edge++;
                j_temp = j;
            }
        }
        if (count_edge < edge)//если вправо больше нельзя, но кол-во вершин недонабрано
        {
            for (int j = 0; j < vert && count_edge < edge; j++)
            {
                if (graph.adj_matrix[i, j] == 0 && j != i)
                {
                    graph.AddEdge(i + 1, j + 1, rnd.Next(1, 21));
                    count_edge++;
```

```
                }
            }
        }
    }
    return graph;
}
/// <summary>
/// Класс:Граф
/// </summary>
class Graph
{
    public int vertex_count = 0;
    public int edge_count = 0;
    public int[,] adj_matrix;
    public bool oriented;
    public Graph(bool oriented, int vertex_count)
    {
        this.oriented = oriented;
        this.vertex_count = vertex_count;
        //this.edge_count = edge_count;
        this.adj_matrix = new int[vertex_count, vertex_count];
    }


    /// <summary>
    /// Метод добавления связи в Граф
    /// </summary>
    /// <param name="from">Вершина От</param>
    /// <param name="to">Вершина Куда</param>
    /// <param name="weight">Вес</param>
    public void AddEdge(int from, int to, int weight)
    {
        adj_matrix[from-1,to-1] = weight;
        if (!oriented) adj_matrix[to - 1, from - 1] = weight;
    }
    /// <summary>
    /// Метод минимального островного дерева. Прим
    /// </summary>
    /// <param name="vert">Кол-во вершин</param>
    /// <returns>Дерево</returns>
    public Graph Prim(int vert)
    {
        var tree = new Graph(false, vert);
        List<int> f_vert = new List<int>();//список для хранения уже найденных вершин
        f_vert.Add(0);//добавление первой вершины
        while (f_vert.Count < vert)
        {
            int min_i = -1;
            int min_j = -1;
            int min_weight=1000;
            foreach (var v in f_vert)
            {
                for(int j=0; j<vert; j++)
                {
                    if (adj_matrix[v, j] > 0 && adj_matrix[v,j] < min_weight &&
adj_matrix[v,j]>0 && !f_vert.Contains(j))//поиск минимального ребра
                    {
                        min_weight = adj_matrix[v, j];
                        min_i = v;
                        min_j = j;
                    }
                }
            }
            Console.Write($"\nI={min_i+1}, J={min_j+1} weight={min_weight}");
            f_vert.Add(min_j);//добавление вершины с минимальным ребром
            tree.AddEdge(min_i+1, min_j+1, min_weight);//добавление связи по минимальному
ребру
        }
        return tree;
```

```
    }
    /// <summary>
    /// Меод вывода матрицы смежности
    /// </summary>
    public void PrintMatrix()
    {
        Console.WriteLine("\nМатрица смежности: ");

        for (int i = 0; i < this.vertex_count; i++)
        {
            Console.Write("{0,2}|", i + 1);
            for (int j = 0; j < this.vertex_count; j++)
            {
                Console.Write("{0,3}", adj_matrix[i, j] + " ");
            }

