NETAJI SUBHAS UNIVERSITY OF TECHNOLOGY

DESIGN AND ANALYSIS OF ALGORITHMS

PROGRAME FILE

(CACSC06)

SUBMITTED BY: -

Name: *HARSH KUMAR*

Branch: CSAI

Roll no.:2021UCA1829

CONTENT:

16	**Write a program to Implement Travelling Salesman Problem.**

Programs

1. **Write a program for Merge sort, Quick sort, Bubble sort, insertion sort and selection sort.**

DESCRIPTION:

- Insertion sort is a sorting algorithm that builds a final sorted array (sometimes called a list) one element at a time. While sorting is a simple concept, it is a basic principle used in complex computer programs such as file search, data compression, and path finding. Running time is an important thing to consider when selecting a sorting algorithm since efficiency is often thought of in terms of speed. Insertion sort has an average and worst-case running time of O(n^2) O(n)2), so in most cases, a faster algorithm is more desirable.
- Merge Sort is one of the most popular sorting algorithms that is based on the principle of Divide and Conquer Algorithm. O(n*log n)

CODE:

```
/*
    Harsh Kumar
    (2021UCA1829)
    DAA(Design and Analysis of Algorithms
*/
// bubble sort in C++
#include <bits/stdc++.h>
#define ll long long int
using namespace std;

void bubbleSort(vector<ll> &arr, int n)
{
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n - i - 1; j++)
            if (arr[j] > arr[j + 1])
                swap(arr[j], arr[j + 1]);
    }
}

int main(){
    ll n;
    cout<<"Enter Arry size::";
    cin>>n;
    vector<ll> arr(n,0);
```

```
    cout<<"Enter the unsorted arry: ";
    for(auto &i:arr)
    cin>>i;
    cout<<"Merge sorted arry: ";
    bubbleSort(arr,n);
    for(auto i:arr)
    cout<<i<<" , ";
}
```

```
/*
    Harsh Kumar
    (2021UCA1829)
    DAA(Design and Analysis of Algorithms
*/
// insertion sort in C++
#include <bits/stdc++.h>
#define ll long long int
using namespace std;

void InsertionSort(vector<ll> &arr, int n)
{
    for(int i=1;i<n;i++){
        ll key=arr[i];
        int j=i-1;
        while(arr[j]>key && j>=0)
        arr[j+1]=arr[j--];
        arr[j+1]=key;
    }
}

int main(){
    ll n;
    cout<<"Enter Arry size::";
    cin>>n;
    vector<ll> arr(n,0);
    cout<<"Enter the unsorted arry: ";
    for(auto &i:arr)
    cin>>i;
    cout<<"Insertion sorted arry: ";
    InsertionSort(arr,n);
    for(auto i:arr)
    cout<<i<<" , ";
}
```

```cpp
/*
    Harsh Kumar
    (2021UCA1829)
    DAA(Design and Analysis of Algorithms
*/
// Merge sort in C++
#include<bits/stdc++.h>
#define ll long long int
using namespace std;

void merge(vector<ll> &arr, int f,int m,int l){
    vector<ll> V1,V2;
    for(int i=f;i<=m;i++)
    V1.push_back(arr[i]);
    for(int i=m+1;i<=l;i++)
    V2.push_back(arr[i]);

    int size1=m-f+1,size2=l-m;
    int i=0,j=0,k=f;
    while(i<size1 && j<size2)
    {
        if(V1[i]>V2[j])
        {
            arr[k++]=V2[j++];
        }
        else if(V1[i]<V2[j])
        {
            arr[k++]=V1[i++];
        }
        else
        {
            arr[k++]=V1[i++];
            arr[k++]=V2[j++];
        }
    }
    while (i<size1)
    {
        arr[k++]=V1[i++];
    }
    while (j<size2)
    {
        arr[k++]=V2[j++];
    }
}

void mergeSort(vector<ll> &arr, int f, int l){
    if(f>=l)
    return;
    int mid=(f+(l-f)/2);
    mergeSort(arr,f,mid);
```

```
    mergeSort(arr,mid+1,l);
    merge(arr,f,mid,l);
}

int main(){
    ll n;
    cout<<"Enter Arry size::";
    cin>>n;
    vector<ll> arr(n,0);
    cout<<"Enter the unsorted arry: ";
    for(auto &i:arr)
    cin>>i;
    cout<<"Merge sorted arry: ";
    mergeSort(arr,0,n-1);
    for(auto i:arr)
    cout<<i<<" , ";
}
```

```
/*
    Harsh Kumar
    (2021UCA1829)
    DAA(Design and Analysis of Algorithms
*/
// Quick sort in C++
#include<bits/stdc++.h>
#define ll long long int
using namespace std;

void swap(ll *x, ll *y){
    ll temp=*x;
    *x=*y;
    *y=temp;
}
void quickSort(vector<ll> &arr, int f,int l){
    if(f>=l)
    return;
    int pivot= arr[l];
    int i=f,j=l-1;
    while(i<=j){
        if(arr[i]<pivot)
        i++;
        else if(arr[j]>=pivot)
        j--;
        else
        swap(&arr[i],&arr[j]);
    }
    swap(&arr[i],&arr[l]);
    quickSort(arr,f,i-1);
    quickSort(arr,i+1,l);
}
```

```
int main(){
    ll n;
    cout<<"Enter Arry size::";
    cin>>n;
    vector<ll> arr(n,0);
    cout<<"Enter the unsorted arry: ";
    for(auto &i:arr)
    cin>>i;
    cout<<"Merge sorted arry: ";
    quickSort(arr,0,n-1);
    for(auto i:arr)
    cout<<i<<" , ";
}
```

```
/*
    Harsh Kumar
    (2021UCA1829)
    DAA(Design and Analysis of Algorithms
*/
// Selection sort in C++
#include <bits/stdc++.h>
#define ll long long int
using namespace std;

void SelectionSort(vector<ll> &arr, int n)
{
    for(int i=0;i<n-1;i++){
        ll val=arr[i];
        int pos=i;
        for(int j=i+1;j<n;j++){
            if(arr[j]<val){
                pos=j;
                val=arr[j];
            }
        }
        arr[pos]=arr[i];
        arr[i]=val;
    }
}

int main(){
    ll n;
    cout<<"Enter Arry size::";
    cin>>n;
    vector<ll> arr(n,0);
    cout<<"Enter the unsorted arry: ";
    for(auto &i:arr)
    cin>>i;
```

```
    cout<<"Insertion sorted arry: ";
    SelectionSort(arr,n);
    for(auto i:arr)
    cout<<i<<" , ";
}
```

OUTPUT:

Bubble Sort

```
Enter Arry size::10
Enter the unsorted arry: 8 56 96 21 45 78 52 36 88 10
Bubble sorted arry: 8 , 10 , 21 , 36 , 45 , 52 , 56 , 78 , 88 , 96 ,
```

Insertion Sort

```
Enter Arry size::10
Enter the unsorted arry: 8 56 96 21 45 78 52 36 88 10
Insertion sorted arry: 8 , 10 , 21 , 36 , 45 , 52 , 56 , 78 , 88 , 96 ,
```

```
Enter Arry size::10
Enter the unsorted arry: 8 56 96 21 45 78 52 36 88 10
Bubble sorted arry: 8 , 10 , 21 , 36 , 45 , 52 , 56 , 78 , 88 , 96 ,
```

Merge Sort

```
Enter Arry size::10
Enter the unsorted arry: 8 56 96 21 45 78 52 36 88 10
Merge sorted arry: 8 , 10 , 21 , 36 , 45 , 52 , 56 , 78 , 88 , 96 ,
```

Quick Sort

```
Enter Arry size::10
Enter the unsorted arry: 8 56 96 21 45 78 52 36 88 10
Quick sorted arry: 8 , 10 , 21 , 36 , 45 , 52 , 56 , 78 , 88 , 96 ,
```

Selection Sort

```
Enter Arry size::10
Enter the unsorted arry: 8 56 96 21 45 78 52 36 88 10
Selection sorted arry: 8 , 10 , 21 , 36 , 45 , 52 , 56 , 78 , 88 , 96 ,
```

2. Write a program for Radix sort, Count sort, Bucket sort, and Shell sort..

DESCRIPTION:

- Radix sort is an integer sorting algorithm that sorts data with integer keys by grouping the keys by individual digits that share the same significant position and value (place value). Radix sort uses counting sort as a subroutine to sort an array of numbers. Because integers can be used to represent strings (by hashing the strings to integers), radix sort works on data types other than just integers. Because radix sort is not comparison based, it is not bounded by Omega(n log n) Ω(nlogn) for running time
- Counting sort is an efficient algorithm for sorting an array of elements that each have a nonnegative integer key, for example, an array, sometimes called a list, of positive integers could have keys that are just the value of the integer as the key, or a list of words could have keys assigned to them by some scheme mapping the alphabet to integers (to sort in alphabetical order, for instance). Unlike other sorting algorithms, such as mergesort, counting sort is an integer sorting algorithm, not a comparison based algorithm. While any comparison based sorting algorithm requires Ω(nlgn)

