Problem Set 2—Due Thursday., April 29, 2 PM

Homework Info Homeworks are to be turned in to me, electronically, in class, or put in my mailbox in the CS office. No late homeworks will be accepted.

(15) Problem 1. Suppose we consider the uniform machine scheduling problem where we have n tasks with processing requirements $p_1 \geq p_2 \geq \ldots p_n$ and m processors with speeds $s_1 \geq s_2 \geq \ldots s_m$, so the ith task would take p_i/s_j units of time to complete on the jth processor.

Consider the setting where all tasks are available at time zero and must be completed by a common deadline D and we allow preemption (so a task can run partly on one machine, be interrupted, and resumed on a different machine, or at a later time on the same machine). We noted in class that a necessary and sufficient set of conditions was that:
$s_1 D \geq p_1$
$(s_1 + s_2)D \geq p_1 + p_2$
$\ldots$
$(s_1 + s_2 + \ldots s_{m-1})D \geq p_1 + \ldots + p_{m-1}$
$(s_1 + s_2 + \ldots s_m)D \geq p_1 + \ldots + p_n$

However, give an example that the simple wrap around algorithm we used for identical processors does not work here (even when the above conditions are met). Recall that the wrap around algorithm would schedule task 1 on processor 1 for p_1/s_1 time units, then start task 2 and run it on processor 1 till finished or we hit time D, and if that happens, we finish the rest of task 2 on processor 2 (starting at time zero), and so on.

Suggest intuitively how we might try and fix the problem of the simple wraparound algorithm (but you are not being asked to describe exactly how to do it).

(25) Problem 2. Now consider the scheduling problem on m identical processors that we discussed in class: we have n tasks where each task i has a processing requirement p_i a release time r_i and a deadline d_i. We want to find a preemptive schedule that completes each task in the time window between its r_i and d_i. We showed that this could be solved using a network flow formulation.

Now consider the following extension of this problem: suppose we discover our original problem is infeasible. We can try and make it feasible by increasing all the deadlines by an amount L. That is, we get a new problem where the deadline of each task i is $d_i + L$. L is called the *lateness* of the schedule (i.e each task is late by at most L compared to its original deadline).

Describe an efficient way to find the least value of L that makes the problem feasible and give the run time of your solution method. You can simplify the problem by assuming L is an integer value, though a full solution would allow L to be a real value.

(20) Problem 3. In our discussion of Matula's algorithm for undirected edge connectivity we used a result that for a dominating set S, if the minimum cut size λ was smaller than the minimum degree of a node (δ), then for any minimum cut $(A, \bar{A})$ S will always have a node in both A and in $\bar{A}$.

Prove this fact (hint: consider for contradiction a cut $(A, \bar{A})$ of size smaller than δ with all of A in S . Using the minimum degree requirement and the fact that S is a dominating set get a contradiction based on the number of edges crossing the cut.

(30) Problem 5. Suppose we have a bipartite flow network (like the type we described in class for the scheduling problem of problem 2). In this network we have two sets of nodes L and R (size n_L, n_R respectively) and we have three types of directed arcs:
(s, u) for each $u \in L$,
(u, v) with $u \in L$ $, v \in R$ Note: not all (u, v) will be in our graph, but you can assume a dense graph where most are
(u, t) for each $u \in R$.

a) Describe what an augmenting path looks like in G_f in terms of nodes in L, R. Suppose n_L is much smaller than n_R, give a bound on the maximum length of an augmenting path

b) We now look at a specific class of bipartite flow networks we will call $G(k)$: in this graph $n_L = k$ and $n_R = k^2$. Each $(s, u), u \in L$ has capacity k. All other arcs have capacity 1.

Analyze the run time of the shortest augmenting path algorithm (where we keep distance labels $d(u)$ indicating our estimate of $u's$ distance to t in G_f) as a function of k . Be sure to note the number of edges m in our graph, n the total number of nodes, and f^* the maximum possible flow value as part of your analysis. ((hint, you need not let the distance labels get as high as n).

c) Same capacities as in part b, but now $n_L = k^2$ and $n_R = k^3$

i) As in b analyze the run time of the shortest A-path algorithm

ii) Give an improved two phase algorithm that stops the shortest A-path algorithm when the labels get too large (and we are close enough to the max flow to just use BFS in our phase two algorithm). Analyze the run time of your two phase algorithm.