/\*

    Program-1

    Implement  Queue  using  two  stacks.  (Note:  the  functions  such  as  Enqueue,

Dequeue and Display need to be implemented.)

    @ Prajwal Sundar

\*/

#include "bits/stdc++.h"

using namespace std;

class Queue

{

    public:

    stack<int> S1; // first stack

    stack<int> S2; // second stack

    // Enqueue a number into the queue

    void enqueue(int n)

    {

        S1.push(n); // push element onto stack

        cout << "Enqueue " << n << " operation : SUCCESS";

    }

    // Dequeue the frontmost element from the queue

    void dequeue()

    {

        if (S1.empty())

        {

            cout << "Error : QUEUE UNDERFLOW";

            return;

        }

        while (!S1.empty())

        {

            int top = S1.top();

            S2.push(top);

            S1.pop();

        }

        S2.pop(); // pop topmost

        cout << "Dequeue : SUCCESS";

        while (!S2.empty())

        {

            int top = S2.top();

            S1.push(top);

            S2.pop();

        }

    }

    // Display the queue

    void display()

    {

        cout << "Your Queue : ";

        if (S1.empty())

        {

            cout << "EMPTY";

            return;

        }

        while (!S1.empty())

        {

            int top = S1.top();

            S2.push(top);

            S1.pop();

        }

        while (!S2.empty())

        {

            int top = S2.top();

            cout << top << " ";

            S1.push(top);

            S2.pop();

        }

    }

};

// Main Function

int main()

{

    cout << "Welcome to C++ Queue using Stacks !" << endl;

    cout << "'E' stands for enqueue, 'D' for dequeue, 'd' for display and 'e' for exit." << endl << endl;

    bool flag = true;

    Queue Q; // queue object

    while (flag)

    {

        int n; // number

        char ch; // choice

        cout << "Enter your choice : ";

        cin >> ch;

        switch(ch)

        {

            case 'E': // enqueue

                cout << "Enter the element you wish to enqueue : ";

                cin >> n;

                Q.enqueue(n); // perform enqueue

                break;

            case 'D': // dequeue

                Q.dequeue();

                break;

            case 'd': // display

                Q.display();

                break;

            case 'e': // exit

                flag = false;

                cout << "Program Execution Terminated.";

                break;

            default:

                cout << "Invalid Choice.";

        }

        cout << endl << endl;

    }

    cout << "Thank you for using C++ Queue using Stacks. Bye Bye !";

}

/\*

    Program-2

    Implement  circular  descending  priority  queue.  (Note:  the  functions  such  as

Enqueue, Dequeue and Display need to be implemented.)

    @ Prajwal Sundar

\*/

#include "bits/stdc++.h"

using namespace std;

// Circular Descending Order Priority Queue

class CDPQ

{

    public:

    int C; // capacity

    int F, R; // front and rear pointers

    int \* Q; // queue

    CDPQ(int n) // constructor

    {

        C = n; // initialize capacity

        F = R = -1; // initialize pointers

        Q = new int [C]; // initialize queue

    }

    void enqueue(int n) // enqueue an element n

    {

        if (((F == 0) && (R == C-1)) || (R+1 == F)) // overflow

        {

            cout << "Error : QUEUE OVERFLOW";

            return;

        }

        else if ((F == -1) && (R == -1)) // first element enqueue

            F = R = 0;

        else if (R == C-1) // circle cross enqueue

            R = 0;

        else // normal enqueue

            R++;

        Q[R] = n; // enqueue

        cout << "Enqueue " << n << " : SUCCESS";

        sort();

    }

    void dequeue() // dequeue frontmost element

    {

        if ((F == -1) && (R == -1))

        {

            cout << "Error : QUEUE UNDERFLOW";

            return;

        }

        cout << "Dequeue " << Q[F] << " : SUCCESS";

        if (F == R) // last element dequeue

            F = R = -1;

        else if (F == C-1) // circle cross dequeue

            F = 0;

        else // normal dequeue

            F++;

    }

    void display()

    {

        cout << "QUEUE : ";

        if ((F == -1) && (R == -1)) // empty

            cout << "EMPTY";

        else if (F <= R) // normal display

            for (int i = F; i <= R; i++) cout << Q[i] << " ";

        else // circle cross display

        {

            for (int i = F; i < C; i++) cout << Q[i] << " ";

            for (int i = 0; i <= R; i++) cout << Q[i] << " ";

        }

    }

    void sort() // sort in descending order using insertion sort

    {

        bool flag = (F <= R);

        if (!flag) R += C;

        int P = R, elem = Q[R%C];

        while ((P > F) && (Q[(P-1)%C] < Q[P%C]))

        {

            swap(Q[P%C], Q[(P-1)%C]); // swap

            P--; // decrement pointer

        }

        Q[P%C] = elem;

        if (!flag) R -= C;

    }

};

// Main Function

int main()

{

    cout << "Welcome to C++ Circular Descending Order Priority Queue !" << endl;

    cout << "'E' stands for enqueue, 'D' for dequeue, 'd' for display and 'e' for exit." << endl << endl;

    int n; // capacity

    cout << "Enter the maximum capcity of your queue : ";

    cin >> n;

    cout << endl;

    bool flag = true;

    CDPQ Q(n); // queue object

    while (flag)

    {

        int n; // number

        char ch; // choice

        cout << "Enter your choice : ";

        cin >> ch;

        switch(ch)

        {

            case 'E': // enqueue

                cout << "Enter the element you wish to enqueue : ";

                cin >> n;

                Q.enqueue(n); // perform enqueue

                break;

            case 'D': // dequeue

                Q.dequeue();

                break;

            case 'd': // display

                Q.display();

                break;

            case 'e': // exit

                flag = false;

                cout << "Program Execution Terminated.";

                break;

            default:

                cout << "Invalid Choice.";

        }

        cout << endl << endl;

    }

    cout << "Thank you for using C++ Circular Descending Order Priority Queue. Bye Bye !";

}

/\*

    Program-3

    Given two unordered circular doubly linked lists, write a program for the printing

common elements of them.