CODE:

```
/*
    Harsh Kumar
    (2021UCA1829)
    DAA(Design and Analysis of Algorithms
*/
// Bucket sort in C++
#include<bits/stdc++.h>
#define ll long long int
using namespace std;

void InsertionSort(vector<float> &arr, int n)
{
    for(int i=1;i<n;i++){
        float key=arr[i];
        int j=i-1;
        while(arr[j]>key && j>=0)
        arr[j+1]=arr[j--];
        arr[j+1]=key;
    }
}

void bucketSort(vector<float> &arr, int n){
```

```
    vector<float> bucket[n];
    for(int i=0;i<n;i++){
        int b=n*arr[i];
        bucket[b].push_back(arr[i]);
    }

    for(auto &i:bucket)
    {
        InsertionSort(i,i.size());
    }

    for(int j=0,k=0;j<n;j++)
    for(int i=0;i<bucket[j].size();i++){
        arr[k++]=bucket[j][i];
    }
}
int main(){
    float n;
    cout<<"Enter Arry size::";
    cin>>n;
    vector<float> arr(n,0);
    cout<<"Enter the unsorted arry: ";
    for(auto &i:arr)
    cin>>i;
    cout<<"Merge sorted arry: ";
    bucketSort(arr,n);
    for(auto i:arr)
    cout<<i<<" , ";
}
```

```
/*
    Harsh Kumar
    (2021UCA1829)
    DAA(Design and Analysis of Algorithms
*/
// counting sort in C++
#include<bits/stdc++.h>
#define ll long long int
using namespace std;


void countingSort(vector<ll> &arr, int n){
    ll max=INT_MIN;
    for(auto i:arr)
    if(i>max)max=i;

    vector<ll> counting(max+1,0), ans(n);
    for(auto i:arr)
    counting[i]++;

    for(int i=1;i<=max;i++)
```

```
    counting[i]+=counting[i-1];

    for(auto i:arr)
    {
        ans[--counting[i]]=i;
    }
    arr=ans;
}
int main(){
    ll n;
    cout<<"Enter Arry size::";
    cin>>n;
    vector<ll> arr(n,0);
    cout<<"Enter the unsorted arry: ";
    for(auto &i:arr)
    cin>>i;
    cout<<"Counting sorted arry: ";
    countingSort(arr,n);
    for(auto i:arr)
    cout<<i<<" , ";
}
```

```
/*
    Harsh Kumar
    (2021UCA1829)
    DAA(Design and Analysis of Algorithms
*/
// radix sort in C++
#include<bits/stdc++.h>
#define ll long long int
using namespace std;

void countingSortExp(vector<ll> &arr, int n, int exp){
    vector<ll> counting(10,0), ans(n);
    for(auto i:arr)
    counting[(i/exp)%10]++;

    for(int i=1;i<10;i++)
    counting[i]+=counting[i-1];
    //this accumulation should run in reverse
    // since the number apper first in arry will append after the same value
number apperaing later
    // the if 65,66 apper... in exp=1: 65,66....in exp=2: 66,65
    for(int i=n-1;i>=0;i--)
    {
        ans[--counting[(arr[i]/exp)%10]]=arr[i];
    }
    arr=ans;
}
```

```
void radixSort(vector<ll> &arr, int n){
    ll max=INT_MIN;
    for(auto i:arr)
    if(i>max)max=i;

    for(int exp=1;max/exp>0;exp*=10)
    countingSortExp(arr,n,exp);
}

int main(){
    ll n;
    cout<<"Enter Arry size::";
    cin>>n;
    vector<ll> arr(n,0);
    cout<<"Enter the unsorted arry: ";
    for(auto &i:arr)
    cin>>i;
    cout<<"radix sorted arry: ";
    radixSort(arr,n);
    for(auto i:arr)
    cout<<i<<" , ";
}
```

```
/*
    Harsh Kumar
    (2021UCA1829)
    DAA(Design and Analysis of Algorithms
*/
// Sehll sort in C++
#include <bits/stdc++.h>
#define ll long long int
using namespace std;

void SehllSort(vector<ll> &arr, int n)
{
    for (int gap = n / 2; gap > 0; gap /= 2)
    {
        for (int i = gap; i < n; i++)
        {
            ll key = arr[i];
            int j = i;
            while (arr[j-gap] > key && j >= gap)
            {
                arr[j] = arr[j-gap];
                j -= gap;
            }
            arr[j] = key;
        }
    }
}
```

```cpp
int main()
{
    ll n;
    cout << "Enter Arry size::";
    cin >> n;
    vector<ll> arr(n, 0);
    cout << "Enter the unsorted arry: ";
    for (auto &i : arr)
        cin >> i;
    cout << "Sehll sorted arry: ";
    SehllSort(arr, n);
    for (auto i : arr)
        cout << i << " , ";
}
```

OUTPUT:

```
Enter Arry size::10
Enter the unsorted arry: 8 56 96 21 45 78 52 36 88 10
Bucket sorted arry: 8 , 10 , 21 , 36 , 45 , 52 , 56 , 78 , 88 , 96 ,
```

```
Enter Arry size::10
Enter the unsorted arry: 8 56 96 21 45 78 52 36 88 10
Counting sorted arry: 8 , 10 , 21 , 36 , 45 , 52 , 56 , 78 , 88 , 96 ,
```

```
Enter Arry size::10
Enter the unsorted arry: 8 56 96 21 45 78 52 36 88 10
Bucket sorted arry: 8 , 10 , 21 , 36 , 45 , 52 , 56 , 78 , 88 , 96 ,
```

```
Enter Arry size::10
Enter the unsorted arry: 8 56 96 21 45 78 52 36 88 10
Shell sorted arry: 8 , 10 , 21 , 36 , 45 , 52 , 56 , 78 , 88 , 96 ,
```

3. Write a program for Linear and Binary search

DESCRIPTION:

- It is widely used to search an element from the unordered list, i.e., the list in which items are not sorted. The worst-case time complexity of linear search is O(n).
- binary search is the search technique that works efficiently on sorted lists. Hence, to search an element into some list using the binary search technique, we must ensure that the list is sorted. O(log n)

CODE:

```
/*
    Harsh Kumar
    (2021UCA1829)
    DAA(Design and Analysis of Algorithms
*/
// Binary Search in C++
#include <bits/stdc++.h>
#define ll long long int
using namespace std;

int BinarySearch(vector<ll> &arr, ll k, int f, int l)
{
    if (f > l)
        return -1;

    int mid = f + (l - f) / 2;
    if (arr[mid] == k)
        return mid;
    if (arr[mid] > k)
        return BinarySearch(arr, k, f, mid - 1);
    else
        return BinarySearch(arr, k, mid + 1, l);
}

int main()
{
    ll n, k;
    cout << "Enter Arry size::";
    cin >> n;
    vector<ll> arr(n, 0);
    cout << "Enter the  arry: ";
    for (auto &i : arr)
        cin >> i;
    cout << "Enter the Element to search: ";
    cin >> k;
```

```
    cout << k << " is at  " << BinarySearch(arr, k,0,n ) << " index";
}
```

```
/*
    Harsh Kumar
    (2021UCA1829)
    DAA(Design and Analysis of Algorithms
*/
// Linear Search in C++
#include <bits/stdc++.h>
#define ll long long int
using namespace std;

int LinearSearch(vector<ll> &arr, int n, ll k)
{
    for (int i = 0; i < n; i++)
    {
        if (arr[i] == k)
            return i;
    }
    return -1;
}

int main()
{
    ll n, k;
    cout << "Enter Arry size::";
    cin >> n;
    vector<ll> arr(n, 0);
    cout << "Enter the  arry: ";
    for (auto &i : arr)
        cin >> i;
    cout << "Enter the Element to search: ";
    cin >> k;
    cout << k << " is at  " << LinearSearch(arr, n, k) << " index";
}
```

OUTPUT:

```
Enter Arry size::10
Enter the  arry: 12 58 63 97 11 25 33 56 87 41
Enter the Element to search: 58
58 is at  -1 index
```

```
Enter the  arry: 8 56 96 21 45 78 52 36 88 10
Enter the Element to search: 8
8 is at  0 index
```

4. Write a program for tower of Hanoi.

DESCRIPTION:

- 1. It is a classic problem where you try to move all the disks from one peg to another peg using only three pegs. ➢
- 2. Initially, all of the disks are stacked on top of each other with larger disks under the smaller disks. ➢
- 3. You may move the disks to any of three pegs as you attempt to relocate all of the disks, but you cannot place the larger disks over smaller disks and only one disk can be transferred at a time. ➢
- This problem can be easily solved by Divide & Conquer algorithm. The worst-case time complexity of linear search is O(n).

CODE:

```
/*
    Harsh Kumar
    (2021UCA1829)
    DAA(Design and Analysis of Algorithms
*/
// Tower of Honoi in C++
#include <bits/stdc++.h>
#define ll long long int
using namespace std;

void toh(int n, char t1, char t2, char t3)
{
    if (n == 0)
        return;

    toh(n - 1, t1, t3, t2);
    cout << "plate " << n << ", from " << t1 << " to " << t2 << endl;
    toh(n - 1, t3, t2, t1);
}
int main()
{
    int n = 3;
    toh(n, 'a', 'b', 'c');
}
```

OUTPUT:

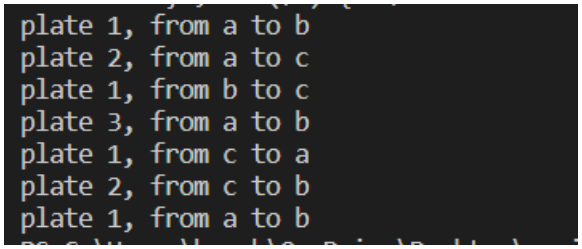


plate 1, from a to b
plate 2, from a to c
plate 1, from b to c
plate 3, from a to b
plate 1, from c to a
plate 2, from c to b
plate 1, from a to b

5. Write a program for inserting elements in AVT, BST, Red-Black Tree.

DESCRIPTION:

- AVL Tree can be defined as height balanced binary search tree in which each node is associated with a balance factor which is calculated by subtracting the height of its right sub-tree from that of its left sub-tree
- A red-black tree is a Binary tree where a particular node has color as an extra attribute, either red or black. By check the node colors on any simple path from the root to a leaf, red-black trees secure that no such path is higher than twice as long as any other so that the tree is generally balanced.
- A binary search tree follows some order to arrange the elements. In a Binary search tree, the value of left node must be smaller than the parent node, and the value of right node must be greater than the parent node. This rule is applied recursively to the left and right subtrees of the root.

