King Saud University
College of Computer and Information Sciences
Department of Computer Science

Data Structures (CSC212)

Third Trimester 2023

Course Project

25 Marks
Due Date: Phase 1 (18 May 2023 11:59pm).
Due Date: Phase 2 (3 June 2023 11:59 pm).

Project Report 2

Group: 2
Section: 72088
Leader name: AlJouri Abdulaziz AlSarami
Instructor name: Lubna Alhinti

Student Name	Student ID
Amal Mousa Aljassas	442201829
Nouv Batey AlQahtani	442201905
AlJouri Abdulaziz AlSarami	442202991

Tasks Distribution

Student Name	Tasks
Amal Mousa Aljassas	▪ Creating and formatting the report. ▪ Provide a criticism of the method *follow*. ▪ Calculating the time complexity of methods ***follow1(String path), follow(MNode<T> t, String path), findkey(int tkey),*** and ***findkey1(int tkey, MNode<T> p)***.
Nouv Batey AlQahtani	▪ Provide a criticism of the method escape. ▪ Calculating the time complexity of methods ***escape (MNode<T> t)*** and ***escape1().***
AlJouri Abdulaziz AlSarami	▪ Provide a criticism of the method **shortPath and shortPath1.** ▪ Calculate the time complexity of both the methods.

Review and Critique:

1. As for the ***follow*** method, their code is very simple and fulfils the requirement, but it does not allow the user to enter the key and then the path, which means that the code was not built to interact with the user's input.
2. As for the ***escape*** method, the code is functional, and the code is performing the preorder traversal of the binary tree. However, there are a few improvements and suggestions that I would like to point out, the code should include a condition to check if the current node is a leaf node in the binary tree before checking its right and left child nodes by checking the nullity of its two children.
   And a recursive implementation could be used instead of using a stack, which would be more memory-efficient and easier to read.
3. As for the ***shortPath*** method, the code provided seems to be correct and fullfills all the functionalities that are required from it. However, I have noticed that it lacks the input validation on the method shortPath(), which could lead to possible null pointer exceptions. After analyzing how these two methods work together, I have noticed that it uses unnecessary redundancy and lacks efficiency in watching out for unnecessary recursive calls.

Time complexity:

1. **Time complexity of method *follow1(String path)*:**

	Statements	Step/ Execution	Freq.	Total
1	`private boolean follow1(String path) {`	0	-	-
2	`return follow (current, path));`	1	$T(n)$	$2^n(8) + 10m + 1$
3	`}`	0	-	-
Total:	$2^n(8) + 10m + 1$			
O:	$O(2^n)$ Exponential Growth			

- **Time complexity of assist methods of method *follow1(String path)*:**
    - Time complexity of method ***follow(MNode<T> t, String path)***:

Assume: m = path.length()

	Statements	Step/ Execution	Freq.	Total
1	**private boolean** follow(MNode<T> t, String path){	0	-	-
2	**if**(t == **null**)	1	1	1
3	**return false**;	1	1	1
4	if (!findkey(t.key))	1	$2^n(8) - 5$	$2^n(8) - 5$
5	**return false**;	1	1	1
6	MNode<T> p = t;	1	1	1
7	**for**(**int** i =0;i<path.length();i++){	1	m+1	m+1
8	**if** ((**char**)p.data != path.charAt(i))	1	m	m
9	**return false**;	1	m	m
10	**if**(i<path.length()-1){	1	m	m
11	**if**(p.left != **null** &&(**char**)p.left.data==path.charAt(i+1))	1	m	m
12	p=p.left;	1	m	m
13	**else if** (p.right != **null** &&(**char**)p.right.data==path.charAt(i+1))	1	m	m
14	p=p.right;	1	m	m
15	**else**	1	m	m
16	**return false**;	1	m	m
17	}	0	-	-
18	}	0	-	-
19	**return true**;	1	1	1
20	}	0	-	-
Total:	$2^n(8) + 10m + 1$			
O:	$O(2^n)$ Exponential Growth			

- Time complexity of method ***findkey(int tkey)***:

	Statements	Step/ Execution	Freq.	Total
1	**private boolean** findkey(**int** tkey) {	0	-	-
2	**return** (findkey1(tkey, root));	1	$T(n)$	$2^n(8) - 5$
3	}	0	-	-
Total:	$2^n(8) - 5$			
O:	$O(2^n)$ Exponential Growth			

- Time complexity of method ***findkey1(int tkey, MNode<T> p)***:

	Statements	Step/ Execution	Freq.	Total
1	**private boolean** findkey1(**int** tkey,MNode<T> p) {	0	-	-
2	**if**(empty() || p == **null**)	1	1	1
3	**return false**;	1	1	1
4	**if**(p.key == tkey)	1	1	1
5	**return true**;	1	1	1
6	**else** {	1	1	1
7	**boolean** n = findkey1(tkey, p.left);	1	$T(n-1)$	$T(n-1)$
8	**if** (n != **false**)	1	1	1
9	**return** n;	1	1	1
10	**else**	1	1	1
11	**return** findkey1(tkey, p.right);	1	$T(n-1)$	$T(n-1)$
12	}	0	-	-
13	}	0	-	-
Total:	$8(2)^n - 2$			
O:	$O(2^n)$ Exponential Growth			

Total:

$T(0) = 3 \rightarrow O(1)$

$T(n) = 5 + T(n-1) + T(n-1)$

$T(n) = 5 + 2T(n-1)$

$T(n) = 15 + 4T(n-2)$

$T(n) = 35 + 8T(n-3)$

$T(n) = 5(2^k - 1) + 2^k T(n-k)$

$T(n) = 5(2^n - 1) + 2^n(3)\ when\ n = k$

$T(n) = 2^n(8) - 5$

$Therefore, \quad O(2^n)$

2. Time complexity of method *escape() and its* assist method escape1() :

	Statements	Step/ Execution	Freq.	Total
1	**public boolean** escape1(){	0	-	-
2	**return** escape(current);	1	$T(n)$	$8n+7$
3	}	0	-	-
Total:	$8n+7$			
O:	$O(n)$ Exponential Growth			

	Statements	Step/ Execution	Freq.	Total
1	**private boolean** escape(MNode<T> t){	0	-	-
2	**if** (t == **null**)	1	1	1
3	**return false**;	1	1	1
4	MNode<T> p = t;	1	1	1
5	LinkedStack <MNode<T>> stack = **new** LinkedStack <>();	1	1	1
6	stack.push(p);	1	1	1
7	**while** (! stack.empty()){	1	n+1	n+1
8	p= stack.pop();	1	n	n
9	**if**((**char**)p.data == 'X')	1	n	n
10	**return true**;	1	n	n
11	**if** (p.right != **null**)	1	n	n
12	stack.push(p.right);	1	n	n
13	**if** (p.left!=**null**)	1	n	n
14	stack.push(p.left);	1	n	n
15	}	0	-	-
16	**return false**;	1	1	1
17	}	0	-	-
Total:	$8n + 7$			
O:	$O(n)$ Exponential Growth			

3. Time complexity of method shortPath():

	Statements	Step/ Execution	Freq.	Total
1	**private String** shortPath(<T> t){	0	-	-
2	**String** path1,path2,path;	1	1	1
3	path1=path2=path="";	1	1	1
4	**if**(!escape(t))	1	1	1
5	**return null;**	1	1	1
6	MNode<T> p = t;	1	1	1
7	path=""+p.data;	1	1	1
8	**if**(escape(p.right))	1	1	1
9	path1=shortPath(p.right);	1	T(n/2)	T(n/2)
10	**if**(escape(p.left))	1	1	1
11	path2=shortPath(p.left);	1	T(n/2)	T(n/2)
12	**if**(path1.equals(""))	1	1	1
13	**return** path+path2;	1	1	1
14	**else if**(path2.equals(""))	1	1	1
15	**return** path+path1;	1	1	1
16	**if**(path1.length()<path2.length())	1	1	1
17	**return** path+path1;	1	1	1
18	**else if** (path2.length()<=path1.length())	1	1	1
19	**return** path+path2;	1	1	1
20	**return null;**	1	1	1
21	}	0	-	-
Total:	$2\left(\frac{T}{n}\right) + 17$			
O:	$O(n\ log)$ Logarithmic Growth			

$$T(n) = 2T\left(\frac{n}{2}\right) + O(1)$$

$$a = 2\ , b = 2\ , d = 1$$

$$\frac{a}{b^d} = \frac{2}{2} = 1$$

Using the master theorem, we conclude that: $O(n^d \log n) = O(n \log)$

- **Time complexity of method shortPath1():**

	Statements	Step/ Execution	Freq.	Total
1	**public String** shortPath1(){	0	-	-
2	**return** shortpath(root);	1	T(n)	2(T/n) + 17
3	}	0	-	-
Total:	$2\left(\frac{T}{n}\right) + 17$			
O:	$O(n\ log)$ Logarithmic Growth			