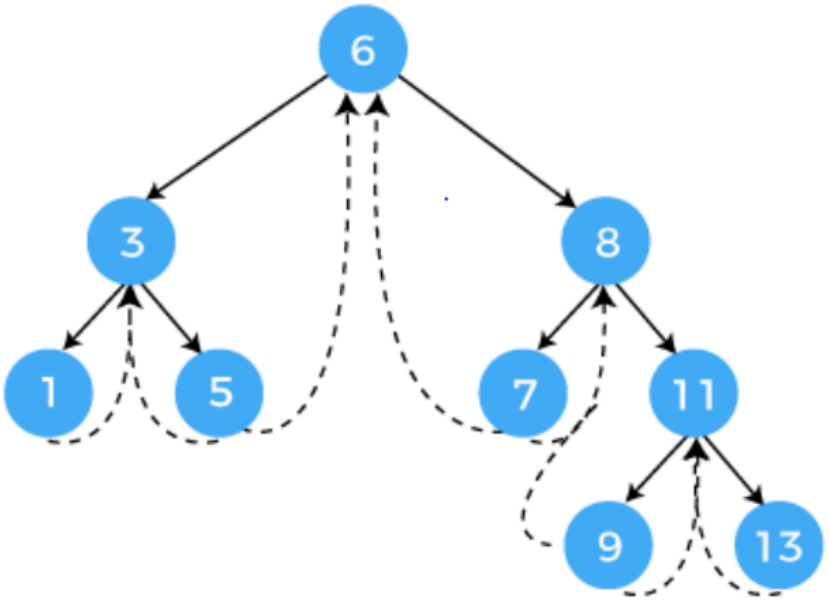
Data Structures and Algorithms Lab(MCSE501P)