CODE:

```
/*
    Harsh Kumar
    (2021UCA1829)
    DAA(Design and Analysis of Algorithms
*/
//AVL Tree Insertion
#include<bits/stdc++.h>
using namespace std;


struct Node
{
    int data;
    Node *left, *right;
    int height;
    Node(int data)
    {
        this->data = data;
        this->left = NULL;
        this->right = NULL;
        this->height = 1;
    }
};

int height(Node *node)
{
```

```
    if (node == NULL)
        return 0;
    return node->height;
}

Node *leftRotate(Node *x)
{
    Node *y = x->right;
    Node *T2 = y->left;

    y->left = x;
    x->right = T2;

    x->height = max(height(x->left), height(x->right)) + 1;
    y->height = max(height(y->left), height(y->right)) + 1;

    return y;
}

Node *rightRotate(Node *y)
{
    Node *x = y->left;
    Node *T2 = x->right;

    x->right = y;
    y->left = T2;

    y->height = max(height(y->left), height(y->right)) + 1;
    x->height = max(height(x->left), height(x->right)) + 1;

    return x;
}

int getBalance(Node *node)
{
    if (node == NULL)
        return 0;
    return height(node->left) - height(node->right);
}

Node *insert(Node *node, int data)
{
    if (node == NULL)
        return new Node(data);

    if (data < node->data)
        node->left = insert(node->left, data);
    else if (data > node->data)
        node->right = insert(node->right, data);
    else
        return node;
```

```
    node->height = max(height(node->left), height(node->right)) + 1;

    int bal = getBalance(node);

    if (bal > 1 && data < node->left->data)
        return rightRotate(node);

    if (bal < -1 && data > node->right->data)
        return leftRotate(node);

    if (bal > 1 && data > node->left->data)
    {
        node->left = leftRotate(node->left);
        return rightRotate(node);
    }

    if (bal < -1 && data < node->right->data)
    {
        node->right = rightRotate(node->right);
        return leftRotate(node);
    }

    return node;
}

void level_order_treversal(Node *root)
    {

    cout<<"\nLEVEL ORDER TRAVERSAL::\n";
        queue<Node *> q;
        q.push(root);
        q.push(NULL);
        while (!q.empty())
        {
            Node *temp = q.front();
            q.pop();
            if (temp == NULL)
            {
                if (!q.empty())
                {
                    q.push(NULL);
                    cout << "\n";
                }
            }
            else
            {
                cout << temp->data << "   ";
                if (temp->left != NULL)
                    q.push(temp->left);
                if (temp->right != NULL)
                    q.push(temp->right);
            }
```

```
        }
    }
int main()
{
    Node *root = NULL;

    root = insert(root, 9);
    root = insert(root, 5);
    root = insert(root, 10);
    root = insert(root, 0);
    root = insert(root, 6);
    root = insert(root, 11);
    root = insert(root, -1);
    root = insert(root, 1);
    root = insert(root, 2);
    level_order_treversal(root);
}
```

```
/*
    Harsh Kumar
    (2021UCA1829)
    DAA(Design and Analysis of Algorithms
*/
//Red Black insertion
#include <bits/stdc++.h>
#define ll long long int
using namespace std;

struct Node {
  int data;
  Node *parent;
  Node *left;
  Node *right;
  int color;
};

class RedBlackTree {
   private:
  Node* root;
  Node* TNULL;

  void initializeNULLNode(Node* node, Node* parent) {
    node->data = 0;
    node->parent = parent;
    node->left = nullptr;
    node->right = nullptr;
    node->color = 0;
  }
  void preOrderHelper(Node* node) {
    if (node != TNULL) {
      cout << node->data << " ";
```

```
    preOrderHelper(node->left);
    preOrderHelper(node->right);
  }
}

void inOrderHelper(Node* node) {
  if (node != TNULL) {
    inOrderHelper(node->left);
    cout << node->data << " ";
    inOrderHelper(node->right);
  }
}

void postOrderHelper(Node* node) {
  if (node != TNULL) {
    postOrderHelper(node->left);
    postOrderHelper(node->right);
    cout << node->data << " ";
  }
}



void deleteFix(Node* x) {
  Node* s;
  while (x != root && x->color == 0) {
    if (x == x->parent->left) {
      s = x->parent->right;
      if (s->color == 1) {
        s->color = 0;
        x->parent->color = 1;
        leftRotate(x->parent);
        s = x->parent->right;
      }

      if (s->left->color == 0 && s->right->color == 0) {
        s->color = 1;
        x = x->parent;
      } else {
        if (s->right->color == 0) {
          s->left->color = 0;
          s->color = 1;
          rightRotate(s);
          s = x->parent->right;
        }

        s->color = x->parent->color;
        x->parent->color = 0;
        s->right->color = 0;
        leftRotate(x->parent);
        x = root;
      }
```

```
    } else {
      s = x->parent->left;
      if (s->color == 1) {
        s->color = 0;
        x->parent->color = 1;
        rightRotate(x->parent);
        s = x->parent->left;
      }

      if (s->right->color == 0 && s->right->color == 0) {
        s->color = 1;
        x = x->parent;
      } else {
        if (s->left->color == 0) {
          s->right->color = 0;
          s->color = 1;
          leftRotate(s);
          s = x->parent->left;
        }

        s->color = x->parent->color;
        x->parent->color = 0;
        s->left->color = 0;
        rightRotate(x->parent);
        x = root;
      }
    }
  }
  x->color = 0;
}

void rbTransplant(Node* u, Node* v) {
  if (u->parent == nullptr) {
    root = v;
  } else if (u == u->parent->left) {
    u->parent->left = v;
  } else {
    u->parent->right = v;
  }
  v->parent = u->parent;
}

void deleteNodeHelper(Node* node, int key) {
  Node* z = TNULL;
  Node* x, *y;
  while (node != TNULL) {
    if (node->data == key) {
      z = node;
    }

    if (node->data <= key) {
      node = node->right;
```

```
    } else {
      node = node->left;
    }
  }

  if (z == TNULL) {
    cout << "Key not found in the tree" << endl;
    return;
  }

  y = z;
  int y_original_color = y->color;
  if (z->left == TNULL) {
    x = z->right;
    rbTransplant(z, z->right);
  } else if (z->right == TNULL) {
    x = z->left;
    rbTransplant(z, z->left);
  } else {
    y = minimum(z->right);
    y_original_color = y->color;
    x = y->right;
    if (y->parent == z) {
      x->parent = y;
    } else {
      rbTransplant(y, y->right);
      y->right = z->right;
      y->right->parent = y;
    }

    rbTransplant(z, y);
    y->left = z->left;
    y->left->parent = y;
    y->color = z->color;
  }
  delete z;
  if (y_original_color == 0) {
    deleteFix(x);
  }
}
void insertFix(Node* k) {
  Node* u;
  while (k->parent->color == 1) {
    if (k->parent == k->parent->parent->right) {
      u = k->parent->parent->left;
      if (u->color == 1) {
        u->color = 0;
        k->parent->color = 0;
        k->parent->parent->color = 1;
        k = k->parent->parent;
      } else {
        if (k == k->parent->left) {
```

```
          k = k->parent;
          rightRotate(k);
        }
        k->parent->color = 0;
        k->parent->parent->color = 1;
        leftRotate(k->parent->parent);
      }
    } else {
      u = k->parent->parent->right;

      if (u->color == 1) {
        u->color = 0;
        k->parent->color = 0;
        k->parent->parent->color = 1;
        k = k->parent->parent;
      } else {
        if (k == k->parent->right) {
          k = k->parent;
          leftRotate(k);
        }
        k->parent->color = 0;
        k->parent->parent->color = 1;
        rightRotate(k->parent->parent);
      }
    }
    if (k == root) {
      break;
    }
  }
  root->color = 0;
}

void printHelper(Node* root, string indent, bool last) {
  if (root != TNULL) {
    cout << indent;
    if (last) {
      cout << "R----";
      indent += "   ";
    } else {
      cout << "L----";
      indent += "|  ";
    }

    string sColor = root->color ? "RED" : "BLACK";
    cout << root->data << "(" << sColor << ")" << endl;
    printHelper(root->left, indent, false);
    printHelper(root->right, indent, true);
  }
}

 public:
RedBlackTree() {
```

```
  TNULL = new Node;
  TNULL->color = 0;
  TNULL->left = nullptr;
  TNULL->right = nullptr;
  root = TNULL;
}

void preorder() {
  preOrderHelper(this->root);
}

void inorder() {
  inOrderHelper(this->root);
}

void postorder() {
  postOrderHelper(this->root);
}


Node* minimum(Node* node) {
  while (node->left != TNULL) {
    node = node->left;
  }
  return node;
}

Node* maximum(Node* node) {
  while (node->right != TNULL) {
    node = node->right;
  }
  return node;
}

Node* successor(Node* x) {
  if (x->right != TNULL) {
    return minimum(x->right);
  }

  Node* y = x->parent;
  while (y != TNULL && x == y->right) {
    x = y;
    y = y->parent;
  }
  return y;
}

Node* predecessor(Node* x) {
  if (x->left != TNULL) {
    return maximum(x->left);
  }
```

```
  Node* y = x->parent;
  while (y != TNULL && x == y->left) {
    x = y;
    y = y->parent;
  }

  return y;
}

void leftRotate(Node* x) {
  Node* y = x->right;
  x->right = y->left;
  if (y->left != TNULL) {
    y->left->parent = x;
  }
  y->parent = x->parent;
  if (x->parent == nullptr) {
    this->root = y;
  } else if (x == x->parent->left) {
    x->parent->left = y;
  } else {
    x->parent->right = y;
  }
  y->left = x;
  x->parent = y;
}

void rightRotate(Node* x) {
  Node* y = x->left;
  x->left = y->right;
  if (y->right != TNULL) {
    y->right->parent = x;
  }
  y->parent = x->parent;
  if (x->parent == nullptr) {
    this->root = y;
  } else if (x == x->parent->right) {
    x->parent->right = y;
  } else {
    x->parent->left = y;
  }
  y->right = x;
  x->parent = y;
}

void insert(int key) {
  Node* node = new Node;
  node->parent = nullptr;
  node->data = key;
  node->left = TNULL;
  node->right = TNULL;
  node->color = 1;
```

```
    Node* y = nullptr;
    Node* x = this->root;

    while (x != TNULL) {
      y = x;
      if (node->data < x->data) {
        x = x->left;
      } else {
        x = x->right;
      }
    }

    node->parent = y;
    if (y == nullptr) {
      root = node;
    } else if (node->data < y->data) {
      y->left = node;
    } else {
      y->right = node;
    }

    if (node->parent == nullptr) {
      node->color = 0;
      return;
    }

    if (node->parent->parent == nullptr) {
      return;
    }

    insertFix(node);
  }

  Node* getRoot() {
    return this->root;
  }

  void deleteNode(int data) {
    deleteNodeHelper(this->root, data);
  }

  void printTree() {
    if (root) {
      printHelper(this->root, "", true);
    }
  }
};

int main() {
  RedBlackTree bst;
  bst.insert(55);
```

```
  bst.insert(40);
  bst.insert(65);
  bst.insert(60);
  bst.insert(75);
  bst.insert(57);

  bst.printTree();
  cout << endl
     << "After deleting" << endl;
  bst.deleteNode(40);
  bst.printTree();
}
```

```
/*
    Harsh Kumar
    (2021UCA1829)
    DAA(Design and Analysis of Algorithms
*/
//BST insertion
#include <bits/stdc++.h>
#define ll long long int
using namespace std;

struct node
{
    ll data;
    node *left;
    node *right;
};
class BST
{
    node *root, *temp;
    node *newNode(ll value)
    {
        node *ptr = new node;
        ptr->left = NULL;
        ptr->right = NULL;
        ptr->data = value;
        return ptr;
    }
    void insert(ll n, node *temp);
public:
    BST()
    {
        root = NULL;
    }
    void insert_node(ll n)
    {
        if (root == NULL)
        {
```

```
            root = newNode(n);
        }
        else
            insert(n, root);
    }
    node* min_value(node *ptr)
    {
        node *temp = ptr;

        while (temp && temp->left != NULL)
            temp = temp->left;

        return temp;
    }

    void level_order_treversal()
    {
        cout << endl
             << " LEVEL ORDER TRAVERSAL:: " << endl;
        queue<node *> S;
        S.push(root);
        S.push(NULL);
        while (!S.empty())
        {
            node *temp = S.front();
            S.pop();
            if (temp == NULL && !S.empty())
            {
                S.push(NULL);
                cout << endl;
            }
            else
            {
                if (temp->left != NULL)
                    S.push(temp->left);
                if (temp->right != NULL)
                    S.push(temp->right);
                cout << "  " << temp->data;
            }
        }
    }
};
void BST::insert(ll n, node *temp)
{
    if (temp->data >= n)
    {
        if (temp->left == NULL)
            temp->left = newNode(n);
        else
            insert(n, temp->left);
    }
    else if (temp->data < n)
```

```
    {
        if (temp->right == NULL)
            temp->right = newNode(n);
        else
            insert(n, temp->right);
    }
}

int main()
{
    BST T1;
    T1.insert_node(3);
    T1.insert_node(2);
    T1.insert_node(5);
    T1.insert_node(8);
    T1.insert_node(4);
    T1.insert_node(1);
    T1.insert_node(15);
    T1.level_order_treversal();
}
```

