/\*

NAME: SHRISTY CHAUDHARY

ROLL No.:01

SECTION: AI & ML

**PROBLEM STATEMENT 1:**

Given an array of non negative integers,design a linear algorithm and implement it

using a program to find whether given key element is present in the array or not.

Also,find total number of comparison for each input case.(Time Complexity=O(n),

where n is the size of input).

\*/

**CODE:**

#include<iostream>

using namespace std;

bool linear\_search(int arr[],int n,int key,int &comparisons)

{

comparisons=0;

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

{

comparisons++;

if(arr[i] == key)

{

return true;

}

}

return false;

}

int main()

{

int arr[] = {2,5,8,12,16,23,38,56,72,91};

int n = sizeof(arr)/sizeof(arr[0]);

int key = 23;

int comparisons;

if(linear\_search(arr,n,key,comparisons))

{

cout<<"Number of comparisons:"<<comparisons<<endl;

cout<<"Key found in array!"<<endl;

}

else{

cout<<"Number of comparisons:"<<comparisons<<endl;

cout<<"Key not found in array!"<<endl;

}

return 0;

}

**OUTPUT:**

Number of comparisons:6

Key found in array!

/\*

NAME: SHRISTY CHAUDHARY

ROLL No.:01

SECTION: AI & ML

**PROBLEM STATEMENT 2:**

Given an already sorted array of positive integers,design an algorithm and implement

it using a program to find whether given key element is present in the array or not.

Also,find total number of comparisons for each input case.(Time complexity = O(logn)

,where n is the size of input).

\*/

**CODE:**

#include<iostream>

using namespace std;

bool binary\_search(int arr[],int n,int key,int &comparisons)

{

int start=0;

int end=n-1;

comparisons=0;

while(start<=end)

{

int mid=(start+end)/2;

comparisons++;

if(arr[mid] == key)

{

return true;

}

else if (arr[mid]<key)

{

start = mid+1;

}

else

{

end=mid-1;

}

}

return false;

}

int main()

{

int arr[]={2,5,8,12,16,23,38,56,72,91};

int n=sizeof(arr)/sizeof(arr[0]);

int key = 23;

int comparisons;

if(binary\_search(arr,n,key,comparisons))

{

cout<<"Numer of comparisons:"<<comparisons<<endl;

cout<<"Key found in array!"<<endl;

}

else

{

cout<<"Numer of comparisons:"<<comparisons<<endl;

cout<<"Key not found in array!"<<endl;

}

return 0;

}

**OUTPUT:**

Numer of comparisons:3

Key found in array!

/\*

NAME: SHRISTY CHAUDHARY

ROLL No.:01

SECTION: AI & ML

**PROBLEM STATEMENT 3:**

Given an already sorted array of positive integers,design an algorithm and implement

it using a program to find whether a given key element is present in the sorted array

or not.For an array arr[n],search at indexes arr[0],arr[2],arr[4],...,arr[2k] and so

on.Once the interval (arr[2k]<key<arr[2k+1]) is found,perform a linear search operation

from the index 2k to find the element key.(Complexity<O(n),where n is the number of

elements need to be scanned for searching):Jump Search.

\*/

**CODE:**

#include<iostream>

#include<cmath>

using namespace std;

int jump\_search(int arr[],int n,int key,int &comparisons)

{

int jump = sqrt(n);

int left = 0;

int right = jump;

comparisons = 0;

while (right<n)

{

comparisons++;

if (arr[right]>=key)

{

for(int i=left;i<=right;i++)

{

comparisons++;

if(arr[i] == key)

{

return i;

}

}

return -1;

}

left = right;

right += jump;

}

right = n;

for(int i=left;i<right;i++)

{

comparisons++;

if(arr[i] == key)

{

return i;

}

}

return -1;

}

int main()

{

int arr[]={2,5,8,12,16,23,38,56,72,91};

int n = sizeof(arr)/sizeof(arr[0]);

int key = 23;

int comparisons;

int index = jump\_search(arr,n,key,comparisons);

if(index != -1)

{

cout<<"Number of comparisons:"<<comparisons<<endl;

cout<<"Key found at index "<<index<<" in array!"<<endl;

}

else

{

cout<<"Number of comparisons:"<<comparisons<<endl;

cout<<"Key not found in array."<<endl;

}

return 0;

}

**OUTPUT:**

Number of comparisons:5

Key found at index 5 in array!

/\*

NAME: SHRISTY CHAUDHARY

ROLL No.:01

SECTION: AI & ML

**PROBLEM STATEMENT 4:**

Given a sorted array of positive integers containing few duplicate elements,design

an algorithm and implement it using a program to find whether the given key element

is present in the array or not.If present,then also find the number of copies of

given key.(Time Complexity = O(log n)).

\*/

**CODE:**

#include<iostream>

using namespace std;

int binary\_search(int arr[],int n,int key,int &comparisons)

{

int low = 0;

int high = n-1;

comparisons = 0;

while (low<= high)

{

comparisons++;

int mid = (low+high)/2;

if(arr[mid]==key)

{

int count =1;

int left = mid-1;

while(left>=0&&arr[left]==key)

{

count++;

left--;

}

int right = mid +1;

while (right<n&&arr[right]==key)

{

count++;

right++;

}

return count;

}

else if(arr[mid]>key)

{

high=mid-1;

}

else

{

low=mid-1;

}

}

return 0;

}

int main()

{

int arr[]={2,3,5,5,7,7,7,8,9,10,10};

int n = sizeof(arr)/sizeof(arr[0]);

int key = 7;

int comparisons;

int count = binary\_search(arr,n,key,comparisons);

if(count>0)

{

cout<<"Number of comparisons:"<<comparisons<<endl;

cout<<"Key found in array!Number of copies:"<<count<<endl;

}

else

{

cout<<"Number of comparisons:"<<comparisons<<endl;

cout<<"Key not found in array."<<endl;

}

return 0;

}

**OUTPUT:**

Number of comparisons:1

Key found in array!Number of copies:3

/\*

NAME: SHRISTY CHAUDHARY

ROLL No.:01

SECTION: AI & ML

**PROBLEM STATEMENT 5:**

Given a sorted array of positive integers,design an algorithm and implement it

using a program to find three indices i,j,k such that arr[i]+arr[j]=arr[k].

\*/

**CODE:**

#include<iostream>

using namespace std;

void find\_triplet(int arr[],int n)

{

for(int k=n-1;k>=2;k--)

{

int i=0;

int j=k-1;

while(i<j)

{

if(arr[i]+arr[j]==arr[k])

{

cout<<"Found Triplet:("<<i<<","<<j<<","<<k<<")"<<endl;

return ;

}

else if(arr[i]+arr[j]>arr[k])

{

j--;

}

else

{

i++;

}

}

}

cout<<"No Triplet Found!"<<endl;

}

int main()

{

int arr[]={1,2,3,4,5,7,9,10};

int n=sizeof(arr)/sizeof(arr[0]);

find\_triplet(arr,n);

return 0;

}

**OUTPUT:**

Found Triplet:(0,6,7)

/\*

NAME: SHRISTY CHAUDHARY

ROLL No.:01

SECTION: AI & ML

**PROBLEM STATEMENT 6:**

Given an array of non negative integers,design an algorithm and a program to

count the number of pairs of integers such that their difference is equal to

a given key,k.

\*/

**CODE:**

#include<iostream>

#include<algorithm>

using namespace std;

int count\_pairs(int arr[],int n,int K)

{

sort(arr,arr+n); //sort the array in non-decreasing order

int count=0;

int i=0,j=1;

while(i<n && j<n)

{

if(arr[j]-arr[i]==K)

{

count++;

i++;

j++;

}

else if(arr[j]-arr[i]>K)

{

i++;

}

else

{

j++;

}

}

return count;

}

int main()

{

int arr[]={1,5,3,4,2};

int n=sizeof(arr)/sizeof(arr[0]);

int K=2;

int num\_pairs=count\_pairs(arr,n,K);

cout<<"Number of pairs with difference "<<K<<" is "<<num\_pairs<<endl;

return 0;

}

**OUTPUT:**

Number of pairs with difference 2 is 3

/\*

NAME: SHRISTY CHAUDHARY

ROLL No.:01

SECTION: AI & ML

**PROBLEM STATEMENT 7:**

Given an unsorted array of integers,design an algorithm and a program to

sort the array using insertion sort.Your program should be able to find

number of comparisons and shifts (shifts-total number of times the array

elements are shifted from their place) required for sorting the array.

\*/

**CODE:**

#include<iostream>

using namespace std;

void insertion\_sort(int arr[],int n,int &comparisons,int &shifts)

{

for(int i=1;i<n;i++)

{

int key=arr[i];

int j=i-1;

while(j>=0 && arr[j]>key)

{

arr[j+1]=arr[j];

j--;

comparisons++;

shifts++;

}

arr[j+1]=key;

shifts++;

}

}

int main()

{

int arr[]={3,7,1,9,2};

int n=sizeof(arr)/sizeof(arr[0]);

int comparisons=0;

int shifts=0;

insertion\_sort(arr,n,comparisons,shifts);

cout<<"Sorted Array:";

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

{

cout<<arr[i]<<" ";

}

cout<<endl;

cout<<"Number of comparisons:"<<comparisons<<endl;

cout<<"Number of shifts:"<<shifts<<endl;

return 0;

}

**OUTPUT:**

Sorted Array:1 2 3 7 9

Number of comparisons:5

Number of shifts:9

/\*

NAME: SHRISTY CHAUDHARY

ROLL No.:01

SECTION: AI & ML

**PROBLEM STATEMENT 8:**

Given an unsorted array of integers,design an algorithm and implement a

program to sort this array using selection sort.Your program should also find

number of comparisons and number of swaps required.

