JFK
65

THE UNIVERSITY OF THE WEST INDIES
ST. AUGUSTINE

EXAMINATIONS OF December 2013

Code and Name of Course: COMP2000 – Data Structures

Date and Time: December 18, 2013, 1.00 p.m. Duration: 2 Hours

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

Answer ALL Questions

1. (a) An *integer* max-heap is stored in an array (**A**, say) such that the size of the heap (**n**, say) is stored in **A[0]** and **A[1]** to **A[n]** contain the elements of the heap with the *largest* value in **A[1]**.

   (i) Write a function **deleteMax** which, given an array like **A**, deletes the largest element and reorganizes the array so that it remains a heap. [5]

   (ii) Given two arrays **A** and **B** containing heaps as described above, write programming code to merge the elements of **A** and **B** into another array **C** such that **C** is in ascending order. Your method must proceed by comparing an element of **A** with one in **B**. You may assume that **deleteMax**, above, is available. [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. 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]

   (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. You may assume that the function call **newNode(n)** creates a node, stores **n** in it and returns a pointer to the node. [6]

2. (a) In a hashing application, the key consists of a string of letters. Write a hash function which, given a key and an integer **max**, returns a hash location between 1 and **max**, inclusive. Your function must use *all* of the key and should not deliberately return the same value for keys consisting of the same letters. [3]

   (b) A hash table of size **n** contains two fields—an integer data field and an integer link field (called *data* and *next*, say). The *next* field is used to link data items in ascending order. A value of -1 indicates the end of the list. The variable **top** (initially set to -1) indicates the location of the smallest data item.

   Integers are inserted in the hash table using "open addressing with double hashing". Assume that the function **h1** produces the initial hash location and **h2** produces the increment. An available location has the value **Empty** and no item is ever deleted from the table.

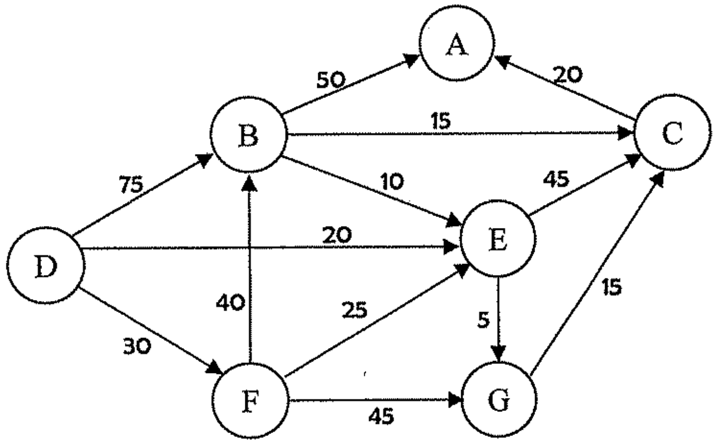
   Write programming code to search for a given value **key**. If found, do nothing. If not found, insert **key** in the table and *link it in its ordered position.* You may assume that the table contains room for a new integer. [10]

2. (c) Given a pointer to the root of a *binary search tree* of integers, write code to delete the root and return a pointer to the root of the new tree. If the root has non-empty left and right subtrees, replace it by its *inorder predecessor*. Each node of the tree contains 3 fields only—**data**, **left** and **right**. [6]

(d) Write a 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. [3]

(e) In a binary tree of n nodes, an 'external' node is attached to each null pointer. If I is the sum of the levels of the (internal) nodes of the tree and E is the sum of the levels of the external nodes, prove that $E - I = 2n$. [3]

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 heading, *list the nodes in alphabetical order*. [9]

3. (b) Suppose that the graph in (a) is undirected, and let **A** be its adjacency matrix representation. Assume that each letter node is represented by its position in the alphabet. **A[i, i] = 0** for all **i**. 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**? [2]

(iii) Write a function which, given a node **k**, accesses **B** and returns the sum of the weights of edges connected to node **k**. It may be helpful to write another function which returns the weight of an edge from node **i** to node **j**. [4]

End of question paper