CS4234: Optimisation Algorithms Tutorial 4

Max-Flow, Min-Cut, Ford-Fulkerson's Analysis

V1.4: Steven Halim *September 14, 2020*

Preliminaries

We will start T04 by asking a quick question from Lecture 05a+05b: What is the Max-Flow of the (random, but reasonably small) Flow Network drawn in (e-)class (using `https://visualgo.net/en/maxflow`)? Run one Ford Fulkerson's based algorithm manually (Tutor will randomly select either basic Ford Fulkerson's method/simple DFS, Edmonds Karp's algorithm/BFS, or Dinic's algorithm/BFS/layer graphs) before executing the animation to show your tutor/classmates that you have fully understand the algorithm.

Discussion Points

Q1: Write Max-Flow as a Linear Program. Again, are you going to solve Max-Flow that way?

Q2: Show how to use (standard) Ford-Fulkerson's algorithm to find a maximum sized matching on *Bipartite* Graph, or formally known as the Max-Cardinality-Bipartite-Matching (MCBM). Prove that the result is a matching, and that it is the maximum-sized matching. Analyze the running time of your algorithm. Note that we have seen the topic of Graph Matching briefly in Lecture 1 (Deterministic Vertex Cover-2), Lecture 4 (part of Christofides's algorithm), and will properly come back to this topic on Lecture 7+9.

Q3a: Assume that you have an undirected graph $G = (V, E)$ with a source vertex s and a target vertex t (the graph is *unweighted*). Give an algorithm that finds the maximum number of *edge-disjoint paths* from s to t. Two paths P_1 and P_2 are called edge-disjoint if they do not share any edges—but they may share a vertex. (Hint: Use (standard) Ford-Fulkerson's algorithm.) What is the running time of your algorithm?

Q3b: What if you want to find the maximum number of *vertex-disjoint paths* from s to t instead? Two paths are called vertex-disjoint if they do not share any vertex.

Q4: Please read `https://onlinejudge.org/external/117/11757.pdf` and try to reduce this problem into a max flow problem, solve it using $O(n^2 \times m)$ Dinic's algorithm (assuming that you have such implementation ready), and analyze its time complexity.

Q5a: Can we write the runtime of Ford-Fulkerson's algorithm as $O(m \times F)$ where F is the eventual max flow value of the input graph? This is an output-sensitive analysis, whereby the runtime speed of the algorithm depends on the size of the output.

Q5b: Now revisit problem in Q4 above and consider this 'Competitive Programmer' analysis that is not frequently found in Computer Science textbooks about max flow algorithms:
$O(min(\sum_{u:(s,u)\in E} c_{su}, \sum_{u:(u,t)\in E} c_{ut}) * m)$.
Explain what are we trying to do here?

Post Tutorial

Max flow is an interesting algorithm that will take time to master (properly). You are advised to use recess week to revise the first half of CS4234.