COM2031 Advanced Algorithms, Autumn Semester 2019

Lab 7: Max Flow Min Cut – the Ford-Fulkerson algorithm

Purpose of the lab

This lab asks you to implement the Ford-Fulkerson algorithm introduced in this week's lecture to compute the maximum flow through a graph.

Ford-Fulkerson algorithm

The pseudocode for the algorithm was given in the lecture as follows:

```
Ford-Fulkerson(G, s, t, c) {
  foreach e ∈ E:  f(e) ← 0
  Gf ← residual graph
  while (there exists augmenting path P) {
    f ← Augment(f, c, P)
    update Gf
  }
  return f
}

Augment(f, c, P) {
  b ← bottleneck(P)
  foreach e ∈ P {
    if (e ∈ E) f(e) ← f(e) + b
    else      f(eR) ← f(e) - b
  }
  return f
}
```

The following site allows you to explore this algorithm:

https://www-m9.ma.tum.de/graph-algorithms/flow-ford-fulkerson/index_en.html

For this problem we have a directed graph where each edge has a capacity. We also have two distinguished nodes: a source node and a sink node.

A graph is given as a set of Vertices V (these are the nodes) and Edges E, where each edge $e \in E$ has a capacity $c(e)$.

There are various Graph Data structures that were covered in last year's COM1029 course. The choice of data structure will affect your implementation.

Given the set of vertices V we can number them from 0 to N-1 with 0 as the source node and N-1 as the sink node, with all the other nodes labelled from 1 to N-2.

Then we can represent the graph as a 2-D array G, where every edge e from i to j with capacity c(e) will correspond to G[i][j] having the value c(e). If there is no edge from i to j then G[i][j] will have the value 0.

Exercise 1: Draw the graph corresponding to the following array:

G	*0*	*1*	*2*	*3*	*4*
0	0	9	2	0	0
1	0	0	5	3	0
2	0	0	0	2	4
3	0	0	0	0	6
4	0	0	0	0	0

Can you see what the minimum cut and maximum flow will be for this graph?

Use the site https://www-m9.ma.tum.de/graph-algorithms/flow-ford-fulkerson/index_en.html to model this graph and to work out the maximum flow.

Implementation

A flow f can also be represented as a 2-D array f, where the entries correspond to the flow along the edges. Each entry for f must be less than or equal to the capacity given in the graph G.

Question: Given a graph G and a flow f how can you compute the residual graph G' ?

[hint: if an edge e from i to j has capacity c(e) and flow f(e) then the residual graph has the remaining capacity c(e)-f(e) for that edge from i to j, and has capacity f(e) for the reverse edge from j to i.]

Main Task: Implement the Ford-Fulkerson Algorithm given above, by completing the Java file Graph_maxflow_mincut.java (i.e. fill in the TODOs).

Use 2-D arrays to represent the graph, the residual graph, and the flow.

Think about how to find an augmenting path, and how to represent it
[hint; use Breadth First Search to find the shortest augmenting path]

To do this you will need to implement all of the elements of the algorithm:

- Initialise the residual graph to be the original graph
- Initialise the flow to be 0 on all edges

Repeat the following until no augmenting path remains:

- Find an augmenting path in the residual graph
- Update the flow with the flow along the augmenting path
- Update the residual graph to account for the updated flow

- When no augmenting path remains then return the flow: this is the maximal flow

Run your algorithm on Example 1 above. Do you get the result you worked out earlier?