CS4234: Optimisation Algorithms Tutorial 4

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

V1.6: Steven Halim *September 12, 2022*

Preliminaries

Before starting T04, we will run a comparison between the two similar Ford-Fulkerson methods that use 'shortest-augmenting-path-first' strategy: Edmonds-Karp vs Dinic's algorithm using `https://visualgo.net/en/maxflow` on a small random flow graph.

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 algorithm to find a maximum sized matching on *Bipartite* Graph, or formally known as the Max-Cardinality-Bipartite-Matching (MCBM) that will be properly discussed on Lecture 6 (please read ahead). 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 6+7.

Q3a: Assume that you have an (un)directed 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. Is this a Max Flow problem? 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. Is this (still) a Max Flow problem?

Q4: Please read `https://onlinejudge.org/external/117/11757.pdf` (that person *just returned last year and almost left this year but he eventually stayed*) 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 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. Assuming that we are using the theoretically faster $O(n^2 \times m)$ Dinic's algorithm, can we analyze its time complexity as:
$O(min(\sum_{u:(s,u)\in E} c_{su}, \sum_{u:(u,t)\in E} c_{ut}, n^2) \times m)$.
Explain what are we trying to do here?

Post Tutorial

Students can discuss PS3 ideas with the tutor if you still struggle as this is the 'last' tutorial before PS3 is due (remember: no tutorial on Monday of recess week...).

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