\*/

**CODE:**

#include <iostream>

using namespace std;

void selectionsort(int arr[],int n,int &comp,int &swaps)

{

for(int i=0;i<n-1;i++)

{

int min\_idx=i;

for(int j=i+1;j<n;j++)

{

comp++;//increment comparison count

if(arr[j]<arr[min\_idx])

min\_idx=j;

}

swaps++;//increment swap count

swap(arr[min\_idx],arr[i]);

}

}

int main()

{

int arr[]={64,25,12,22,11};

int n=sizeof(arr)/sizeof(arr[0]);

int comp=0,swaps=0;

selectionsort(arr,n,comp,swaps);

cout<<"Sorted Array:";

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

{

cout<<arr[i]<<" ";

}

cout<<endl;

cout<<"Number of comparisons:"<<comp<<endl;

cout<<"Number of Swaps:"<<swaps<<endl;

return 0;

}

**OUTPUT:**

Sorted Array:11 12 22 25 64

Number of comparisons:10

Number of Swaps:4

/\*

NAME: SHRISTY CHAUDHARY

ROLL No.:01

SECTION: AI & ML

**PROBLEM STATEMENT 9:**

Given an unsorted array of positive integers,design an algorithm and increment

it using a program to find whether there are any duplicate elements in the array

or not.(use sorting)(Time Complexity=O(n log n)).

\*/

**CODE:**

#include <iostream>

#include <algorithm>

using namespace std;

bool hasDuplicates(int arr[],int n)

{

sort(arr,arr+n);

for(int i=1;i<n;i++)

{

if(arr[i] == arr[i-1])

{

return true;

}

}

return false;

}

int main()

{

int arr[]={5,2,7,2,4,7,8,2,3};

int n=sizeof(arr)/sizeof(arr[0]);

if(hasDuplicates(arr,n))

{

cout<<"Duplicates found in the array."<<endl;

}

else

{

cout<<"No Duplicates found in the array."<<endl;

}

}

**OUTPUT:**

Duplicates found in the array.

/\*

NAME: SHRISTY CHAUDHARY

ROLL No.:01

SECTION: AI & ML

**PROBLEM STATEMENT 10:**

Given an unsorted array of integers,design an algorithm and implement it using

a program to sort an array of elements by dividing the array into two subarrays

and combining these subarrays after sorting each one of them.Your program should

also find number of comparisons and inversions during sorting the array.

\*/

**CODE:**

#include <iostream>

using namespace std;

void merge(int arr[],int l,int m,int r,int &comparisons,int &inversions)

{

int n1=m-l+1;

int n2=r-m;

int L[n1],R[n2];

for(int i=0;i<n1;i++)

{

L[i]=arr[l+i];

}

for(int j=0;j<n2;j++)

{

R[j]=arr[m+1+j];

}

int i=0,j=0,k=l;

while(i<n1 && j<n2)

{

comparisons++;

if(L[i]<=R[j])

{

arr[k++]=L[i++];

}

else

{

arr[k++]=R[j++];

inversions+=n1-i;

}

}

while(i<n1)

{

arr[k++]=L[i++];

}

while(j<n2)

{

arr[k++]=R[j++];

}

}

void mergesort(int arr[],int l,int r,int &comparisons,int &inversions)

{

if(l>=r)

{

return;

}

int m=l+(r-l)/2;

mergesort(arr,l,m,comparisons,inversions);

mergesort(arr,m+1,r,comparisons,inversions);

merge(arr,l,m,r,comparisons,inversions);

}

int main()

{

int n,comparisons=0,inversions=0;

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

cin>>n;

int arr[n];

cout<<"Enter the elements of the array:\n";

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

{

cin>>arr[i];

}

mergesort(arr,0,n-1,comparisons,inversions);

cout<<"Sorted Array:";

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

{

cout<<arr[i]<<" ";

}

cout<<endl;

cout<<"Number of comparisons:"<<comparisons<<endl;

cout<<"Number of inversions:"<<inversions<<endl;

return 0;

}

**OUTPUT:**

Enter the size of the array:5

Enter the elements of the array:

5 4 3 2 1

Sorted Array:1 2 3 4 5

Number of comparisons:5

Number of inversions:10

/\*

NAME: SHRISTY CHAUDHARY

ROLL No.:01

SECTION: AI & ML

**PROBLEM STATEMENT 11:**

Given an unsorted array of integers,design an algorithm and implement it using

a program to sort an array of elements by partioning the array into two subarrays

based on a pivot element such that one of the sub array holds values smaller than

the pivot element while another subarray holds values greater than the pivot element.

Pivot element should be selected randomly from the array.your program should also

find number of comparisons and swaps required for sorting the array.

\*/

**CODE:**

#include <iostream>

#include <cstdlib>

#include <ctime>

using namespace std;

int partition(int arr[],int low,int high,int &comparisons,int &swaps)

{

int pivot = arr[high];

int i=low-1;

for(int j=low;j<=high-1;j++)

{

comparisons++;

if(arr[j]<=pivot)

{

i++;

swaps++;

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

}

}

swaps++;

swap(arr[i+1],arr[high]);

return i+1;

}

void quicksort(int arr[],int low,int high,int &comparisons,int &swaps)

{

if(low<high)

{

int pivot=rand()%(high-low+1)+low;

swaps++;

swap(arr[pivot],arr[high]);

int p=partition(arr,low,high,comparisons,swaps);

quicksort(arr,low,p-1,comparisons,swaps);

quicksort(arr,p+1,high,comparisons,swaps);

}

}

int main()

{

int n,comparisons=0,swaps=0;

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

cin>>n;

int arr[n];

cout<<"Enter the elements of the array:\n";

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

{

cin>>arr[i];

}

srand(time(NULL));

quicksort(arr,0,n-1,comparisons,swaps);

cout<<"Sorted Array:";

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

{

cout<<arr[i]<<" ";

}

cout<<endl;

cout<<"Number of comparisons:"<<comparisons<<endl;

cout<<"Number of swaps:"<<swaps<<endl;

return 0;

}

**OUTPUT:**

Enter the size of the array:5

Enter the elements of the array:

5 4 3 2 1

Sorted Array:1 2 3 4 5

Number of comparisons:7

Number of swaps:7

/\*

NAME: SHRISTY CHAUDHARY

ROLL No.:01

SECTION: AI & ML

**PROBLEM STATEMENT 12:**

Given an unsorted array of integers,design an algorithm and implement it

using a program to find Kth smallest or largest element in the array.(Worst

case time complexity=O(n)).

\*/

**CODE:**

#include <iostream>

#include <cstdlib>

#include <ctime>

using namespace std;

int partition(int arr[],int left,int right)

{

int pivot=arr[right];//Choose the rightmost element as pivot

int i=left-1;//Index of smaller element

for(int j=left;j<=right-1;j++)

{

//If current element is smaller than equal to pivot

if(arr[j]<=pivot)

{

i++;//Increment index of smaller element

swap(arr[i],arr[j]);//swap arr[i] and arr[j]

}

}

swap(arr[i+1],arr[right]);//swap arr[i+1] and arr[right]

return i+1;

}

int quickselect(int arr[],int left,int right,int k)

{

if(left==right)

{

//If the array contains only one element

return arr[left];

}

//choose a random pivot element

int pivot\_index=left+rand()%(right-left+1);

//Partition the array around the pivot and get the pivot's final index

int pivot\_final\_index=partition(arr,left,right);

//If the pivot is the Kth element,return it

if(k==pivot\_final\_index)

{

return arr[k];

}

//If the pivot is smaller than the Kth element,search the right subarray

if(k>pivot\_final\_index)

{

return quickselect(arr,pivot\_final\_index+1,right,k);

}

//If the pivot is larger than the Kth element,search the left subarray

else

{

return quickselect(arr,left,pivot\_final\_index-1,k);

}

}

int main()

{

int arr[]={5,2,8,3,9,1};

int n=sizeof(arr)/sizeof(arr[0]);

int k=3;

int Kth\_smallest = quickselect(arr,0,n-1,k-1);

cout<<"The "<<k<<"th smallest element is "<<Kth\_smallest<<endl;

return 0;

}

**OUTPUT:**

The 3th smallest element is 3

/\*

NAME: SHRISTY CHAUDHARY

ROLL No.:01

SECTION: AI & ML

**PROBLEM STATEMENT 13:**

Given an unsorted array of alphabets containing duplicate elements.Design an

algorithm and implement it using a program to find which alphabet has maximum

number of occurences and print it.(Time Complexity = O(n)).(Hint:Use

counting sort)

\*/

**CODE:**

#include <iostream>

#include <cstring>

using namespace std;

char findMaxOccurrence(char arr[],int n)

{

int count[26]={0};//initialize count array to 0

int maxCount=0;

char maxChar=' ';

//count frequency of each alphabet

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

{

count[arr[i]-'a']++;

}

//find alphabet with maximum count

for(int i=0;i<26;i++)

{

if(count[i]>maxCount)

{

maxCount=count[i];

maxChar='a'+i;

}

}

return maxChar;

}

int main()

{

char arr[]="abcbadde";

int n=strlen(arr);

char maxChar = findMaxOccurrence(arr,n);

cout<<"Alphabet with maximum occurrence is:"<<maxChar<<endl;

return 0;

}

**OUTPUT:**

Alphabet with maximum occurrence is:a

/\*

NAME: SHRISTY CHAUDHARY

ROLL No.:01

SECTION: AI & ML

**PROBLEM STATEMENT 14:**

Given an unsorted array of integers,design an algorithm and implement it using

a program to find whether two elements exist such that their sum is equal to

the given key element.(Time Complexity = O(n log n)).

