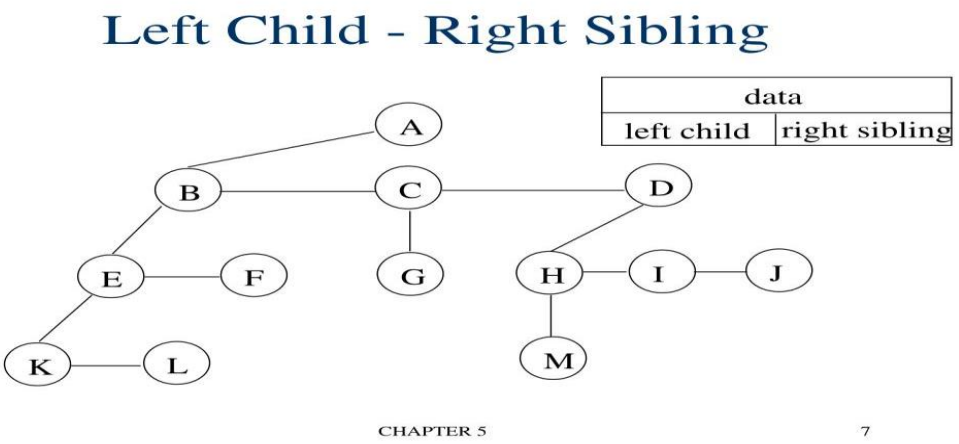
Data Structures and Algorithms: Assignment 2

مسألة الجد المميز

الحل : البنية المستخدمة للحل هي TREE وسوف أقوم بتمثيلها على طريقة Left Child Right Sibling

The left-child right-sibling representation is a way of representing an n-ary tree (a tree where each node can have any number of children) using a binary tree (a tree where each node can have at most two children). In this representation, each node holds two references: one to its leftmost child and one to its immediate right sibling. The leftmost child is the first child of the node, and the right sibling is the next child of the same parent node. This representation allows us to represent an n-ary tree using only two pointers per node, instead of having to store an array of pointers to all the children of each node.



This is a class named Node that represents a node in a tree. It has five instance variables: name, Parent, gender, firstChild, and nextSibling. The name and gender instance variables store the name and gender of the node, respectively. The Parent instance variable stores the name of the parent node. The firstChild and nextSibling instance variables store references to the first child and next sibling of the node, respectively.

The class has a constructor that takes in three parameters: name, ParentName, and gender. It initializes the instance variables with the given values and sets the firstChild and nextSibling instance variables to null.

The class also has several getter and setter methods for its instance variables. The getParent, setParent, getName, setName, getGender, setGender, getFirstChild, setFirstChild, getNextSibling, and setNextSibling methods are used to get and set the values of the corresponding instance variables.

The class also has a toString method that returns a string representation of the node. It returns a string containing the values of the instance variables in a specific format.

```
CLASS Node
    PRIVATE name
    PRIVATE Parent = null
    PRIVATE gender
    firstChild
    nextSibling

    CONSTRUCTOR Node(name, ParentName, gender)
        this.name = name
        this.gender = gender
        this.Parent = ParentName
        this.firstChild = null
        this.nextSibling = null
    ENDCONSTRUCTOR

    FUNCTION getParent()
```

```
    RETURN Parent
ENDFUNCTION

FUNCTION setParent(parent)
    Parent = parent
ENDFUNCTION

FUNCTION getName()
    RETURN name
ENDFUNCTION

FUNCTION setName(name)
    this.name = name
ENDFUNCTION

FUNCTION getGender()
    RETURN gender
ENDFUNCTION

FUNCTION setGender(gender)
    this.gender = gender
ENDFUNCTION

FUNCTION getFirstChild()
    RETURN firstChild
ENDFUNCTION

FUNCTION setFirstChild(firstChild)
    this.firstChild = firstChild
```

```
    ENDFUNCTION

    FUNCTION getNextSibling()
        RETURN nextSibling
    ENDFUNCTION

    FUNCTION setNextSibling(nextSibling)
        this.nextSibling = nextSibling
    ENDFUNCTION

    FUNCTION toString()
        RETURN "Node{" + "name='" + name + "', Parent='" + Parent + "', gender=" + gender + '}'
    ENDFUNCTION
ENDCLASS
```

```
CLASS LCRSTree
Node root
Static S
Static MAX
Static numOfEdges
```