    @ Prajwal Sundar

\*/

#include "bits/stdc++.h"

using namespace std;

// Structure of a Node

struct Node

{

    int data; // data

    struct Node \* prev; // address of previous node

    struct Node \* next; // address of next node

    Node(int n) // constructor

    {

        data = n; // set data

        prev = next = NULL;

    }

};

// Get a list as user input

struct Node \* input()

{

    int n; // size

    cout << "Enter the list size : ";

    cin >> n;

    struct Node \* head = NULL; // head pointer

    struct Node \* ptr = head; // pointer

    cout << "Enter your list : ";

    for (int i = 0; i < n; i++)

    {

        int tmp; cin >> tmp; // get the number

        struct Node \* node = new Node(tmp); // node

        if (i == 0) // first number

        {

            head = node; // set head

            head->next = head->prev = head; // set links

            ptr = head;

        }

        else // general case

        {

            ptr->next = node;

            node->prev = ptr; // doubly links

            node->next = head;

            head->prev = node; // circular links

            ptr = node; // update last node

        }

    }

    return head;

}

// Main Function

int main()

{

    cout << "Welcome to C++ Circular Doubly Linked Lists !" << endl << endl;

    cout << "LIST-I :" << endl;

    struct Node \* headA = input();

    cout << endl << "LIST-II :" << endl;

    struct Node \* headB = input();

    struct Node \* ptrA = headA;

    struct Node \* ptrB = headB; // pointers to lists

    int c = 0; // number of common elements

    cout << endl << "COMMON ELEMENTS : " << endl;

    do

    {

        do

        {

            if (ptrA->data == ptrB->data)

            {

                c++;

                cout << ptrA->data << " ";

            }

            ptrB = ptrB->next; // increment pointerB

        } while (ptrB != headB);

        ptrA = ptrA->next; // increment pointerA

    } while (ptrA != headA);

    if (!c) cout << "NONE";

    cout << endl << endl << "Thank you for using C++ Circular Doubly Linked Lists. Bye Bye !";

}

/\*

    Program-4

    Given a singly linked list, write a program to find

        (i) the last element from the beginning whose n%k == 0,

        (ii)  the first from the end whose n%k == 0,

    where n is the number of  elements in the list and k is an integer constant. For example,

    if n = 19 and k = 3 then  (i) 18th node should be returned. (ii) 16th node should be returned.

    @ Prajwal Sundar

\*/

#include "bits/stdc++.h"

using namespace std;

// Structure of a Linked List Node

struct Node

{

    int data; // data

    struct Node \* next; // address to next node

    Node(int n) // constructor

    {

        data = n; // set data

        next = NULL; // set next pointer as NULL

    }

};

// Reverse a Linked List

struct Node \* rev(struct Node \* head)

{

    if (!head) return NULL;

    struct Node \* prev = NULL;

    struct Node \* curr = head;

    struct Node \* next = NULL; // 3 pointer approach

    while (curr)

    {

        next = curr->next;

        curr->next = prev; // reverse link

        prev = curr;

        curr = next; // update pointers

    }

    return prev; // new head

}

// Main Function

int main()

{

    cout << "Welcome to C++ Singly Linked Lists !" << endl << endl;

    int n; // number of elements

    cout << "Enter the number of elements in your linked list : ";

    cin >> n;

    struct Node \* head = NULL; // head of linked list

    struct Node \* ptr = head; // pointer to head

    cout << "Enter your linked list : ";

    for (int i = 0; i < n; i++)

    {

        int tmp; cin >> tmp;

        struct Node \* node = new Node(tmp);

        if (ptr)

        {

            ptr->next = node;

            ptr = node; // update pointer

        }

        else

        {

            head = node; // set head

            ptr = head; // set pointer

        }

    }

    int k; // divisor

    cout << "Enter k : ";

    cin >> k;

    head = rev(head); // reverse the linked list

    bool flag = false;

    ptr = head;

    while (ptr)

    {

        if (!(ptr->data % k)) // remainder 0 obtained

        {

            flag = true;

            cout << "Element " << ptr->data << " satisfies n % k = 0";

            break;

        }

        ptr = ptr->next; // move forward

    }

    if (!flag) cout << "No element in your linked list satisfies the given condition.";

    cout << endl << endl << "Thank you for using C++ Singly Linked Lists. Bye Bye !";

}

/\*

    Program-5

    Given a circular linked list with even and odd numbers, write a program to  make

changes to the list in such a way that all even numbers appear at the beginning.

