Homework 2

Design and Analysis of Data Structures and Algorithms (Spring 2012)
Rutgers University
Swastik Kopparty

Due Date: Wednesday, March 7, 2012

Questions

1. Consider the problem of neatly printing a paragraph on a printer. The input text is a sequence of n words of lengths $l_1, l_2, \ldots, l_n$ measured in characters. We want to print this paragraph neatly on a number of lines that holds a maximum of M characters each. Our criterion of "neatness" is as follows. If a given line contains words i through j, and we leave exactly one space between words, the number of extra spaces at the end of the line is

$$M - (j - i) - \sum_{k=i}^{j} l_k.$$

   We wish to minimize the sum, over all lines except the last, of the squares of the numbers of extra space characters at the end of lines. Give a dynamic-programming algorithm to print a paragraph of n words neatly on a printer. Analyze the running time and space requirements of your algorithm.

2. Let $G = (V, E, w)$ be a graph with nonnegative weights on the edges. In this exercise we will see an algorithm that finds the $n \times n$ matrix of shortest distances between each pair of vertices.

   We will use dynamic programming. Let $OPT_k[u, v]$ be the shortest distance between u and v amongst paths that use at most k edges. Computing OPT_n is our final goal.

   (a) Show how one can compute the matrix OPT_k from the matrix OPT_{k-1}. Use this to design a dynamic programming algorithm for computing OPT_n. It should run in time $O(n^4)$.

   (b) Show how one can compute the matrix OPT_k from the matrix $OPT_{\lceil k/2 \rceil}$. Use this to design a dynamic programming algorithm for computing OPT_n. It should run in time $O(n^3 \log n)$.

   (c) Given the shortest distance matrix, describe how one can compute a shortest *path* between two vertices u, v.

3. Let $G = (V, E, w)$ be a graph with weights on the edges. Let $s \in V$. Dijkstra's algorithm to find all distances from s maintains a set S (initially equal to $\{s\}$) and a vector of distances d, and at each stage it includes into S the vertex $x \in V \setminus S$ which minimizes $d_x = \min_{v \in S}(d(v) + w(v, x))$, and updates $d[x] = d_x$.

   (a) Give an example instance to show that this algorithm will fail if the graph has negative edge weights.

(b) Consider the following new variant of Dijkstra's algorithm. To find all distances from s, the new algorithm maintains a set S (initially empty) and a vector of distances d, and at each stage it includes into S the vertex $x \in V \setminus S$ which minimizes $d_x = \min_{v \in S} w(v, x)$, and updates $d[x] = d[v^*] + d_x$ (where v^* is the $v \in S$ that minimizes $w(v, x)$).

Give an example instance to show that this algorithm will fail on some graph with nonnegative edge weights.

4. Describe the structure of a binomial heap at each stage as the following operations are executed. The heap is initially empty.

   (a) INSERT(3)
   (b) INSERT(10)
   (c) INSERT(5)
   (d) INSERT(8)
   (e) EXTRACT-MIN
   (f) INSERT(6)
   (g) EXTRACT-MIN
   (h) INSERT(11)
   (i) EXTRACT-MIN

5. Describe the structure of a disjoint-set forest at each stage as the following operations are executed. Assume that the disjoint-set forest uses both the following heuristics:

   - The weighted-union heuristic (for a UNION operation, the root of the smaller tree is made to point to the root of the larger tree; don't worry about how the data structure keeps track of which tree is larger). If the trees have equal size, assume that the root of the first argument is made to point to the root of the second argument.
   - The path-compression heuristic (after a FIND operation, all the vertices encountered on the path to the root are made to point directly to the root).

   The universe is the set $\{1, 2, \ldots, 10\}$, and initially all the elements are in singleton sets (so the initial state is $\{\{1\}, \{2\}, \ldots, \{10\}\}$).

   (a) UNION(1,2)
   (b) UNION(3,4)
   (c) UNION(5,6)
   (d) UNION(3,5)
   (e) UNION(1,3)
   (f) UNION(7,8)
   (g) UNION(9,10)
   (h) UNION(7,9)
   (i) FIND(3)
   (j) UNION(1,10)
   (k) FIND(7)

(l) FIND(3)

(m) FIND(4)

(n) FIND(5)