This is a constructor for a class named LCRSTree. It takes in three parameters: name, parentName, and gender. The constructor creates a new Node object with the given name, parentName, and gender parameters and assigns it to the root instance variable of the LCRSTree object being constructed.

```
CONSTRUCTOR LCRSTree(name, parentName, gender)
    root = NEW Node(name, parentName, gender)
ENDCONSTRUCTOR
```

This function takes in three parameters: parentName, name, and gender. It inserts a new node with the given name and gender as a child of the node with the given parentName in the tree rooted at the global variable root. If the node with the given parentName is not found, it prints an error message and returns. Otherwise, it checks if the parent node has any children. If it does not have any children, it creates a new Node object and assigns it to the firstChild instance variable of the parent node. If the parent node already has children, it iterates through its children to find the last one and creates a new Node object as its next sibling. The function does not return any value.

```
FUNCTION insert(parentName, name, gender)
  parent = search(parentName)
  IF parent == null
    PRINTLN "Parent node not found"
    RETURN
  ENDIF
  // if the father doesn't have any children yet
  IF parent.firstChild == null
    parent.firstChild = NEW Node(name, parentName, gender)
  ELSE
    sibling = parent.firstChild
    // reaching the last sibling added and insert a brother or a sister
    WHILE sibling.nextSibling != null
      sibling = sibling.nextSibling
    ENDWHILE
    sibling.nextSibling = NEW Node(name, parentName, gender)
  ENDIF
ENDFUNCTION
```

This function takes in a name parameter and deletes the node with the given name from the tree rooted at the global variable root. If the root is null, it prints an error message and returns. If the root has the given name, it sets root to null and returns. Otherwise, it searches for a node with the given name and its parent node. If the parent node is not found, it prints an error message and returns. The function then checks if the first child of the parent node has the given name. If it does, it sets the firstChild instance variable of the parent node to the next sibling of the first child and returns.

Otherwise, it iterates through the children of the parent node to find a child with the given name. If such a child is found, it sets its previous sibling's nextSibling instance variable to its next sibling and returns. If no such child is found, it prints an error message. The function does not return any value.

```
FUNCTION delete(name)
   IF root == null
      PRINTLN "Tree is empty"
      RETURN
   ENDIF
   IF root.getName() == name
      root = null
      RETURN
   ENDIF
   // these two statements to bring the father of a node according to its name
   theName = search(name)
   parent = search(theName.getParent())

   IF parent == null
      PRINTLN "Node not found"
      RETURN
   ENDIF
   nodeToDelete = parent.firstChild
   // if the node to delete is in the first child position
   IF nodeToDelete.getName() == name
      parent.firstChild = nodeToDelete.nextSibling
      RETURN
   ENDIF
   WHILE nodeToDelete.nextSibling != null
      IF nodeToDelete.nextSibling.getName() == name
         nodeToDelete.nextSibling = nodeToDelete.nextSibling.nextSibling
```

```
        RETURN
      ENDIF
      nodeToDelete = nodeToDelete.nextSibling
    ENDWHILE
    PRINTLN "Node not found"
ENDFUNCTION
```

This function takes in a Name parameter and returns the node with the given Name in the tree rooted at the global variable root. If the root is null, it prints an error message and returns null. Otherwise, it calls the searchHelper function on root and Name and returns its result. The searchHelper function is not shown in this code snippet.

```
FUNCTION search(Name)
  IF root == null
    PRINTLN "Tree is empty"
    RETURN null
  ENDIF
  RETURN searchHelper(root, Name)
ENDFUNCTION
```

This function takes a node and a Name as input and returns a Node. It checks if the given node is null, if it is then it returns null. If the name of the given node is equal to the given Name, it returns the node. Otherwise, it calls itself recursively on the first child of the given node and checks if the returned value is not null. If it is not null, it returns the child. Finally, it calls itself recursively on the next sibling of the given node and returns its value. This function performs a depth-first search on a tree data structure to find a node with a given name.

```
  FUNCTION searchHelper(node: Node, Name: String) : Node
  IF node is null THEN
    RETURN null
  END IF
  IF node's name is equal to Name THEN
    RETURN node
```

```
    END IF
    child = searchHelper(node's first child, Name)
    IF child is not null THEN
        RETURN child
    END IF
    RETURN searchHelper(node's next sibling, Name)
END FUNCTION
```

