SPEC 67

THE UNIVERSITY OF THE WEST INDIES
ST. AUGUSTINE

EXAMINATIONS OF DECEMBER 2005

Code and Name of Course: COMP2000 – Data Structures   **Paper:**

Date and Time: Tuesday 20th, December 2005   1:00 P.M.   **Duration: 2 Hours**

INSTRUCTIONS TO CANDIDATES: This paper has 4 pages and 3 questions

Answer all questions

1. (a) Write a function which, given a pointer to the root of binary tree, returns the *sum of the levels* of the nodes in the tree by performing a "level order" traversal. Assume the root is at level 0 and an appropriate queue is available with the usual operations. Pseudocode may be used in the body of the function. [4]

(b) Write a *recursive* function which, given a pointer to the root of a binary tree, returns the *sum of the levels* of the nodes in the tree. Assume the root is at level 0. [4]

(c) Each node of a *binary search tree* has fields **left**, **right**, **key** (an integer) and **parent**, with the usual meanings. Write a function which, given a pointer to the root of the tree and an integer **n**, searches for **n**. If found, return a pointer to the node. If not found, add **n** to the tree, ensuring that *all fields* are set correctly; return a pointer to the new node. [5]

(d) Write a function which, given a pointer to a binary search tree as in (c), above, deletes the node containing the smallest number, ensuring that all fields are set correctly. Return a pointer to the root of the modified tree. [5]

(e) (i) A complete binary tree contains 127 nodes. What is its height? [1]

(ii) A data file contains 127 distinct integers in ascending order. Give a detailed algorithm to read the integers and build the *optimal binary search tree* from them. [6]

2. (a) In a certain application, keys which hash to the same location are held on a linked list. The hashtable location contains a pointer to the first item on the list and a new key is placed at the end of the list. Each item in the linked list consists of an integer **key**, an integer **count** and a pointer to the next element in the list. Storage for a linked list item is allocated as needed. Assume that the hash table is of size *n* and the call **H(key)** returns a location from **1** to *n*, inclusive.

   (i) Write programming code, including all relevant declarations, to initialize the hash table; [2]

   (ii) Write a function which, given the key **nkey**, searches for it If not found, add **nkey** in its appropriate position and set **count** to **0**. If found, add **1** to **count**; if count reaches **10**, delete the node from its current postion, place it at the head of its list and set **count** to **0**. [8]

(b) A function **makeHeap** is passed an integer array **A**. If **A[0]** contains **n**, then **A[1]** to **A[n]** contain numbers in arbitrary order.

   (i) Write **makeHeap** such that **A[1]** to **A[n]** contain a max-heap (*largest* value at the root). Your function must create the heap by processing the elements in the order **A[2]**, **A[3]**,...,**A[n]**. [6]

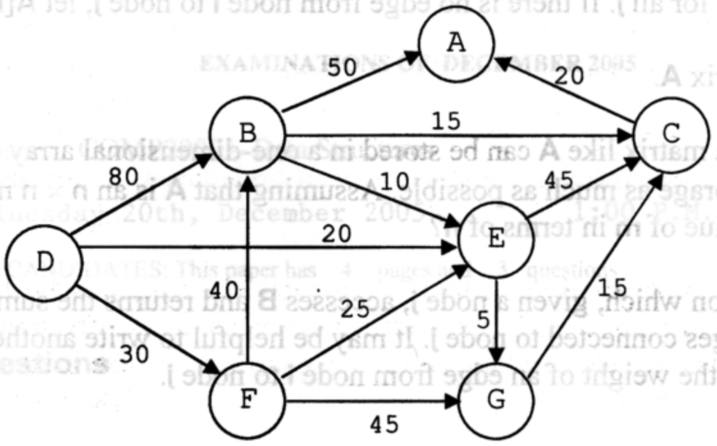
   (ii) If the array **A** contains the following values initially:

   ```
   8  25  42  14  36  50  21  63  48
   ```

   show the contents of **A[1]** to **A[j]** after element **A[j]** is processed. [3]

   (iii) Given an array like **A**, with **A[1]** to **A[n]** containing a max-heap, write a function to sort the array in ascending order. [6]

(3) (a) Given the following graph:



(i) Give the depth-first and breadth-first traversals of the graph starting at D. Edges of a node are processed in alphabetical order. [2]

(ii) Make a copy of the graph *without* the edge weights. Assume that a depth-first traversal is performed starting at D and that *edges of a node are processed in alphabetical order*. Indicate the discovery and finish times for each node and label each edge with T (tree edge), B (back edge), F (forward edge) or C (cross edge), according to its type. [4]

(iii) State, with a reason, whether or not it is possible to topologically sort the nodes of the graph. If it *is* possible, give one solution indicating how you arrived at that solution. [2]

(iv) Derive the minimal-cost paths from node D to every other node using Dijkstra's algorithm. For each node, give the cost and the path to get to the node. At each stage of the derivation, show the minimal cost fields, the parent fields and the priority queue. In your table, list the nodes in alphabetical order. [9]

(b) Suppose that the graph in (a) is undirected; let **A** be its adjacency matrix representation. Assume that each letter node is represented by its position in the alphabet. **A[j, j] = 0** for all **j**. If there is no edge from node **i** to node **j**, let **A[i, j] = 999**.

(i) Draw the matrix **A**. [2]

(ii) Explain how a matrix like **A** can be stored in a one-dimensional array **B[1..m]** conserving storage as much as possible. Assuming that **A** is an **n × n** matrix, what is the value of **m** in terms of **n**?

Write a function which, given a node **j**, accesses **B** and returns the sum of the weights of edges connected to node **j**. It may be helpful to write another function which returns the weight of an edge from node **i** to node **j**. [6]

End of question paper