            Console.WriteLine();
        }
        Console.Write("   ");
        for (int j = 0; j < this.vertex_count; j++)
        {
            Console.Write("{0,2}|", j + 1);
        }
    }
}
```

Реализация:

1)Данный Граф реализован через матрицу смежности. Двумерная таблица, для которой столбцы и строки соответствуют вершинам, а значения в таблицы соответствуют ребрам, для невзвешенного графа они могут быть просто 1 если связь есть и идет в нужном направлении и 0 если ее нет, а для взвешенного графа будут стоять конкретные значения.) Создаю класс Графа и внутри него содержатся поля adj_matrix (сама матрица смежности), vertex_count, edge_count для подсчета кол-ва вершин и рёбер соответственно, oriented для флага, указывающего тип графа.

2) Создаю метод AddEdge(отвечает за добавление новых элементов в список рёбер). Если граф неориентированный, значения под нижние диагонали продублируются.

4) создаю методы для вывода PrintMatrix() в консоль.

5)И также метод Prim(), возвращающий минимальное островное дерево. Работает он по алгоритму прима.

- Выбираем произвольную вершину, с которой начинается построение дерева
- До тех пор, пока остаются вершины, не включенные в дерево
    - Выбираем ребро минимального веса между деревом и вершиной вне дерева
    - Добавляем выбранное ребро и вершину в итоговое дерево

6)Создаю в функции RandomGraphGen генератор случайных графок, который принимает в качестве параметров: кол-во вершин и номер теста. Генератор проходит по каждой вершине и в зависимости от выбранного номера теста и кол-ва вершин генерирует кол-во ребёр для этой вершины в диапазоне от 1 до 20. Чтобы у всех элементов была возможность обратиться друг к другу (даже в условиях ориентированного графа). Следующая за предыдущей вершина (если j < vert, где j-проверяемая вершина) сдвигается на то кол-во элементов, сколько было рёбер у предыдущей вершины. Пока кол-во вершин не будет больше, чем оставшихся вершин. В таком случае, рёбра будут снова от первого элемента отсчитываться.

7)В функции main я замеряю время, потраченное на поиск обоими методами и записываю выводы в файлы test1.txt

Результаты:

1)Демонстрация работы методов RandomGraphGen(10, 0)

Матрица смежности:

1	0	14	17	3	7	16	2	18	10	9
2	14	0	4	11	20	8	11	17	4	10
3	17	4	0	7	7	2	16	4	7	20
4	3	11	7	0	14	3	7	7	1	16
5	7	20	7	14	0	6	8	6	18	15
6	16	8	2	3	6	0	15	12	13	12
7	2	11	16	7	8	15	0	17	3	7
8	18	17	4	7	6	12	17	0	6	10
9	9	10	20	16	15	12	7	10	12	0
	1	2	3	4	5	6	7	8	9	10

Дерево Прима:

1	0	0	0	3	0	0	2	0	0	0
2	0	0	0	0	0	0	0	0	4	0
3	0	0	0	0	0	2	0	4	0	0
4	3	0	0	0	0	3	0	0	1	0
5	0	0	0	0	0	6	0	0	0	0
6	0	0	2	3	6	0	0	0	0	0
7	2	0	0	0	0	0	0	0	0	7
8	0	0	4	0	1	0	0	0	0	0
9	0	0	0	0	0	0	7	0	0	0
	1	2	3	4	5	6	7	8	9	10

2) Замерил время построения минимального островного дерева для графов с 10, 20, 50, 100 вершинами и вынес на график.

Лучшее	Худшее	Среднее
1	23	10
28	76	50.2
99	194	149.6
215	369	303.2

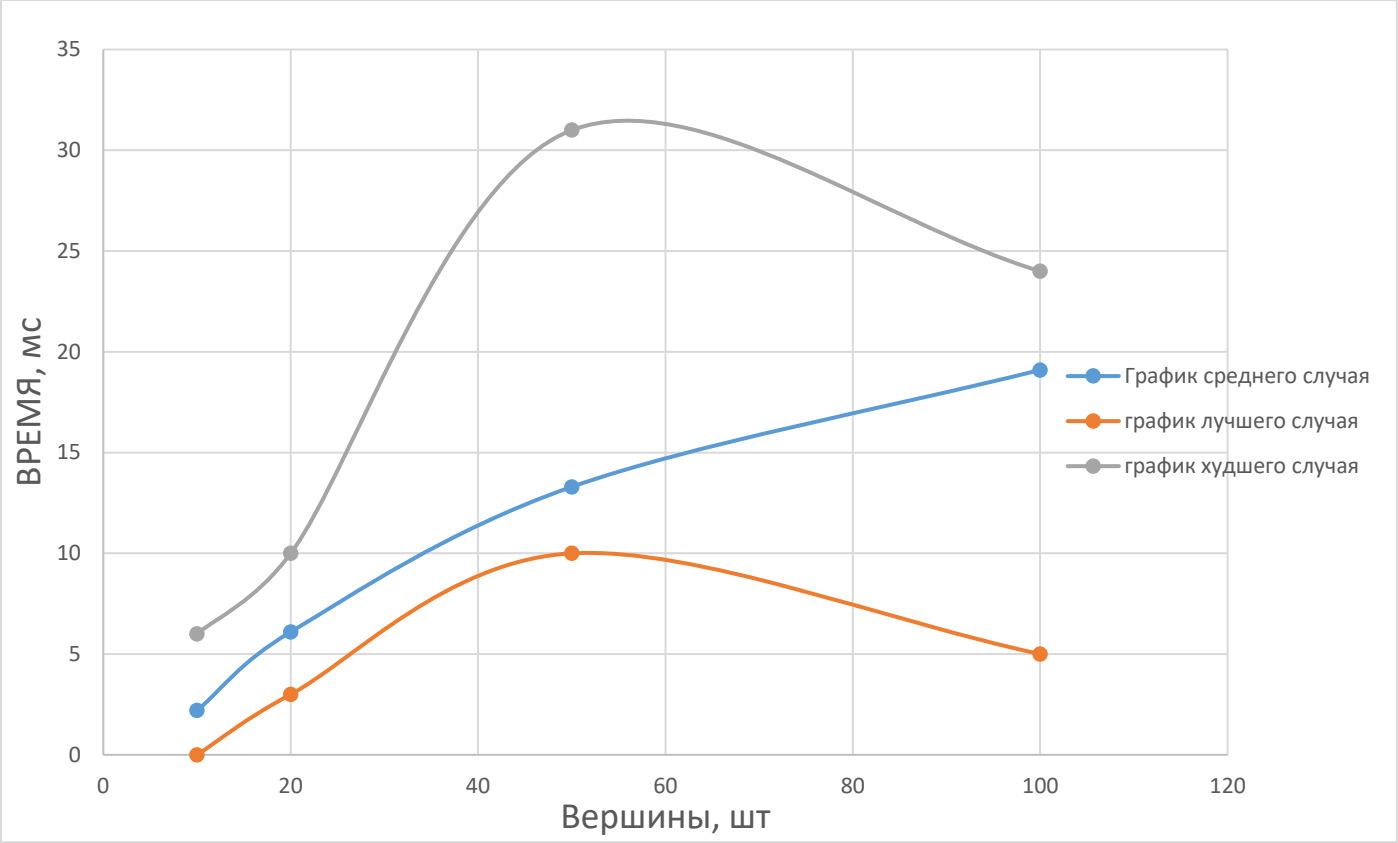

Рис.5. График времени, за которое создается минимальное островное дерево Прима

На графике видно, что с увеличением кол-ва вершин увеличивается и время, за которое строится дерево.

Заключение.

Реализация смежного графа оказалась достаточно простой, как и реализация минимального островного дерева по алгоритму Прима. На графиках можно было увидеть, что время выполнения поиска растет с количеством исследуемых вершин.