\*/

**CODE:**

#include <iostream>

#include <algorithm>

using namespace std;

bool isSumPresent(int arr[],int n,int key)

{

//Sort the array

sort(arr,arr+n);

//Initialize left and right pointers

int left=0,right=n-1;

//Loop through the array using two pointers

while(left<right)

{

int sum = arr[left]+arr[right];

if(sum == key)

{

//Pair found

return true;

}

else if(sum>key)

{

//Decrease right pointer

right--;

}

else

{

//Increase left pointer

left++;

}

}

//Pair not found

return false;

}

int main()

{

int arr[]={2,4,6,8,10};

int n=sizeof(arr)/sizeof(arr[0]);

int key=12;

if(isSumPresent(arr,n,key))

{

cout<<"Pair with sum "<<key<<" is present in the array."<<endl;

}

else

{

cout<<"Pair with sum "<<key<<" is not present in the array."<<endl;

}

return 0;

}

**OUTPUT:**

Pair with sum 12 is present in the array.

/\*

NAME: SHRISTY CHAUDHARY

ROLL No.:01

SECTION: AI & ML

**PROBLEM STATEMENT 15:**

You have been given two sorted integer arrays of size m and n. Design an algorithm and

implement it using a program to find list of elements which are common to both. (Time

Complexity = O(m+n))

\*/

**CODE:**

#include <iostream>

using namespace std;

void findCommonElements(int arr1[], int m, int arr2[], int n)

{

int i = 0, j = 0;

while (i < m && j < n)

{

if (arr1[i] == arr2[j])

{

cout << arr1[i] << " ";

i++;

j++;

}

else if (arr1[i] < arr2[j])

{

i++;

}

else

{

j++;

}

}

}

int main()

{

int m, n;

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

cin >> m;

int arr1[m];

cout << "Enter the elements of the first array: ";

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

{

cin >> arr1[i];

}

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

cin >> n;

int arr2[n];

cout << "Enter the elements of the second array: ";

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

{

cin >> arr2[i];

}

cout << "Common elements: ";

findCommonElements(arr1, m, arr2, n);

return 0;

}

**OUTPUT:**

Enter the size of the first array: 5

Enter the elements of the first array: 4 5 8 7 6

Enter the size of the second array: 8

Enter the elements of the second array: 2 4 7 6 3 5 1 9

Common elements: 4

/\*

NAME: SHRISTY CHAUDHARY

ROLL No.:01

SECTION: AI & ML

**PROBLEM STATEMENT 16:**

Given a (directed/undirected) graph, design an algorithm and implement it

using a program to find if a path exists between two given vertices or not.

(Hint: use DFS)

\*/

#include <iostream>

using namespace std;

#define MAX\_VERTICES 100

class Graph

{

int numVertices;

int adjMatrix[MAX\_VERTICES][MAX\_VERTICES];

public:

Graph(int vertices)

{

numVertices = vertices;

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

{

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

{

adjMatrix[i][j] = 0;

}

}

}

void addEdge(int src, int dest)

{

adjMatrix[src][dest] = 1;

}

bool hasPathDFS(int src, int dest, bool visited[])

{

if (src == dest)

{

return true;

}

visited[src] = true;

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

{

if (adjMatrix[src][i] == 1 && !visited[i])

{

if (hasPathDFS(i, dest, visited))

{

return true;

}

}

}

return false;

}

};

int main()

{

int numVertices;

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

cin >> numVertices;

Graph graph(numVertices);

int numEdges;

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

cin >> numEdges;

cout << "Enter the edges (source destination):" <<endl;

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

{

int src, dest;

cin >> src >> dest;

graph.addEdge(src, dest);

}

int srcVertex, destVertex;

cout << "Enter the source and destination vertices: ";

cin >> srcVertex >> destVertex;

bool visited[MAX\_VERTICES] = {false};

if (graph.hasPathDFS(srcVertex, destVertex, visited))

{

cout << "Yes, Path Exists" <<endl;

}

else

{

cout << "No, Such Path Exists" << endl;

}

return 0;

}

**OUTPUT:**

Enter the number of vertices in the graph: 6

Enter the number of edges in the graph: 5

Enter the edges (source destination):

8 4 6 2 7

6 4 2 6 7

Enter the source and destination vertices: 8 7

No, Such Path Exists

/\*

NAME: SHRISTY CHAUDHARY

ROLL No.:01

SECTION: AI & ML

**PROBLEM STATEMENT 17:**

Given a graph, design an algorithm and implement it using a program to find

if a graph is bipartite or not. (Hint: use BFS)

\*/

**CODE:**

#include <iostream>

#include <queue>

using namespace std;

#define MAX\_VERTICES 100

class Graph

{

int numVertices;

int adjMatrix[MAX\_VERTICES][MAX\_VERTICES];

public:

Graph(int vertices)

{

numVertices = vertices;

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

{

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

{

adjMatrix[i][j] = 0;

}

}

}

void addEdge(int src, int dest)

{

adjMatrix[src][dest] = 1;

adjMatrix[dest][src] = 1;

}

bool isBipartite()

{

int color[MAX\_VERTICES] = {0};

int visited[MAX\_VERTICES] = {false};

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

{

if (!visited[i])

{

color[i] = 1;

visited[i] = true;

queue<int> q;

q.push(i);

while (!q.empty())

{

int currVertex = q.front();

q.pop();

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

{

if (adjMatrix[currVertex][j] == 1 && !visited[j])

{

visited[j] = true;

color[j] = 1 - color[currVertex];

q.push(j);

}

else if (adjMatrix[currVertex][j] == 1 && color[currVertex] == color[j])

{

return false;

}

}

}

}

}

return true;

}

};

int main()

{

int numVertices;

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

cin >> numVertices;

Graph graph(numVertices);

int numEdges;

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

cin >> numEdges;

cout << "Enter the edges (source destination):" <<endl;

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

{

int src, dest;

cin >> src >> dest;

graph.addEdge(src, dest);

}

if (graph.isBipartite())

{

cout << "Yes, Bipartite" << endl;

}

else

{

cout << "Not Bipartite" << endl;

return 0;

}

**OUTPUT:**

Enter the number of vertices in the graph: 5

Enter the number of edges in the graph: 4

Enter the edges (source destination):

2 4 5 6 9

4 5 2 9 6

Yes, Bipartite

/\*

NAME: SHRISTY CHAUDHARY

ROLL No.:01

SECTION: AI & ML

**PROBLEM STATEMENT 18:**

Given a directed graph, design an algorithm and implement it using a

program to find whether cycle exists in the graph or not

\*/

**CODE:**

#include <iostream>

using namespace std;

#define MAX\_VERTICES 100

class Graph

{

int numVertices;

int adjMatrix[MAX\_VERTICES][MAX\_VERTICES];

public:

Graph(int vertices)

{

numVertices = vertices;

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

{

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

{

adjMatrix[i][j] = 0;

}

}

}

void addEdge(int src, int dest)

{

adjMatrix[src][dest] = 1;

}

bool isCyclicUtil(int vertex, bool visited[], bool stack[])

{

visited[vertex] = true;

stack[vertex] = true;

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

{

if (adjMatrix[vertex][i] == 1)

{

if (!visited[i] && isCyclicUtil(i, visited, stack))

{

return true;

}

else if (stack[i])

{

return true;

}

}

}

stack[vertex] = false;

return false;

}

bool isCyclic()

{

bool visited[MAX\_VERTICES] = {false};

bool stack[MAX\_VERTICES] = {false};

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

{

if (!visited[i] && isCyclicUtil(i, visited, stack))

{

return true;

}

return false;

}

}

};

int main()

{

int numVertices;

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

cin >> numVertices;

Graph graph(numVertices);

int numEdges;

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

cin >> numEdges;

cout << "Enter the edges (source destination):" << endl;

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

{

int src, dest;

cin >> src >> dest;

graph.addEdge(src, dest);

}

if (graph.isCyclic())

{

cout << "Cycle exists in the graph." << endl;

}

else

{

cout << "No cycle exists in the graph." << endl;

}

return 0;

}

**OUTPUT:**

Enter the number of vertices in the graph: 4

Enter the number of edges in the graph: 3

Enter the edges (source destination):

1 5 6 8

4 6 3 7

No cycle exists in the graph.