This function takes in four parameters: fatherName, name, nameOfTheBroOrTheSis, and gender. It searches for a node with the given name and its parent node. If the parent node's name matches the given fatherName, the function then iterates through the children of the parent node to find a child with the given nameOfTheBroOrTheSis and gender. If such a child is found, it is returned. Otherwise, the function returns null.

```
FUNCTION getBrotherOrSister(fatherName, name, nameOfTheBroOrTheSis, gender)
    son = search(name)
    father = search(son.getParent())
    IF fatherName == father.getName()
        sibling = father.getFirstChild()
        WHILE sibling != null
            IF sibling.getName() == nameOfTheBroOrTheSis AND sibling.getGender() == gender
                RETURN sibling
            ENDIF
            sibling = sibling.getNextSibling()
        ENDWHILE
    ENDIF
    RETURN null
ENDFUNCTION
```

This function takes in a root node as a parameter and returns the height of the tree rooted at that node. If the root is null, the function returns 0. Otherwise, it iterates through the children of the root

node and recursively calls itself on each child to find their heights. The maximum height among all children is stored in maxChildHeight. The function then returns maxChildHeight + 1, which is the height of the tree rooted at the given root node.

```
FUNCTION getHeight(root)
   IF root == null
      RETURN 0
   ENDIF
   maxChildHeight = -1
   child = root.getFirstChild()
   WHILE child != null
      childHeight = getHeight(child)
      IF childHeight > maxChildHeight
         maxChildHeight = childHeight
      ENDIF
      child = child.getNextSibling()
   ENDWHILE
   RETURN maxChildHeight + 1
ENDFUNCTION
```

This function prints the direct children of the root node. If the root is null, it prints a message indicating that there are no children. Otherwise, it prints the first child of the root node and then iterates through its siblings, printing each one. The function does not return any value.

```
FUNCTION directChildrenOfTheUniqueGrandFather()
   IF root == null
      PRINT "there are no childrens"
      RETURN
   ENDIF
   firstChild = root.getFirstChild()
   PRINT firstChild.toString() + " "
   sibling = firstChild.getNextSibling()
```

```
  WHILE sibling != null
    PRINT sibling.toString()
    sibling = sibling.getNextSibling()
  ENDWHILE
ENDFUNCTION
```

This function returns the height of the tree rooted at the root node by calling the getHeight function on root. The height of the tree is equal to the level of the tree starting from zero. The function does not take any parameters and returns an integer value.

```
FUNCTION getTheLevelOfTheTreeStartingFromZero()
  RETURN getHeight(root)
ENDFUNCTION
```

This function takes in four parameters: fatherName, name, nameOfTheBroOrTheSis, and gender. It searches for a node with the given name and its parent node. If the parent node's name matches the given fatherName, the function then iterates through the children of the parent node to find a child with the given nameOfTheBroOrTheSis and gender. If such a child is found, it is returned. Otherwise, the function returns null.

```
FUNCTION getBrotherOrSister(fatherName, name, nameOfTheBroOrTheSis, gender)
  son = search(name)
  father = search(son.getParent())
  IF fatherName == father.getName()
    sibling = father.getFirstChild()
    WHILE sibling != null
      IF sibling.getName() == nameOfTheBroOrTheSis AND sibling.getGender() == gender
        RETURN sibling
      ENDIF
      sibling = sibling.getNextSibling()
    ENDWHILE
  ENDIF
  RETURN null
```

```
ENDFUNCTION
```

This function takes in four parameters: Father, name, nameOfTheAuntOrTheUncle, and gender. It searches for a node with the given name and its parent node. Then it searches for the parent of the parent node. The function then iterates through the children of the grandparent node to find a child with the given nameOfTheAuntOrTheUncle and gender. If such a child is found, it is returned. Otherwise, the function returns null.

```
FUNCTION getAuntOrUncleOfSomeone(Father, name, nameOfTheAuntOrTheUncle, gender)
   son = search(name)
   father = search(son.getParent())
   parentOfTheFather = search(father.getParent())
   AuntOrUncle = parentOfTheFather.getFirstChild()
   WHILE AuntOrUncle != null
      IF AuntOrUncle.getName() == nameOfTheAuntOrTheUncle AND AuntOrUncle.getGender() ==
gender
         RETURN AuntOrUncle
      ENDIF
      AuntOrUncle = AuntOrUncle.getNextSibling()
   ENDWHILE
   RETURN null
ENDFUNCTION
```