OUTPUT:

```
LEVEL ORDER TRAVERSAL::
9
1   10
0   5   11
-1   2   6
```

```
R----55(BLACK)
   L----40(BLACK)
   R----65(RED)
      L----60(BLACK)
      |  L----57(RED)
      R----75(BLACK)

After deleting
R----65(BLACK)
   L----57(RED)
   |  L----55(BLACK)
   |  R----60(BLACK)
   R----75(BLACK)
```

```
LEVEL ORDER TRAVERSAL::
3
2  5
1  4  8
15
```

6. Write a program for deleting elements in AVT, BST, Red-Black Tree..

DESCRIPTION:

- AVL Tree can be defined as height balanced binary search tree in which each node is associated with a balance factor which is calculated by subtracting the height of its right sub-tree from that of its left sub-tree
- A red-black tree is a Binary tree where a particular node has color as an extra attribute, either red or black. By check the node colors on any simple path from the root to a leaf, red-black trees secure that no such path is higher than twice as long as any other so that the tree is generally balanced.
- A binary search tree follows some order to arrange the elements. In a Binary search tree, the value of left node must be smaller than the parent node, and the value of right node must be greater than the parent node. This rule is applied recursively to the left and right subtrees of the root.

CODE:

```
/*
    Harsh Kumar
    (2021UCA1829)
    DAA(Design and Analysis of Algorithms
*/
//AVL  TREE

#include<bits/stdc++.h>
using namespace std;


struct Node
{
    int data;
    Node *left, *right;
    int height;
    Node(int data)
    {
        this->data = data;
        this->left = NULL;
        this->right = NULL;
        this->height = 1;
    }
};
```

```
int height(Node *node)
{
    if (node == NULL)
        return 0;
    return node->height;
}

Node *leftRotate(Node *x)
{
    Node *y = x->right;
    Node *T2 = y->left;

    y->left = x;
    x->right = T2;

    x->height = max(height(x->left), height(x->right)) + 1;
    y->height = max(height(y->left), height(y->right)) + 1;

    return y;
}

Node *rightRotate(Node *y)
{
    Node *x = y->left;
    Node *T2 = x->right;

    x->right = y;
    y->left = T2;

    y->height = max(height(y->left), height(y->right)) + 1;
    x->height = max(height(x->left), height(x->right)) + 1;

    return x;
}

int getBalance(Node *node)
{
    if (node == NULL)
        return 0;
    return height(node->left) - height(node->right);
}

Node *insert(Node *node, int data)
{
    if (node == NULL)
        return new Node(data);

    if (data < node->data)
        node->left = insert(node->left, data);
    else if (data > node->data)
        node->right = insert(node->right, data);
```

```
    else
        return node;

    node->height = max(height(node->left), height(node->right)) + 1;

    int bal = getBalance(node);

    if (bal > 1 && data < node->left->data)
        return rightRotate(node);

    if (bal < -1 && data > node->right->data)
        return leftRotate(node);

    if (bal > 1 && data > node->left->data)
    {
        node->left = leftRotate(node->left);
        return rightRotate(node);
    }

    if (bal < -1 && data < node->right->data)
    {
        node->right = rightRotate(node->right);
        return leftRotate(node);
    }

    return node;
}

Node *minValNode(Node *node)
{
    while (node->left != NULL)
        node = node->left;
    return node;
}

Node *deleteNode(Node *root, int no)
{
    if (root == NULL)
        return root;

    if (no < root->data)
        root->left = deleteNode(root->left, no);
    else if (no > root->data)
        root->right = deleteNode(root->right, no);
    else
    {
        if (root->left == NULL || root->right == NULL)
        {
            Node *temp = root->right ? root->right : root->left;

            if (temp == NULL)
            {
```

```
                temp = root;
                root = NULL;
            }
            else
                *root = *temp;
            delete(temp);
        }
        else
        {
            Node *temp = minValNode(root->right);
            root->data = temp->data;
            root->right = deleteNode(root->right, temp->data);
        }
    }
    if (root == NULL)
        return root;

    root->height = max(height(root->left), height(root->right)) + 1;
    int bal = getBalance(root);

    if (bal > 1 && getBalance(root->left) >= 0)
        return rightRotate(root);

    if (bal > 1 && getBalance(root->left) < 0)
    {
        root->left = leftRotate(root->left);
        return rightRotate(root);
    }

    if (bal < -1 && getBalance(root->right) <= 0)
        return leftRotate(root);

    if (bal < -1 && getBalance(root->right) > 0)
    {
        root->right = rightRotate(root->right);
        return leftRotate(root);
    }

    return root;
}

void preOrder(Node *root)
{
    if (root != NULL)
    {
        cout << root->data << " ";
        preOrder(root->left);
        preOrder(root->right);
    }
}
void level_order_treversal(Node *root)
    {
```

```
    cout<<"\nLEVEL ORDER TRAVERSAL::\n";
        queue<Node *> q;
        q.push(root);
        q.push(NULL);
        while (!q.empty())
        {
            Node *temp = q.front();
            q.pop();
            if (temp == NULL)
            {
                if (!q.empty())
                {
                    q.push(NULL);
                    cout << "\n";
                }
            }
            else
            {
                cout << temp->data << "   ";
                if (temp->left != NULL)
                    q.push(temp->left);
                if (temp->right != NULL)
                    q.push(temp->right);
            }
        }
    }
int main()
{
    Node *root = NULL;

    root = insert(root, 9);
    root = insert(root, 5);
    root = insert(root, 10);
    root = insert(root, 0);
    root = insert(root, 6);
    root = insert(root, 11);
    root = insert(root, -1);
    root = insert(root, 1);
    root = insert(root, 2);

    cout << "Preorder traversal of the AVL tree is:\n";
    preOrder(root);

    root = deleteNode(root, 10);

    cout << "\nPreorder traversal after deletion is:\n";
    preOrder(root);
}
```

```
/*
```

```
    Harsh Kumar
    (2021UCA1829)
    DAA(Design and Analysis of Algorithms
*/
//RED Black Tree
#include <bits/stdc++.h>
#define ll long long int
using namespace std;

struct Node {
  int data;
  Node *parent;
  Node *left;
  Node *right;
  int color;
};

class RedBlackTree {
   private:
  Node* root;
  Node* TNULL;

  void initializeNULLNode(Node* node, Node* parent) {
    node->data = 0;
    node->parent = parent;
    node->left = nullptr;
    node->right = nullptr;
    node->color = 0;
  }
  void preOrderHelper(Node* node) {
    if (node != TNULL) {
      cout << node->data << " ";
      preOrderHelper(node->left);
      preOrderHelper(node->right);
    }
  }

  void inOrderHelper(Node* node) {
    if (node != TNULL) {
      inOrderHelper(node->left);
      cout << node->data << " ";
      inOrderHelper(node->right);
    }
  }

  void postOrderHelper(Node* node) {
    if (node != TNULL) {
      postOrderHelper(node->left);
      postOrderHelper(node->right);
      cout << node->data << " ";
    }
  }
```

```
void deleteFix(Node* x) {
  Node* s;
  while (x != root && x->color == 0) {
    if (x == x->parent->left) {
      s = x->parent->right;
      if (s->color == 1) {
        s->color = 0;
        x->parent->color = 1;
        leftRotate(x->parent);
        s = x->parent->right;
      }

      if (s->left->color == 0 && s->right->color == 0) {
        s->color = 1;
        x = x->parent;
      } else {
        if (s->right->color == 0) {
          s->left->color = 0;
          s->color = 1;
          rightRotate(s);
          s = x->parent->right;
        }

        s->color = x->parent->color;
        x->parent->color = 0;
        s->right->color = 0;
        leftRotate(x->parent);
        x = root;
      }
    } else {
      s = x->parent->left;
      if (s->color == 1) {
        s->color = 0;
        x->parent->color = 1;
        rightRotate(x->parent);
        s = x->parent->left;
      }

      if (s->right->color == 0 && s->right->color == 0) {
        s->color = 1;
        x = x->parent;
      } else {
        if (s->left->color == 0) {
          s->right->color = 0;
          s->color = 1;
          leftRotate(s);
          s = x->parent->left;
        }
```

```
        s->color = x->parent->color;
        x->parent->color = 0;
        s->left->color = 0;
        rightRotate(x->parent);
        x = root;
      }
    }
  }
  x->color = 0;
}

void rbTransplant(Node* u, Node* v) {
  if (u->parent == nullptr) {
    root = v;
  } else if (u == u->parent->left) {
    u->parent->left = v;
  } else {
    u->parent->right = v;
  }
  v->parent = u->parent;
}

void deleteNodeHelper(Node* node, int key) {
  Node* z = TNULL;
  Node* x, *y;
  while (node != TNULL) {
    if (node->data == key) {
      z = node;
    }

    if (node->data <= key) {
      node = node->right;
    } else {
      node = node->left;
    }
  }

  if (z == TNULL) {
    cout << "Key not found in the tree" << endl;
    return;
  }

  y = z;
  int y_original_color = y->color;
  if (z->left == TNULL) {
    x = z->right;
    rbTransplant(z, z->right);
  } else if (z->right == TNULL) {
    x = z->left;
    rbTransplant(z, z->left);
  } else {
    y = minimum(z->right);
```

```
    y_original_color = y->color;
    x = y->right;
    if (y->parent == z) {
      x->parent = y;
    } else {
      rbTransplant(y, y->right);
      y->right = z->right;
      y->right->parent = y;
    }

    rbTransplant(z, y);
    y->left = z->left;
    y->left->parent = y;
    y->color = z->color;
  }
  delete z;
  if (y_original_color == 0) {
    deleteFix(x);
  }