/\*

NAME: SHRISTY CHAUDHARY

ROLL No.:01

SECTION: AI & ML

**PROBLEM STATEMENT 19:**

After end term examination, Akshay wants to party with his friends. All his

friends are living as paying guest and it has been decided to first gather

at Akshay’s house and then move towards party location. The problem is that

no one knows the exact address of his house in the city.Akshay as a computer

science wizard knows how to apply his theory subjects in his real life and

came up with an amazing idea to help his friends. He draws a graph by looking

in to location of his house and his friends’ location (as a node in the graph)

on a map. He wishes to find out shortest distance and path covering that

distance from each of his friend’s location to his house and then whatsapp them

this path so that they can reach his house in minimum time. Akshay has developed

the program that implements Dijkstra’s algorithm but not sure about correctness of

results. Can you also implement the same algorithm and verify the correctness of

Akshay’s results? (Hint: Print shortest path and distance from friends’

location to Akshay’s house)

\*/

**CODE:**

#include <iostream>

#include <limits.h>

using namespace std;

#define MAX\_VERTICES 100

class Graph

{

int numVertices;

int adjMatrix[MAX\_VERTICES][MAX\_VERTICES];

public:

Graph(int vertices)

{

numVertices = vertices;

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

{

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

{

adjMatrix[i][j] = 0;

}

}

}

void addEdge(int src, int dest, int weight)

{

adjMatrix[src][dest] = weight;

}

int findMinDistance(int distance[], bool visited[])

{

int minDistance = INT\_MAX;

int minIndex;

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

{

if (!visited[i] && distance[i] <= minDistance)

{

minDistance = distance[i];

minIndex = i;

}

}

return minIndex;

}

void printPath(int parent[], int vertex)

{

if (parent[vertex] == -1)

{

cout << vertex;

return;

}

printPath(parent, parent[vertex]);

cout << " -> " << vertex;

}

void dijkstra(int src)

{

int distance[MAX\_VERTICES];

bool visited[MAX\_VERTICES];

int parent[MAX\_VERTICES];

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

{

distance[i] = INT\_MAX;

visited[i] = false;

parent[i] = -1;

}

distance[src] = 0;

for (int count = 0; count < numVertices - 1; count++)

{

int u = findMinDistance(distance, visited);

visited[u] = true;

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

{

if (!visited[v] && adjMatrix[u][v] != 0 && distance[u] != INT\_MAX &&

distance[u] + adjMatrix[u][v] < distance[v])

{

distance[v] = distance[u] + adjMatrix[u][v];

parent[v] = u;

}

}

}

cout << "Shortest paths from friends' locations to Akshay's house:" << endl;

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

{

if (i != src)

{

cout << "From friend " << i << ": ";

if (distance[i] != INT\_MAX)

{

printPath(parent, i);

cout << ", Distance: " << distance[i] << endl;

}

else

{

cout << "No path exists" << endl;

}

}

}

}

};

int main()

{

int numVertices;

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

cin >> numVertices;

Graph graph(numVertices);

int numEdges;

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

cin >> numEdges;

cout << "Enter the edges and weights (source destination weight):" << endl;

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

{

int src, dest, weight;

cin >> src >> dest >> weight;

graph.addEdge(src, dest, weight);

}

int akshayHouse;

cout << "Enter the location of Akshay's house: ";

cin >> akshayHouse;

graph.dijkstra(akshayHouse);

return 0;

}

**OUTPUT:**

Enter the number of vertices in the graph: 5

Enter the number of edges in the graph: 4

Enter the edges and weights (source destination weight):

2 6 4

1 5 1

6 2 3

3 5 7

Enter the location of Akshay's house: 5

Shortest paths from friends' locations to Akshay's house:

From friend 0: No path exists

From friend 1: No path exists

From friend 2: No path exists

From friend 3: No path exists

From friend 4: No path exists

/\*

NAME: SHRISTY CHAUDHARY

ROLL No.:01

SECTION: AI & ML

**PROBLEM STATEMENT 20:**

Design an algorithm and implement it using a program to solve previous

question's problem using Bellman- Ford's shortest path algorithm.

\*/

**CODE:**

#include <iostream>

#include <limits.h>

using namespace std;

#define MAX\_VERTICES 100

#define MAX\_EDGES 100

class Edge

{

public:

int src, dest, weight;

};

class Graph

{

int numVertices, numEdges;

Edge edges[MAX\_EDGES];

public:

Graph(int vertices, int edges)

{

numVertices = vertices;

numEdges = edges;

}

void addEdge(int src, int dest, int weight, int index)

{

edges[index].src = src;

edges[index].dest = dest;

edges[index].weight = weight;

}

void bellmanFord(int src)

{

int distance[MAX\_VERTICES];

int parent[MAX\_VERTICES];

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

{

distance[i] = INT\_MAX;

parent[i] = -1;

}

distance[src] = 0;

for (int i = 0; i < numVertices - 1; i++)

{

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

{

int u = edges[j].src;

int v = edges[j].dest;

int weight = edges[j].weight;

if (distance[u] != INT\_MAX && distance[u] + weight < distance[v])

{

distance[v] = distance[u] + weight;

parent[v] = u;

}

}

}

cout << "Shortest paths from friends' locations to Akshay's house:" << endl;

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

{

if (i != src)

{

cout << "From friend " << i << ": ";

if (distance[i] != INT\_MAX)

{

printPath(parent, i);

cout << ", Distance: " << distance[i] << endl;

}

else

{

cout << "No path exists" << endl;

}

}

}

}

void printPath(int parent[], int vertex)

{

if (parent[vertex] == -1)

{

cout << vertex;

return;

}

printPath(parent, parent[vertex]);

cout << " -> " << vertex;

}

};

int main()

{

int numVertices, numEdges;

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

cin >> numVertices;

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

cin >> numEdges;

Graph graph(numVertices, numEdges);

cout << "Enter the edges and weights (source destination weight):" << endl;

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

{

int src, dest, weight;

cin >> src >> dest >> weight;

graph.addEdge(src, dest, weight, i);

}

int akshayHouse;

cout << "Enter the location of Akshay's house: ";

cin >> akshayHouse;

graph.bellmanFord(akshayHouse);

return 0;

}

**OUTPUT:**

Enter the number of vertices in the graph: 6

Enter the number of edges in the graph: 4

Enter the edges and weights (source destination weight):

1 2 0

2 3 4

3 4 1

4 2 1

Enter the location of Akshay's house: 2

Shortest paths from friends' locations to Akshay's house:

From friend 0: No path exists

From friend 1: No path exists

From friend 3: 2 -> 3, Distance: 4

From friend 4: 2 -> 3 -> 4, Distance: 5

From friend 5: No path exists

/\*

NAME: SHRISTY CHAUDHARY

ROLL No.:01

SECTION: AI & ML

**PROBLEM STATEMENT 21:**

Given a directed graph with two vertices ( source and destination). Design

an algorithm and implement it using a program to find the weight of the

shortest path from source to destination with exactly k edges on the path.

\*/

**CODE:**

#include <iostream>

#include <limits.h>

using namespace std;

#define MAX\_VERTICES 100

#define INF INT\_MAX

class Graph

{

int numVertices;

int adjMatrix[MAX\_VERTICES][MAX\_VERTICES];

public:

Graph(int vertices)

{

numVertices = vertices;

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

{

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

{

adjMatrix[i][j] = INF;

}

}

}

void addEdge(int src, int dest, int weight)

{

adjMatrix[src][dest] = weight;

}

int shortestPath(int src, int dest, int k)

{

int dp[MAX\_VERTICES][MAX\_VERTICES];

for (int e = 0; e <= k; e++)

{

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

{

dp[e][v] = INF;

if (e == 0 && v == src)

dp[e][v] = 0;

if (e == 1 && adjMatrix[src][v] != INF)

dp[e][v] = adjMatrix[src][v];

if (e > 1)

{

for (int u = 0; u < numVertices; u++)

{

if (adjMatrix[u][v] != INF && u != v && dp[e - 1][u] != INF)

dp[e][v] = std::min(dp[e][v], adjMatrix[u][v] + dp[e - 1][u]);

}

}

}

}

return dp[k][dest];

}

};

int main()

{

int numVertices;

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

cin >> numVertices;

Graph graph(numVertices);

int numEdges;

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

cin >> numEdges;

cout << "Enter the edges and weights (source destination weight):" << endl;

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

{

int src, dest, weight;

cin >> src >> dest >> weight;

graph.addEdge(src, dest, weight);

}

int src, dest, k;

cout << "Enter the source vertex: ";

cin >> src;

cout << "Enter the destination vertex: ";

cin >> dest;

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

cin >> k;

int shortestWeight = graph.shortestPath(src, dest, k);

if (shortestWeight == INF)

cout << "No path exists with exactly " << k << " edges from source to destination." << endl;

else

cout << "Shortest weight of the path with exactly " << k << " edges from source to destination: "<< shortestWeight << endl;

return 0;

}

**OUTPUT:**

Enter the number of vertices in the graph: 4

Enter the number of edges in the graph: 3

Enter the edges and weights (source destination weight):

1 2 1

2 2 0

2 1 3

Enter the source vertex: 1

Enter the destination vertex: 2

Enter the number of edges on the path: 1

Shortest weight of the path with exactly 1 edges from source to destination: 1

/\*

NAME: SHRISTY CHAUDHARY

ROLL No.:01

SECTION: AI & ML

**PROBLEM STATEMENT 22:**

Assume that a project of road construction to connect some cities is given

to your friend. Map of these cities and roads which will connect them (after

construction) is provided to him in the form of a graph. Certain amount of

rupees is associated with construction of each road. Your friend has to calculate

the minimum budget required for this project. The budget should be designed in

such a way that the cost of connecting the cities should be minimum and number

of roads required to connect all the cities should be minimum (if there are N

cities then only N-1 roads need to be constructed). He asks you for help. Now,

you have to help your friend by designing an algorithm which will find minimum

cost required to connect these cities. (use Prim's algorithm)

\*/

**CODE:**

#include <iostream>

#include <limits.h>

using namespace std;

#define MAX\_VERTICES 100

class Graph

{

int numVertices;

int adjMatrix[MAX\_VERTICES][MAX\_VERTICES];

public:

Graph(int vertices)

{

numVertices = vertices;

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

{

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

{

adjMatrix[i][j] = INT\_MAX;

}

}

}

void addEdge(int src, int dest, int weight)

{

adjMatrix[src][dest] = weight;

adjMatrix[dest][src] = weight;

}

int minKey(int key[], bool mstSet[])

{

int min = INT\_MAX, minIndex;

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

{

if (!mstSet[v] && key[v] < min)

{

min = key[v];

minIndex = v;

}

}

return minIndex;

}

void printMST(int parent[], int cost)

{

cout << "Minimum Cost to connect all cities: " << cost << endl;

cout << "Minimum Spanning Tree (MST):" << endl;

for (int v = 1; v < numVertices; v++)

{

cout << parent[v] << " - " << v << endl;

