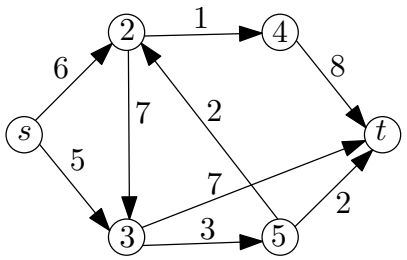
Spring 2015

18.310A Homework 5

Due Mon April 6th at 10AM in lecture

Instructions: Collaboration on homework is permitted, but you must write the solutions yourself; no copying is allowed. Please list the names of your collaborators; if you worked alone, state this. Also indicate any sources you consulted beyond the lecture notes.

1. Consider the following flow problem instance, with source s and sink t:



   Find a maximum flow with the augmenting path algorithm (show the steps), and also derive from the optimum flow an s-t cut of the same value.

2. Suppose we are given a set B of tasks that need to performed by a set A of people. We are also given a set $E \subseteq A \times B$ such that person a is suitable for task b if and only if $(a, b) \in E$. Each person can be assigned to at most one task, and every task should be done by at most one person. The goal is to find the maximum number of tasks that can be completed. Model this problem as a maximum flow problem, and argue why the optimum flow gives an optimum assignment of persons to tasks.

3. In this exercise, you'll derive yet another way of sorting an array by successively sorting subarrays. We need the following notion. *h-sorting* an array A of length n means sorting all subarrays consisting of the elements $A[i], A[i+h], A[i+2h], \cdots$ for all $i = 1, 2, \cdots, h$. h-sorting an array thus results in an array that is *h-sorted* which means that $A[i] \leq A[i+h]$ for all $i = 1, \cdots, n-h$. (1-sorting is thus equivalent to sorting.)

   (a) Let A be an array of size n which is already h-sorted for some $h \leq n$. Consider any $k \neq h$, and let B be the array resulting from k-sorting A. Write a proof that B is both h-sorted and k-sorted.

   (b) Suppose that an array A is both 2-sorted and 3-sorted. Suppose we now apply InsertionSort where InsertionSort processes elements 2 to n in turn and, when processing element i, inserts it into the sorted subarray consisting of the first $i-1$ elements by first comparing element i to element $i-1$ then to element $i-2$ and so on until it finds the right place to insert it.

Argue that InsertionSort performs at most 2 comparisons when trying to insert element i at the right position (and the second comparison is really not needed).

(c) Let JumpSort be the algorithm that does h-sorting with increments $h_{ij} = 2^i 3^j$ where i ranges from $\lfloor \log_2(n) \rfloor$ down to 0, and j ranges from $\lfloor \log_3(n) \rfloor$ down to 0. We don't really care in which order we process all the possible pairs (i, j) except that we insist on using both the increments $2^{i+1}3^j$ and $2^i 3^{j+1}$ some time before using the increment $2^i 3^j$. Does JumpSort output a sorted array? Why?

(d) How many comparisons does JumpSort with this sequence of increments do in the worst-case on an array of length n? (As usual, we care only about the leading term as n grows.)