

Tutorial Sheet 12

- This tutorial sheet contains problems in network flow. When presenting your solutions, please draw/describe the flow network clearly. The correctness and running time arguments, if required, can be brief (1-2 sentences).
- Problems marked with (★) will not be asked in the tutorial quiz.

1. In a standard $s - t$ maximum flow problem, we assume edges have capacities, and there is no limit on how much flow is allowed to flow through a node. In this problem, we consider a variant of the maximum flow problem with node capacities (and no edge capacities). Let $G = (V, E)$ be a directed graph, with source s , sink t , and nonnegative node capacities u_v for each $v \in V$. Given a flow f in this graph, the flow through a node v is defined as

$$\sum_{e \in \delta^-(v)} f_e,$$

where $\delta^-(v)$ denotes the edges coming into v . We say that a flow is feasible if it satisfies the usual flow-conservation constraints and the node-capacity constraint: the flow through a node v cannot exceed u_v . Design a polynomial-time algorithm to find a $s - t$ maximum flow in such a node-capacitated network.

2. Let M be an $n \times n$ matrix with each entry equal to either 0 or 1. Let m_{ij} denote the entry in row i and column j . A diagonal entry is one of the form m_{ii} for some i . *Swapping* rows i and j of the matrix M denotes the following action: We swap the values m_{ik} and m_{jk} for every $k \in \{1, 2, \dots, n\}$. Swapping two columns is defined analogously. We say that M is *re-arrangeable* if it is possible to swap some of the pairs of rows and some of the pairs of columns (in any sequence) so that after the swapping, all the diagonal entries of M are equal to 1.
 - a) Give an example of a matrix M which is not re-arrangeable but for which at least one entry in each row and each column is equal to 1.
 - b) Give a polynomial-time algorithm that determines whether a matrix M with 0 – 1 entries, is re-arrangeable.
3. You have a collection of n software applications, $\{1, \dots, n\}$, running on an old system, and now you would like to port some of these to the new system. If you move application i to the

new system, you expect a net (monetary) benefit of $b_i \geq 0$. The different software applications interact with one another; if applications i and j have extensive interaction, then you will incur an expense if you move one of i or j to the new system but not both – let's denote this expense by $x_{ij} \geq 0$. So, if the situation were really this simple, you would port all n applications, achieving a total benefit of $\sum_i b_i$.

Unfortunately, there's a problem. Due to small but fundamental incompatibilities between the two systems, there's no way to port the application 1 to the new system; it will have to remain on the old system. Nevertheless, it might still pay to port some of the other applications, accruing the associated benefits and incurring the expense of the interaction between applications on different systems. So, here's the question: Which of the remaining applications, if any, should be moved? Give a polynomial-time algorithm to find a set $S \subseteq \{2, \dots, n\}$ for which the sum of the benefits minus the expenses of moving the applications in S to the new system is maximized.

4. Extend the proof of Hall's theorem to show that, for every bipartite graph $G = (V \cup W, E)$ with $|V| \leq |W|$,

$$\text{maximum cardinality of a matching in } G = \min_{S \subseteq V} (|V| - (|S| - |N(S)|)).$$

5. Recall that a vertex cover is a subset S of vertices such that every edge is incident to some vertex in S . Prove that the problem of computing a minimum-cardinality vertex cover in bipartite graphs can be solved in polynomial time.
6. Give a polynomial-time algorithm for the following minimization version of the max-flow problem. You are given a directed graph $G = (V, E)$ with a source s and sink t , and numbers (capacities) ℓ_e for each edge $e \in E$. We define a flow f and the value of a flow, as usual, requiring that all nodes except s and t satisfy flow conservation. However, the given numbers are *lower bounds* on edge flow, i.e., they require that $f_e \geq \ell_e$ for each edge e , and there is no upper bound on flow values on the edges.
 - a) Give a polynomial-time algorithm that finds a feasible flow of minimum possible value.
Hint: Start with any flow from s to t which obeys the lower bounds, and try to send a maximum flow from t to s in a suitable modification of the graph G .
 - b) (*) Prove an analogue of the max-flow min-cut theorem for this problem.
7. (*) In sociology, one often studies a graph G in which nodes represent people and edges represent those who are friends with each other. Let's assume that friendship is symmetric for this question, so we can consider an undirected graph. Now, suppose we want to study this graph G , looking for a close-knit group of people. One way to formalize this notion would be as follows. For a subset S of nodes, let $e(S)$ denote the number of edges in S , i.e., the

number of edges with both ends in S . We define the cohesiveness of S as $e(S)/|S|$. A natural thing to search for would be the set S of people achieving the maximum cohesiveness. Give a polynomial-time algorithm that takes a rational number α and determines whether there exists a set S with cohesiveness at least α . Give a polynomial-time algorithm to find a set S of nodes with maximum cohesiveness.

8. (★) Consider the following generalization of the maximum flow problem. The input consists of (a) a directed graph $G = (V, E)$, (b) a source vertex $s \in V$ and a destination vertex $t \in V$, and (c) for each edge $e \in E$, there is an *upper* bound u_e and a *lower* bound ℓ_e on the flow through that edge. A flow $f := \{f_e\}_{e \in E}$ is said to satisfy capacity constraints if, for every edge e , it holds that $\ell_e \leq f_e \leq u_e$. The conservation constraints for each vertex are defined as before. Assume that u_e and ℓ_e are nonnegative integers. The goal, as before, is to compute a feasible flow with maximum value in the given graph.

Show that the maximum flow problem with both upper and lower bounds on capacities can be reduced to the maximum flow problem with only upper bounds on capacities. That is, given an instance of the generalized maximum flow problem (with both upper and lower bounds), show how to construct an instance of the standard maximum flow problem (with only upper bounds) such that given a maximum flow in the instance of the standard problem, you can recover a maximum flow in the instance of the generalized problem.