Day 9 25/11/2024

BST:

```
#include<bits/stdc++.h>
using namespace std;
struct Node{
  int data;
  Node* left;
  Node* right;
};
Node* createNode(int data){
  Node* newNode = new Node();
  newNode->data = data;
  newNode->left = newNode->right = nullptr;
  return newNode;
}
Node* insertNode(Node* root, int data){
  if(root == nullptr){
	return createNode(data);
  }
  else if(data<root->data){
	root->left = insertNode(root->left,data);
  }
  else{
	root->right = insertNode(root->right,data);
  }
  return root;
}
void inOrder(Node* root,vector<int>& arr){
  if(root!=nullptr){
	inOrder(root->left,arr);
	arr.push_back(root->data);
	inOrder(root->right,arr);
  }
}
void preOrder(Node* root,vector<int>& arr){
  if(root!=nullptr){
	arr.push_back(root->data);
	preOrder(root->left,arr);
	preOrder(root->right,arr);
  }
}
void postOrder(Node* root,vector<int>& arr){
  if(root!=nullptr){
          postOrder(root->left,arr);
          postOrder(root->right,arr);
          arr.push_back(root->data);
  }
}
Node* searchNode(Node* root, int key){
  if(root == nullptr || root->data == key){
          return root;
  }
  else if(root->data < key){
          return searchNode(root->right,key);
  }
  else{
          return searchNode(root->left,key);
  }
}
Node* minNode(Node* node){
  Node* currNode = node;
  while(currNode && currNode->left != nullptr){
          currNode = currNode->left;
  }
  return currNode;
}
Node* maxNode(Node* node){
  Node* currNode = node;
  while(currNode && currNode->right != nullptr){
          currNode = currNode->right;
  }
  return currNode;
}
Node* deleteNode(Node* root,int data){
  if(root == nullptr){
          return root;
  }
  if(data < root->data){
          root->left = deleteNode(root->left,data);
  }
  else if(data > root->data){
          root->right = deleteNode(root->right,data);
  }
  else{
          if(root->left == nullptr){
          Node* temp = root->right;
```

```
            delete root;
            return temp;
            }
            else if(root->right == nullptr){
            Node* temp = root->left;
            delete root;
            return temp;
            }
            Node* temp = minNode(root->right);
            root->data = temp->data;
            root->right = deleteNode(root->right,temp->data);
  }
  return root;
}
int main(){
  Node* root = nullptr;
  vector<int> tree = {50,60,10,80,20};
  for(int i:tree){
            root = insertNode(root,i);
  }
  root = deleteNode(root,80);
  cout<<"InOrder:\n";
  vector<int> in;
  inOrder(root,in);
  for(int i:in){
            cout<<i<<" ";
  }
  cout<<"\n";
  cout<<"PreOrder:\n";
  vector<int> pre;
  preOrder(root,pre);
  for(int i:pre){
            cout<<i<<" ";
  }
  cout<<"\n";
  cout<<"PostOrder:\n";
  vector<int> post;
  postOrder(root,post);
  for(int i:post){
            cout<<i<<" ";
  }
  return 0;
}
```

Validate BST:

```
#include<bits/stdc++.h>
using namespace std;
struct Node{
  int data;
  Node* left;
  Node* right;
};
Node* createNode(int data){
  Node* newNode = new Node();
  newNode->data = data;
  newNode->left = newNode->right = nullptr;
  return newNode;
}
Node* insertNode(Node* root, int data){
  if(root == nullptr){
          return createNode(data);
  }
  if(root->left == nullptr){
          root->left = createNode(data);
  }
  else{
          root->right = createNode(data);
  }
  return root;
}
int minValue(Node* node){
  if(node == nullptr){
          return INT_MAX;
  }
  return min({node->data,minValue(node->left),minValue(node->right)});
}
int maxValue(Node* node){
  if(node == nullptr){
          return INT_MIN;
  }
  return max({node->data,maxValue(node->left),maxValue(node->right)});
}
bool isBst(Node* node){
  if(node == nullptr){
          return true;
  }
  else if(node->left != nullptr && maxValue(node->left)>=node->data){
            return false;
  }
  else if(node->right != nullptr && minValue(node->right)<=node->data){
            return false;
  }
  return isBst(node->left) && isBst(node->right);
}
int main(){
  Node* root = nullptr;
  vector<int> tree = {50,10,60};
  for(int i:tree){
            root = insertNode(root,i);
  }
  bool isTree = isBst(root);
  if(isTree){
            cout<<"BST boi";
  }
  else{
            cout<<"No BST boi";
  }
  return 0;
}
```