This function takes in a name parameter and returns the degree of the node with the given name. It searches for a node with the given name and then iterates through its children, counting them. The function returns the count of children, which is the degree of the node.

```
FUNCTION getDegreeOfANodeByName(name)
   theNode = search(name)
   childes = theNode.getFirstChild()
   count = 0
   WHILE childes != null
      count = count + 1
```

```
    childes = childes.getNextSibling()
  ENDWHILE
  RETURN count
ENDFUNCTION
```

This function takes a Node object as an input and returns an integer value representing the degree of the node. The degree of a node is defined as the number of its children. The function initializes a childes variable to the first child of the root node and a count variable to 0. It then enters a while loop that iterates until childes is null. In each iteration, the count variable is incremented by 1 and the childes variable is updated to the next sibling of the current childes. Finally, the function returns the value of count, which represents the degree of the root node.

```
FUNCTION getDegreeOfANode(root)
int childes = root.getFirstChild()
  count = 0
  WHILE childes != null
    count = count + 1
    childes = childes.getNextSibling()
ENDWHILE
  return count
ENDFUNCTION
```

This function takes a Node object as an input and returns an integer value representing the maximum degree of the tree rooted at the root node. The degree of a node is defined as the number of its children. The function first checks if the root node is null. If it is, the function returns 0. Otherwise, it makes two recursive calls to itself, one with the first child of the root node and one with the next sibling of the root node. It then updates the value of a global variable MAX to be the maximum of its current value and the degree of the root node, which is obtained by calling the getDegreeOfANode function. Finally, the function returns the value of MAX, which represents the maximum degree of the tree rooted at the root node.

Note that this function relies on a global variable MAX, which should be initialized to a value that is smaller than or equal to the minimum possible degree of any node in the tree before calling this function.

Note that the degree of a tree is the maximum number of childs of a node in this tree

```
FUNCTION getDegreeOfTheTree(root: Node) -> int
```

```
        IF root == null
      return 0
   getDegreeOfTheTree(root.getFirstChild())
   getDegreeOfTheTree(root.getNextSibling())
   MAX = max(MAX, getDegreeOfANode(root))
   return MAX
```

This function takes in six parameters: Father, name, nameOfTheAuntOrTheUncle, gender, nameOfTheSon, and genderOfTheSon. It calls the getAuntOrUncleOfSomeone function to find the aunt or uncle of someone with the given Father, name, nameOfTheAuntOrTheUncle, and gender parameters. Then it iterates through the children of the aunt or uncle to find a child with the given nameOfTheSon and genderOfTheSon. If such a child is found, it is returned. Otherwise, the function returns null.

```
FUNCTION getAuntOrUncleSonOfSomeone(Father, name, nameOfTheAuntOrTheUncle, gender, nameOfTheSon, genderOfTheSon)
   TheAuntOrTheUncleChosen = getAuntOrUncleOfSomeone(Father, name, nameOfTheAuntOrTheUncle, gender)
   TheAuntOrTheUncleChosenChild = TheAuntOrTheUncleChosen.getFirstChild()
   WHILE TheAuntOrTheUncleChosenChild != null
      IF TheAuntOrTheUncleChosenChild.getName() == nameOfTheSon AND TheAuntOrTheUncleChosenChild.getGender() == genderOfTheSon
         RETURN TheAuntOrTheUncleChosenChild
      ENDIF
      TheAuntOrTheUncleChosenChild = TheAuntOrTheUncleChosenChild.getNextSibling()
   ENDWHILE
   RETURN null
ENDFUNCTION
```

This function takes in two parameters: root and name. It searches for a node with the given name and its parent node. If the parent node is null, it appends a string to the global variable s and returns. Otherwise, it recursively calls itself on the root and the name of the parent node. After the recursive

call returns, it appends the name of the parent node to the global variable s and increments the global variable numOfEdges. The function does not return any value.

```
FUNCTION allTheWayToAshraf(root, name)
   getFather = search(name)
   parent = search(getFather.getParent())
   IF parent == null
      s = s + "Ashraf's path : "
      RETURN
   ENDIF
   allTheWayToAshraf(root, parent.getName())
   s = s + parent.getName() + " "
   numOfEdges = numOfEdges + 1
ENDFUNCTION
```