}
void insertFix(Node* k) {
  Node* u;
  while (k->parent->color == 1) {
    if (k->parent == k->parent->parent->right) {
      u = k->parent->parent->left;
      if (u->color == 1) {
        u->color = 0;
        k->parent->color = 0;
        k->parent->parent->color = 1;
        k = k->parent->parent;
      } else {
        if (k == k->parent->left) {
          k = k->parent;
          rightRotate(k);
        }
        k->parent->color = 0;
        k->parent->parent->color = 1;
        leftRotate(k->parent->parent);
      }
    } else {
      u = k->parent->parent->right;

      if (u->color == 1) {
        u->color = 0;
        k->parent->color = 0;
        k->parent->parent->color = 1;
        k = k->parent->parent;
      } else {
        if (k == k->parent->right) {
          k = k->parent;
          leftRotate(k);
        }
```

```
        k->parent->color = 0;
        k->parent->parent->color = 1;
        rightRotate(k->parent->parent);
      }
    }
    if (k == root) {
      break;
    }
  }
  root->color = 0;
}

void printHelper(Node* root, string indent, bool last) {
  if (root != TNULL) {
    cout << indent;
    if (last) {
      cout << "R----";
      indent += "   ";
    } else {
      cout << "L----";
      indent += "|  ";
    }

    string sColor = root->color ? "RED" : "BLACK";
    cout << root->data << "(" << sColor << ")" << endl;
    printHelper(root->left, indent, false);
    printHelper(root->right, indent, true);
  }
}

 public:
RedBlackTree() {
  TNULL = new Node;
  TNULL->color = 0;
  TNULL->left = nullptr;
  TNULL->right = nullptr;
  root = TNULL;
}

void preorder() {
  preOrderHelper(this->root);
}

void inorder() {
  inOrderHelper(this->root);
}

void postorder() {
  postOrderHelper(this->root);
}
```

```
Node* minimum(Node* node) {
  while (node->left != TNULL) {
    node = node->left;
  }
  return node;
}

Node* maximum(Node* node) {
  while (node->right != TNULL) {
    node = node->right;
  }
  return node;
}

Node* successor(Node* x) {
  if (x->right != TNULL) {
    return minimum(x->right);
  }

  Node* y = x->parent;
  while (y != TNULL && x == y->right) {
    x = y;
    y = y->parent;
  }
  return y;
}

Node* predecessor(Node* x) {
  if (x->left != TNULL) {
    return maximum(x->left);
  }

  Node* y = x->parent;
  while (y != TNULL && x == y->left) {
    x = y;
    y = y->parent;
  }

  return y;
}

void leftRotate(Node* x) {
  Node* y = x->right;
  x->right = y->left;
  if (y->left != TNULL) {
    y->left->parent = x;
  }
  y->parent = x->parent;
  if (x->parent == nullptr) {
    this->root = y;
  } else if (x == x->parent->left) {
    x->parent->left = y;
```

```
    } else {
      x->parent->right = y;
    }
    y->left = x;
    x->parent = y;
  }

  void rightRotate(Node* x) {
    Node* y = x->left;
    x->left = y->right;
    if (y->right != TNULL) {
      y->right->parent = x;
    }
    y->parent = x->parent;
    if (x->parent == nullptr) {
      this->root = y;
    } else if (x == x->parent->right) {
      x->parent->right = y;
    } else {
      x->parent->left = y;
    }
    y->right = x;
    x->parent = y;
  }

  void insert(int key) {
    Node* node = new Node;
    node->parent = nullptr;
    node->data = key;
    node->left = TNULL;
    node->right = TNULL;
    node->color = 1;

    Node* y = nullptr;
    Node* x = this->root;

    while (x != TNULL) {
      y = x;
      if (node->data < x->data) {
        x = x->left;
      } else {
        x = x->right;
      }
    }

    node->parent = y;
    if (y == nullptr) {
      root = node;
    } else if (node->data < y->data) {
      y->left = node;
    } else {
      y->right = node;
```

```
    }

    if (node->parent == nullptr) {
      node->color = 0;
      return;
    }

    if (node->parent->parent == nullptr) {
      return;
    }

    insertFix(node);
  }

  Node* getRoot() {
    return this->root;
  }

  void deleteNode(int data) {
    deleteNodeHelper(this->root, data);
  }

  void printTree() {
    if (root) {
      printHelper(this->root, "", true);
    }
  }
};

int main() {
  RedBlackTree bst;
  bst.insert(55);
  bst.insert(40);
  bst.insert(65);
  bst.insert(60);
  bst.insert(75);
  bst.insert(57);

  bst.printTree();
  cout << endl
     << "After deleting" << endl;
  bst.deleteNode(40);
  bst.printTree();
}
```

```
/*
    Harsh Kumar
    (2021UCA1829)
    DAA(Design and Analysis of Algorithms
*/
```

```
//BST
#include <bits/stdc++.h>
#define ll long long int
using namespace std;

struct node
{
    ll data;
    node *left;
    node *right;
};
class BST
{
public:
    node *root, *temp;
    node *newnode(ll value)
    {
        node *ptr = new node;
        ptr->left = NULL;
        ptr->right = NULL;
        ptr->data = value;
        return ptr;
    }
    void insert(ll n, node *temp);

    BST()
    {
        root = NULL;
    }
    void insert_node(ll n)
    {
        if (root == NULL)
        {
            root = newnode(n);
        }
        else
            insert(n, root);
    }
    node *deletenode(node *root, int no);

    void level_order_treversal()
    {

    cout<<"\nLEVEL ORDER TRAVERSAL::\n";
        queue<node *> q;
        q.push(root);
        q.push(NULL);
        while (!q.empty())
        {
            node *temp = q.front();
            q.pop();
            if (temp == NULL)
```

```
            {
                if (!q.empty())
                {
                    q.push(NULL);
                    cout << "\n";
                }
            }
            else
            {
                cout << temp->data << "   ";
                if (temp->left != NULL)
                    q.push(temp->left);
                if (temp->right != NULL)
                    q.push(temp->right);
            }
        }
    }
};
void BST::insert(ll n, node *temp)
{
    if (temp->data >= n)
    {
        if (temp->left == NULL)
            temp->left = newnode(n);
        else
            insert(n, temp->left);
    }
    else if (temp->data < n)
    {
        if (temp->right == NULL)
            temp->right = newnode(n);
        else
            insert(n, temp->right);
    }
}
node *BST::deletenode(node *root, int no)
{
    if (root == NULL)
        return root;

    if (no < root->data)
    {
        root->left = deletenode(root->left, no);
        return root;
    }
    else if (no > root->data)
    {
        root->right = deletenode(root->right, no);
        return root;
    }

    if (root->left == NULL)
```

```
    {
        node *temp = root->right;
        delete (root);
        return temp;
    }
    else if (root->right == NULL)
    {
        node *temp = root->left;
        delete (root);
        return temp;
    }
    else
    {
        node *predParent = root;
        node *pred = root->left;

        while (pred->right != NULL)
        {
            predParent = pred;
            pred = pred->right;
        }

        if (predParent != root)
            predParent->right = pred->left;
        else
            predParent->left = pred->left;
        root->data = pred->data;
        delete pred;
        return root;
    }
}

int main()
{
    BST T1;
    T1.insert_node(3);
    T1.insert_node(2);
    T1.insert_node(5);
    T1.insert_node(8);
    T1.insert_node(4);
    T1.insert_node(1);
    T1.insert_node(15);
    T1.level_order_treversal();

    cout<<endl<<T1.deletenode(T1.root, 8)<<" node deleted\n";

    T1.level_order_treversal();
}
```

OUTPUT:

```
Preorder traversal of the AVL tree is:
9 1 0 -1 5 2 6 10 11
Preorder traversal after deletion is:
1 0 -1 9 5 2 6 11
```

```
R----55(BLACK)
   L----40(BLACK)
   R----65(RED)
      L----60(BLACK)
      |  L----57(RED)
      R----75(BLACK)

After deleting
R----65(BLACK)
   L----57(RED)
   |  L----55(BLACK)
   |  R----60(BLACK)
   R----75(BLACK)
```

```
LEVEL ORDER TRAVERSAL::
3
2   5
1   4   8
15
0x11516c0 node deleted

LEVEL ORDER TRAVERSAL::
3
2   5
1   4   15
```

7. Write a program to return the maximum height of the tree..

DESCRIPTION:

- The height or depth of a binary tree can be defined as the maximum or the largest number of edges from a leaf node to the root node or root node to the leaf node. The root node will be at level zero that means if the root node doesn't have any of the child nodes connected to it then the height or depth of the particular binary tree is said to be zero

CODE:

```
/*
    Harsh Kumar
    (2021UCA1829)
    DAA(Design and Analysis of Algorithms
*/
#include<bits/stdc++.h>
using namespace std;

class node{
    public:
    int Data;
    node* left;
    node* right;
    node(int d){
        this->Data=d;
        this->left=NULL;
        this->right=NULL;
    }
};
node* createTree(node* root){
    cout<<"\n DATA: ";
    int data;
    cin>>data;
    if(data==-1){
        return root;
    }
    root= new node(data);
    cout<<"Left of "<<data;
    root->left=createTree(root->left);
    cout<<"Right of "<<data;
    root->right=createTree(root->right);
}
```

```
//Tree HEIGHT
int height(struct node* root){
    if(root==NULL)
    return 0;
    int Hl=height(root->left),Hr=height(root->right);
    if(Hl>=Hr)
    return Hl+1;
    else return Hr+1;
}

int main(){
    node* root=createTree(root);
    cout<<height(root);
}
```

OUTPUT:

```
 DATA: -1
Right of 56
 DATA: 12
Left of 12
 DATA: 12
Left of 12
 DATA: 86
Left of 86
 DATA: -1
Right of 86
 DATA: -1
Right of 12
 DATA: -1
Right of 12
 DATA: -1
Right of 12
 DATA: -1
6
```

8. Write a program to Find maximum and minimum of array using the dynamic programing.

DESCRIPTION:

- The maximum subarray problem is the task of finding the largest possible sum of a contiguous subarray, within a given one-dimensional array A[1…n] of numbers.

