Experiment Number 10

Name ::	Rishabh Anand	UID ::	19BCS4525
Branch ::	CSE - IoT	Sec/Grp ::	1/A
Semester ::	5th	Date ::	14th Nov, 2021
Subject ::	Adv Programming Lab	CODE ::	CSP-347

1. Aim :

Demonstrate insert ,delete and search in Treap.

2. Task :

1. Insert in Treap
2. Delete in Treap
3. Search in Treap

3. Algorithm :

- Create new node with key equals to x and value equals to a random value then perform standard BST insertion. A newly inserted node gets a random priority, so max heap property may be violated, use rotations to make sure that inserted node's priority follow maxheap property.
- During insertion, we recursively traverse all ancestors of the inserted node if
-
  - new node is inserted in left subtree and root of left subtree has highest priority, perform right rotation.
  - If new node is inserted in right subtree and root of right subtree has higher priority, perform left rotation.
- The delete implementation here is slightly different from pervious one. If node is a leaf delete it, if node has on child NULL and other as non-NULL, replace node with non-empty child.
- If node has both children as non-NULL, find max of left and right children.
  - If priority of right child is greater, perform left rotation at node
  - If priority of left child is greater, perform right rotation at node.

4. Source Code :

```
#include <bits/stdc++.h>


using namespace std;


struct TreapNode
{
    int key;
    int priority;
    TreapNode *left, *right;
};


TreapNode *rightRotate(TreapNode *y)
{
    TreapNode *x = y->left, *T2 = x->right;
    x->right = y;
    y->left = T2;
    return x;
}


TreapNode *leftRotate(TreapNode *x)
{
    TreapNode *y = x->right, *T2 = y->left;
    y->left = x;
    x->right = T2;
    return y;
}


TreapNode *newNode(int key)
{
    TreapNode *temp = new TreapNode;
    temp->key = key;
    temp->priority = rand() % 100;
    temp->left = temp->right = NULL;
    return temp;
}
```

```
TreapNode *search(TreapNode *root, int key)
{
    if (root == NULL || root->key == key)
        return root;
    if (root->key < key)
        return search(root->right, key);
    return search(root->left, key);
}


TreapNode *insert(TreapNode *root, int key)
{
    if (!root)
        return newNode(key);
    if (key <= root->key)
    {
        root->left = insert(root->left, key);
        if (root->left->priority > root->priority)
            root = rightRotate(root);
    }
    else
    {
        root->right = insert(root->right, key);
        if (root->right->priority > root->priority)
            root = leftRotate(root);
    }
    return root;
}


TreapNode *deleteNode(TreapNode *root, int key)
{
    if (root == NULL)
        return root;
    if (key < root->key)
        root->left = deleteNode(root->left, key);
    else if (key > root->key)
        root->right = deleteNode(root->right, key);
    else if (root->left == NULL)
    {
        TreapNode *temp = root->right;
        delete (root);
        root = temp;
    }
```

```
    else if (root->right == NULL)
    {
        TreapNode *temp = root->left;
        delete (root);
        root = temp;
    }
    else if (root->left->priority < root->right->priority)
    {
        root = leftRotate(root);
        root->left = deleteNode(root->left, key);
    }
    else
    {
        root = rightRotate(root);
        root->right = deleteNode(root->right, key);
    }
    return root;
}


void inorder(TreapNode *root)
{
    if (root)
    {
        inorder(root->left);
        cout << "key: " << root->key << " | priority: %d "
             << root->priority;
        if (root->left)
            cout << " | left child: " << root->left->key;
        if (root->right)
            cout << " | right child: " << root->right->key;
        cout << endl;
        inorder(root->right);
    }
}
```

```
int main()
{
    srand(time(NULL));
    struct TreapNode *root = NULL;
    root = insert(root, 50);
    root = insert(root, 30);
    root = insert(root, 20);
    root = insert(root, 40);
    root = insert(root, 70);
    root = insert(root, 60);
    root = insert(root, 80);
    cout << "Inorder traversal of the given tree \n";
    inorder(root);
    cout << "\nDelete 20\n";
    root = deleteNode(root, 20);
    cout << "Inorder traversal of the modified tree \n";
    inorder(root);
    cout << "\nDelete 30\n";
    root = deleteNode(root, 30);
    cout << "Inorder traversal of the modified tree \n";
    inorder(root);
    cout << "\nDelete 50\n";
    root = deleteNode(root, 50);
    cout << "Inorder traversal of the modified tree \n";
    inorder(root);
    TreapNode *res = search(root, 50);
    (res == NULL) ? cout << "\n50 Not Found " : cout << "\n50 found";
    return 0;
}
```

5. Observations :

```
❯ g++ code.cpp -o code;./code
Inorder traversal of the given tree
key: 20 | priority: %d 56 | right child: 30
key: 30 | priority: %d 3
key: 40 | priority: %d 79 | left child: 20 | right child: 60
key: 50 | priority: %d 51
key: 60 | priority: %d 60 | left child: 50
key: 70 | priority: %d 87 | left child: 40 | right child: 80
key: 80 | priority: %d 45

Delete 20
Inorder traversal of the modified tree
key: 30 | priority: %d 3
key: 40 | priority: %d 79 | left child: 30 | right child: 60
key: 50 | priority: %d 51
key: 60 | priority: %d 60 | left child: 50
key: 70 | priority: %d 87 | left child: 40 | right child: 80
key: 80 | priority: %d 45

Delete 30
Inorder traversal of the modified tree
key: 40 | priority: %d 79 | right child: 60
key: 50 | priority: %d 51
key: 60 | priority: %d 60 | left child: 50
key: 70 | priority: %d 87 | left child: 40 | right child: 80
key: 80 | priority: %d 45

Delete 50
Inorder traversal of the modified tree
key: 40 | priority: %d 79 | right child: 60
key: 60 | priority: %d 60
key: 70 | priority: %d 87 | left child: 40 | right child: 80
key: 80 | priority: %d 45

50 Not Found
➜ ~/w/S/As/temp on master !4 ?11 ❯
```

Learning Outcomes :

- Learnt about treap data structure.
- Insertion, deletion, searching in treap data structure.
- Application and uses of treap data structure.

S. No.	Parameters	Marks Obtained	Maximum Marks
1.			
2.			
3.			