

Homework 10 – Due Friday, November 14, 2008

Please refer to the general information handout for the full homework policy and options.

Reminders

- Your solutions are due before the lecture. Late homework will not be accepted.
- Collaboration is permitted, but you must write the solutions *by yourself without assistance*, and be ready to explain them orally to a member of the course staff if asked. You must also identify your collaborators. *Getting solutions from outside sources such as the Web or students not enrolled in the class is strictly forbidden.*
- To facilitate grading, please write down your solution to each problem on a separate sheet of paper. Make sure to include all identifying information and your collaborators on each sheet. Your solutions to different problems will be graded separately, possibly by different people, and returned to you independently of each other.
- For problems that require you to provide an algorithm, you must give a precise description of the algorithm, **together with a proof of correctness** and an analysis of its running time. You may use algorithms from class as subroutines. You may also use any facts that we proved in class or from the book.

General guidelines for reductions. Model your solutions on the reduction of MAXIMUM MATCHING to MAXIMUM FLOW given in class. To reduce problem B to problem A :

1. Explain how to transform an instance \mathcal{I}_B of B into an instance \mathcal{I}_A of A .
2. Explain how to transform a solution \mathcal{S}_A for \mathcal{I}_A into a solution \mathcal{S}_B for \mathcal{I}_B .
3. (**large fraction of the points**) Prove that \mathcal{S}_B is a correct solution for \mathcal{I}_B , provided that \mathcal{S}_A is a correct solution for \mathcal{I}_A . In case of optimization problems, it usually involves proving that the value of \mathcal{S}_A is equal to the value of \mathcal{S}_B . (Often, it is easier to prove \geq and \leq separately.)
4. Analyze the efficiency of the resulting algorithm for problem B that uses your reduction and the most suitable algorithm for problem A that we studied. Make sure that the running time is expressed in terms of the length of \mathcal{I}_B , not \mathcal{I}_A .

Exercises These should not be handed in, but the material they cover may appear on exams:

1. (a) Reduce the maximum flow problem for a network with several source nodes (s_1, \dots, s_k) and several sink nodes (t_1, \dots, t_ℓ) into the single-source single-sink maximum flow problem.
(b) Some networks have capacity constraints on the flow amounts that can flow through their intermediate vertices. Explain how the maximum flow problem for such a network can be reduced to MAXIMUM FLOW with edge capacity constraints only.

2. Review Sections 7.6 (edge-disjoint paths) and 7.11 (project selection). These were covered without slides in class. To review Section 7.6, look at exercises 32 and 33.
3. As usual, read, solve and check your answers on the solved exercises. They might be helpful for some of the homework and exam problems.

Problems to be handed in

Page limits: The answer to each problem should fit in **2 pages (or one double-sided sheet)** of paper. Longer answers will be penalized.

1. **(Vertex-disjoint paths)** Section 7.6 of the textbook describes using the Ford-Fulkerson to find *edge-disjoint* paths in a unweighted directed (and undirected) graphs. The number of edge-disjoint paths is a measure of how resistant a graph is to deletion of edges.

Consider instead, an analogous notion that captures resistance to the deletion of vertices (think blocked intersections or failed routers). Two paths from s to t are *vertex-disjoint* if they pass through disjoint (i.e. non-intersecting) sets of vertices. Note that vertex-disjointness implies edge-disjointness, but not the other way around.

- (Exercise, do not hand in) Give an example of paths that are edge-disjoint but not vertex-disjoint.

In this problem, we consider the problem of finding a maximum-size vertex-disjoint collection of paths that connect two given vertices s and t .

- (a) Consider the greedy algorithm which works as follows:

```

▷ Given a directed, unweighted graph  $G = (V, E)$  and  $s, t \in V$ :
1 Initialize set  $S$  to  $\emptyset$ 
2 while (there exists a path from  $s$  to  $t$  in  $G$ ):
3     do Find a path  $p$  from  $s$  to  $t$  in  $G$  (using, say, DFS).
4         Add  $p$  to  $S$ 
5         Remove all intermediate vertices in  $p$  from  $V$ 
           (along with corresponding edges from  $E$ ).
6 Output  $S$ 

```

Give an example of a graph for which the algorithm above will not always return a maximum-size set of vertex-disjoint paths.

- (b) Give a polynomial-time algorithm that takes a directed graph G along with vertex names s, t and outputs a maximum-size set of vertex-disjoint paths from s to t .
(Hint: Reduce to the edge-disjoint paths problem in related graph G' which has two vertices for every vertex of G .)
- (c) Prove that there exist k vertex-disjoint paths from s to t if and only if there is no set of $k - 1$ vertices (not including s or t) whose deletion from G disconnects s from t .
(Hint: Use the min-cut/max-flow equivalence.)