CODE:

```
/*
    Harsh Kumar
    (2021UCA1829)
    DAA(Design and Analysis of Algorithms
*/
#include <bits/stdc++.h>
using namespace std;

struct ans
{
    int min;
    int max;
};

ans getmm(int arr[], int n)
{
    ans mm;
    int i;

    if (n == 1)
    {
        mm.max = arr[0];
        mm.min = arr[0];
        return mm;
    }

    if (arr[0] > arr[1])
    {
        mm.max = arr[0];
        mm.min = arr[1];
    }
    else
    {
        mm.max = arr[1];
        mm.min = arr[0];
    }

    for (i = 2; i < n; i++)
```

```
    {
        if (arr[i] > mm.max)
            mm.max = arr[i];

        else if (arr[i] < mm.min)
            mm.min = arr[i];
    }
    return mm;
}
int main()
{
    int arr[] = {15,85,96,7,852,145,20,63};
    int arr_size = 8;

    ans mm = getmm(arr, arr_size);

    cout << "Minimum  "<<mm.min << endl;
    cout << "Maximum  "<< mm.max;

}
```

OUTPUT:

```
Minimum  7
Maximum  852
```

9. Write a program for job Sequencing problem.

DESCRIPTION:

- The height or depth of a binary tree can be defined as the maximum or the largest number of edges from a leaf node to the root node or root node to the leaf node. The root node will be at level zero that means if the root node doesn't have any of the child nodes connected to it then the height or depth of the particular binary tree is said to be zero

CODE:

```
/*
    Harsh Kumar
    (2021UCA1829)
    DAA(Design and Analysis of Algorithms
*/
#include <bits/stdc++.h>
#define ll long long int
using namespace std;

struct Job
{

    char id;
    int dead;
    int profit;
};

bool comparison(Job a, Job b)
{
    return (a.profit > b.profit);
}

void JobSequence(Job arr[], int n)
{
    sort(arr, arr + n, comparison);

    int result[n];
    bool slot[n];

    for (int i = 0; i < n; i++)
    {
        slot[i] = false;
    }

    for (int i = 0; i < n; i++)
```

```
    {
        for (int j = min(n, arr[i].dead) - 1; j >= 0; j--)
        {
            if (slot[j] == false)
            {
                result[j] = i;
                slot[j] = true;
                break;
            }
        }
    }

    for (int i = 0; i < n; i++)
    {
        if (slot[i])
        {
            cout << arr[result[i]].id << " ";
        }
    }
}

int main()
{
    Job arr[7] = {{'1', 3, 35},
                  {'2', 4, 30},
                  {'3', 4, 25},
                  {'4', 2, 20},
                  {'5', 3, 15},
                  {'6', 1, 12},
                  {'7', 2, 5}};
    cout << "Job sequence for maximum profit is : " << endl;
    JobSequence(arr, 7);
}
```

OUTPUT:

```
Job sequence for maximum profit is :
4 3 1 2
```

10. Write a program to get the maximum total value in the knapsack..

DESCRIPTION:

- The fractional knapsack problem is also one of the techniques which are used to solve the knapsack problem. In fractional knapsack, the items are broken in order to maximize the profit. The problem in which we break the item is known as a Fractional knapsack problem.

CODE:

```
/*
    Harsh Kumar
    (2021UCA1829)
    DAA(Design and Analysis of Algorithms
*/
#include <bits/stdc++.h>
#define ll long long int
using namespace std;
struct Item
{
    int value;
    int weight;
};
struct comparitor_
{
    bool operator()(Item min, Item max)
    {

        return (double)max.value / max.weight > (double)min.value /
min.weight;
    }
};
double fractionalKnapsack(int capacity, Item arr[], int n)
{
    priority_queue<Item, vector<Item>, comparitor_> pq;
    for (int i = 0; i < n; i++)
    {
        pq.push(arr[i]);
    }

    double total_value = 0;
    while (capacity && pq.size())
    {
        auto top = pq.top();
```

```
        pq.pop();
        if (top.weight <= capacity)
        {
            total_value += top.value;
            capacity -= top.weight;
        }
        else
        {
            total_value += (double)top.value / top.weight * capacity;
            capacity = 0;
        }
    }
    return total_value;
}

int main(){
    Item arr[]={{25,3},{20,2},{15,1},{40,4},{50,5}};
    ll maxW=6;
    cout<<"MAX total value"<<fractionalKnapsack(maxW,arr,5);
}
```

OUTPUT:

```
MAX total value65
```

11. Write a program for the fractional and dynamic knapsack problem.

DESCRIPTION:

- The fractional knapsack problem is also one of the techniques which are used to solve the knapsack problem. In fractional knapsack, the items are broken in order to maximize the profit. The problem in which we break the item is known as a Fractional knapsack problem.
- The 0/1 knapsack problem means that the items are either completely or no items are filled in a knapsack. For example, we have two items having weights 2kg and 3kg, respectively. If we pick the 2kg item then we cannot pick 1kg item from the 2kg item (item is not divisible); we have to pick the 2kg item completely. This is a 0/1 knapsack problem in which either we pick the item completely or we will pick that item. The 0/1 knapsack problem is solved by the dynamic programming.

CODE:

```
/*
    Harsh Kumar
    (2021UCA1829)
    DAA(Design and Analysis of Algorithms
*/
//FRactional Knapsack
#include <bits/stdc++.h>
#define ll long long int
using namespace std;

void swap(pair<pair<ll,ll>,float> *x, pair<pair<ll,ll>,float> *y){
    pair<pair<ll,ll>,float> temp=*x;
    *x=*y;
    *y=temp;
}
void quickSort(vector<pair<pair<ll,ll>,float>> &arr, int f,int l){
    if(f>=l)
    return;
    int pivot= arr[l].second;
    int i=f,j=l-1;
    while(i<=j){
        if(arr[i].second<pivot)
        i++;
        else if(arr[j].second>=pivot)
        j--;
        else
        swap(&arr[i],&arr[j]);
    }
    swap(&arr[i],&arr[l]);
```

```
    quickSort(arr,f,i-1);
    quickSort(arr,i+1,l);
}


double fractionalKnapsack(int capacity, vector<pair<pair<ll,ll>,float>> &arr,
int n)
{

    for(auto &i:arr){
        i.second=(float)i.first.first/(float)i.first.second;
    }
    quickSort(arr,0,arr.size()-1);
    reverse(arr.begin(),arr.end());
    double total_value = 0;
   for(auto &i:arr){
        if(capacity==0)
        {
            i.second=0;
        }
        else if(i.first.second<=capacity)
        {
            capacity-=i.first.second;
            total_value+=i.first.first;
            i.second=1;
        }
        else{
            i.second=(float)capacity/(float)i.first.second;
            capacity=0;
            total_value+=i.first.first*i.second;
        }

    }
    return total_value;
}

int main(){
     vector<pair<pair<ll,ll>,float>> knapsack
={make_pair(make_pair(25,3),0),make_pair(make_pair(20,2),0),make_pair(make_pai
r(15,1),0),make_pair(make_pair(40,4),0),make_pair(make_pair(50,5),0)};
    ll maxW=6;
    cout<<"MAX total value : "<<fractionalKnapsack(maxW,knapsack,5);
    cout<<endl;
    for(auto i:knapsack){
        cout<<"(value: "<<i.first.first<<", fraction:
"<<i.second<<")    ,    ";
    }
}
```

```
/*
    Harsh Kumar
```

```
    (2021UCA1829)
    DAA(Design and Analysis of Algorithms
*/
//Zero one knapsack
#include<bits/stdc++.h>
using namespace std;

int memo[1001][1001];
int knapsack(int W, int i, vector<int>& wt, vector<int>& val) {
    if (i == wt.size()) return 0;
    if (memo[i][W] == -1) {
        if (W < wt[i])
            memo[i][W] = knapsack(W, i + 1, wt, val);
        else
            memo[i][W] = max(val[i] + knapsack(W - wt[i], i + 1, wt, val),
knapsack(W, i + 1, wt, val));
    }
    return memo[i][W];
}

int main() {
    vector<int> val = {25, 20, 15, 40, 50};
    vector<int> wt = {3, 2, 1, 4, 5};
    int W = 6;
    memset(memo, -1, sizeof(memo));
    cout << knapsack(W, 0, wt, val);
    return 0;
}
```

OUTPUT:

```
MAX total value : 65
(value: 15, fraction: 1)    ,   (value: 40, fraction: 1)    ,   (value: 20, fraction: 0.5)
,   (value: 50, fraction: 0)    ,   (value: 25, fraction: 0)    ,
```

```
65
```

12. Write a program to Implement Minimum Spanning trees: Prim's algorithm and Kruskal's algorithm

DESCRIPTION:

- : Kruskal's algorithm to find the minimum cost spanning tree uses the greedy approach. This algorithm treats the graph as a forest and every node it has as an individual tree. A tree connects to another only and only if, it has the least cost among all available options and does not violate MST properties.
- Prim's algorithm to find minimum cost spanning tree (as Kruskal's algorithm) uses the greedy approach. Prim's algorithm shares a similarity with the shortest path first algorithms