    @ Prajwal Sundar

\*/

#include "bits/stdc++.h"

using namespace std;

// Structure of a Linked List Node

struct Node

{

    int data; // data

    struct Node \* next; // link to next node

    Node(int n) // constructor

    {

        data = n; // set data

        next = NULL;

    }

};

// Modify list such that all even numbers appear first

struct Node \* modify(struct Node \* head)

{

    struct Node \* odd = NULL;

    struct Node \* oddPtr = odd; // odd numbers

    struct Node \* even = NULL;

    struct Node \* evenPtr = even; // even numbers

    struct Node \* ptr = head;

    do

    {

        if (ptr->data % 2) // odd number encountered

        {

            if (odd)

            {

                oddPtr->next = ptr;

                oddPtr = ptr; // increment odd pointer

            }

            else // first odd number

                odd = oddPtr = ptr;

        }

        else // even number encountered

        {

            if (even)

            {

                evenPtr->next = ptr;

                evenPtr = ptr; // increment even pointer

            }

            else // first even number

                even = evenPtr = ptr;

        }

        ptr = ptr->next;

    } while (ptr != head);

    if (even && odd) // at least one of both odd and even exist

    {

        evenPtr->next = odd;

        oddPtr->next = even;

        return even; // even comes first

    }

    else if (even) // only even numbers exist

    {

        evenPtr->next = even; // circular link

        return even;

    }

    else if (odd) // only odd numbers exist

    {

        oddPtr->next = odd; // circular link

        return odd;

    }

    else // no numbers at all

        return NULL;

}

// Display Circular Linked List

void display(struct Node \* head)

{

    if (!head)

    {

        cout << "EMPTY";

        return;

    }

    struct Node \* ptr = head;

    do

    {

        cout << ptr->data << " ";

        ptr = ptr->next;

    } while (ptr != head);

}

// Main Function

int main()

{

    cout << "Welcome to C++ Circular Linked Lists !" << endl << endl;

    int n; // size of linked list

    cout << "Enter the number of elements in your linked list : ";

    cin >> n;

    struct Node \* head = NULL; // head of linked list

    struct Node \* ptr = head; // pointer

    cout << "Enter your list : ";

    for (int i = 0; i < n; i++)

    {

        int tmp; cin >> tmp;

        struct Node \* node = new Node(tmp); // node

        if (ptr)

        {

            ptr->next = node; // insert

            ptr = node; // move pointer

        }

        else

        {

            head = node; // set head

            ptr = head; // set pointer

        }

    }

    ptr->next = head; // establish circular link

    head = modify(head); // perform modifications

    cout << "Modified List : ";

    display(head);

    cout << endl << endl << "Thank you for using C++ Circular Linked Lists. Bye Bye !";

}

/\*

    Program-6

    Given a BST and two integers (minimum and maximum integers) as parameters,

write a program to remove (prune) elements that are not within that range.

    @ Prajwal Sundar

\*/

#include "bits/stdc++.h"

using namespace std;

// Structure of a Binary Tree Node

struct Node

{

    int data; // data

    struct Node \* left; // address of left child

    struct Node \* right; // address of right child

    Node(int n) // constructor

    {

        data = n;

        left = right = NULL;

    }

};

struct Node \* form()

{

    int n; cin >> n;

    struct Node \* root = new Node(n);

    int l; // left child existence

    cout << "Does " << n << " have a left child ? Enter 0/1 : ";

    cin >> l;

    if (l)

    {

        cout << "Enter the left child of " << n << " : ";

        root->left = form(); // left subtree

    }

    int r; // right child existence

    cout << "Does " << n << " have a right child ? Enter 0/1 : ";

    cin >> r;

    if (r)

    {

        cout << "Enter the right child of " << n << " : ";

        root->right = form(); // right subtree

    }

    return root;

}

// Prune a given BST

struct Node \* prune(struct Node \* root, int L, int U)

{

    if (!root) return NULL;

    root->left = prune(root->left, L, U); // prune left subtree

    root->right = prune(root->right, L, U); // prune right subtree

    // Now come to root

    if (root->data < L) // lesser than minimum element

    {

        struct Node \* right = root->right;

        delete root;

        return right;

    }

    else if (root->data > U) // greater than maximum element

    {

        struct Node \* left = root->left;

        delete root;

        return left;

    }

    else // in range

        return root;

}

// Binary Tree Display Function

void display(struct Node \* root)

{

    if (!root) cout << "N ";

    else

    {

        cout << root->data << " "; // root

        display(root->left); // left

        display(root->right); // right

    }

}

// Main Function

int main()

{

    cout << "Welcome to C++ Binary Tree Pruner !" << endl << endl;

    cout << "Enter root value : ";

    struct Node \* root = form(); // get binary tree as user input

    int L, U; // lower and upper limits

    cout << endl << "Enter the range you wish to retain : ";

    cin >> L >> U;

    root = prune(root, L, U); // prune BST

    cout << "Pruned BST : ";

    display(root);

    cout << endl << endl << "Thank you for using C++ Binary Tree Pruner. Bye Bye !";

}

/\*

    Program-7

    Give an algorithm for checking the existence of path with given sum. That means,

given a sum, check whether there exists a path from root to any of the nodes.

