**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра САПР**

отчет

**по курсовой работе №2**

**по дисциплине «Алгоритмы и структуры данных»**

Тема: Потоки в сетях**. Вариант: 2**

|  |  |  |
| --- | --- | --- |
| Студент гр. 0302 |  | Савенко Н.С |
| Преподаватель |  | Тутуева А.В. |

Санкт-Петербург

2022

**Постановка задачи**

Входные данные: текстовый файлы со строками в формате V1, V1, P, где V1, V2 направленная дуга

транспортной сети, а P – ее пропускная способность. Исток всегда обозначен как S, сток – как T

Пример файла для сети с изображения выше:

S O 3

S P 3

O Q 3

O P 2

P R 2

Q R 4

Q T 2

R T 3

Найти максимальный поток в сети используя алгоритм Эдмонса-Карпа

**Описание реализуемых алгоритмов**

Эдмонса-Карпа

1. Обнуляем все потоки. Остаточная сеть изначально совпадает с исходной сетью.
2. В остаточной сети находим *кратчайший* путь из источника в сток. Если такого пути нет, останавливаемся.
3. Пускаем через найденный путь (он называется **увеличивающим путём** или **увеличивающей цепью**) максимально возможный поток:
   1. На найденном пути в остаточной сети ищем ребро с минимальной пропускной способностью {\displaystyle c\_{\min }}.
   2. Для каждого ребра на найденном пути увеличиваем поток на {\displaystyle c\_{\min }}, а в противоположном ему — уменьшаем на {\displaystyle c\_{\min }}.
   3. Модифицируем остаточную сеть. Для всех рёбер на найденном пути, а также для противоположных им рёбер, вычисляем новую пропускную способность. Если она стала ненулевой, добавляем ребро к остаточной сети, а если обнулилась, стираем его.
4. Возвращаемся на шаг 2.

Чтобы найти кратчайший путь в графе, используем поиск в ширину:

1. Создаём очередь вершин *О*. Вначале *О* состоит из единственной вершины *s*.
2. Отмечаем вершину *s* как посещённую, без родителя, а все остальные как непосещённые.
3. Пока очередь не пуста, выполняем следующие шаги:
   1. Удаляем первую в очереди вершину *u*.
   2. Для всех дуг (*u*, *v*), исходящих из вершины *u*, для которых вершина *v* ещё не посещена, выполняем следующие шаги:
      1. Отмечаем вершину *v* как посещённую, с родителем *u*.
      2. Добавляем вершину *v* в конец очереди.
      3. Если *v* = *t*, выходим из обоих циклов: мы нашли кратчайший путь.
4. Если очередь пуста, возвращаем ответ, что пути нет вообще и останавливаемся.
5. Если нет, идём от *t* к *s*, каждый раз переходя к родителю. Возвращаем путь в обратном порядке.

**Оценка временной сложности**

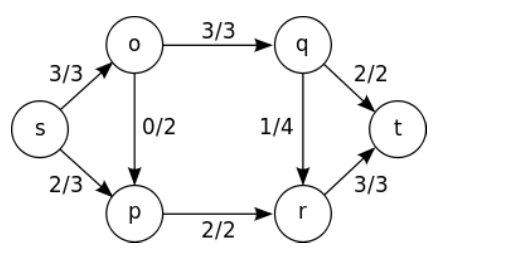
1. CalculateMaxFlow

*O(VE)*

**Описание Unit тестов**

В unit тестах поводится проверка методов чтения из файла, парсинга ребер и расчета максимального потока.

**Примеры работы**

В качестве примера демонстрируется граф 

**Изображение выглядит как текст, монитор, снимок экрана, экран

Автоматически созданное описание**

Изображение выглядит как текст, снимок экрана, экран, черный

Автоматически созданное описание

**Листинг**

namespace Kursach2;

public class EdmonsKarp

{

public IList<FlowLink> Links { get; init; }

public EdmonsKarp()

{

Links = new List<FlowLink>();

}

public EdmonsKarp(string filePath) : this()

{

LoadFromFile(filePath);

}

public bool LoadFromFile(string filePath)

{

try

{

string[] lines = File.ReadAllLines(filePath);

ParseLines(lines);

return true;

}

catch (Exception e)

{

return false;

}

}

// In this implementation, we do not need to save the initial bandwidth of the channels

public long CalculateMaxFlow(string from, string to)

{

long maxFlow = 0;

while (TryFindLowerCost(from, to, out var flow))

{

var minFlow = flow.Min(link => link.cost);

foreach (var flowLink in flow)

{

var linkWithUpdatedCost = flowLink with {cost = flowLink.cost - minFlow};

Links.Remove(flowLink);

Links.Add(linkWithUpdatedCost);

if (Links.FirstOrDefault(link => link.from == flowLink.to && link.from == flowLink.to) != null)

{

var opposite = Links.First(link => link.from == flowLink.to && link.from == flowLink.to);

var newOpposite = opposite with {cost = opposite.cost + minFlow};

Links.Remove(opposite);

Links.Add(newOpposite);

}

}

maxFlow += minFlow;

}

return maxFlow;

}

private bool TryFindLowerCost(string from, string to, out List<FlowLink>? flow)

{

flow = new List<FlowLink>();

var linkQueue = new MyQueue<MeetablePoint>(new [] { new MeetablePoint(from, null) });

var metPoints = new List<string>(new [] { from });

var findedFlag = false;

while (linkQueue.TryDequeue(out var point) && findedFlag)

{

var relativeLinks = Links.Where(link => link.from == point.name && !metPoints.Contains(link.to));

foreach (var relativeLink in relativeLinks)

{

metPoints.Add(relativeLink.to);

linkQueue.Enqueue(new MeetablePoint(relativeLink.to, relativeLink.from));

if (relativeLink.from == to)

{

findedFlag = true;

break;

}

}

}

if (linkQueue.Count == 0)

{

return false;

}

var lastPoint = metPoints.First(point => point == to);

while (lastPoint != from)

{

var link = Links.First(link => link.to == lastPoint);

flow.Add(link);

lastPoint = link.from;

}

return true;

}

public void ParseLinks(string source)

{

var fields = source.Split();

if (int.TryParse(fields[2], out var depCost))

{

AddLink(new FlowLink(fields[0], fields[1], depCost));

}

}

public void ParseLines(string[] lines)

{

foreach (var line in lines)

{

ParseLinks(line);

}

}

public void AddLink(FlowLink link)

{

Links.Add(link);

}

}

namespace Kursach2;

//In this implementation, we do not need to save the initial bandwidth of the channels, for example. 1/3. Therefore links do not have this field

public record FlowLink(string from, string to, int cost);

namespace Kursach2;

public record MeetablePoint(string name, string? parent);

namespace Kursach2

{

internal class MyQueue<T>

{

public T? Head { get; private set; }

readonly Laba1.LinkedList<T> list;

public MyQueue(T[] items) : this()

{

foreach(var item in items)

{

Enqueue(item);

}

}

public MyQueue()

{

list = new Laba1.LinkedList<T>();

}

public void Enqueue(T element)

{

list.push\_front(element);

}

public T Dequeue()

{

return list.PopLastData();

}

public T Peek() => list.at(list.Count - 1);

public int Count => list.Count;

internal bool TryDequeue(out T point)

{

point = default;

if (list.Count == 0) return false;

point = Dequeue();

return true;

}

}

}