}

}

void primMST()

{

int parent[MAX\_VERTICES];

int key[MAX\_VERTICES];

bool mstSet[MAX\_VERTICES];

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

{

key[v] = INT\_MAX;

mstSet[v] = false;

}

key[0] = 0;

parent[0] = -1;

for (int count = 0; count < numVertices - 1; count++)

{

int u = minKey(key, mstSet);

mstSet[u] = true;

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

{

if (adjMatrix[u][v] && !mstSet[v] && adjMatrix[u][v] < key[v])

{

parent[v] = u;

key[v] = adjMatrix[u][v];

}

}

}

int cost = 0;

for (int v = 1; v < numVertices; v++)

{

cost += adjMatrix[v][parent[v]];

}

printMST(parent, cost);

}

};

int main()

{

int numVertices;

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

cin >> numVertices;

Graph graph(numVertices);

int numEdges;

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

cin >> numEdges;

cout << "Enter the roads and their construction cost (source destination cost):" << endl;

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

{

int src, dest, cost;

cin >> src >> dest >> cost;

graph.addEdge(src, dest, cost);

}

graph.primMST();

return 0;

}

**OUTPUT:**

Enter the number of cities: 4

Enter the number of roads: 6

Enter the roads and their construction cost (source destination cost):

1 2 1

2 4 0

2 3 2

4 3 6

3 2 1

4 1 0

Minimum Cost to connect all cities: 2147

Minimum Spanning Tree (MST):

0 - 1

0 - 2

0 - 3

/\*

NAME: SHRISTY CHAUDHARY

ROLL No.:01

SECTION: AI & ML

**PROBLEM STATEMENT 23:**

Implement the previous problem using Kruskal's algorithm.

\*/

**CODE:**

#include <iostream>

#include <algorithm>

using namespace std;

#define MAX\_VERTICES 100

struct Edge

{

int src, dest, weight;

};

class Graph

{

int numVertices, numEdges;

Edge edges[MAX\_VERTICES];

public:

Graph(int vertices, int edges)

{

numVertices = vertices;

numEdges = edges;

}

void addEdge(int index, int src, int dest, int weight)

{

edges[index].src = src;

edges[index].dest = dest;

edges[index].weight = weight;

}

static bool compareEdges(const Edge& edge1, const Edge& edge2)

{

return edge1.weight < edge2.weight;

}

int find(int parent[], int vertex)

{

if (parent[vertex] == vertex)

return vertex;

return find(parent, parent[vertex]);

}

void unionSets(int parent[], int rank[], int src, int dest)

{

int srcRoot = find(parent, src);

int destRoot = find(parent, dest);

if (rank[srcRoot] < rank[destRoot])

parent[srcRoot] = destRoot;

else if (rank[srcRoot] > rank[destRoot])

parent[destRoot] = srcRoot;

else

{

parent[destRoot] = srcRoot;

rank[srcRoot]++;

}

}

void kruskalMST()

{

Edge mst[MAX\_VERTICES];

int parent[MAX\_VERTICES], rank[MAX\_VERTICES];

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

{

parent[i] = i;

rank[i] = 0;

}

sort(edges, edges + numEdges, compareEdges);

int mstIndex = 0;

int edgeIndex = 0;

while (mstIndex < numVertices - 1 && edgeIndex < numEdges)

{

Edge nextEdge = edges[edgeIndex++];

int srcRoot = find(parent, nextEdge.src);

int destRoot = find(parent, nextEdge.dest);

if (srcRoot != destRoot)

{

mst[mstIndex++] = nextEdge;

unionSets(parent, rank, srcRoot, destRoot);

}

}

int cost = 0;

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

{

cost += mst[i].weight;

}

printMST(mst, mstIndex, cost);

}

void printMST(const Edge mst[], int mstSize, int cost)

{

cout << "Minimum Cost to connect all cities: " << cost << endl;

cout << "Minimum Spanning Tree (MST):" << endl;

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

{

cout << mst[i].src << " - " << mst[i].dest << endl;

}

}

};

int main()

{

int numVertices;

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

cin >> numVertices;

int numEdges;

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

cin >> numEdges;

Graph graph(numVertices, numEdges);

cout << "Enter the roads and their construction cost (source destination cost):" << endl;

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

{

int src, dest, cost;

cin >> src >> dest >> cost;

graph.addEdge(i, src, dest, cost);

}

graph.kruskalMST();

return 0;

}

**OUTPUT:**

Enter the number of cities: 4

Enter the number of roads: 3

Enter the roads and their construction cost (source destination cost):

1 2 4

2 3 1

4 1 0

Minimum Cost to connect all cities: 5

Minimum Spanning Tree (MST):

4 - 1

2 - 3

1 – 2

/\*

NAME: SHRISTY CHAUDHARY

ROLL No.:01

SECTION: AI & ML

**PROBLEM STATEMENT 24:**

Assume that same road construction project is given to another person. The

amount he will earn from this project is directly proportional to the budget

of the project. This person is greedy, so he decided to maximize the budget

by constructing those roads who have highest construction cost. Design an

algorithm and implement it using a program to find the maximum budget required

for the project.

\*/

**CODE:**

#include <iostream>

#include <algorithm>

using namespace std;

#define MAX\_VERTICES 100

struct Edge

{

int src, dest, weight;

};

class Graph

{

int numVertices, numEdges;

Edge edges[MAX\_VERTICES];

public:

Graph(int vertices, int edges)

{

numVertices = vertices;

numEdges = edges;

}

void addEdge(int index, int src, int dest, int weight)

{

edges[index].src = src;

edges[index].dest = dest;

edges[index].weight = weight;

}

static bool compareEdges(const Edge& edge1, const Edge& edge2)

{

return edge1.weight > edge2.weight;

}

int find(int parent[], int vertex)

{

if (parent[vertex] == vertex)

return vertex;

return find(parent, parent[vertex]);

}

void unionSets(int parent[], int rank[], int src, int dest)

{

int srcRoot = find(parent, src);

int destRoot = find(parent, dest);

if (rank[srcRoot] < rank[destRoot])

parent[srcRoot] = destRoot;

else if (rank[srcRoot] > rank[destRoot])

parent[destRoot] = srcRoot;

else

{

parent[destRoot] = srcRoot;

rank[srcRoot]++;

}

}

void kruskalMST()

{

Edge mst[MAX\_VERTICES];

int parent[MAX\_VERTICES], rank[MAX\_VERTICES];

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

{

parent[i] = i;

rank[i] = 0;

}

sort(edges, edges + numEdges, compareEdges);

int mstIndex = 0;

int edgeIndex = 0;

while (mstIndex < numVertices - 1 && edgeIndex < numEdges)

{

Edge nextEdge = edges[edgeIndex++];

int srcRoot = find(parent, nextEdge.src);

int destRoot = find(parent, nextEdge.dest);

if (srcRoot != destRoot)

{

mst[mstIndex++] = nextEdge;

unionSets(parent, rank, srcRoot, destRoot);

}

}

int budget = 0;

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

{

budget += mst[i].weight;

}

printMST(mst, mstIndex, budget);

}

void printMST(const Edge mst[], int mstSize, int budget)

{

cout << "Maximum Budget for the project: " << budget << endl;

cout << "Maximum Spanning Tree (MST):" << endl;

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

{

cout << mst[i].src << " - " << mst[i].dest << endl;

}

}

};

int main()

{

int numVertices;

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

cin >> numVertices;

int numEdges;

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

cin >> numEdges;

Graph graph(numVertices, numEdges);

cout << "Enter the roads and their construction cost (source destination cost):" << endl;

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

{

int src, dest, cost;

cin >> src >> dest >> cost;

graph.addEdge(i, src, dest, cost);

}

graph.kruskalMST();

return 0;

}

**OUTPUT:**

Enter the number of cities: 4

Enter the number of roads: 5

Enter the roads and their construction cost (source destination cost):

1 2 1

2 1 3

3 4 2

4 3 3

4 1 2

Maximum Budget for the project: 8

Maximum Spanning Tree (MST):

2 - 1

4 - 3

4 – 1

/\*

NAME: SHRISTY CHAUDHARY

ROLL No.:01

SECTION: AI & ML

**PROBLEM STATEMENT 25:**

Given a graph, Design an algorithm and implement it using a program to

implement Floyd-Warshall all pair shortest path algorithm.