    @ Prajwal Sundar

\*/

#include "bits/stdc++.h"

using namespace std;

// Structure of a Binary Tree Node

struct Node

{

    int data; // data

    struct Node \* left; // address of left child

    struct Node \* right; // address of right child

    Node(int n) // constructor

    {

        data = n;

        left = right = NULL;

    }

};

void sum(struct Node \* root, int s, string str, int \* c)

{

    if (!root) return;

    s -= root->data; // reduce sum

    if (!s) // sum obtained

    {

        cout << "Sum Obtained in Path : " << str << endl;

        (\*c)++; // increment count variable

    }

    sum(root->left, s, str+"L", c);

    sum(root->right, s, str+"R", c);

}

struct Node \* form()

{

    int n; cin >> n;

    struct Node \* root = new Node(n);

    int l; // left child existence

    cout << "Does " << n << " have a left child ? Enter 0/1 : ";

    cin >> l;

    if (l)

    {

        cout << "Enter the left child of " << n << " : ";

        root->left = form(); // left subtree

    }

    int r; // right child existence

    cout << "Does " << n << " have a right child ? Enter 0/1 : ";

    cin >> r;

    if (r)

    {

        cout << "Enter the right child of " << n << " : ";

        root->right = form(); // right subtree

    }

    return root;

}

int main()

{

    cout << "Welcome to C++ Binary Tree Path Sum Checker !" << endl << endl;

    cout << "Enter the root : ";

    struct Node \* root = form();

    int s; // sum

    cout << endl << "Enter the sum you wish to obtain : ";

    cin >> s;

    int c = 0; // count variable

    sum(root, s, "", &c);

    if (!c) cout << "Path with given sum does not exist.";

    else cout << "Therefore, number of paths with given sum = " << c;

    cout << endl << endl << "Thank you for using C++ Binary Tree Path Sum Checker. Bye Bye !";

}

/\*

    Program-8

    Given a tree with a special property where leaves are represented with ‘L’ and

internal node with ‘I’. Also, assume that each node has either 0 or 2 children. Given

preorder traversal of this tree, write a program to construct the tree and display it

in the tree format as shown below.

    @ Prajwal Sundar

\*/

#include "bits/stdc++.h"

using namespace std;

// Structure of a Binary Tree Node

struct Node

{

    string data; // data

    struct Node \* left; // address of left child

    struct Node \* right; // address of right child

    Node(string ch) // constructor

    {

        data = ch; // set data

        left = right = NULL;

    }

};

// Return tree of a given string

struct Node \* form(string str)

{

    struct Node \* root = new Node("-");

    bool mode = false; // left = false, right = true

    stack<struct Node \*> S; // stack

    S.push(root);

    int len = str.size();

    for (int i = 0; i < len; i++)

    {

        if (str[i] == 'I')

        {

            struct Node \* node = new Node("I");

            if (!mode) S.top()->left = node;

            else S.top()->right = node; // push based on mode

            S.push(node); // push newly formed internal node

            mode = false; // reset mode

        }

        else if (str[i] == 'L')

        {

            if (!mode)

            {

                S.top()->left = new Node("L");

                mode = true; // change mode to right

            }

            else

            {

                S.top()->right = new Node("L");

                S.pop(); // pop topmost node

            }

        }

    }

    return root->left;

}

// Get height of the binary tree

int height(struct Node \* root)

{

    if (!root) return -1;

    else return 1 + max(height(root->left), height(root->right));

}

// Fill Matrix Function

void fill(string \*\* M, struct Node \* root, int r, int c, int h)

{

    M[r][c] = root->data;

    int n = pow(2, h-1);

    if (root->left) // recursion in left-subtree

    {

        int i, j, x;

        for (i = r+1, j = c-1, x = 0; x < n; i++, j--, x++) M[i][j] = "/";

        fill(M, root->left, i, j, h-1);

    }

    if (root->right) // recursion in right-subtree

    {

        int i, j, x;

        for (i = r+1, j = c+1, x = 0; x < n; i++, j++, x++) M[i][j] = "\\";

        fill(M, root->right, i, j, h-1);

    }

}

// Display a tree

void display(struct Node \* root)

{

    int h = height(root);

    int r = pow(2, h) + h; // number of rows in matrix

    int c = pow(2, h+1) + (2\*h) - 1; // number of columns in matrix

    string \*\* M = new string \* [r]; // string matrix to store tree

    for (int i = 0; i < r; i++)

    {

        M[i] = new string [c];

        for (int j = 0; j < c; j++) M[i][j] = " "; // empty string by default

    }

    fill(M, root, 0, r-1, h); // fill matrix function

    for (int i = 0; i < r; i++)

    {

        for (int j = 0; j < c; j++) cout << M[i][j];

        cout << endl; // next line

    }

}

// Main Function

int main()

{

    cout << "Welcome to C++ Trees !" << endl << endl;

    string str;

    cout << "Enter your pre-order string : ";

    cin >> str;

    struct Node \* root = form(str); // form tree

    cout << "Your Tree is : " << endl;

    display(root);

    cout << endl << "Thank you for using C++ Trees. Bye Bye !";

}

/\*

    Program-9

    Write a program for finding the maximum-weight spanning tree in a graph.