CODE:

```
/*
    Harsh Kumar
    (2021UCA1829)
    DAA(Design and Analysis of Algorithms
*/
//Krukshal
#include<bits/stdc++.h>
using namespace std;

#define V 5
#define iPair pair<int, int>
int parent[V];


int find(int i)
{
    while (parent[i] != i)
        i = parent[i];
    return i;
}

void union1(int i, int j)
{
    int a = find(i);
    int b = find(j);
    parent[a] = b;
}

void KruskalMST(int cost[][V])
{
    int mincost = 0;
```

```
    for (int i = 0; i < V; i++)
        parent[i] = i;

    int edge_count = 0;
    while (edge_count < V - 1)
    {
        int min = INT_MAX, a = -1, b = -1;
        for (int i = 0; i < V; i++)
        {
            for (int j = 0; j < V; j++)
            {
                if (find(i) != find(j) && cost[i][j] < min)
                {
                    min = cost[i][j];
                    a = i, b = j;
                }
            }
        }
        union1(a, b);
        cout << "Edge " << edge_count++ << ": (" << a << ", " << b << ") cost:
" << min << "\n";
        mincost += min;
    }
    cout << "Minimum cost = " << mincost;
}

int find(vector<int> &parent, int u)
{
    while (u != parent[u])
        u = parent[u];
    return u;
}

int KruskalMST(vector<vector<int>>& points)
{
    int n = points.size(), mst_wt = 0, vert = 0;

    vector<int> parent(n);
    for (int i = 0; i < n; i++)
        parent[i] = i;

    vector<array<int, 3>> edges;
    for (int i = 0; i < n; i++)
        edges.push_back({points[i][2], points[i][0], points[i][1]});

    make_heap(edges.begin(), edges.end(), greater<array<int, 3>>());

    while (vert < n - 1 && !edges.empty())
    {
        pop_heap(edges.begin(), edges.end(), greater<array<int, 3>>());
        array<int, 3> cur = edges.back();
        int dist = cur[0], u = cur[1], v = cur[2];
```

```
        edges.pop_back();

        int set_u = find(parent, u), set_v = find(parent, v);
        if (set_u != set_v)
        {
            mst_wt += dist;
            cout << u << " - " << v << "\n";
            parent[set_u] = set_v;
        }
    }
    return mst_wt;
}

int main()
{
    int cost[][V] = {
        { INT_MAX, 2, INT_MAX, 6, INT_MAX },
        { 2, INT_MAX, 3, 8, 5 },
        { INT_MAX, 3, INT_MAX, INT_MAX, 7 },
        { 6, 8, INT_MAX, INT_MAX, 9 },
        { INT_MAX, 5, 7, 9, INT_MAX },
    };

    KruskalMST(cost);


    vector<vector<int>> points = {{0, 1, 4}, {0, 7, 8}, {1, 2, 8}, {1, 7, 11},
{2, 3, 7}, {2, 8, 2},
        {2, 5, 4}, {3, 4, 9}, {3, 5, 14}, {4, 5, 10}, {5, 6, 2}, {6, 7, 1},
        {6, 8, 6}, {7, 8, 7}
    };

    cout << "\n\nEdges of MST are:\n";
    int mst_wt = KruskalMST(points);

    cout << "Weight of MST is: " << mst_wt;

    return 0;
}
/*
    Harsh Kumar
    (2021UCA1829)
    DAA(Design and Analysis of Algorithms
*/
//Prim's
#include<bits/stdc++.h>
#define iPair pair<int, int>
#define V 9
using namespace std;

vector<iPair> graph[V];
```

```cpp
void addEdge(int u, int v, int wt)
{
    graph[u].push_back({v, wt});
    graph[v].push_back({u, wt});
}

int primMST()
{
    priority_queue<iPair, vector<iPair>, greater<iPair>> pq;
    int src = 0;
    vector<int> parent(V, -1);
    vector<int> inMST(V, false);
    vector<int> key(V, INT_MAX);

    pq.push({0, src});
    key[src] = 0;

    while (!pq.empty())
    {
        int u = pq.top().second;
        pq.pop();
        inMST[u] = true;

        for (auto i : graph[u])
        {
            int v = i.first;
            int wt = i.second;

            if (inMST[v] == false && key[v] > wt)
            {
                key[v] =  wt;
                pq.push({key[v], v});
                parent[v] = u;
            }
        }
    }
    for (int i = 1; i < V; i++)
        cout << parent[i] << "-" << i << "\n";
    return 0;
}

int main()
{
    addEdge(0, 1, 4);
    addEdge(0, 7, 8);
    addEdge(1, 7, 11);
    addEdge(2, 3, 7);
    addEdge(2, 8, 2);
    addEdge(2, 5, 4);
    addEdge(3, 4, 9);
    addEdge(3, 5, 14);
    addEdge(4, 5, 10);
```

```
    addEdge(5, 6, 2);
    addEdge(6, 7, 1);
    addEdge(6, 8, 6);
    addEdge(7, 8, 7);

    cout << "Edges of MST are:\n";
    int mst_wt = primMST();

    cout << "Weight of MST is: " << mst_wt;

    return 0;
}
```

OUTPUT:

```
Edge 0: (0, 1) cost: 2
Edge 1: (1, 2) cost: 3
Edge 2: (1, 4) cost: 5
Edge 3: (0, 3) cost: 6
Minimum cost = 16

Edges of MST are:
6 - 7
2 - 8
5 - 6
0 - 1
2 - 5
2 - 3
0 - 7
3 - 4
Weight of MST is: 37
```

```
Edges of MST are:
0-1
5-2
2-3
3-4
6-5
7-6
0-7
2-8
Weight of MST is: 0
```

13. Write a program for most efficient way to multiply matrices together.

DESCRIPTION:

- :Given the dimension of a sequence of matrices in an array arr[], where the dimension of the ith matrix is (arr[i-1] * arr[i]), the task is to find the most efficient way to multiply these matrices together such that the total number of element multiplications is minimum.

CODE:

```
/*
    Harsh Kumar
    (2021UCA1829)
    DAA(Design and Analysis of Algorithms
*/
//MCM
#include <bits/stdc++.h>
using namespace std;
int MCM(int i, int j,int arr[])
 {
     if(i>=j)return 0;

     int ans=INT_MAX;
     for(int k=i;k<=j-1;k++)
     {
         int tempAns = MCM(i,k,arr) + MCM(k+1,j,arr)
                       + arr[i-1]*arr[k]*arr[j];

         ans=min(ans,tempAns);
     }
     return ans;
 }

int main()
{
    int arr[] = { 1, 2, 3, 4, 3 };
    int N = sizeof(arr) / sizeof(arr[0]);

    cout << "Minimum number of multiplications is "
         << MCM(1, N-1, arr);
    return 0;
}
```

OUTPUT:

```
Minimum number of multiplications is 30
```

14. Write a program to implement Strassen's Matrix Multiplication and Matrix chain multiplication.

DESCRIPTION:

- strassen's Multiplying two matrices in minimum number of comparisons..
- It works on the seven equations
- M1:=(A+C)×(E+F)
- M2:=(B+D)×(G+H)
- M3:=(A−D)×(E+H)
- M4:=A×(F−H)
- M5:=(C+D)×(E)
- M6:=(A+B)×(H)
- M7:=D×(G−E)

CODE:

```
/*
    Harsh Kumar
    (2021UCA1829)
    DAA(Design and Analysis of Algorithms
*/
//MCM
#include <bits/stdc++.h>
using namespace std;
int MCM(int i, int j,int arr[])
 {
     if(i>=j)return 0;

     int ans=INT_MAX;
     for(int k=i;k<=j-1;k++)
     {
         int tempAns = MCM(i,k,arr) + MCM(k+1,j,arr)
                       + arr[i-1]*arr[k]*arr[j];

         ans=min(ans,tempAns);
     }
     return ans;
 }

int main()
{
    int arr[] = { 1, 2, 3, 4, 3 };
    int N = sizeof(arr) / sizeof(arr[0]);
```

```
    cout << "Minimum number of multiplications is "
         << MCM(1, N-1, arr);
    return 0;
}
```

```
/*
    Harsh Kumar
    (2021UCA1829)
    DAA(Design and Analysis of Algorithms
*/
//Strasses
#include <bits/stdc++.h>
using namespace std;

int nextpowerof2(int k)
{
    return pow(2, int(ceil(log2(k))));
}
void display(vector<vector<int>> &matrix, int m, int n)
{
    for (int i = 0; i < m; i++)
    {
        for (int j = 0; j < n; j++)
        {
            if (j != 0)
            {
                cout << "\t";
            }
            cout << matrix[i][j];
        }
        cout << endl;
    }
}

void add(vector<vector<int>> &A, vector<vector<int>> &B, vector<vector<int>>
&C, int size)
{
    int i, j;

    for (i = 0; i < size; i++)
    {
        for (j = 0; j < size; j++)
        {
            C[i][j] = A[i][j] + B[i][j];
        }
    }
}

void sub(vector<vector<int>> &A, vector<vector<int>> &B, vector<vector<int>>
&C, int size)
{
```

```
    int i, j;

    for (i = 0; i < size; i++)
    {
        for (j = 0; j < size; j++)
        {
            C[i][j] = A[i][j] - B[i][j];
        }
    }
}

void STRASSEN_algorithmA(vector<vector<int>> &A, vector<vector<int>> &B,
vector<vector<int>> &C, int size)
{
    if (size == 1)
    {
        C[0][0] = A[0][0] * B[0][0];
        return;
    }
    else
    {
        int new_size = size / 2;
        vector<int> z(new_size);
        vector<vector<int>>
            a11(new_size, z), a12(new_size, z), a21(new_size, z),
a22(new_size, z),
            b11(new_size, z), b12(new_size, z), b21(new_size, z),
b22(new_size, z),
            c11(new_size, z), c12(new_size, z), c21(new_size, z),
c22(new_size, z),
            p1(new_size, z), p2(new_size, z), p3(new_size, z), p4(new_size,
z),
            p5(new_size, z), p6(new_size, z), p7(new_size, z),
            aResult(new_size, z), bResult(new_size, z);

        int i, j;

        // dividing into sub-matrices:
        for (i = 0; i < new_size; i++)
        {
            for (j = 0; j < new_size; j++)
            {
                a11[i][j] = A[i][j];
                a12[i][j] = A[i][j + new_size];
                a21[i][j] = A[i + new_size][j];
                a22[i][j] = A[i + new_size][j + new_size];

                b11[i][j] = B[i][j];
                b12[i][j] = B[i][j + new_size];
                b21[i][j] = B[i + new_size][j];
                b22[i][j] = B[i + new_size][j + new_size];
            }
```