Name: Mahesh Jagtap
Reg No.24MCS1017
Date 02/10/2024

1. Implement Threaded binary tree structure for the below tree



Double Threaded Binary Tree

```
#include <iostream>
#include <cstdlib>
#define MAX_VALUE 200
using namespace std;

class Node
{
        public:
    int key;
    Node *left, *right;
    bool leftThread, rightThread;
};


class ThreadedBinarySearchTree
{
        private:
    Node *root;
  public:
    /* Constructor */
    ThreadedBinarySearchTree()
    {
```

```
    root = new Node();
    root->right = root->left = root;
    root->leftThread = true;
    root->key = MAX_VALUE;
}


void makeEmpty()
{
    root = new Node();
    root->right = root->left = root;
    root->leftThread = true;
    root->key = MAX_VALUE;
}


void insert(int key)
{
    Node *p = root;
    for (;;)
    {
        if (p->key < key)
        {
            if (p->rightThread)
                break;
            p = p->right;
        }
        else if (p->key > key)
        {
            if (p->leftThread)
                break;
            p = p->left;
        }
        else
        {
            /* redundant key */
            return;
        }
    }
    Node *tmp = new Node();
    tmp->key = key;
    tmp->rightThread = tmp->leftThread = true;
    if (p->key < key)
    {
        /* insert to right side */
        tmp->right = p->right;
        tmp->left = p;
        p->right = tmp;
```

```
            p->rightThread = false;
        }
        else
        {
            tmp->right = p;
            tmp->left = p->left;
            p->left = tmp;
            p->leftThread = false;
        }
    }

    bool search(int key)
    {
        Node *tmp = root->left;
        for (;;)
        {
            if (tmp->key < key)
            {
                if (tmp->rightThread)
                    return false;
                tmp = tmp->right;
            }
            else if (tmp->key > key)
            {
                if (tmp->leftThread)
                    return false;
                tmp = tmp->left;
            }
            else
            {
                return true;
            }
        }
    }

    void Delete(int key)
    {
        Node *dest = root->left, *p = root;
        for (;;)
        {
            if (dest->key < key)
            {
                /* not found */
                if (dest->rightThread)
                    return;
                p = dest;
                dest = dest->right;
            }
```

```
    else if (dest->key > key)
    {
        /* not found */
        if (dest->leftThread)
            return;
        p = dest;
        dest = dest->left;
    }
    else
    {
        /* found */
        break;
    }
}
Node *target = dest;
if (!dest->rightThread && !dest->leftThread)
{
    /* dest has two children*/
    p = dest;
    /* find largest node at left child */
    target = dest->left;
    while (!target->rightThread)
    {
        p = target;
        target = target->right;
    }
    /* using replace mode*/
    dest->key = target->key;
}
if (p->key >= target->key)
{
    if (target->rightThread && target->leftThread)
    {
        p->left = target->left;
        p->leftThread = true;
    }
    else if (target->rightThread)
    {
        Node *largest = target->left;
        while (!largest->rightThread)
        {
            largest = largest->right;
        }
        largest->right = p;
        p->left = target->left;
    }
    else
    {
```

```
            Node *smallest = target->right;
            while (!smallest->leftThread)
            {
                smallest = smallest->left;
            }
            smallest->left = target->left;
            p->left = target->right;
        }
    }
    else
    {
        if (target->rightThread && target->leftThread)
        {
            p->right = target->right;
            p->rightThread = true;
        }
        else if (target->rightThread)
        {
            Node *largest = target->left;
            while (!largest->rightThread)
            {
                largest = largest->right;
            }
            largest->right =  target->right;
            p->right = target->left;
        }
        else
        {
            Node *smallest = target->right;
            while (!smallest->leftThread)
            {
                smallest = smallest->left;
            }
            smallest->left = p;
            p->right = target->right;
        }
    }
}

/* Function to print tree */
void printTree()
{
    Node *tmp = root, *p;
    for (;;)
    {
        p = tmp;
        tmp = tmp->right;
        if (!p->rightThread)
```

```
            {
                while (!tmp->leftThread)
                {
                    tmp = tmp->left;
                }
            }
            if (tmp == root)
                break;
            cout<<tmp->key<<"   ";
        }
        cout<<endl;
    }
};



#include <sstream>

int main()
{
    ThreadedBinarySearchTree tbst;
    cout<<"ThreadedBinarySearchTree Test\n";
    char ch;
    int choice, val;
    string input;
    /*  Perform tree operations  */
    do
    {
        cout<<"\nThreadedBinarySearchTree Operations\n";
        cout<<"1. Insert (multiple numbers separated by space)"<<endl;
        cout<<"2. Delete"<<endl;
        cout<<"3. Search"<<endl;
        cout<<"4. Clear"<<endl;
        cout<<"Enter Your Choice: ";
        cin>>choice;
        cin.ignore(); // Clear the input buffer
        switch (choice)
        {
        case 1 :
            cout<<"Enter integers to insert (separated by space): ";
            getline(cin, input); // Get the entire line of input

            // Parse multiple integers from the input
            {
                stringstream ss(input);
                int val;
                while (ss >> val) // Extract integers and insert them in sequence
                {
```