    @ Prajwal Sundar

\*/

#include "bits/stdc++.h"

using namespace std;

struct Edge

{

    int x, y, w;

};

bool comp(struct Edge A, struct Edge B)

{

    return (A.w > B.w);

}

class Graph

{

    public:

    int v; // number of vertices

    int e; // number of edges

    int W; // total weight of graph

    int \*\* M; // adjacency matrix

    vector<struct Edge> E; // vector of edges

    Graph(int n) // constructor

    {

        v = n; // initialize number of vertices

        e = 0; // initialize number of edges

        W = 0; // initialize total weight of graph

        M = new int \* [v]; // initialize adjacency matrix

        for (int i = 0; i < v; i++)

        {

            M[i] = new int [v];

            for (int j = 0; j < v; j++) M[i][j] = 0;

        }

    }

    void addEdge(int x, int y, int w) // add edge with given weight

    {

        M[x][y] = M[y][x] = w; // set edge

        e++; // increment number of edges

        W += w; // add weight of new edge

        E.push\_back({x, y, w});

    }

    void displayGraph() // display all graph edges

    {

        for (int i = 0; i < e; i++)

            cout << "Edge of weight " << E[i].w << " between (" << E[i].x << "," << E[i].y << ")" << endl;

    }

};

// Absolute Parents Function

int getAbsParent(int \* parents, int v)

{

    if (parents[v] < 0) return v;

    else return getAbsParent(parents, parents[v]);

}

// Return MST of maximum weight

Graph Kruskal(Graph G)

{

    sort(G.E.begin(), G.E.end(), comp); // arrange edges in descending order of weight

    Graph MST(G.v); // maximum spanning tree

    int parents[G.v]; // array pointing to parents

    memset(parents, -1, sizeof(parents));

    for (int i = 0; (i < G.e && MST.e < G.v-1); i++)

    {

        int xp = getAbsParent(parents, G.E[i].x);

        int yp = getAbsParent(parents, G.E[i].y);

        if (xp == yp) continue; // cycle formation

        else if (parents[xp] < parents[yp]) // make x the parent of y

        {

            parents[xp] += parents[yp];

            parents[yp] = xp;

        }

        else // make y the parent of x

        {

            parents[yp] += parents[xp];

            parents[xp] = yp;

        }

        MST.addEdge(G.E[i].x, G.E[i].y, G.E[i].w); // add edge to graph

    }

    return MST;

}

// Main Function

int main()

{

    cout << "Welcome to C++ Maximum Spanning Trees !" << endl << endl;

    int v; // number of vertices

    cout << "Enter the number of vertices : ";

    cin >> v;

    int e; // number of edges

    cout << "Enter the number of edges : ";

    cin >> e;

    Graph G(v);

    cout << "Enter each edge as (vertex1 vertex2 weight) :-" << endl;

    for (int i = 0; i < e; i++)

    {

        int x, y, w;

        cin >> x >> y >> w;

        G.addEdge(x, y, w); // add edge to graph object

    }

    Graph MST = Kruskal(G); // get maximum spanning tree using Kruskal's algorithm

    cout << endl << "The maximum Spanning Tree has the following edges :" << endl;

    MST.displayGraph();

    cout << "Weight of MST : " << MST.W;

    cout << endl << endl << "Thank you for using C++ Maximum Spanning Trees. Bye Bye !";

}

/\*

    Program-10

    Write a program to return the reverse of the directed graph (each edge from v to w

is replaced by an edge from w to v).

    @ Prajwal Sundar

\*/

#include "bits/stdc++.h"

using namespace std;

class Graph

{

    public:

    int v; // number of vertices

    int e; // number of edges

    int \*\* M; // adjacency matrix

    Graph(int n) // constructor

    {

        v = n; // initialize number of vertices

        e = 0; // initialize number of edges

        M = new int \* [v];

        for (int i = 0; i < v; i++)

        {

            M[i] = new int [v];

            for (int j = 0; j < v; j++) M[i][j] = 0; // no edge present initially

        }

    }

    void addEdge(int x, int y)

    {

        M[x][y] = 1; // add edge to graph

    }

    void printEdges() // print all edges present in graph

    {

        for (int i = 0; i < v; i++)

            for (int j = 0; j < v; j++)

                if (M[i][j]) cout << "(" << i << "," << j << ") ";

    }

    void transpose() // reverse the graph

    {

        for (int i = 0; i < v; i++)

            for (int j = 0; j < i; j++)

                swap(M[i][j], M[j][i]);

    }

};

// Main Function

int main()

{

    cout << "Welcome to C++ Graph Reverser !" << endl << endl;

    int v; // vertices

    cout << "Enter the number of vertices : ";

    cin >> v;

    int e; // edges

    cout << "Enter the number of edges : ";

    cin >> e;

    Graph G(v); // graph

    cout << "Enter each edge as (source destination) :-" << endl;

    for (int i = 0; i < e; i++)

    {

        int x, y;

        cin >> x >> y;

        G.addEdge(x, y);

    }

    G.transpose(); // reverse direction of all edges

    cout << endl << "Reversed Graph :" << endl;

    G.printEdges();

    cout << endl << endl << "Thank you for using C++ Graph Reverser. Bye Bye !";

}

/\*

    Program-11

    Write a program to implement Warshall’s algorithm on weighted as well as unweighted graphs.