Binary tree to BST:

```
class Solution{
  public:
    Node *binaryTreeToBST (Node *root)
    {
        vector<int> inOrderArray;
        inOrder(root,inOrderArray);
        vector<int> sortedArray(inOrderArray.begin(), inOrderArray.end());
        sort(sortedArray.begin(),sortedArray.end());
        int idx = 0;
        changeTree(root,idx,sortedArray);
        return root;
    }
    void inOrder(Node *node, vector<int>& inOrderArray){
        if(node==nullptr) return;
        inOrder(node->left,inOrderArray);
        inOrderArray.push_back(node->data);
        inOrder(node->right,inOrderArray);
    }
    void changeTree(Node *node,int &idx, vector<int>& sortedArray){
        if(node==nullptr) return;
        changeTree(node->left,idx,sortedArray);
        node->data = sortedArray[idx++];
        changeTree(node->right,idx,sortedArray);
    }
};
```

Bottom View:

```
class Solution {
  public:
    vector <int> bottomView(Node *root) {
        vector<int>ans;
        map<int,int> mp;
        if(root == nullptr) return ans;
        queue<pair<Node*,int>> q;
        q.push({root, 0});
        while(!q.empty()){
            auto it = q.front();
            q.pop();
            auto node = it.first;
            int line = it.second;
            mp[line] = node->data;
            if(node->left != nullptr) q.push({node->left,line-1});
            if(node->right!=nullptr) q.push({node->right,line+1});
        }
        for(auto it : mp){
            ans.push_back(it.second);
        }
        return ans;
    }
};
```

Top View:

```
class Solution {
  public:
    vector<int> topView(Node *root) {
        vector<int>ans;
        map<int,int> mp;
        if(root == nullptr) return ans;
        queue<pair<Node*,int>> q;
        q.push({root, 0});
        while(!q.empty()){
            auto it = q.front();
            q.pop();
            auto node = it.first;
            int line = it.second;
            if(mp.find(line) == mp.end()) mp[line] = node->data;
            if(node->left != nullptr) q.push({node->left,line-1});
            if(node->right!=nullptr) q.push({node->right,line+1});
        }
        for(auto it : mp){
            ans.push_back(it.second);
        }
        return ans;
    }
};
```

Right View:

```
class Solution {
public:
    vector<int> rightSideView(TreeNode* root) {
        vector<int> arr;
        int level = 0;
        reversePreOrder(root, level,arr);
        return arr;
    }
    void reversePreOrder(TreeNode* node, int level, vector<int>& arr){
        if(node==nullptr) return;
        if(level == arr.size()) arr.push_back(node->val);
        reversePreOrder(node->right,level+1,arr);
        reversePreOrder(node->left,level+1,arr);
    }
};
```

Left View:

```
class Solution {
public:
    vector<int> LeftSideView(TreeNode* root) {
        vector<int> arr;
        int level = 0;
        reversePreOrder(root, level,arr);
        return arr;
    }
    void reversePreOrder(TreeNode* node, int level, vector<int>& arr){
        if(node==nullptr) return;
        if(level == arr.size()) arr.push_back(node->val);
        reversePreOrder(node->left,level+1,arr);
        reversePreOrder(node->right,level+1,arr);
    }
};
```