CS 332 Exam 2

Spring, 2005

Problem 1: Do Exercises 8.1-1 and 8.2-2 on page 155 of the book "Introduction to Algorithms, 1st edition" by Cormen, Leiserson, and Rivest. These two problems are about the QuickSort algorithm. Use the version of the Partition() function given on page 154 of CLR.

Problem 2: The following two problems are based on Exercises 17.3-1 and 17.3-2 from the book "Introduction to Algorithms, 1st edition" by Cormen, Leiserson, and Rivest, page 344.

- (a) Prove that a binary tree that is not full cannot correspond to an optimal prefix code, that is, show that you can create another prefix code that encodes some of the letters in the alphabet using fewer bits.
- (b) Give the Huffman code for each character in the following alphabet with the following frequency table (which is based on the first eight Fibonacci numbers).

  a:1 b:1 c:2 d:3 e:5 f:8 g:13 h:21

  Show all of the steps involved in building up the Huffman tree for the code, starting with the forest of single node trees.

Problem 3: Draw a single binary tree *T* such that

- Each node of *T* stores a single character.
- A preorder traversal of *T* yields *examfun*.
- An inorder traversal of *T* yields *mafxuen*.

Problem 4: Explain why an undirected graph with *n* vertices can have at most $\frac{n(n-1)}{2}$ edges. (Hint: Think of the adjacency matrix representation for the graph.)

Problem 5: Can a simple, undirected graph with 8 vertices and 16 edges (and no self edges or multiple edges) have 3 connected components? If so, give an example; otherwise argue why this cannot occur.

Problem 6: Consider the undirected graph G with vertices $V = \{ a, b, c, d, e, f, g \}$ and edges

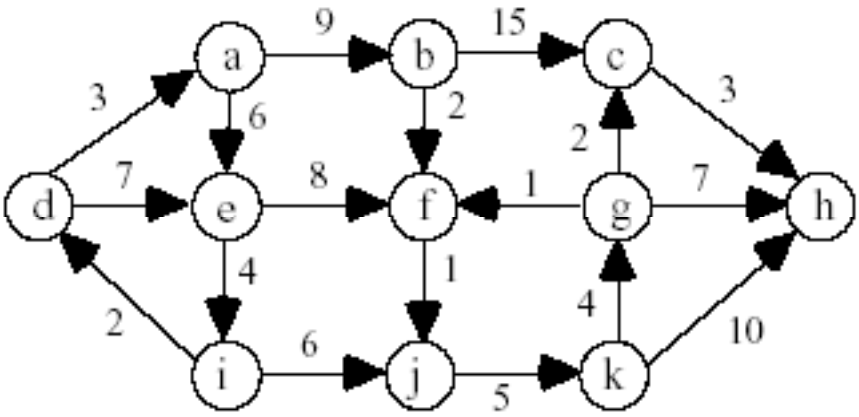
$$E = \{ (a, b), (a, c), (a, d), (a, f), (a, g), (b, c), (b, g), (d, e), (d, f), (e, f) \}.$$

(a) Draw a picture of the graph. Then draw an adjacency list representation of the graph. Have each vertex see its neighbors in alphabetical order.

(b) Draw the BFS spanning tree achieved by starting at vertex *a*, numbering each node as it is inserted into the tree and visiting neighbors in alphabetic order. (Use the algorithm that pushes only unseen vertices into the queue.)

(c) Draw the DFS spanning tree achieved by starting at vertex *a*, numbering each node as it is inserted into the tree and visiting neighbors in alphabetic order. (Use the algorithm that pushes only unseen vertices into the stack.)

Problem 7: Draw the DFS spanning tree that is induced by performing a depth-first-search starting at node *a* in the following directed graph. Iterate through adjacent vertices in alphabetical order. Label each vertex by its discovery and finish times (as in the book by Cormen, Leiserson, and Rivest (CLR), p. 479). Using dotted lines, draw the edges of G that are not in the DFS spanning tree and label them as forward edges (F), back edges (B), or cross edges (C) (as defined in CLR on page 482, and as shown in the examples on pages 479 and 481).



Problem 8: Dijkstra's algorithm assumes that the edge weights of the given graph are non-negative. Here we consider the consequence of allowing negative edge weights.

A *negative-weight cycle* of a weighted graph is a cyclic path whose path weight is negative. If a graph G has some edges with negative weights but no negative-weight cycles, then the single-source shortest path problem is well-defined but Dijkstra's algorithm is not guaranteed to correctly solve the problem in the presence of negative edge weights. Your problem is to show this.

Construct a connected, directed, weighted graph with four vertices and at least one negative weight edge (but no negative-weight cycles) such that Dijkstra's algorithm gives an incorrect solution to the single-source shortest path problem.