Министерство науки и высшего образования Российской Федерации  
Федеральное государственное автономное учреждение высшего образования  
«Пермский национальный исследовательский политехнический университет»

Электротехнический факультет  
Кафедра «Информационные технологии и автоматизированные системы»

**О Т Ч Ё Т**

Дискретная математика

Семестр 4

Лабораторная работа №6

Выполнил:  
студент группы РИС-22-2б  
Зубов Р.А.

Проверила:  
старший преподаватель кафедры ИТАС  
Рустамханова Г.И.

Пермь, 2024

Постановка задачи

На вход поступает матрица смежности (10x10) без петель. Граф не ориентирован. С помощью алгоритма Прима и Краскала необходимо найти остов, при этом необходимо узнать:

1. Какие ребра с каким весом
2. Суммарный вес

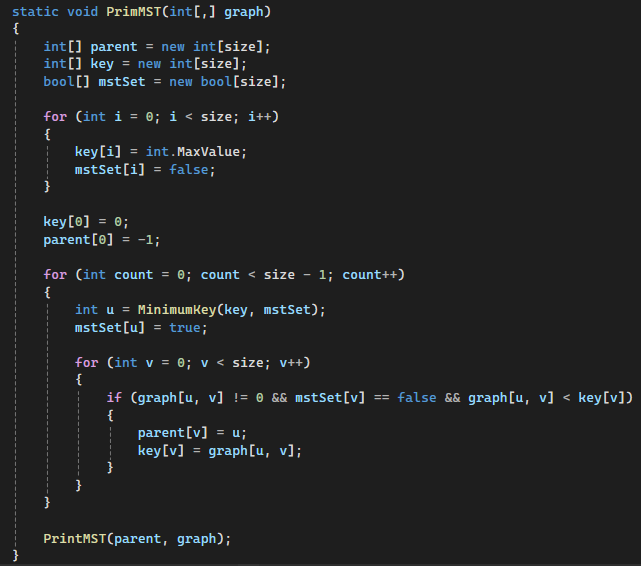
Алгоритмы работы программы

Алгоритм Прима. Суть самого алгоритма Прима сводится к жадному перебору рёбер. На входе имеется пустой подграф, который и будем достраивать до потенциального минимального остовного дерева.

* Изначально наш подграф состоит из одной любой вершины исходного графа.
* Затем из рёбер инцидентных этой вершине, выбирается такое минимальное ребро, которое связала бы две абсолютно разные компоненты связности, одной из которых и является наш подграф. То есть, как только у нас появляется возможность добавить новую вершину в наш подграф, мы тут же включаем ее по минимальмально возможному весу.
* Продолжаем выполнять предыдущий шаг до тех пор, пока не найдем искомое MST.

Алгоритм Прима реализован в классе *Prim*. На вход поступает матрица смежности, начальная вершина берется 1. В цикле перебираются вершины, с помощью функции *MinimumKey()* находят вершину с минимальным весом и подставляют в качестве следующей вершины. Далее анализируется, из какой вершины исходит ребро с минимальным весом, при этом чтобы не образовывался цикл в графе.

После завершения алгоритма с помощью функции *PrintMST()* выводится результат работы программы: какое ребро с каким связано и какой вес у ребра, а также суммарный вес графа.



*Рисунок 1 – Алгоритм Прима*

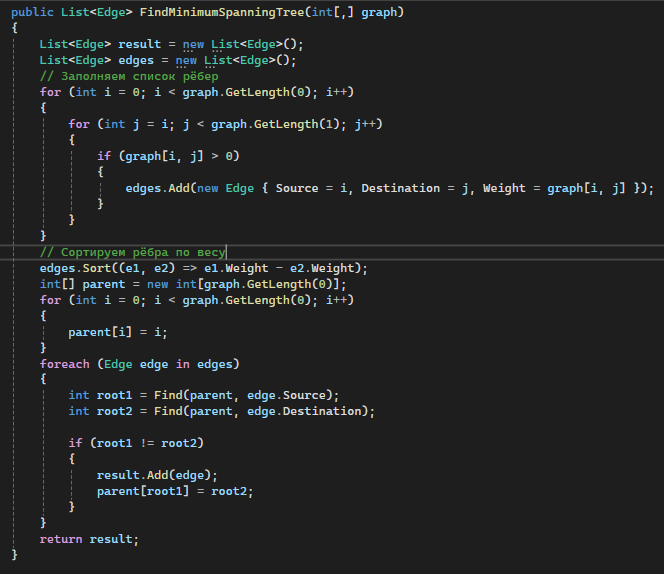
Алгоритм Краскала. Данный алгоритм называется жадным из-за того, что мы на каждом шаге пытаемся найти оптимальный вариант, который приведет к оптимальному решению в целом.

Механизм, по которому работает данный алгоритм, очень прост. На входе имеется пустой подграф, который и будем достраивать до потенциального минимального остовного дерева. Будем рассматривать только связные графы, в другом случае при применении алгоритма Краскала мы будем получать не минимальное остовное дерево, а просто остовной лес.