    @ Prajwal Sundar

\*/

#include "bits/stdc++.h"

using namespace std;

// Floyd-Warshall Algorithm on given Matrix

int \*\* warshall(int \*\* M, int n)

{

    for (int k = 0; k < n; k++)

    {

        int \*\* T = new int \* [n];

        for (int i = 0; i < n; i++)

        {

            T[i] = new int [n];

            for (int j = 0; j < n; j++)

                T[i][j] = min(M[i][j], M[i][k] + M[k][j]);

        }

        M = T;

    }

    return M;

}

// Main Function

int main()

{

    cout << "Welcome to C++ Floyd Warshall Algorithm !" << endl << endl;

    int v; // number of vertices

    cout << "Enter the number of vertices : ";

    cin >> v;

    int e; // number of edges

    cout << "Enter the number of edges : ";

    cin >> e;

    int \*\* M = new int \* [v]; // adjacency matrix

    for (int i = 0; i < v; i++)

    {

        M[i] = new int [v];

        for (int j = 0; j < v; j++)

        {

            if (i == j) M[i][j] = 0;

            else M[i][j] = INT\_MAX;

        }

    }

    cout << "Enter each edge as (source destination weight) :-" << endl;

    int maxW = INT\_MIN;

    for (int i = 0; i < e; i++)

    {

        int x, y, w; cin >> x >> y >> w;

        M[x][y] = w; // set value in adjacency matrix

        maxW = max(maxW, w);

    }

    for (int i = 0; i < v; i++)

        for (int j = 0; j < v; j++)

            if (M[i][j] == INT\_MAX) M[i][j] = maxW+1;

    M = warshall(M, v); // apply shortest path algorithm

    cout << endl << "All Pair Shortest Paths are :-" << endl;

    cout << "\t";

    for (int i = 0; i < v; i++) cout << "V" << i << "\t";

    cout << endl;

    for (int i = 0; i < v; i++)

    {

        cout << "V" << i << "\t";

        for (int j = 0; j < v; j++)

        {

            if (M[i][j] == maxW+1) cout << "-" << "\t";

            else cout << M[i][j] << "\t";

        }

        cout << endl;

    }

    cout << endl << "Thank you for using C++ Floyd Warshall Algorithm. Bye Bye !";

}

/\*

    Program-12

    Given an array A[] consisting of 0’s, 1’s and 2’s, Write a program to sort this array

A[] using Quick Sort.

    @ Prajwal Sundar

\*/

#include "bits/stdc++.h"

using namespace std;

// Partition Function

int partition(int \* A, int L, int U)

{

    int i = L-1, j; // pointers

    for (j = L; j < U; j++)

    {

        if (A[j] < A[U])

        {

            i++; // increment pivot positioner

            swap(A[i], A[j]); // swap

        }

    }

    i++; // final pivot positioner

    swap(A[i], A[U]); // swap with pivot

    return i; // return pivot position

}

// Quick Sort Function

void sort(int \* A, int L, int U)

{

    if (L < U)

    {

        int P = partition(A, L, U); // get partition

        sort(A, L, P-1); // sort elements left of partition

        sort(A, P+1, U); // sort elements right of partition

    }

}

// Main Function

int main()

{

    cout << "Welcome to C++ Quick Sort !" << endl << endl;

    int n; // number of elements of array

    cout << "Enter the size of your array : ";

    cin >> n;

    int A[n]; // array

    cout << "Enter your array : ";

    for (int i = 0; i < n; i++) cin >> A[i];

    sort(A, 0, n-1); // quick sort function

    cout << "Sorted Array : ";

    for (int i = 0; i < n; i++) cout << A[i] << " ";

    cout << endl << endl << "Thank you for using C++ Quick Sort. Bye Bye !";

}

/\*

    Program-13

    Write a program for finding the kth smallest element in min-heap.

    @ Prajwal Sundar

\*/

#include "bits/stdc++.h"

using namespace std;

struct Element

{

    int num;

    int pos;

};

bool comp(struct Element A, struct Element B)

{

    return (A.num < B.num);

}

// Get kth smallest element from a min-heap

int kSmall(vector<int> MH, int k)

{

    int n = MH.size(); // size of min-heap

    vector<struct Element> V; // candidates for smallest node property

    V.push\_back({MH[0], 0});

    int ans = INT\_MAX;

    for (int i = 1; i <= k; i++)

    {

        sort(V.begin(), V.end(), comp); // sort vector

        ans = V[0].num; // smallest element currently

        int L = (2 \* V[0].pos) + 1; // position of left child

        int R = (2 \* V[0].pos) + 2; // position of right child

        if (L < n) V.push\_back({MH[L], L});

        if (R < n) V.push\_back({MH[R], R}); // push children

        V.erase(V.begin()); // remove first element

    }

    return ans;

}

// Main Function

int main()

{

    cout << "Welcome to C++ Min Heaps !" << endl << endl;

    int n; // size of heap

    cout << "Enter the number of elements in your heap : ";

    cin >> n;

    vector<int> MH; // min heap vector

    cout << "Enter your min-heap : ";

    for (int i = 0; i < n; i++)

    {

        int tmp; cin >> tmp;

        MH.push\_back(tmp);

    }

    int k; // kth smallest element

    cout << "Enter k to get kth smallest element : ";

    cin >> k;

    if ((k >= 1) && (k <= n)) cout << "The kth smallest element in your min-heap is : " << kSmall(MH, k);

    else cout << "k should be in the range [1,n].";

    cout << endl << endl << "Thank you for using C++ Min Heaps. Bye Bye !";

}

/\*

    Program-14

    Implement TSP problem using Dynamic Programming approach.