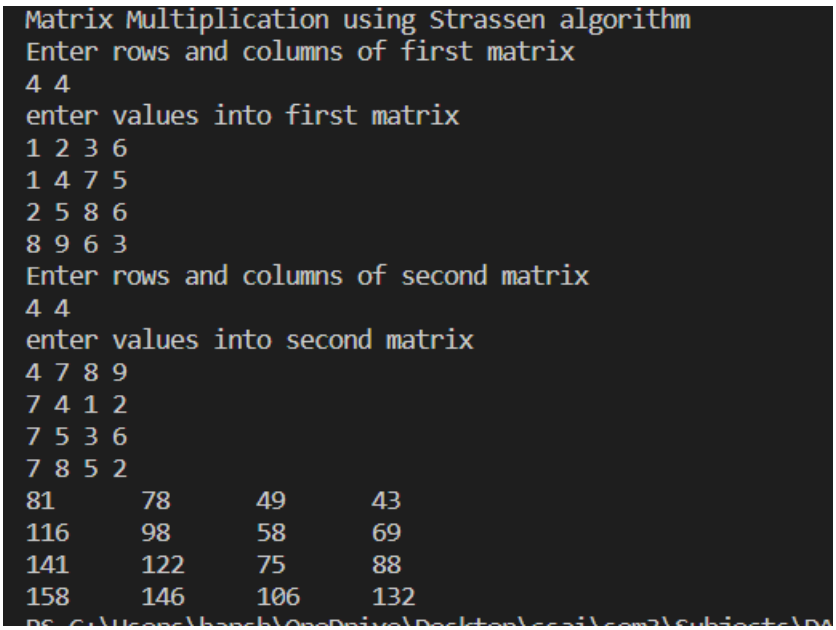
```
        }

        add(a11, a22, aResult, new_size);
        add(b11, b22, bResult, new_size);
        STRASSEN_algorithmA(aResult, bResult, p1, new_size);

        add(a21, a22, aResult, new_size);
        STRASSEN_algorithmA(aResult, b11, p2, new_size);

        sub(b12, b22, bResult, new_size);
        STRASSEN_algorithmA(a11, bResult, p3, new_size);

        sub(b21, b11, bResult, new_size);
        STRASSEN_algorithmA(a22, bResult, p4, new_size);

        add(a11, a12, aResult, new_size);
        STRASSEN_algorithmA(aResult, b22, p5, new_size);

        sub(a21, a11, aResult, new_size);
        add(b11, b12, bResult, new_size);
        STRASSEN_algorithmA(aResult, bResult, p6, new_size);

        sub(a12, a22, aResult, new_size);
        add(b21, b22, bResult, new_size);
        STRASSEN_algorithmA(aResult, bResult, p7, new_size);

        add(p3, p5, c12, new_size);
        add(p2, p4, c21, new_size);

        add(p1, p4, aResult, new_size);
        add(aResult, p7, bResult, new_size);
        sub(bResult, p5, c11, new_size);

        add(p1, p3, aResult, new_size);
        add(aResult, p6, bResult, new_size);
        sub(bResult, p2, c22, new_size);

        // Grouping results obtained
        for (i = 0; i < new_size; i++)
        {
            for (j = 0; j < new_size; j++)
            {
                C[i][j] = c11[i][j];
                C[i][j + new_size] = c12[i][j];
                C[i + new_size][j] = c21[i][j];
                C[i + new_size][j + new_size] = c22[i][j];
            }
        }
    }
}
void STRASSEN_algorithm(vector<vector<int>> &A, vector<vector<int>> &B, int m,
int n, int a, int b)
```

```
{

    int k = max({m, n, a, b});

    int s = nextpowerof2(k);

    vector<int> z(s);
    vector<vector<int>> Aa(s, z), Bb(s, z), Cc(s, z);

    for (unsigned int i = 0; i < m; i++)
    {
        for (unsigned int j = 0; j < n; j++)
        {
            Aa[i][j] = A[i][j];
        }
    }
    for (unsigned int i = 0; i < a; i++)
    {
        for (unsigned int j = 0; j < b; j++)
        {
            Bb[i][j] = B[i][j];
        }
    }
    STRASSEN_algorithmA(Aa, Bb, Cc, s);
    vector<int> temp1(b);
    vector<vector<int>> C(m, temp1);
    for (unsigned int i = 0; i < m; i++)
    {
        for (unsigned int j = 0; j < b; j++)
        {
            C[i][j] = Cc[i][j];
        }
    }
    display(C, m, b);
}

bool check(int n, int a)
{
    if (n == a)
        return true;
    else
        return false;
}

int main()
{
    int m, n, a, b;
    cout << "Matrix Multiplication using Strassen algorithm" << endl;
    cout << "Enter rows and columns of first matrix" << endl;
    cin >> m >> n;
    cout << "enter values into first matrix" << endl;
    vector<vector<int>> A;
```

```
    // first matrix
    for (int i = 0; i < m; i++)
    {
        vector<int> temp;
        for (int j = 0; j < n; j++)
        {
            int i;
            cin >> i;
            temp.push_back(i);
        }
        A.push_back(temp);
    }
    cout << "Enter rows and columns of second matrix" << endl;
    cin >> a >> b;
    cout << "enter values into second matrix" << endl;
    vector<vector<int>> B;
    // second matrix
    for (int i = 0; i < a; i++)
    {
        vector<int> temp;
        for (int j = 0; j < b; j++)
        {
            int i;
            cin >> i;
            temp.push_back(i);
        }
        B.push_back(temp);
    }

    bool k = check(n, a);
    if (k)
    {
        STRASSEN_algorithm(A, B, m, n, a, b);
    }
    else
    {
        cout << "martrix multiplication not possible";
    }
    return 0;
}
```

OUTPUT:

```
Minimum number of multiplications is 30
```


Matrix Multiplication using Strassen algorithm
Enter rows and columns of first matrix
4 4
enter values into first matrix
1 2 3 6
1 4 7 5
2 5 8 6
8 9 6 3
Enter rows and columns of second matrix
4 4
enter values into second matrix
4 7 8 9
7 4 1 2
7 5 3 6
7 8 5 2
81 78 49 43
116 98 58 69
141 122 75 88
158 146 106 132

15. Write a program to implement Longest Common Subsequence.

DESCRIPTION:

- In a set of sequences are given, the longest common subsequence problem is to find a common subsequence of all the sequences that is of maximal length.
- The longest common subsequence problem is a classic computer science problem, the basis of data comparison programs such as the diff-utility, and has applications in bioinformatics. It is also widely used by revision control systems, such as SVN and Git, for reconciling multiple changes made to a revision-controlled collection of files.

CODE:

```
/*
    Harsh Kumar
    (2021UCA1829)
    DAA(Design and Analysis of Algorithms
*/
#include <bits/stdc++.h>
using namespace std;

int longestCommonSubsequence(string x, string y) {
        int n=x.size(),m=y.size();
        string s;
        int dp[n+1][m+1];
        for(int i=0;i<=n;i++)dp[i][0]=0;
        for(int i=0;i<=m;i++)dp[0][i]=0;
        for(int i=1;i<=n;i++){
            for(int j=1;j<=m;j++){
                if(x[i-1]==y[j-1])dp[i][j]=1+dp[i-1][j-1];
                else {
                    dp[i][j]=max(dp[i][j-1],dp[i-1][j]);
                }
            }
        }
        int i=n,j=m;
        while(i>0 && j>0){
            if(x[i-1]==y[j-1]){
                s+=x[i-1];
                i--;j--;
            }
            else{
                if(dp[i][j-1]>dp[i-1][j])j--;
                else i--;
            }
        }
```

```
        reverse (s.begin(),s.end());
        cout<<s;
        return dp[n][m];
    }

int main()
{
    string A= "DUMBITTTER";
    string B = "UMBRILLRR";
    cout << "\nLength of Longest Subsequence is " <<
longestCommonSubsequence(A,B);

    return 0;
}
```

OUTPUT:

```
UMBIR
Length of Longest Subsequence is 5
```

---

16. Write a program to Implement Travelling Salesman Problem.

DESCRIPTION:

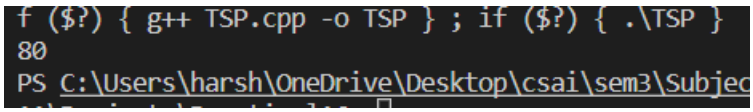
- Travelling Salesman Problem is a problem in which we need to find the shortest route that covers each city exactly once and returns to the starting point. Hamiltonian Cycle is another problem in Java that is mostly similar to Travelling Salesman Problem.
- The main difference between TSP and the Hamiltonian cycle is that in Hamiltonian Cycle, we are not sure whether a tour that visits each city exactly once exists or not, and we have to determine it. In the Travelling Salesman Problem, a Hamiltonian cycle is always present because the graph is complete, and the problem is to find a Hamiltonian cycle with minimum weight.

CODE:

```
/*
    Harsh Kumar
    (2021UCA1829)
    DAA(Design and Analysis of Algorithms
*/
#include <bits/stdc++.h>
using namespace std;
#define V 4
int travllingSalesmanProblem(int graph[][V], int s)
{
    vector<int> vertex;
    for (int i = 0; i < V; i++)
        if (i != s)
            vertex.push_back(i);

    int min_path = INT_MAX;
    do {
        int current_pathweight = 0;
        int k = s;
        for (int i = 0; i < vertex.size(); i++) {
            current_pathweight += graph[k][vertex[i]];
            k = vertex[i];
        }
        current_pathweight += graph[k][s];
        min_path = min(min_path, current_pathweight);

    } while (
        next_permutation(vertex.begin(), vertex.end()));
```

```
    return min_path;
}
int main()
{
    int graph[][V] = { { 0, 10, 15, 20 },
                       { 10, 0, 35, 25 },
                       { 15, 35, 0, 30 },
                       { 20, 25, 30, 0 } };
    int s = 0;
    cout << travllingSalesmanProblem(graph, s) << endl;
}
```

OUTPUT:


f ($?) { g++ TSP.cpp -o TSP } ; if ($?) { .\TSP }
80
PS C:\Users\harsh\OneDrive\Desktop\csai\sem3\Subjec