\*/

**CODE:**

#include <iostream>

#include <climits>

using namespace std;

#define INF INT\_MAX

class Graph

{

int numVertices;

int\*\* adjacencyMatrix;

public:

Graph(int vertices)

{

numVertices = vertices;

// Create the adjacency matrix

adjacencyMatrix = new int\*[numVertices];

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

{

adjacencyMatrix[i] = new int[numVertices];

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

{

adjacencyMatrix[i][j] = INF;

}

}

}

void addEdge(int src, int dest, int weight)

{

adjacencyMatrix[src][dest] = weight;

}

void floydWarshall()

{

int dist[numVertices][numVertices];

// Initialize the distance matrix with the values of the adjacency matrix

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

{

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

{

dist[i][j] = adjacencyMatrix[i][j];

}

}

// Update the distance matrix by considering all intermediate vertices

for (int k = 0; k < numVertices; k++)

{

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

{

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

{

if (dist[i][k] != INF && dist[k][j] != INF && dist[i][k] + dist[k][j] < dist[i][j])

{

dist[i][j] = dist[i][k] + dist[k][j];

}

}

}

}

printSolution(dist);

}

void printSolution(int dist[][100])

{

cout << "Shortest distances between all pairs of vertices:" << endl;

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

{

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

{

if (dist[i][j] == INF)

{

cout << "INF ";

}

else

{

cout << dist[i][j] << " ";

}

}

cout << endl;

}

}

};

int main()

{

int numVertices;

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

cin >> numVertices;

Graph graph(numVertices);

int numEdges;

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

cin >> numEdges;

cout << "Enter the edges and their weights (source destination weight):" << endl;

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

{

int src, dest, weight;

cin >> src >> dest >> weight;

graph.addEdge(src, dest, weight);

}

graph.floydWarshall();

return 0;

}

**OUTPUT:**

Enter the number of vertices: 4

Enter the edges:

[0][0]: 0

[0][1]: 12

[0][2]: 45

[0][3]: 2

[1][0]: 1

[1][1]: 0

[1][2]: 45

[1][3]: 32

[2][0]: 77

[2][1]: 43

[2][2]: 0

[2][3]: 2

[3][0]: 42

[3][1]: 3

[3][2]: 88

[3][3]: 0

The original graph is:

0 12 45 2

1 0 45 32

77 43 0 2

42 3 88 0

The shortest path matrix is:

0 5 45 2

1 0 45 3

6 5 0 2

4 3 48 0

/\*

NAME: SHRISTY CHAUDHARY

ROLL No.:01

SECTION: AI & ML

**PROBLEM STATEMENT 26:**

Given a knapsack of maximum capacity w. N items are provided, each having its

own value and weight. You have to Design an algorithm and implement it using

a program to find the list of the selected items such that the final selected

content has weight w and has maximum value. You can take fractions of items,i.e.

the items can be broken into smaller pieces so that you have to carry only a

fraction xi of item i, where 0 ≤xi≤ 1.

\*/

**CODE:**

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

struct Item

{

int weight;

int value;

double ratio;

};

bool compareItems(const Item& item1, const Item& item2)

{

return item1.ratio > item2.ratio;

}

void knapsack(int maxCapacity, vector<Item>& items)

{

sort(items.begin(), items.end(), compareItems);

double totalWeight = 0.0;

double totalValue = 0.0;

vector<double> fractions(items.size(), 0.0);

for (int i = 0; i < items.size(); i++)

{

if (totalWeight + items[i].weight <= maxCapacity)

{

fractions[i] = 1.0;

totalWeight += items[i].weight;

totalValue += items[i].value;

}

else

{

double remainingCapacity = maxCapacity - totalWeight;

fractions[i] = remainingCapacity / items[i].weight;

totalWeight += remainingCapacity;

totalValue += items[i].value \* fractions[i];

break; // Knapsack is full

}

}

cout << "Selected Items:" << endl;

for (int i = 0; i < items.size(); i++)

{

if (fractions[i] > 0.0)

{

cout << "Item " << (i + 1) << ": " << fractions[i] << " fraction" << endl;

}

}

cout << "Total Weight: " << totalWeight << endl;

cout << "Total Value: " << totalValue << endl;

}

int main()

{

int maxCapacity;

cout << "Enter the maximum capacity of the knapsack: ";

cin >> maxCapacity;

int numItems;

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

cin >> numItems;

vector<Item> items(numItems);

cout << "Enter the weight and value for each item:" << endl;

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

{

cout << "Item " << (i + 1) << ":" << endl;

cout << "Weight: ";

cin >> items[i].weight;

cout << "Value: ";

cin >> items[i].value;

items[i].ratio = static\_cast<double>(items[i].value) / items[i].weight;

}

knapsack(maxCapacity, items);

return 0;

}

**OUTPUT:**

Enter the maximum capacity of the knapsack: 5

Enter the number of items: 4

Enter the weight and value for each item:

Item 1:

Weight: 2

Value: 1

Item 2:

Weight: 3

Value: 2

Item 3:

Weight: 6

Value: 4

Item 4:

Weight: 2

Value: 4

Selected Items:

Item 1: 1 fraction

Item 2: 1 fraction

Total Weight: 5

Total Value: 6

/\*

NAME: SHRISTY CHAUDHARY

ROLL No.:01

SECTION: AI & ML

**PROBLEM STATEMENT 27:**

Given an array of elements. Assume arr[i] represents the size of file i.

Write an algorithm and a program to merge all these files into single file

with minimum computation. For given two files A and B with sizes m and n,

computation cost of merging them is O(m+n). (Hint: use greedy approach)

\*/

**CODE:**

#include <iostream>

#include <queue>

using namespace std;

int minimumComputationCost(int arr[], int n)

{

priority\_queue<int, vector<int>, greater<int>> minHeap;

// Insert all file sizes into the min-heap

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

{

minHeap.push(arr[i]);

}

int totalComputation = 0;

// Merge files until there is only one file remaining

while (minHeap.size() > 1)

{

// Extract the two smallest file sizes

int file1 = minHeap.top();

minHeap.pop();

int file2 = minHeap.top();

minHeap.pop();

// Merge the two files and calculate the computation cost

int mergedFile = file1 + file2;

totalComputation += mergedFile;

// Insert the merged file size back into the min-heap

minHeap.push(mergedFile);

}

return totalComputation;

}

int main()

{

int n;

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

cin >> n;

int arr[n];

cout << "Enter the sizes of the files: ";

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

{

cin >> arr[i];

}

int minComputation = minimumComputationCost(arr, n);

cout << "Minimum Computation Cost: " << minComputation << endl;

return 0;

}

**OUTPUT:**

Enter the number of files: 5

Enter the sizes of the files: 2 3 4 1 6

Minimum Computation Cost: 35

/\*

NAME: SHRISTY CHAUDHARY

ROLL No.:01

SECTION: AI & ML

**PROBLEM STATEMENT 28:**

Given a list of activities with their starting time and finishing time. Your

goal is to select maximum number of activities that can be performed by a

single person such that selected activities must be non-conflicting. Any

activity is said to be non-conflicting if starting time of an activity is

greater than or equal to the finishing time of the other activity. Assume

that a person can only work on a single activity at a time.

\*/

**CODE:**

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

struct Activity

{

int start;

int finish;

};

bool compareActivities(const Activity& activity1, const Activity& activity2)

{

return activity1.finish < activity2.finish;

}

void activitySelection(vector<Activity>& activities)

{

sort(activities.begin(), activities.end(), compareActivities);

vector<Activity> selectedActivities;

selectedActivities.push\_back(activities[0]);

int lastSelected = 0;

for (int i = 1; i < activities.size(); i++)

{

if (activities[i].start >= activities[lastSelected].finish)

{

selectedActivities.push\_back(activities[i]);

lastSelected = i;

}

}

cout << "Selected Activities:" <<endl;

for (const auto& activity : selectedActivities)

{

cout << "Start: " << activity.start << ", Finish: " << activity.finish << endl;

}

}

int main()

{

int n;

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

cin >> n;

vector<Activity> activities(n);

cout << "Enter the starting and finishing times for each activity:" << endl;

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

{

cout << "Activity " << (i + 1) << ":" << endl;

cout << "Starting Time: ";

cin >> activities[i].start;

cout << "Finishing Time: ";

cin >> activities[i].finish;

}

activitySelection(activities);

return 0;

}

**OUTPUT:**

Enter the number of activities: 5

Enter the starting and finishing times for each activity:

Activity 1:

Starting Time: 4

Finishing Time: 2

Activity 2:

Starting Time: 5

Finishing Time: 6

Activity 3:

Starting Time: 1

Finishing Time: 5

Activity 4:

Starting Time: 6

Finishing Time: 8

Activity 5:

Starting Time: 2

Finishing Time: 4

Selected Activities:

Start: 4, Finish: 2

Start: 2, Finish: 4

Start: 5, Finish: 6

Start: 6, Finish: 8

/\*

NAME: SHRISTY CHAUDHARY

ROLL No.:01

SECTION: AI & ML

**PROBLEM STATEMENT 29:**

Given a long list of tasks. Each task takes specific time to accomplish it

and each task has a deadline associated with it. You have to design an

algorithm and implement it using a program to find maximum number of tasks

that can be completed without crossing their deadlines and also find list

of selected tasks.

\*/

**CODE:**

#include <iostream>

#include <algorithm>

using namespace std;

struct Task

{

int time;

int deadline;

};

bool compareTasks(const Task& task1, const Task& task2)

{

return task1.deadline < task2.deadline;

}

void taskScheduling(Task tasks[], int n)

{

sort(tasks, tasks + n, compareTasks);

int maxTasks = 0;

int currentTime = 0;

Task selectedTasks[n];

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

{

if (currentTime + tasks[i].time <= tasks[i].deadline)

{

maxTasks++;

selectedTasks[maxTasks - 1] = tasks[i];

currentTime += tasks[i].time;

}

}

cout << "Maximum Number of Tasks: " << maxTasks << endl;

cout << "Selected Tasks:" << endl;

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

{

cout << "Time: " << selectedTasks[i].time << ", Deadline: " << selectedTasks[i].deadline << endl;

}

}

int main()

{

int n;

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

cin >> n;

Task tasks[n];

cout << "Enter the time and deadline for each task:" << endl;

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

{

cout << "Task " << (i + 1) << ":" << endl;

cout << "Time: ";

cin >> tasks[i].time;

cout << "Deadline: ";

cin >> tasks[i].deadline;

}

taskScheduling(tasks, n);

return 0;

}

**OUTPUT:**

Enter the number of tasks: 5

Enter the time and deadline for each task:

Task 1:

Time: 4

Deadline: 6

Task 2:

Time: 1

Deadline: 4

Task 3:

Time: 5

Deadline: 8

Task 4:

Time: 4

Deadline: 7

Task 5:

Time: 6

Deadline: 8

Maximum Number of Tasks: 2

Selected Tasks:

Time: 1, Deadline: 4

Time: 4, Deadline: 6

/\*

NAME: SHRISTY CHAUDHARY

ROLL No.:01

SECTION: AI & ML

**PROBLEM STATEMENT 30:**

Given an unsorted array of elements, design an algorithm and implement it

using a program to find whether majority element exists or not. Also find

median of the array. A majority element is an element that appears more than

n/2 times, where n is the size of array.