    @ Prajwal Sundar

\*/

#include "bits/stdc++.h"

using namespace std;

// Graph Class

class Graph

{

    public:

    int v, e; // number of vertices and edges

    int \*\* M; // adjacency matrix

    Graph(int n) // constructor

    {

        v = n; // set number of vertices

        e = 0; // no edges initially

        M = new int \* [v];

        for (int i = 0; i < v; i++)

        {

            M[i] = new int [v];

            for (int j = 0; j < v; j++) M[i][j] = 0;

        }

    }

    void addEdge(int x, int y, int w)

    {

        if (!M[x][y]) e++; // increment number of edges

        M[x][y] = w; // set edge with given weight

    }

    int getWeight(int x, int y)

    {

        return M[x][y];

    }

};

// Travelling Salesman Problem using Dynamic Programming

// Graph G and Starting Vertex s

void TSP(Graph G, int s)

{

    int r = G.v; // number of rows of matrix

    int c = pow(2, G.v); // number of columns of matrix

    int \*\* C = new int \* [r]; // cost matrix

    int \*\* N = new int \* [r]; // next matrix

    for (int i = 0; i < r; i++)

    {

        C[i] = new int [c]; N[i] = new int [c];

        for (int j = 0; j < c; j++)

        {

            C[i][j] = 0; // 0 cost

            N[i][j] = -1; // no where to go next

        }

    }

    vector<int> \* V = new vector<int> [r+1]; // vector

    int \* ones = new int [c]; // number of one's

    ones[0] = 0; V[0].push\_back(0);

    for (int i = 1; i < c; i++)

    {

        ones[i] = ones[i/2] + (i%2);

        V[ones[i]].push\_back(i);

    }

    // Base Cases

    for (int i = 0; i < G.v; i++)

    {

        C[i][c-1] = G.getWeight(i, s);

        N[i][c-1] = s;

    }

    // Recursive Cases

    for (int n = 1; n < r; n++)

    {

        // n - number of vertices remaining (number of zeros)

        for (int ptr = 0; ptr < V[r-n].size(); ptr++)

        {

            int j = V[r-n][ptr]; // column number

            for (int i = 0; i < r; i++)

            {

                int mincost = INT\_MAX, next = -1; // minimum cost and next vertex visited

                for (int x = 0; x < r; x++)

                {

                    if (!(j & (1 << x))) // xth vertex is unvisited

                    {

                        int cost = G.getWeight(i, x) + C[x][j+int(pow(2,x))]; // find cost

                        if (cost < mincost)

                        {

                            mincost = cost;

                            next = x; // update minimum cost and next vertex

                        }

                    }

                }

                C[i][j] = mincost;

                N[i][j] = next; // record results

            }

        }

    }

    int ptrR = s, ptrC = (int) pow(2, s);

    cout << endl << "Minimum Cost : " << C[ptrR][ptrC] << endl;

    cout << "Path : ";

    do

    {

        cout << ptrR << " -> ";

        int next = N[ptrR][ptrC];

        ptrR = next;

        ptrC += (int) pow(2, next);

    } while (ptrR != s);

    cout << s;

}

// Main Function

int main()

{

    cout << "Welcome to C++ Travelling Salesman Problem !" << endl << endl;

    int v; // number of vertices

    cout << "Enter the number of vertices in your graph : ";

    cin >> v;

    int e; // number of edges

    cout << "Enter the number of edges in your graph : ";

    cin >> e;

    Graph G(v); // graph object

    cout << "Enter each edge as (source destination weight) :-" << endl;

    for (int i = 0; i < e; i++)

    {

        int x, y, w; cin >> x >> y >> w;

        G.addEdge(x, y, w);

    }

    int s; // start vertex

    cout << "Enter the start vertex : ";

    cin >> s;

    TSP(G, s);

    cout << endl << endl << "Thank you for using C++ Travelling Salesman Problem. Bye Bye !";

}

/\*

    Program-15

    Implement Strassen’s Matrix multiplication using Divide and Conquer approach.

    @ Prajwal Sundar

\*/

#include "bits/stdc++.h"

using namespace std;

// Padding Operation

int \*\* padding(int \*\* M, int r, int c, int n)

{

    int \*\* matrix = new int \* [n];

    for (int i = 0; i < r; i++)

    {

        matrix[i] = new int [n];

        for (int j = 0; j < c; j++) matrix[i][j] = M[i][j];

        for (int j = c; j < n; j++) matrix[i][j] = 0;

    }

    for (int i = r; i < n; i++)

    {

        matrix[i] = new int [n];

        for (int j = 0; j < n; j++) matrix[i][j] = 0;

    }

    return matrix;

}

// Addition Operation

int \*\* Add(int \*\* A, int rA, int cA, int \*\* B, int rB, int cB, int n)

{

    int \*\* C = new int \* [n];

    for (int i = 0; i < n; i++)

    {

        C[i] = new int [n];

        for (int j = 0; j < n; j++)

            C[i][j] = A[i+rA][j+cA] + B[i+rB][j+cB]; // addition operation

    }

    return C;

}

// Subtraction Operation

int \*\* Sub(int \*\* A, int rA, int cA, int \*\* B, int rB, int cB, int n)

{

    int \*\* C = new int \* [n];

    for (int i = 0; i < n; i++)

    {

        C[i] = new int [n];

        for (int j = 0; j < n; j++)

            C[i][j] = A[i+rA][j+cA] - B[i+rB][j+cB]; // subtraction operation

    }

    return C;

}

// Strassen's Matrix Multiplication on two n x n matrices, where n is a power of 2

int \*\* Mul(int \*\* A, int rA, int cA, int \*\* B, int rB, int cB, int n)

