BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI (RAJ.)

II Semester 2018-19 CS F211 Data Structures and Algorithms

Online Test-2

Date: 14th April, 2019 Duration:160 minutes Weightage: 56M

Note: (a) Do not change the prototypes of given functions and names of global variables.

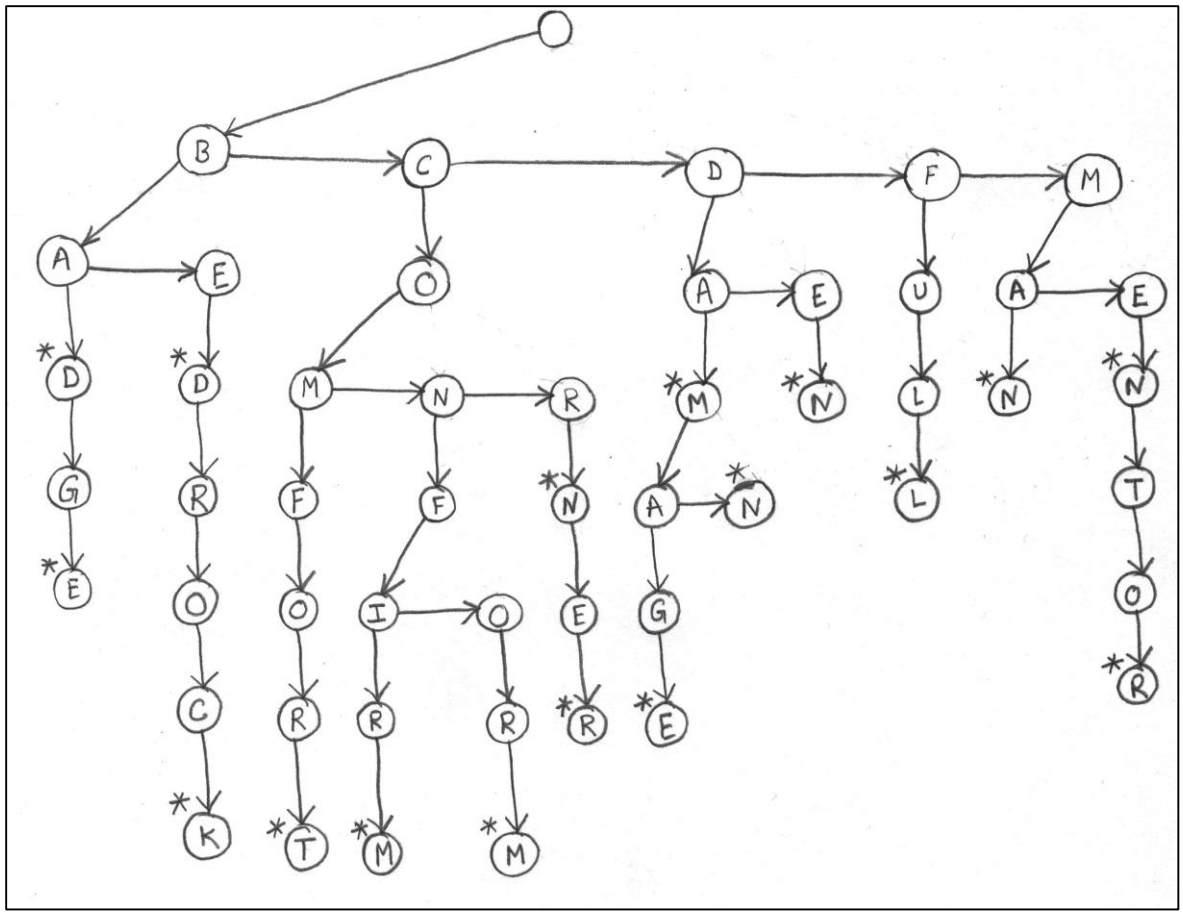
(b) If you want to write any function other than asked in the question paper, write it in the same file where you are calling it. Each individual file will be tested independently with their own driver.

Trie is an efficient information reTrieval data structure. Every node of Trie consists of multiple branches. Each branch represents a possible character of keys. We need to mark the last node of every key as stop_word. A simple structure to represent node of **Capital Case** English words in Trie is as follows:

```
typedef struct node NODE;
struct node
{
 char c;                 //stores character of the key. Assume all characters are capital case letters
 NODE *child;            //Left Child
 NODE *sibling;          //Right Sibling
 boolean stop_word;      //TRUE if word is in the Dictionary (i.e. it is a key); FALSE otherwise
};
```

For today's problem Trie is implemented as **Left-Child Right-Sibling** representation. In this, a node holds just two references: first a reference to its leftmost child, and the other to its immediate next right sibling. For more understanding see the following Trie representation for set of keys = {BADGE, CORN, BED, COMFORT, BEDROCK, CONFIRM, CONFORM, FULL, CORNER, DAMN, DAM, DAMAGE, BAD, DEN, MAN, MEN, MENTOR}. Star (i.e. *) on a node indicates that stop_word field is TRUE for this node; else it is FALSE. Root does not contain any character.

[Note: Note that characters stored in siblings of any node are sorted in increasing order]



File node.h contains structure for implementing node. It also contains global variable "**root**" which points to the Root of Trie. Memory for root is assigned in main() function.

File main.c is a driver file. It also contains implementation of two functions: create_node() and print_list(). See the file for their functionality.

Now, implement the following functions in respective files.

Q1. `void insert (NODE** head, NODE* new_node)` [5M]

Precondition: head stores the address of pointer to a node and new_node stores the address of new_node. "head" refers to any node whose address is stored in the child pointer of some other node. For example, in the figure given, at level-1, the only head is the node with value "B". Nodes with respective values "A", "O", "A", "U", "A" are heads at level-2, and so on. The **new_node**'s fields are set, i.e. its character field is set to appropriate value, all pointer fields are set to NULL, and **stop_word** field is set to FALSE.

Post condition: Stores new_node as one of the siblings of head such that all resulting siblings of head are sorted in increasing order as per character stored.

insert function will have binary marking.

Q2. `NODE* search(char ch, NODE* head)` [6M]

Precodition: head stores the pointer of a node and its definition is as explained in Q1 above. Character ch is to be searched within head and all its siblings.

Postcondition: It returns the address of the node in which ch is present. Otherwise, if head is NULL or ch is not present, it returns NULL.

search function will have binary marking.

Q3. `boolean search_word(char *word)` [15M]

Precondition: "word" points to the upper-case English word with length > 1.

Post condition: the function returns TRUE if "word" is present in the Trie; FALSE otherwise. Note that global variable "root" stores the root of the Trie data structure. *You have to strictly implement this fun as non-recursive.*

Q4. `void insert_word (char *word)` [20M]

Precondition: "word" points to the upper-case English word with length > 1.

Post condition: The function inserts each character of "word" in Trie appropriately. Note that global variable "root" stores the root of the Trie datastructure. *You have to strictly implement this function as non-recursive.*

Q5. `void traverse_tree(NODE *roo)` [10M]

Precondition: "roo" points to the root of Trie. Note that here root of Trie is passed explicitly.

Post condition: If all the words stored in Trie are arranged in lexicographically increasing order, this function will print the last character of each word in same order. It does so ***recursively***. For example, for the Trie shown in question, it prints D E D K T M M N R M E N N L N N R. *You have to strictly implement this function as a recursive function.*

traverse_tree function will have binary marking.