\*/

**CODE:**

#include <iostream>

#include <algorithm>

using namespace std;

void findMajorityElementAndMedian(int arr[], int n)

{

sort(arr, arr + n);

int count = 1;

int maxCount = 1;

int majorityElement = arr[0];

for (int i = 1; i < n; i++)

{

if (arr[i] == arr[i - 1])

{

count++;

}

else

{

if (count > maxCount)

{

maxCount = count;

majorityEhttps:;

}

count = 1;

}

}

if (count > maxCount)

{

maxCount = count;

majorityElement = arr[n - 1];

}

if (maxCount > n / 2)

{

cout << "Majority element exists: " << majorityElement << endl;

}

else

{

cout << "Majority element does not exist." << endl;

}

int median;

if (n % 2 == 0)

{

median = (arr[n / 2 - 1] + arr[n / 2]) / 2;

}

else

{

median = arr[n / 2];

}

cout << "Median: " << median << endl;

}

int main()

{

int n;

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

cin >> n;

int arr[n];

cout << "Enter the elements of the array:" << endl;

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

{

cin >> arr[i];

}

findMajorityElementAndMedian(arr, n);

return 0;

}

**OUTPUT:**

Enter the size of the array: 5

Enter the elements of the array:

2 4 8 6 7

Majority element does not exist.

Median: 6

/\*

NAME: SHRISTY CHAUDHARY

ROLL No.:01

SECTION: AI & ML

**PROBLEM STATEMENT 31:**

Given a sequence of matrices, write an algorithm to find most efficient way

to multiply these matrices together. To find the optimal solution, you need

to find the order in which these matrices should be multiplied.

\*/

**CODE:**

#include <iostream>

#include <climits>

using namespace std;

void matrixChainMultiplication(int matrixDimensions[], int n)

{

int m[n][n];

int s[n][n];

for (int i = 1; i < n; i++)

{

m[i][i] = 0;

}

for (int l = 2; l < n; l++)

{

for (int i = 1; i < n - l + 1; i++)

{

int j = i + l - 1;

m[i][j] = INT\_MAX;

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

{

int cost = m[i][k] + m[k + 1][j] + matrixDimensions[i - 1] \* matrixDimensions[k] \* matrixDimensions[j];

if (cost < m[i][j])

{

m[i][j] = cost;

s[i][j] = k;

}

}

}

}

cout << "Optimal Order of Matrix Multiplication: ";

printOptimalOrder(s, 1, n - 1);

cout << endl;

cout << "Minimum Scalar Multiplications: " << m[1][n - 1] << endl;

}

void printOptimalOrder(int s[][10], int i, int j)

{

if (i == j)

{

cout << "A" << i;

}

else

{

cout << "(";

printOptimalOrder(s, i, s[i][j]);

printOptimalOrder(s, s[i][j] + 1, j);

cout << ")";

}

}

int main()

{

int n;

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

cin >> n;

int matrixDimensions[n];

cout << "Enter the dimensions of the matrices:" << endl;

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

{

cin >> matrixDimensions[i];

}

matrixChainMultiplication(matrixDimensions, n);

return 0;

}

**OUTPUT:**

Enter the number of matrices: 2

Enter the dimensions of the matrices:

2 3

Optimal Order of Matrix Multiplication: A1

Minimum Scalar Multiplications: 0

/\*

NAME: SHRISTY CHAUDHARY

ROLL No.:01

SECTION: AI & ML

**PROBLEM STATEMENT 32:**

Given a set of available types of coins. Let suppose you have infinite supply

of each type of coin. For a given value N, you have to Design an algorithm

and implement it using a program to find number of ways in which these coins

can be added to make sum value equals to N

\*/

**CODE:**

#include <iostream>

using namespace std;

int countWays(int coins[], int numCoins, int targetSum)

{

int dp[targetSum + 1];

dp[0] = 1;

for (int i = 1; i <= targetSum; i++)

{

dp[i] = 0;

}

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

{

for (int j = coins[i]; j <= targetSum; j++)

{

dp[j] += dp[j - coins[i]];

}

}

return dp[targetSum];

}

int main()

{

int numCoins;

cout << "Enter the number of coin types: ";

cin >> numCoins;

int coins[numCoins];

cout << "Enter the coin values:" << endl;

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

{

cin >> coins[i];

}

int targetSum;

cout << "Enter the target sum: ";

cin >> targetSum;

int ways = countWays(coins, numCoins, targetSum);

cout << "Number of ways to make a sum of " << targetSum << " using the given coins: " << ways << endl;

return 0;

}

**OUTPUT:**

Enter the number of coin types: 4

Enter the coin values:

20 4 6 70

Enter the target sum: 110

Number of ways to make a sum of 110 using the given coins: 38

/\*

NAME: SHRISTY CHAUDHARY

ROLL No.:01

SECTION: AI & ML

**PROBLEM STATEMENT 33:**

Given a set of elements, you have to partition the set into two subsets such

that the sum of elements in both subsets is same. Design an algorithm and

implement it using a program to solve this problem.

\*/

**CODE:**

#include <iostream>

using namespace std;

bool canPartition(int arr[], int n)

{

int sum = 0;

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

{

sum += arr[i];

}

if (sum % 2 != 0)

{

return false;

}

int targetSum = sum / 2;

bool dp[n + 1][targetSum + 1];

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

{

dp[i][0] = true;

}

for (int i = 1; i <= targetSum; i++)

{

dp[0][i] = false;

}

for (int i = 1; i <= n; i++)

{

for (int j = 1; j <= targetSum; j++)

{

if (arr[i - 1] == j)

{

dp[i][j] = true;

}

else if (arr[i - 1] > j)

{

dp[i][j] = dp[i - 1][j];

}

else

{

dp[i][j] = dp[i - 1][j] || dp[i - 1][j - arr[i - 1]];

}

}

}

return dp[n][targetSum];

}

int main()

{

int n;

cout << "Enter the number of elements in the set: ";

cin >> n;

int arr[n];

cout << "Enter the elements of the set: ";

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

{

cin >> arr[i];

}

if (canPartition(arr, n))

{

cout << "The set can be partitioned into two subsets with equal sums." << endl;

}

else

{

cout << "The set cannot be partitioned into two subsets with equal sums." << endl;

}

return 0;

}

**OUTPUT:**

Enter the number of elements in the set: 8

Enter the elements of the set: 5 3 4 7 8 9 14 20

The set can be partitioned into two subsets with equal sums.

/\*

NAME: SHRISTY CHAUDHARY

ROLL No.:01

SECTION: AI & ML

**PROBLEM STATEMENT 34:**

Given two sequences, Design an algorithm and implement it using a program to

find the length of longest subsequence present in both of them. A subsequence

is a sequence that appears in the same relative order, but not necessarily

contiguous.

\*/

**CODE:**

#include <iostream>

using namespace std;

int longestCommonSubsequenceLength(const string& seq1, const string& seq2)

{

int m = seq1.length();

int n = seq2.length();

int dp[m + 1][n + 1];

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

{

dp[i][0] = 0;

}

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

{

dp[0][j] = 0;

}

for (int i = 1; i <= m; i++)

{

for (int j = 1; j <= n; j++)

{

if (seq1[i - 1] == seq2[j - 1])

{

dp[i][j] = dp[i - 1][j - 1] + 1;

}

else

{

dp[i][j] = std::max(dp[i - 1][j], dp[i][j - 1]);

}

}

}

return dp[m][n];

}

int main()

{

string seq1, seq2;

cout << "Enter the first sequence: ";

cin >> seq1;

cout << "Enter the second sequence: ";

cin >> seq2;

int length = longestCommonSubsequenceLength(seq1, seq2);

cout << "The length of the longest common subsequence is: " << length << endl;

return 0;

}

**OUTPUT:**

Enter the first sequence: AGHRTUKUK

Enter the second sequence: SNFHEISWKLOAW

The length of the longest common subsequence is: 2

/\*

NAME: SHRISTY CHAUDHARY

ROLL No.:01

SECTION: AI & ML

**PROBLEM STATEMENT 35:**

Given a knapsack of maximum capacity w. N items are provided, each having its

own value and weight. Design an algorithm and implement it using a program to

find the list of the selected items such that the final selected content has

weight <= w and has maximum value. Here, you cannot break an item i.e. either

pick the complete item or don't pick it. (0-1 property).