* Вначале мы производим сортировку рёбер по не убыванию по их весам.
* Добавляем i-ое ребро в наш подграф только в том случае, если данное ребро соединяет две разные компоненты связности, одним из которых является наш подграф. То есть, на каждом шаге добавляется минимальное по весу ребро, один конец которого содержится в нашем подграфе, а другой - еще нет.
* Алгоритм завершит свою работу после того, как множество вершин нашего подграфа совпадет с множеством вершин исходного графа.

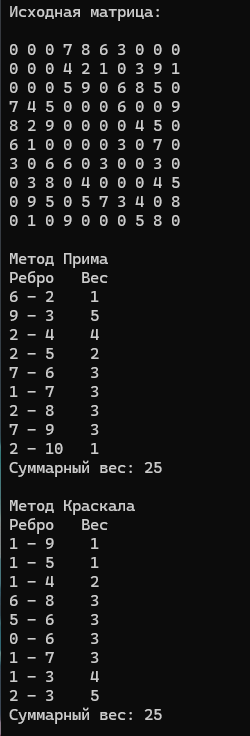
Алгоритм Краскала реализован в классе *KruskalAlgorithm.* В данном классе реализованы методы для нахождения минимального остовного дерева (*FindMinimumSpanningTree()*), метод для поиска корня вершины (*Find()*) и метод вывода результатов (*PrintMST()*).

Для работы с графом был создан класс *Edge*, в котором хранятся ребра графа: какое ребро с каким связано и вес графа. Алгоритм Краскала работает рекурсивно.



*Рисунок 2 – Алгоритм Краскала*

**Результаты работы программы:**



*Рисунок 3 – результат работы программы*

**Листинг:**

namespace DLab6

{

public class Edge

{

public int Source { get; set; }

public int Destination { get; set; }

public int Weight { get; set; }

}

public class PrimKruskal

{

#region Prim

private static int size = 10;

static int MinimumKey(int[] key, bool[] mstSet)

{

int min = int.MaxValue, minIndex = 0;

for (int v = 0; v < size; v++)

{

if (mstSet[v] == false && key[v] < min)

{

min = key[v];

minIndex = v;

}

}

return minIndex;

}

static void PrintMST(int[] parent, int[,] graph)

{

Console.WriteLine("\nМетод Прима");

Console.WriteLine("Ребро \tВес");

int sum = 0;

for (int i = 1; i < size; i++)

{

Console.WriteLine((parent[i] + 1) + " - " + (i + 1) + "\t " + graph[i, parent[i]]);

sum += graph[i, parent[i]];

}

Console.WriteLine("Суммарный вес: " + sum);

}

public void PrimMST(int[,] graph)

{

int[] parent = new int[size];

int[] key = new int[size];

bool[] mstSet = new bool[size];

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

{

key[i] = int.MaxValue;

mstSet[i] = false;

}

key[0] = 0;

parent[0] = -1;

for (int count = 0; count < size - 1; count++)

{

int u = MinimumKey(key, mstSet);

mstSet[u] = true;

for (int v = 0; v < size; v++)

{

if (graph[u, v] != 0 && mstSet[v] == false && graph[u, v] < key[v])

{

parent[v] = u;

key[v] = graph[u, v];

}

}

}

PrintMST(parent, graph);

}

#endregion

public List<Edge> FindMinimumSpanningTree(int[,] graph)

{

List<Edge> result = new List<Edge>();

List<Edge> edges = new List<Edge>();

// Заполняем список рёбер

for (int i = 0; i < graph.GetLength(0); i++)

{

for (int j = i; j < graph.GetLength(1); j++)

{

if (graph[i, j] > 0)

{

edges.Add(new Edge { Source = i, Destination = j, Weight = graph[i, j] });

}

}

}

// Сортируем рёбра по весу

edges.Sort((e1, e2) => e1.Weight - e2.Weight);

int[] parent = new int[graph.GetLength(0)];

for (int i = 0; i < graph.GetLength(0); i++)

{

parent[i] = i;

}

foreach (Edge edge in edges)

{

int root1 = Find(parent, edge.Source);

int root2 = Find(parent, edge.Destination);

if (root1 != root2)

{

result.Add(edge);

parent[root1] = root2;

}

}

return result;

}

// Метод для поиска корня вершины

private int Find(int[] parent, int vertex)

{

while (vertex != parent[vertex])

{

vertex = parent[vertex];

}

return vertex;

}

// Метод вывода ребер и веса

public void PrintMST(int[,] arr)

{

List<Edge> result = FindMinimumSpanningTree(arr);

Console.WriteLine("\nМетод Краскала");

Console.WriteLine("Ребро \tВес");

int sum = 0;

foreach (Edge edge in result)

{

Console.WriteLine($"{edge.Source} - {edge.Destination} \t {edge.Weight}");

sum += edge.Weight;

}

Console.WriteLine("Суммарный вес: " + sum);

}

}

}