Week 9 Applied Sheet

Objectives: The applied sessions, in general, give practice in problem solving, in analysis of algorithms and data structures, and in mathematics and logic useful in the above.

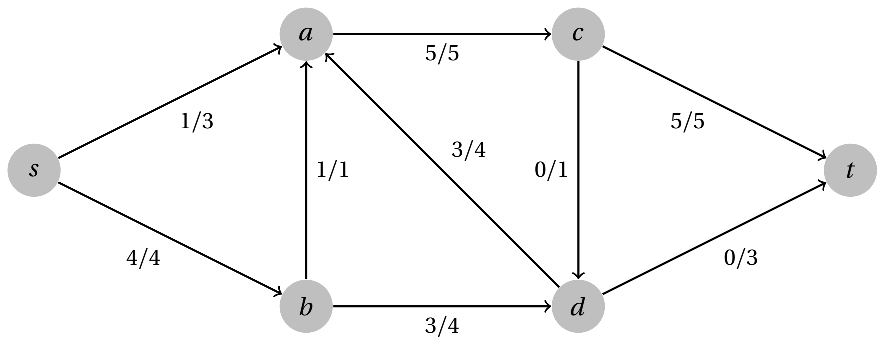
Instructions to the class: You should actively participate in the class. The preparation problems are not assessed. We strongly recommend that everyone tries to solve the preparation problems before the applied session as you will benefit the most from the applied session if you come properly prepared. However, you can still attend the applied session if you have not solved those problems beforehand.

Instructions to Tutors: The purpose of the applied session is not to solve the practical exercises! The purpose is to check answers, and to discuss particular sticking points, not to simply make answers available.

Supplementary problems: The supplementary problems provide additional practice for you to complete after your applied session, or as pre-exam revision. Problems that are marked as **(Advanced)** difficulty are beyond the difficulty that you would be expected to complete in the exam, but are nonetheless useful practice problems as they will teach you skills and concepts that you can apply to other problems.

Problems

Problem 1. **(Preparation)** Consider the following flow network. Edge labels of the form f/c denote the current flow f and the total capacity c.



(a) Draw the corresponding residual network.

(b) Identify an augmenting path in the residual network and state its capacity.

(c) Augment the flow of the network along the augmenting path, showing the resulting flow network.

Problem 2. Complete the Ford-Fulkerson method for the network in Problem 1, showing the final flow network with a maximum flow.

Problem 3. Using your solution to Problem 2, list the vertices in the two components of a minimum $s-t$ cut in the network in Problem 1. Identify the edges that cross the cut and verify that their capacity adds up to the value of the maximum flow.

Problem 4. Let G be a flow network and let f be a valid flow on G. Prove that the net outflow out of s is equal to the net inflow into t.

Problem 5. Consider a variant of the maximum network flow problem in which we allow for multiple source vertices and multiple sink vertices. We retain all of the capacity and flow conservation constraints of the original maximum flow problem. As in the original problem, all of the sources and sinks are excluded from the flow conservation constraint. Describe a simple method for solving this problem.

Problem 6. Given a list of n integers $r_1, r_2, ..., r_n$ and m integers $c_1, c_2, ..., c_m$, we wish to determine whether there exists an $n \times m$ matrix consisting of zeros and ones whose row sums are $r_1, r_2, ..., r_n$ respectively and whose column sums are $c_1, c_2, ..., c_m$ respectively. It is assumed that the sum of the r's is equal to the sum of the c's (otherwise it is trivially impossible to satisfy the constraints). Describe an algorithm for solving this problem by using a flow network.

Problem 7. Consider the problem of allocating a set of jobs to one of two supercomputers. Each job must be allocated to exactly one of the two computers. The two computers are configured slightly differently, so for each job, you know how much it will cost on each of the computers. There is an additional constraint. Some of the jobs are related, and it would be preferable, but not required, to run them on the same computer. For each pair of related jobs, you know how much more it will cost if they are run on separate computers. Give an efficient algorithm for determining the optimal way to allocate the jobs to the computers, where your goal is to minimise the total cost.

Problem 8. Consider a variant of the maximum network flow problem in which vertices also have capacities. That is for each vertex except s and t, there is a maximum amount of flow that can enter and leave it. Describe a simple transformation that can be made to such a flow network so that this problem can be solved using an ordinary maximum flow algorithm[1].

Problem 9. Two paths in a graph are *edge disjoint* if they have no edges in common. Given a directed graph, we would like to determine the maximum number of edge-disjoint paths from vertex s to vertex t.

(a) Describe how to determine the maximum number of edge-disjoint $s-t$ paths.

(b) What is the time complexity of this approach?

(c) How could we modify this approach to find *vertex-disjoint paths* (i.e. paths with no vertices in common)?

Supplementary Problems

Problem 10. Implement the Ford-Fulkerson method using:

- DFS for finding augmenting paths.
- BFS for finding augmenting paths.

Problem 11. You are a radio station host and are in charge of scheduling the songs for the coming week. Every song can be classified as being from a particular era, and a particular genre. Your boss has given you a strict set of requirements that the songs must conform to. Among the songs chosen, for each of the n eras $1 \leq i \leq n$, you must play exactly n_i songs of that era, and for each of the m genres $1 \leq j \leq m$, you must play exactly m_j songs from that genre. Of course, $\sum n_i = \sum m_j$. Devise an algorithm that given a list of all of the songs you could choose from, their era and their genre, determines a subset of them that satisfies the requirements, or determines that it is not possible.

[1] Do not try to modify the Ford-Fulkerson algorithm. In general, it is always safer when solving a problem to reduce the problem to another known problem by transforming the input, rather than modifying the algorithm for the related problem.