This function returns the depth of a node with the name “Ashraf” in the tree rooted at the global variable root. If the global variable numOfEdges is 0, it calls the allTheWayToAshraf function on root and “Ashraf”. The function then returns the value of the global variable numOfEdges, which is the depth of the node with the name “Ashraf”.

```
FUNCTION depthOfAshraf()
   IF numOfEdges == 0
      allTheWayToAshraf(root, "Ashraf")
   ENDIF
   RETURN numOfEdges
ENDFUNCTION
```

This function takes in a r parameter and returns the level of a node with the name “Ashraf” in the tree rooted at the global variable root. If the global variable numOfEdges is 0, it calls the allTheWayToAshraf function on root and “Ashraf”. The function then returns the value of the global variable numOfEdges, which is the level of the node with the name “Ashraf”.

```
FUNCTION getTheLevelOfAshraf(r)
   IF numOfEdges == 0
```

```
    allTheWayToAshraf(root, "Ashraf")
  ENDIF
  RETURN numOfEdges
ENDFUNCTION
```

This function returns the level of the next mutation. It calls the getTheLevelOfAshraf function to get the level of a node with the name "Ashraf" in the tree rooted at the global variable root. It then adds 3 * 4 + 3 to this value and returns the result. The function does not take any parameters and returns an integer value.

```
FUNCTION getTheLevelOfTheNextMutation()
  result = getTheLevelOfAshraf() + (3 * 4) + 3
  RETURN result
ENDFUNCTION
```

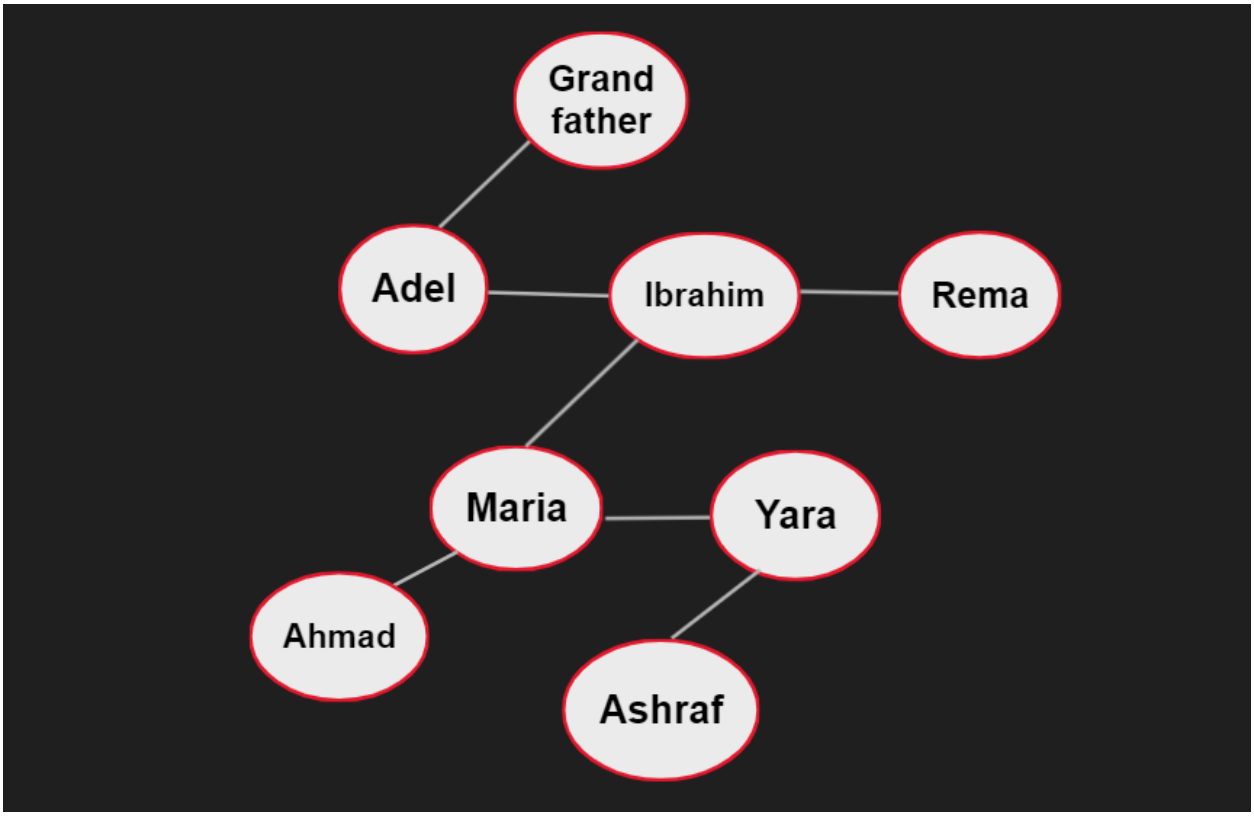
This function takes in two parameters: node and level. It prints the tree rooted at the given node in a pre-order traversal. The level parameter indicates the level of indentation for printing the current node. If the node is null, the function returns. Otherwise, it prints the name of the node with the appropriate indentation. Then it recursively calls itself on the first child of the node and its siblings, incrementing the level parameter by 1. The function does not return any value.

```
FUNCTION printTree(node, level)
  IF node == null
    RETURN
  ENDIF

  // Print the current node
  FOR i = 0 TO level - 1
    PRINT "  "
  ENDFOR
  PRINTLN node.getName()

  // Recursively print the left child and its siblings
```

```
    IF node.getFirstChild() != null
        printTree(node.getFirstChild(), level + 1)
        sibling = node.getFirstChild().getNextSibling()
        WHILE sibling != null
            printTree(sibling, level + 1)
            sibling = sibling.getNextSibling()
        ENDWHILE
    ENDIF
ENDFUNCTION
```