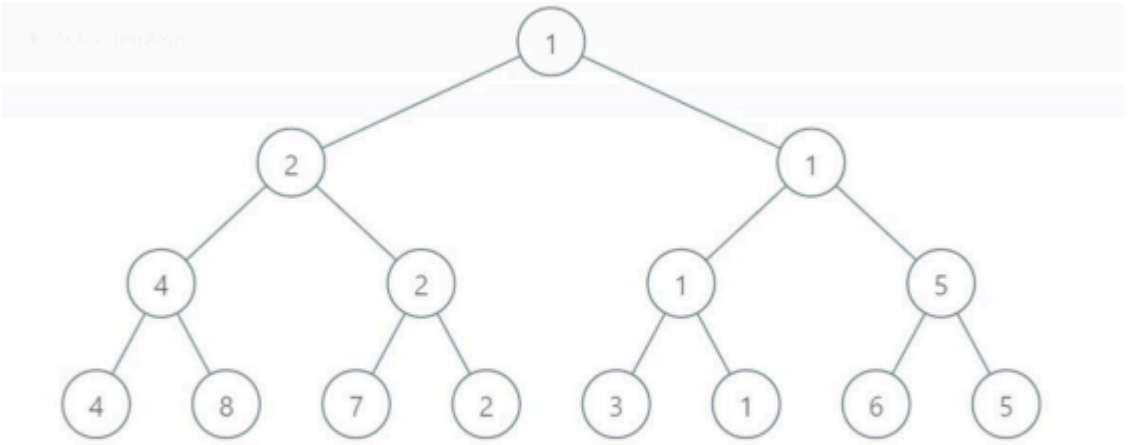
```
                tbst.insert(val);
              }
            }
            break;
        case 2 :
            cout<<"Enter integer element to delete: ";
            cin>>val;
            tbst.Delete(val);
            break;
        case 3 :
            cout<<"Enter integer element to search: ";
            cin>>val;
            if (tbst.search(val) == true)
              cout<<"Element "<<val<<" found in the tree"<<endl;
            else
              cout<<"Element "<<val<<" not found in the tree"<<endl;
            break;
        case 4 :
            cout<<"\nTree Cleared\n";
            tbst.makeEmpty();
            break;
        default :
            cout<<"Wrong Entry \n ";
            break;
        }
        /*  Display tree  */
        cout<<"\nTree = ";
        tbst.printTree();
        cout<<"\nDo you want to continue (Type y or n): ";
        cin>>ch;
    }
    while (ch == 'Y'|| ch == 'y');
    return 0;
}
```

Output:

```
ThreadedBinarySearchTree Test

ThreadedBinarySearchTree Operations
1. Insert (multiple numbers separated by space)
2. Delete
3. Search
4. Clear
Enter Your Choice: 1
Enter integers to insert (separated by space): 13 9 11 8 7 1 5 3 6

Tree = 1   3   5   6   7   8   9   11   13

Do you want to continue (Type y or n): y

ThreadedBinarySearchTree Operations
1. Insert (multiple numbers separated by space)
2. Delete
3. Search
4. Clear
Enter Your Choice: 3
Enter integer element to search: 7
Element 7 found in the tree

Tree = 1   3   5   6   7   8   9   11   13

Do you want to continue (Type y or n): y
```

