CSci 421
Introduction to Algorithms

Homework Assignment 4
Due: Friday, Feb 6, 2001
Midterm – Friday, 2/13

Winter 2004 Handout 4

W. L. Ruzzo January 31, 2004

Reading Assignment: Sections 7.1, 7.6, 7.3, 7.9. (Rest of ch 7 next week, I think.)

Homework:

1. Run the string alignment algorithm given in lecture (similar to Fig 6.27, pg 158) on strings $S = tcatag$ and $T = tataag$. Build the cost matrix and traceback pointers as in the example given in lecture. Assume that aligning two identical letters gives a score of $+2$, whereas aligning a letter with a mismatched letter or a gap ("–") gives a score of -1.

2. For many applications of the string alignment algorithm, including most biological applications, it's not true that "all gaps are created equal." For example, if a gap in an alignment reflects a rare evolutionary event where there was an insertion or deletion in one sequence with respect to the other, then 10 separate gaps of length 1 may be much less likely that one gap of length 10. E.g. abcdewxyz is more likely to be related to abcde0123456789wxyz than to 0a1b2c3d4e5w5x7y8z9.

   Extend the string alignment algorithm to handle either of the following gap cost models. (You may choose which of these options you want to do; extra credit for doing both.)

   (a) **General gaps costs:** For general gap costs, part of your input is a table defining a cost function $C(j), 1 \leq j \leq max(m,n)$ which gives the cost of a gap of length j. You may assume $C(j) < 0$ for all j.

   (b) **"Affine" gap costs:** For affine gap costs, you are given two negative constants C_i and C_e, and a gap of length j costs $C_i + C_e * j$. Typically, $C_i \ll C_e < 0$, so that it is fairly expensive to initiate a gap (C_i), and less expensize (per letter) to extend one (C_e).

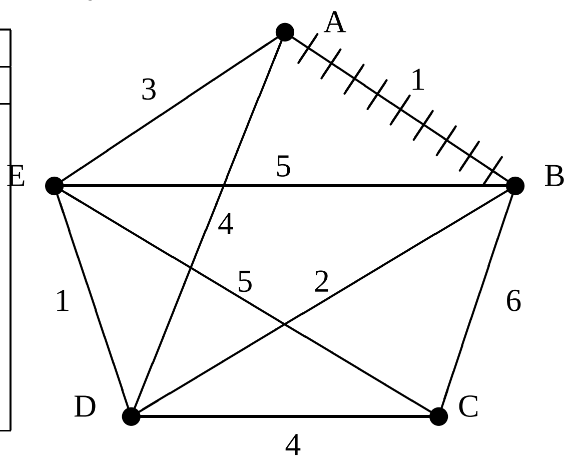
   As usual, give the algorithm, explain why it is correct, and analyze its running time. [The fastest known algorithm for general gap costs is slower than the basic $O(mn)$ algorithm presented in lecture, but there is a version handling affine gap costs in that time.]

3. You are given a binary tree with n leaves, with the ith leaf labeled by a letter S_i from a fixed alphabet Σ. You are also given a function c, which assigns a cost $c(S, S') \geq 0$ to each pair of letters S, S' in Σ. Assume $c(S, S') = c(S', S)$ and $c(S, S) = 0$ for all $S, S' \in \Sigma$. The problem is to give each internal node x of the tree a label $S_x \in \Sigma$ so as to minimize the total over all tree edges of the cost of that edge, where the cost of an edge whose end points are labeled U and V is $c(U, V)$. Give an efficient algorithm to solve this problem, explain why it is correct, and analyze its running time as a function of n and $|\Sigma|$. Assume that each evaluation of the cost function c costs $O(1)$ operations; e.g., via table lookup using a table built in to your algorithm. Hint: use dynamic programming. As we've seen before, you might need a stronger induction hypothesis; i.e., you'll calculate more at each internal node than just whether to put letter "S" there. For instance, it might help to know the total cost of the subtree rooted there, assuming that it's labeled "S".

[Although it doesn't matter for purposes of this homework, this algorithm is sometimes used in estimating evolutionary trees. The S_i are letters at corresponding positions in DNA or protein sequences from n different modern species, the tree is their presumed evolutionary tree, and the goal is to try to reconstruct the corresponding letters in the ancestral sequences. For example, with the cost function $c(S, S') = (\text{ if } S == S' \text{ then } 0 \text{ else } 1)$, the min cost is simply counting the minimum number of "mutation" events in the evolutionary tree necessary to explain the observed variability in the modern species with respect to their presumed common ancestors.

4. Simulate both Kruskal's minimum spanning tree algorithm (presented in lecture) and Prim's algorithm (presented in section 7.6) on the graph shown below. For each, in a table like the one below, list, in order, the edges considered, their weight, and the action taken by the algorithm, namely "discarded", or "selected to be in tree". Highlight the selected edges, say, by drawing a squiggly line along each selected edge in the figure. Assume the first edge is the one done for you below.

Kruskal's Algorithm			Prim's Algorithm		
Edge	Weight	Action	Edge	Weight	Action
1. AB	1	Selected	1. AB	1	Selected
2.			2.		
3.			3.		
4.			4.		
5.			5.		
6.			6.		
7.			7.		
8.			8.		
9.			9.		



5. Let G denote a connected undirected graph with weighted edges. The weights need not be distinct. A *cut* of G is simply a partition of its vertices into two nonempty subsets; an edge *crosses the cut* if one end point is in one part of the partition and the other endpoint is in the other part.

   Prove or disprove the following:
   Let e be an edge of G. Then the following statements are equivalent:

   (a) There exists a minimum spanning tree containing the edge e;

   (b) Every simple cycle containing e contains another edge at least as heavy as e;

   (c) There exists a cut crossed by e for which every edge crossing the cut is at least as heavy as e;