mD3
36

THE UNIVERSITY OF THE WEST INDIES
ST. AUGUSTINE

EXAMINATIONS OF December 2007

Code and Name of Course: COMP2000 – Data Structures

Date and Time: Monday 10th December 2007 A.M. Duration: 2 Hours

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

Answer all questions

1. (a) The integer elements of an almost complete binary tree are stored in an array **A[1..n]**, with the root in location 1. 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. [6]

   (b) Draw a non-degenerate binary tree of 5 nodes such that the pre-order and level-order traversals produce identical results. [3]

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

Inorder:	G D P K E N F A T L
Postorder:	G P D K F N T A L E

   Draw the tree. [5]

   (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]

   (f) Write a structured, non-recursive algorithm to traverse a binary tree in post-order. The only pointer fields in each node are **left** and **right**. You may assume the existence of the usual functions for manipulating a stack. [5]

2. (a) Integers are inserted in an integer hash table **list[1]** to **list[n]** using "open addressing with double hashing". Assume that the function **h1** produces the initial hash location and the function **h2** produces the increment. An available location has the value **Empty** and a deleted location has the value **Deleted**.

Write a function to search for a given value **key**. If found, the function returns the location containing **key**. If not found, the function inserts **key** in the *first* deleted location encountered (if any) in searching for **key**, or an **Empty** location, and returns the location in which **key** was inserted. You may assume that **list** contains room for a new integer. [6]

(b) In another hashing application, keys which hash to the same location are held on a linked list *sorted in ascending order* with the hashtable location containing a pointer to the first item on the list. Each item in the linked list consists of an integer key and a pointer to the next element in the list. Use the names *key* and *next* for these fields. Storage for a linked list item is allocated as needed. Assume that the hash table is of size 53.
Write programming code, *including all relevant declarations*, to

(i) initialize the hash table; [2]

(ii) search for the number *m*. If found, return the pointer to the node containing *m*. If not found, insert *m* in its appropriate position. [6]

(c) An **n** x **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**? [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]

(d) 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]**. [5]

3. (a) A directed graph **G(V, E)** is represented using *adjacency lists*. The graph has *n* nodes stored in an array **G[1..n]**. Each node, **G[j]**, has the following fields:

```
char id; //the name of the node; a single character e.g. A, F, T
int colour;
int parent; //an array subscript indicating the location of a parent node
GEdgePtr first; //pointer to the first edge from the node; null, if none
```

Each edge node has the following fields:

```
int child; //a subscript in G; G[child] is the child node
int weight;
GEdgePtr next; //pointer to the next edge; null, if none
```

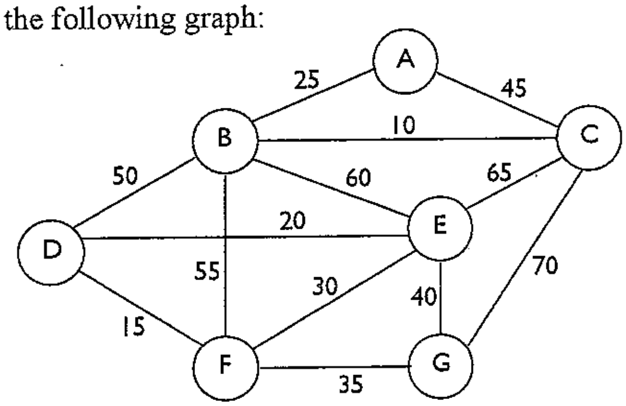
(i) Write a function which, given **G** and **n**, outputs the graph, listing the nodes in the order in which they appear in **G**. Each node is followed by the name and weight of the edges leaving it. [4]

(ii) Write a function which, given **G**, **n** and a node name **S**, performs a depth-first traversal of **G** starting at **S**. Output the name of a node as it is visited and set the **parent** fields so that a depth-first path can be determined. [8]

(iii) Assuming that a depth-first traversal has already been done as in (ii), write a function which, given **G**, **n** and a node *name* **D**, prints the names of the nodes, in the order visited, on the depth-first path from the source node to **D**. [5]

(b) The *indegree* of a node is the number of edges leading into it. Assume that the field **inDegree** is added to a graph node defined in part (a), above. Write a function which, given **G** and **n**, correctly sets the **inDegree** field of all the nodes. [3]

(c) Given the following graph:



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

(ii) Draw the minimum spanning tree obtained using Kruskal's algorithm. [3]

End of question paper