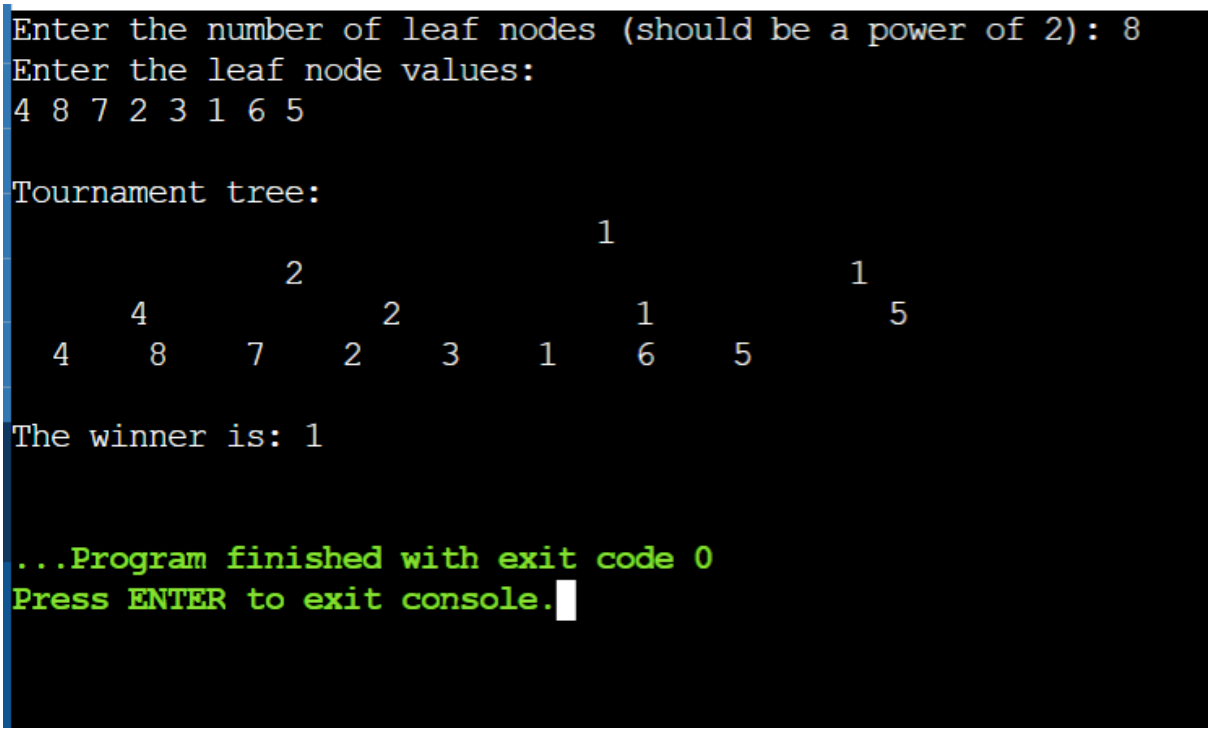
```
ThreadedBinarySearchTree Operations
1. Insert (multiple numbers separated by space)
2. Delete
3. Search
4. Clear
Enter Your Choice: 2
Enter integer element to delete: 6

Tree = 1   3   5   7   8   9   11   13

Do you want to continue (Type y or n): y

ThreadedBinarySearchTree Operations
1. Insert (multiple numbers separated by space)
2. Delete
3. Search
4. Clear
Enter Your Choice: 4

Tree Cleared

Tree =

Do you want to continue (Type y or n): n
```

2. Implement tournament tree for the below tree




```
#include <iostream>
#include <vector>
#include <cmath>
#include <iomanip> // for setw()

using namespace std;
int compete(int a, int b) {
    return min(a, b);
}

void buildTournamentTree(vector<int>& tree, int n) {
    for (int i = n - 1; i > 0; --i) {
        tree[i] = compete(tree[2 * i], tree[2 * i + 1]);
    }
}

void printTree(const vector<int>& tree, int n) {
    int levels = log2(n) + 1;
    int index = 1;
    for (int level = 0; level < levels; ++level) {
        int nodesAtThisLevel = pow(2, level);
        int space = pow(2, levels - level) - 1;

        cout << setw(space * 2) << "";
```

```
        for (int i = 0; i < nodesAtThisLevel && index < 2 * n; ++i, ++index) {
            cout << tree[index];
            if (i < nodesAtThisLevel - 1) {
                cout << setw(space * 4) << ""; // Spacing between nodes
            }
        }
        cout << endl;
    }
}

int main() {
    int n;
    cout << "Enter the number of leaf nodes (should be a power of 2): ";
    cin >> n;

    vector<int> leaves(n);

    cout << "Enter the leaf node values:\n";
    for (int i = 0; i < n; ++i) {
        cin >> leaves[i];
    }

    int totalNodes = 2 * n;
    vector<int> tournamentTree(totalNodes);


    for (int i = 0; i < n; ++i) {
        tournamentTree[n + i] = leaves[i];
    }

    buildTournamentTree(tournamentTree, n);

    cout << "\nTournament tree:\n";
    printTree(tournamentTree, n);

    cout << "\nThe winner is: " << tournamentTree[1] << endl;

    return 0;
}
```


Enter the number of leaf nodes (should be a power of 2): 8
Enter the leaf node values:
4 8 7 2 3 1 6 5
Tournament tree:
1
2 1
4 2 1 5
4 8 7 2 3 1 6 5
The winner is: 1
...Program finished with exit code 0
Press ENTER to exit console.