JFK

65

THE UNIVERSITY OF THE WEST INDIES
ST. AUGUSTINE

EXAMINATIONS OF December 2010

Code and Name of Course: COMP2000 – Data Structures

Date and Time: Wednesday 8th December 2010 1 pm. Duration: 2 Hours

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

Answer ALL Questions

1. (a) The following are the inorder and postorder traversals of the nodes of a binary tree:

   Inorder: H E R L F P G B U M
   Postorder: H R E L G P U B M F

   Draw the tree. [5]

   (b) 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. [5]

   (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. [6]

   (d) The integer elements of an almost complete binary tree are stored in an array **A[1..n]**, with the root in location 1.

   (i) *Without sorting*, write a function to rearrange the elements of **A** so that the *smallest* integer is in location 1, and the smallest integer of each subtree is also in the root of that subtree. The elements are to be processed in the order **A[2]**, **A[3]**, and so on, up to **A[n]**. [6]

   (ii) If the array **A** contains the following values initially (**n** = 8):

   42 25 14 36 50 21 12 31

   show the contents of **A[1]** to **A[j]** after element **A[j]** is processed (**j** = 2, 8). [3]

Go to next page

2. (a) A hash table is to be used to implement a *self-organizing list*. 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 *head 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
- to declare a list node element
- to create and initialize a list node element
- to declare and initialize the hash table [4]

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

(b) An **n** × **n** matrix **A** is used to store the points obtained in cricket matches among **n** teams. A team gets 3 points for a win, 1 point for a tie and 0 points for a loss. **A[i, j]** is set to 3 if team **i** beats team **j**; it is set to 1 if the match is tied and it is set to 0 if team **i** loses to team **j**.

In order to conserve storage, the values in the (strictly) lower triangle of **A** are stored in an array **B[1..m]** in row order.

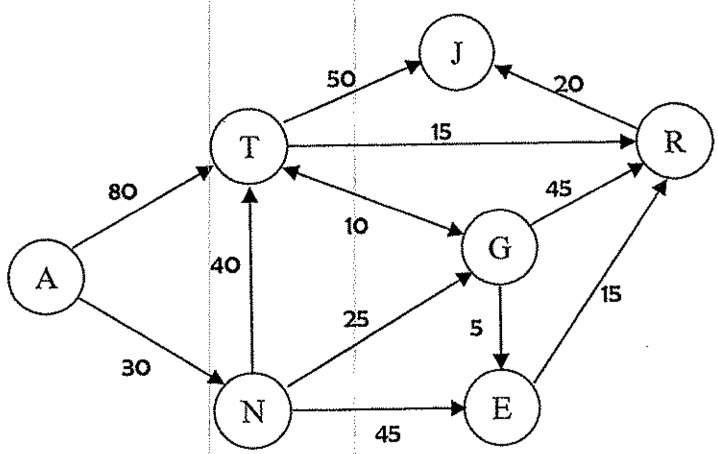
(i) What is the value of **m** in terms of **n**? [1]

(ii) Write a function **score(i, j)** which, by accessing **B**, returns the value of **A[i, j]**. If **i** or **j** is invalid, the function returns −1. [5]

(iii) Using the function in (ii), write another function which, given **t**, returns the total number of points earned by team **t**. [3]

(c) A function is given an integer array **A** and two subscripts **m** and **n**. The function must rearrange the elements **A[m]** to **A[n]** and return a subscript **d** such that all elements to the left of **d** are less than or equal to **A[d]** and all elements to the right of **d** are greater than **A[d]**. Write the function. [4]

(3) (a) Given the following graph:



(i) Draw the adjacency list representation of the graph. [2]

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

(iii) Make a copy of the graph *without* the edge weights. Assume that a depth-first traversal is performed starting at A 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. [5]

(iv) Derive the minimal-cost paths from node **A** to every other node using Dijkstra's algorithm. 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*.

For each node, give the cost and the path to get to the node from **A**. [10]

(b) Assuming that the graph in (a) is undirected, draw the minimum spanning tree obtained by using Kruskal's algorithm. Show the steps in your derivation. [3]

(c) Assuming that the graph in (a) is undirected, draw the minimum spanning tree obtained by using Prim's algorithm, starting with node N. Show the steps in your derivation. [3]

End of question paper