Main Function :

```
LCRSTree T = new LCRSTree("Grandfather" , null,'M');
T.insert("Grandfather" , "Adel" , 'M');
T.insert("Grandfather" , "Ibrahim" , 'M');
T.insert("Grandfather" , "Rema" , 'F');
T.insert("Ibrahim" , "Maria" , 'F');
T.insert("Maria" , "Ahmad" , 'M');
T.insert("Ibrahim" , "Yara" , 'F');
T.insert("Yara" , "Ashraf" , 'M');
```




```
     T.printTree(T.root , 0);
////////////////////////////
Output :
Grandfather
  Adel
  Ibrahim
    Maria
      Ahmad
    Yara
      Ashraf
  Rema
```

```
T.delete("Maria");
T.printTree(T.root , 0);
////////////////////////
```

```
Output:
Grandfather
  Adel
  Ibrahim
    Yara
      Ashraf
  Rema
```

```
System.out.println(T.search("Yara
"));
//////////////////////
Output:
Node{name='Yara',
Parent='Ibrahim', gender=F}
```

```
System.out.println(T.getHeight(T.
root));
//////////////
Output :
3
```

```
T.directChildrenOfTheUniqueGrandF
ather();
//////////////////////
Output :
Node{name='Adel',
Parent='Grandfather', gender=M}
Node{name='Ibrahim',
Parent='Grandfather', gender=M}
Node{name='Rema',
Parent='Grandfather', gender=F}
```

```
System.out.println(T.getTheLevelO
fTheTreeStartingFromZero());
/////////////////
Output :
3
```

```
Node holder =
T.getBrotherOrSister("Grandfather
" , "Ibrahim" , "Rema" , 'F');
System.out.println(holder.toStrin
g());
//////////////////
Output :
```

```
Node{name='Rema',
Parent='Grandfather', gender=F}
```

```
Node holder =
T.getAuntOrUncleOfSomeone("Grandf
ather" , "Ashraf" , "Maria" ,
'F');
System.out.println(holder.toStrin
g());
///////////////
Output :
Node{name='Maria',
Parent='Ibrahim', gender=F}
```

```
System.out.println(T.getDegreeOfA
NodeByName("Grandfather"));
////////////
Output :
3
```

```
System.out.println(T.getDegreeOfT
heTree(T.root));
/////////
Output :
```

```
3
```

```
Node holder = T.getAuntOrUncleSonOfSomeone("Yara" , "Ashraf" , "Maria" , 'F'
, "Ahmad" , 'M');
System.out.println(holder.toString());
//////////////
Output :
Node{name='Ahmad', Parent='Maria', gender=M}
```

```
T.allTheWayToAshraf(T.root ,
"Ashraf");
System.out.println(T.s);
//////////////////
Output:
Ashraf's path : Grandfather
Ibrahim    Yara
```

```
System.out.println(T.depthOfAsh
raf());
//////////////
Output:
3
```

```
System.out.println(T.getTheLeve
lOfAshraf());
///////////////
```

```
Output:
3
```

```
System.out.println(T.getTheLeve
lOfTheNextMutation());
////////////
Output:
18
```