ESO207A - Assignment 2.1

Group members:
Arnav Gupta(200186)
Kushagra Sharma(200539)

Q - Implementing Union-Find-Split ADT

We are given two trees T1 and T2, which are 2-3 trees representing the sets S1 and S2 (these sets contain natural numbers). Additionally, the minimum element in S2 is less than the maximum element of S1. Now, we have to perform a merge operation on these sets and return a tree T which is the representation of the set S1 ∪ S2.

Since all the elements of one set will lie to the left(or right) of the other set {since max(T1)<min(T2)}, we observe that the fastest way would be to just add the whole tree T1(or T2) to the other tree at the appropriate height (since in the final tree, all elements must be at the same depth).

For this, we first calculate the heights of both the trees, by recursively going down the tree from node to the (leftmost) leaf. This is because we have defined our node in such a way that the left field of the node will be filled before middle and right, so if the left field is empty, we can guarantee that a leaf node has been reached)

Let h1 be the height of T1 and h2 be the height of T2.

CASE 1: If h1=h2 - Both trees are at the same height.

We can join them in O(1) time by making a new root node and adding T1 and T2 as its children(T1 as the left child and T2 as the right child). The new resultant tree will be a 2-3 tree representing the set S1 ∪ S2.

CASE 2: If h1>h2 - T1 has a greater height than T2.

We traverse down, along the rightmost chain of nodes of, T1 to a height of (h1-h2-1) and add T2 as its rightmost child. This works because the leaf nodes of T2 will have the same height as that of T1 and will lie in the correct order since they are greater than all the values in T1.
This operation is O(h1) , since in the worst case all the rightmost nodes of T1 already have 3 children and we would have to recursively go all the way up to the root node, while splitting the nodes into 2 nodes with 2 children each.

CASE 3: If h1<h2 - T1 has a lower height than T2.

This is analogous to the previous case. We will now go down, along the leftmost chain of nodes of, T2 to a height of (h2-h1-1) and add T1 as its leftmost child. All the leaf nodes of T1 will be at the same height as those of T2 and in the correct order since they are less than all the values in T2.
This operation is O(h2), since we might have to recursively go all the way up to the root node of T2 while splitting the nodes.

How inserting a Node into the tree proceeds:
Let n be the node which is being inserted as a child on node p.
If node p has two children : a and b. We can directly insert n1 as its child(appropriately, depending on the values stored in n). Which means after insertion p will have three children : (a,b, n).
However if p already has three children :(a,b,c), then adding n will violate the property of a 2-3 tree. So we split p, and create a new neighbour of p, say p2. We transfer a child of p to p2 and add n as a child of p2. Thus both p and p2 have 2 children each, which retains the 2-3 tree property.
Recursively, we now add p2 to the parent of p. If this chain goes all the way up to the root node, we create a new root node and add the two subtrees as its children.

MERGE ALGORITHM
Data: Trees U and T
Returns: Tree V after merging U and T.

```
// Tree U has lower values than T
// h1 is height of U and h2 is height of T.
// Takes O(h1+h2) time.
h1= getHeight(U)
h2= getHeight(T)
if(h1==h2) then
        V = newNode //Make a new node
        V->left = U
        V->right =T // Add U and T as its children
else if(h1>h2) then
        node* ptr= U;
        for(i=0,i<h1-h2-1;i++) ptr=ptr->right //Go down rightmost chain
        Insert(ptr,T) //Add T as its child
else if(h1<h2) then
        node* ptr= T;
        for(i=0;i<h2-h1-1;i++) ptr=ptr->left //Go down leftmost chain
        insert(U,ptr) // Add U as its child
// If insert returns two nodes(if recursion goes all the way to the root)
node* V = makeRoot(n1,n2) //n1 and n2 returned from insert.
// else if only one node is returned from the insert function
node*V = n1 //Node returned from insert function
return V
```

INSERT ALGORITHM

Data: node pointers U and T (and integers containing minimum values so that time complexity is better)

