Assignment 9

1 Applications of Max-Flow Min-Cut

Review the statement of max-flow min-cut theorem and prove the following two statements.

(a) Let $G = (L \cup R, E)$ be a unweighted bipartite graph. Then G has a L-perfect matching (a matching with size $|L|$) if and only if, for every set $X \subseteq L$, X is connected to at least $|X|$ vertices in R. You must prove both directions.

Hint: Use the max-flow min-cut theorem.

(b) Let G be an unweighted directed graph and $s, t \in V$ be two distinct vertices. Then the maximum number of edge-disjoint s - t paths equals the minimum number of edges whose removal disconnects t from s (i.e., no directed path from s to t after the removal).

Hint: show how to decompose a flow of value k into k disjoint paths, and how to transform any set of k edge-disjoint paths into a flow of value k.

2 Trade Surplus

Consider the following definition. We are given a set of n countries that are engaged in trade with one another. For each country i, we have the value s_i of its budget surplus; this number may be positive or negative, with a negative number indicating a deficit. For each pair of countries i, j, we have the total value e_{ij} of all exports from i to j; this number is always non-negative. We say that a subset S of the countries is *free-standing* if the sum of the budget surpluses of the countries in S, minus the total value of all exports from countries in S to countries not in S, is non-negative.

Give a polynomial-time algorithm that takes this data for a set of n countries, and decides whether it contains a non-empty free-standing subset that is not equal to the full set. Analyze its running time.

3 Flow Disconnecting with Multiple Terminals

Suppose we are given a directed network $G = (V, E)$ with a root node r, and a set of *terminals* $T \subseteq V$. We'd like to disconnect many terminals from r, while cutting relatively few edges.

We make this trade-off precise as follows. For a set of edges $F \subseteq E$, let $q(F)$ denote the number of nodes $v \in T$ such that there is no $r - v$ path in the subgraph $(V, E - F)$. Give a polynomial-time algorithm to find a set F of edges that maximizes the quantity $q(F) - |F|$. (Note that setting F equal to the empty set is an option). Analyze the running time of your proposed algorithm.

4 Job Scheduling of Multi-Processors

Suppose you're managing a collection of processors and must schedule a sequence of jobs over time. The jobs have the following characteristics. Each job j has an arrival time a_j when it is first available for processing, a length ℓ_j which indicates how much processing time it needs, and a deadline d_j by which it must be finished. (We'll assume $0 < \ell_j \leq d_j - a_j$). Each job can be run on any of the processors, but only on one at a time; it can also be pre-empted and resumed from where it left off on another processor.

Moreover, the collection of processors is not entirely static either: you have an overall pool of k possible processors; but for each processor i, there is an interval of time $[t_i, t'_i]$ during which it is available; it is unavailable at all other times.

Given all this data about job requirements and processor availability, you'd like to decide whether the jobs can all be completed or not. Give a polynomial-time algorithm that either produces a schedule completing all jobs by their deadlines, or reports (correctly) that no such schedule exists. You may assume that all the parameters associated with the problem are integers.

Example. Suppose we have two jobs J_1 and J_2. J_1 arrives at time 0, is due at time 4, and has length 3. J_2 arrives at time 1, is due at time 3, and has length 2. We also have two processors P_1 and P_2. P_1 is available between times 0 and 4; P_2 is available between times 2 and 3. In this case, there is a schedule that gets both jobs done:

- At time 0 , we start job J_1 on processor P_1.
- At time 1 , we pre-empt J_1 to start J_2 on P_1.
- At time 2 , we resume J_1 on P_2. (J_2 continues processing on P_1.)
- At time 3, J_2 completes by its deadline. P_2 ceases to be available, so we move J_1 back to P_1 to finish its remaining one unit of processing there.
- At time 4, J_1 completes its processing on P_1.