\*/

**CODE:**

#include <iostream>

#include <vector>

using namespace std;

struct Item

{

int weight;

int value;

};

void knapsack(int capacity, const vector<Item>& items)

{

int n = items.size();

vector<vector<int>> dp(n + 1, vector<int>(capacity + 1, 0));

for (int i = 1; i <= n; i++)

{

for (int w = 1; w <= capacity; w++)

{

if (items[i - 1].weight <= w)

{

dp[i][w] = max(items[i - 1].value + dp[i - 1][w - items[i - 1].weight], dp[i - 1][w]);

}

else

{

dp[i][w] = dp[i - 1][w];

}

}

}

int maxValue = dp[n][capacity];

vector<Item> selectedItems;

int w = capacity;

for (int i = n; i > 0 && maxValue > 0; i--)

{

if (maxValue != dp[i - 1][w])

{

selectedItems.push\_back(items[i - 1]);

maxValue -= items[i - 1].value;

w -= items[i - 1].weight;

}

}

cout << "Selected Items:\n";

for (const auto& item : selectedItems)

{

cout << "Weight: " << item.weight << ", Value: " << item.value << endl;

}

cout << "Total Value: " << dp[n][capacity] << endl;

}

int main()

{

int capacity;

cout << "Enter the maximum capacity of the knapsack: ";

cin >> capacity;

int n;

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

cin >> n;

vector<Item> items(n);

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

{

cout << "Enter the weight and value of item " << i + 1 << ": ";

cin >> items[i].weight >> items[i].value;

}

knapsack(capacity, items);

return 0;

}

**OUTPUT:**

Enter the maximum capacity of the knapsack: 5

Enter the number of items: 4

Enter the weight and value of item 1: 2 5

Enter the weight and value of item 2: 4 1

Enter the weight and value of item 3: 2 6

Enter the weight and value of item 4: 6 4

Selected Items:

Weight: 2, Value: 6

Weight: 2, Value: 5

Total Value: 11

/\*

NAME: SHRISTY CHAUDHARY

ROLL No.:01

SECTION: AI & ML

**PROBLEM STATEMENT 36:**

Given a string of characters, design an algorithm and implement it using a

program to print all possible permutations of the string in lexicographic order

\*/

**CODE:**

#include <iostream>

#include <cstring>

using namespace std;

void permute(char\* str, int start, int length)

{

if (start == length - 1)

{

cout << str << endl;

return;

}

for (int i = start; i < length; i++)

{

swap(str[i], str[start]);

permute(str, start + 1, length);

swap(str[i], str[start]);

}

}

int main()

{

char str[100];

cout << "Enter a string: ";

cin >> str;

cout << "All possible permutations in lexicographic order:\n";

permute(str, 0, strlen(str));

return 0;

}

**OUTPUT:**

Enter a string: ABC

All possible permutations in lexicographic order:

ABC

ACB

BAC

BCA

CBA

CAB

/\*

NAME: SHRISTY CHAUDHARY

ROLL No.:01

SECTION: AI & ML

**PROBLEM STATEMENT 37:**

Given an array of characters, you have to find distinct characters from this

array. Design an algorithm and implement it using a program to solve this

problem using hashing. (Time Complexity = O(n))

\*/

**CODE:**

#include <iostream>

using namespace std;

#include <unordered\_set>

void findDistinctChars(char arr[], int n)

{

unordered\_set<char> charSet;

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

{

if (charSet.find(arr[i]) == charSet.end())

{

charSet.insert(arr[i]);

}

}

cout << "Distinct characters: ";

for (const auto& ch : charSet)

{

cout << ch << " ";

}

cout << endl;

}

int main()

{

int n;

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

cin >> n;

char arr[n];

cout << "Enter the elements of the array: ";

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

{

cin >> arr[i];

}

findDistinctChars(arr, n);

return 0;

}

**OUTPUT:**

Enter the size of the array: 8

Enter the elements of the array: a b c t a f t w

Distinct characters: w f t c b a

/\*

NAME: SHRISTY CHAUDHARY

ROLL No.:01

SECTION: AI & ML

**PROBLEM STATEMENT 38:**

Given an array of integers of size n, design an algorithm and write a program

to check whether this array contains duplicate within a small window of size k < n.

\*/

**CODE:**

#include <iostream>

using namespace std;

bool containsDuplicateWithinWindow(int arr[], int n, int k)

{

for (int i = 0; i < n - k + 1; i++)

{

for (int j = i + 1; j < i + k; j++)

{

if (arr[i] == arr[j])

{

return true;

}

}

}

return false;

}

int main()

{

int n, k;

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

cin >> n;

int arr[n];

cout << "Enter the elements of the array: ";

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

{

cin >> arr[i];

}

cout << "Enter the window size (k): ";

cin >> k;

bool hasDuplicates = containsDuplicateWithinWindow(arr, n, k);

if (hasDuplicates)

{

cout << "The array contains duplicates within a window of size " << k << endl;

}

else

{

cout << "The array does not contain duplicates within a window of size " << k << endl;

}

return 0;

}

**OUTPUT:**

Enter the size of the array: 4

Enter the elements of the array: 5 6 4 7

Enter the window size (k): 5

The array does not contain duplicates within a window of size 5

/\*

NAME: SHRISTY CHAUDHARY

ROLL No.:01

SECTION: AI & ML

**PROBLEM STATEMENT 39:**

Given an array of nonnegative integers, Design an algorithm and implement

it using a program to find two pairs (a,b) and (c,d) such that a\*b = c\*d,

where a, b, c and d are distinct elements of array.

\*/

**CODE:**

#include <iostream>

using namespace std;

bool findEqualProductPairs(int arr[], int n)

{

for (int i = 0; i < n - 1; i++)

{

for (int j = i + 1; j < n; j++)

{

int product1 = arr[i] \* arr[j];

for (int k = 0; k < n - 1; k++)

{

for (int l = k + 1; l < n; l++)

{

int product2 = arr[k] \* arr[l];

if (product1 == product2 && arr[i] != arr[j] && arr[i] != arr[k] && arr[i] != arr[l] && arr[j] != arr[k] && arr[j] != arr[l] && arr[k] != arr[l])

{

cout << "Pairs found: (" << arr[i] << ", " << arr[j] << ") and (" << arr[k] << ", " << arr[l] << ")" << endl;

return true;

}

}

}

}

}

return false;

}

int main()

{

int n;

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

cin >> n;

int arr[n];

cout << "Enter the elements of the array: ";

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

{

cin >> arr[i];

}

bool foundPairs = findEqualProductPairs(arr, n);

if (!foundPairs)

{

cout << "No pairs found" << endl;

}

return 0;

}

**OUTPUT:**

Enter the size of the array: 10

Enter the elements of the array: 31 23 4 1 39 2 20 27 8 10

Pairs found: (4, 2) and (1, 8)

/\*

NAME: SHRISTY CHAUDHARY

ROLL No.:01

SECTION: AI & ML

**PROBLEM STATEMENT 40:**

Given a number n, write an algorithm and a program to find nth ugly number.

Ugly numbers are those numbers whose only prime factors are 2, 3 or 5. The

sequence 1, 2, 3, 4, 5, 6, 8, 9, 10, 12, 15, 16, 18, 20, 24,..... is sequence

of ugly numbers

\*/

**CODE:**

#include <iostream>

using namespace std;

int getNthUglyNumber(int n)

{

int ugly[n]; // Array to store ugly numbers

ugly[0] = 1; // First ugly number

int p2 = 0, p3 = 0, p5 = 0; // Pointers to previous ugly numbers

int nextMultipleOf2 = 2, nextMultipleOf3 = 3, nextMultipleOf5 = 5; // Next multiples

for (int i = 1; i < n; i++)

{

int nextUglyNumber = min(nextMultipleOf2, min(nextMultipleOf3, nextMultipleOf5));

ugly[i] = nextUglyNumber;

if (nextUglyNumber == nextMultipleOf2)

{

p2++;

nextMultipleOf2 = ugly[p2] \* 2;

}

if (nextUglyNumber == nextMultipleOf3)

{

p3++;

nextMultipleOf3 = ugly[p3] \* 3;

}

if (nextUglyNumber == nextMultipleOf5)

{

p5++;

nextMultipleOf5 = ugly[p5] \* 5;

}

}

return ugly[n - 1];

}

int main()

{

int n;

cout << "Enter the value of n: ";

cin >> n;

int nthUglyNumber = getNthUglyNumber(n);

cout << "The " << n << "th ugly number is: " << nthUglyNumber << endl;

return 0;

}

**OUTPUT:**

Enter the value of n: 20

The 20th ugly number is: 36

/\*

NAME: SHRISTY CHAUDHARY

ROLL No.:01

SECTION: AI & ML

**PROBLEM STATEMENT 41:**

Given a directed graph, write an algorithm and a program to find mother vertex

in a graph. A mother vertex is a vertex v such that there exists a path from

v to all other vertices of the graph.

\*/

**CODE:**

#include <iostream>

#define MAX\_VERTICES 1000

using namespace std;

class Graph

{

int numVertices;

int adjacencyMatrix[MAX\_VERTICES][MAX\_VERTICES];

public:

Graph(int n) : numVertices(n)

{

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

{

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

{

adjacencyMatrix[i][j] = 0;

}

}

}

void addEdge(int u, int v)

{

adjacencyMatrix[u][v] = 1;

}

void DFS(int v, bool visited[])

{

visited[v] = true;

for (int u = 0; u < numVertices; u++)

{

if (adjacencyMatrix[v][u] == 1 && !visited[u])

{

DFS(u, visited);

}

}

}

int findMotherVertex()

{

bool visited[MAX\_VERTICES] = {false};

int lastFinished = 0;

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

{

if (!visited[v])

{

DFS(v, visited);

lastFinished = v;

}

}

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

{

visited[i] = false;

}

DFS(lastFinished, visited);

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

{

if (!visited[i])

{

return -1;

}

}

return lastFinished;

}

};

int main()

{

int numVertices, numEdges;

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

cin >> numVertices;

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

cin >> numEdges;

Graph graph(numVertices);

cout << "Enter the edges:\n";

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

{

int u, v;

cin >> u >> v;

graph.addEdge(u, v);

}

int motherVertex = graph.findMotherVertex();

if (motherVertex != -1)

{

cout << "The mother vertex is: " << motherVertex << endl;

}

else

{

cout << "No mother vertex found." << endl;

}

return 0;

}

**OUTPUT:**

Enter the number of vertices: 6

Enter the number of edges: 6

Enter the edges:

0 3

0 4

3 1

4 2

1 5

4 5

The mother vertex is: 0