{

    int \*\* C = new int \* [n];

    for (int i = 0; i < n; i++) C[i] = new int [n];

    if (n == 1)

    {

        C[0][0] = A[rA][cA] \* B[rB][cB]; // perform multiplication

        return C; // return

    }

    // Temperory M matrices

    int \*\* M1 = Add(A, rA, cA, A, rA + n/2, cA + n/2, n/2); // A11 + A22

    int \*\* M2 = Add(B, rB, cB, B, rB + n/2, cB + n/2, n/2); // B11 + B22

    int \*\* M3 = Add(A, rA + n/2, cA, A, rA + n/2, cA + n/2, n/2); // A21 + A22

    int \*\* M4 = Sub(B, rB, cB + n/2, B, rB + n/2, cB + n/2, n/2); // B12 - B22

    int \*\* M5 = Sub(B, rB + n/2, cB, B, rB, cB, n/2); // B21 - B11

    int \*\* M6 = Add(A, rA, cA, A, rA, cA + n/2, n/2); // A11 + A12

    int \*\* M7 = Sub(A, rA + n/2, cA, A, rA, cA, n/2); // A21 - A11

    int \*\* M8 = Add(B, rB, cB, B, rB, cB + n/2, n/2); // B11 + B12

    int \*\* M9 = Sub(A, rA, cA + n/2, A, rA + n/2, cA + n/2, n/2); // A12 - A22

    int \*\* M10 = Add(B, rB + n/2, cB, B, rB + n/2, cB + n/2, n/2); // B21 + B22

    // Intermediate Strassen's 7 Matrices

    int \*\* P = Mul(M1, 0, 0, M2, 0, 0, n/2);

    int \*\* Q = Mul(M3, 0, 0, B, rB, cB, n/2);

    int \*\* R = Mul(A, rA, cA, M4, 0, 0, n/2);

    int \*\* S = Mul(A, rA + n/2, cA + n/2, M5, 0, 0, n/2);

    int \*\* T = Mul(M6, 0, 0, B, rB + n/2, cB + n/2, n/2);

    int \*\* U = Mul(M7, 0, 0, M8, 0, 0, n/2);

    int \*\* V = Mul(M9, 0, 0, M10, 0, 0, n/2);

    // Some more intermediate matrices

    int \*\* M11 = Add(P, 0, 0, S, 0, 0, n/2);

    int \*\* M12 = Sub(V, 0, 0, T, 0, 0, n/2);

    int \*\* M13 = Add(P, 0, 0, R, 0, 0, n/2);

    int \*\* M14 = Sub(U, 0, 0, Q, 0, 0, n/2);

    // Final Strassen's Results

    int \*\* C11 = Add(M11, 0, 0, M12, 0, 0, n/2);

    int \*\* C12 = Add(R, 0, 0, T, 0, 0, n/2);

    int \*\* C21 = Add(Q, 0, 0, S, 0, 0, n/2);

    int \*\* C22 = Add(M13, 0, 0, M14, 0, 0, n/2);

    // Copy Values into C Matrix

    for (int i = 0; i < n/2; i++)

    {

        for (int j = 0; j < n/2; j++)

        {

            C[i][j] = C11[i][j];

            C[i][j+n/2] = C12[i][j];

            C[i+n/2][j] = C21[i][j];

            C[i+n/2][j+n/2] = C22[i][j]; // copy

        }

    }

    return C; // return multiplied result

}

// Display a Matrix

void display(int \*\* M, int r, int c)

{

    for (int i = 0; i < r; i++)

    {

        for (int j = 0; j < c; j++) cout << M[i][j] << "\t";

        cout << endl;

    }

}

// Main Function

int main()

{

    cout << "Welcome to C++ Strassen's Matrix Multiplication !" << endl << endl;

    int rA, cA;

    cout << "Enter the dimensions of your first matrix : ";

    cin >> rA >> cA;

    int rB, cB;

    cout << "Enter the dimensions of your second matrix : ";

    cin >> rB >> cB;

    if (cA != rB) // multiplication not possible

        cout << "Sorry, multiplication not possible as the number of columns of the first matrix and number of rows of the second matrix are not equal." << endl;

    else

    {

        cout << endl << "Enter matrix A below :-" << endl;

        int \*\* A = new int \* [rA];

        for (int i = 0; i < rA; i++)

        {

            A[i] = new int [cA];

            for (int j = 0; j < cA; j++) cin >> A[i][j];

        }

        cout << endl << "Enter matrix B below :-" << endl;

        int \*\* B = new int \* [rB];

        for (int i = 0; i < rB; i++)

        {

            B[i] = new int [cB];

            for (int j = 0; j < cB; j++) cin >> B[i][j];

        }

        vector<int> D = {rA, cA, rB, cB}; int n = INT\_MIN;

        for (int i = 0; i < 4; i++) n = max(n, D[i]); // get maximum dimension

        n = (int) pow(2, ceil(log(n)/log(2))); // convert to next nearest power of 2

        int \*\* X = padding(A, rA, cA, n);

        int \*\* Y = padding(B, rB, cB, n); // convert to square matrices by padding

        int \*\* C = Mul(X, 0, 0, Y, 0, 0, n); // strassen

        cout << endl << "Multiplied Result :-" << endl;

        display(C, rA, cB);

    }

    cout << endl << "Thank you for using C++ Strassen's Matrix Multiplication. Bye Bye !";

}