Returns: 2 node pointers and minimum value in second node. (one node could be null if recursion stops before reaching the root)

```
// Takes O(height of tree) time.
// twoNode and threeNode mean that the node has 2 or 3 children respectively
// Adding T to U
if U==twoNode(a,b,m(b)) then //m(b)=min value in b
        U=threeNode(a,b,T,m(b),m(T))
        Return {U,NULL,0}
else if U==threeNode(a,b,c,m(b),m(c)) then
        node* newNode = twoNode(c,T,m(T))
        U=twoNode(a,b,m(b))
        Insert(U->parent,newNode) //Recursively adding nodes to the parents
        if(U->parent==NULL)Return {U,newNode,m(c)} // Returning two nodes if U is a root
        node

// Adding U to T
if T==twoNode(a,b,m(b)) then //m(b)=min value in b
        T=threeNode(U,a,b,m(a),m(b))
        Return {T,NULL,0}
else if T==threeNode(a,b,c,m(b),m(c)) then
        node* newNode = twoNode(b,c,m(c))
        T=twoNode(U,a,m(a))
        Insert(newNode,T->parent) //Recursively adding nodes to the parents
        if(T->parent==NULL)Return {U,newNode,m(b)} // Returning two nodes if T is a root
        node
```

GET-HEIGHT ALGORITHM

Data: Tree T

Returns: Height of tree T

```
height=0;
node*ptr=T;
while(ptr->left!=NULL) do //Exits when a leaf node is reached.
        height++
        ptr=ptr->left
Return height
```

EXTRACT ALGORITHM

Data: Tree T

Output: Set represented by T,i.e. Values stored in its leaf nodes.

```
// Breadth First Search (BFS) implemented using queues
// q.push() : Adds an element to the end of the queue
// q.pop() : Removes the first element in the queue
// q.front() : Gives the first element in the queue
```

```
Queue q
q.push(T)
while(!q.empty()) do
node* ptr= q.front()
q.pop()
if(ptr->left!=NULL)q.push(ptr->left) //Adding left child to the queue
if(ptr->middle!=NULL)q.push(ptr->middle)
if(ptr->right!=NULL)q.push(ptr->right)
if(ptr->left==NULL)
Print(ptr->data)  //Printing the value if it is a leaf node. (A leaf node does not have children)
```

Time Complexity Analysis

In the subsequent lines, we have taken h1 as the height of the first tree, and h2 as the height of second tree.

Firstly, calculating the height and the minimum value in a tree takes O(logN) time, where N is the number of leaf nodes in the tree. This is because the structure of the 2-3 tree is such that the height of the 2-3 tree is O(logN). This had also been proved in the lectures. These steps are done in time (a*h1 + b*h2) (a,b are constants). This is the same as O(h1+h2) time.

If the heights of the trees are equal then we just create a new root have the two of its subtrees as T1 and T2 in O(1) and return. It is easy to see that the total time complexity is O(h(T1)+h(T2)).

Otherwise, we consider the case h1>h2. In this case, we traverse down T1 up to (h1-h2-1) nodes, as outlined in the algorithm. This takes O(h1) time since in the worst case we traverse all the way down in T1. Now, we need to insert the smaller tree at this place. If the property of a node having 2 or 3 children is broken here, we simply split the 4 nodes into groups of 2 and travel towards the root to maintain this property again. It can be seen that the creation of any node takes O(1) time. The worst case here is when all the nodes upto the root have 3 children, so we have to split the nodes and travel all the way up to the root. This, again, has O(h1) time. The total time complexity is therefore given by

(a*h1 + b*h2) + c*h1 + d*(e + f*h1) = O(h1+h2)

c*h1 is the time for going down T1 to find the correct place for insertion of T2.
d*(e + f*h1) is the time for inserting T2, and if required do splits on the nodes and travel towards the root.

The other case is analogous and symmetric to the previous case and therefore has the same time complexity. Hence we conclude that the time complexity of our algorithm is **O(h1+h2).**

It should be noted that this is also the worst-case complexity of the algorithm. It is possible because of the fact that 2-3 trees are balanced i.e. each leaf is at the same depth.