S.No: 1	Exp. Name: ***Write a C program to Sort the given elements in Ascending order using Bubble Sort***	**Date: 2023-02-24**

ID: 21SCSE1010303

E2UC403B-2023-Section-14

Galgotias University Greater Noida

Aim:

Write a program to `sort` (`Ascending order`) the given elements using `bubble sort technique`.

At the time of execution, the program should print the message on the console as:

```
Enter value of n :
```

For example, if the user gives the **input** as:

```
Enter value of n : 3
```

Next, the program should print the messages one by one on the console as:

```
Enter element for a[0] :
Enter element for a[1] :
Enter element for a[2] :
```

if the user gives the **input** as:

```
Enter element for a[0] : 22
Enter element for a[1] : 33
Enter element for a[2] : 12
```

then the program should **print** the result as:

```
Before sorting the elements in the array are
Value of a[0] = 22
Value of a[1] = 33
Value of a[2] = 12
After sorting the elements in the array are
Value of a[0] = 12
Value of a[1] = 22
Value of a[2] = 33
```

Note: Do use the **printf()** function with a **newline** character (`\n`).

Source Code:

Program504.c

```
#include<stdio.h>
void swap(int *xp, int *yp)
{
        int temp =*xp;
        *xp= *yp;
        *yp= temp;
}
void bubbleSort(int arr[], int n)
{
        int i, j;
        for (i=0; i<n -1; i++)
        for (j=0; j<n-i-1;j++)
        if (arr[j]> arr[j+1])
        swap(&arr[j], &arr[j+1]);

}
void printpreArray(int arr[], int size)
{
        int j;
        printf("Before sorting the elements in the array are\n");
        for(j=0; j< size; j++)
        printf("Value of a[%d] = %d\n",j,arr[j]);
        //printf("\n");
}
void printArray(int arr[], int size)
{
        int i;
        printf("After sorting the elements in the array are\n");
        for (i=0; i<size; i++)
        printf("Value of a[%d] = %d\n",i,arr[i]);
        //printf("\n");
}
        int main()
        {
                int arr[10];
                int i,n;
                printf("Enter value of n : ");
                scanf("%d",&n);
                for(i=0; i<n; i++)
                { printf("Enter element for a[%d] : ", i);
                scanf("%d",&arr[i]);
                }

        printpreArray(arr,n);
        bubbleSort(arr,n);
        printArray(arr,n);
        return 0;
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output

```
Enter value of n :
5
Enter element for a[0] :
2
Enter element for a[1] :
7
Enter element for a[2] :
6
Enter element for a[3] :
4
Enter element for a[4] :
1
Before sorting the elements in the array are
Value of a[0] = 2
Value of a[1] = 7
Value of a[2] = 6
Value of a[3] = 4
Value of a[4] = 1
After sorting the elements in the array are
Value of a[0] = 1
Value of a[1] = 2
Value of a[2] = 4
Value of a[3] = 6
Value of a[4] = 7
```

Test Case - 2

User Output

```
Enter value of n :
4
Enter element for a[0] :
28
Enter element for a[1] :
34
Enter element for a[2] :
26
Enter element for a[3] :
29
Before sorting the elements in the array are
Value of a[0] = 28
Value of a[1] = 34
Value of a[2] = 26
Value of a[3] = 29
After sorting the elements in the array are
Value of a[0] = 26
Value of a[1] = 28
Value of a[2] = 29
Value of a[3] = 34
```

User Output
Enter value of n :
8
Enter element for a[0] :
7
Enter element for a[1] :
3
Enter element for a[2] :
9
Enter element for a[3] :
2
Enter element for a[4] :
5
Enter element for a[5] :
4
Enter element for a[6] :
6
Enter element for a[7] :
1
Before sorting the elements in the array are
Value of a[0] = 7
Value of a[1] = 3
Value of a[2] = 9
Value of a[3] = 2
Value of a[4] = 5
Value of a[5] = 4
Value of a[6] = 6
Value of a[7] = 1
After sorting the elements in the array are
Value of a[0] = 1
Value of a[1] = 2
Value of a[2] = 3
Value of a[3] = 4
Value of a[4] = 5
Value of a[5] = 6
Value of a[6] = 7
Value of a[7] = 9

Test Case - 4
User Output
Enter value of n :
4
Enter element for a[0] :
-23
Enter element for a[1] :
-14
Enter element for a[2] :
-56

-35
Before sorting the elements in the array are
Value of a[0] = -23
Value of a[1] = -14
Value of a[2] = -56
Value of a[3] = -35
After sorting the elements in the array are
Value of a[0] = -56
Value of a[1] = -35
Value of a[2] = -23
Value of a[3] = -14

Test Case - 5
User Output
Enter value of n :
5
Enter element for a[0] :
28
Enter element for a[1] :
45
Enter element for a[2] :
-1
Enter element for a[3] :
-5
Enter element for a[4] :
2
Before sorting the elements in the array are
Value of a[0] = 28
Value of a[1] = 45
Value of a[2] = -1
Value of a[3] = -5
Value of a[4] = 2
After sorting the elements in the array are
Value of a[0] = -5
Value of a[1] = -1
Value of a[2] = 2
Value of a[3] = 28
Value of a[4] = 45

S.No: 2	Exp. Name: ***Write a C program to Search an element using Binary Search process***	Date: 2023-02-25

Aim:

Write a program to `search` a key element in the given array of elements using `binary search`.

At the time of execution, the program should print the message on the console as:

```
Enter value of n :
```

For example, if the user gives the **input** as:

```
Enter value of n : 3
```

Next, the program should print the messages one by one on the console as:

```
Enter element for a[0] :
Enter element for a[1] :
Enter element for a[2] :
```

if the user gives the **input** as:

```
Enter element for a[0] : 89
Enter element for a[1] : 33
Enter element for a[2] : 56
```

Next, the program should print the message on the console as:

```
Enter key element :
```

if the user gives the **input** as:

```
Enter key element : 56
```

then the program should **print** the result as:

```
After sorting the elements in the array are
Value of a[0] = 33
Value of a[1] = 56
Value of a[2] = 89
The key element 56 is found at the position 1
```

Similarly if the key element is given as **25** for the above one dimensional array elements then the program should print the output as "**The Key element 25 is not found in the array**".

Note: Do use the **printf()** function with a **newline** character (`\n`) at the end.

Source Code:

Program510.c

```
#include<stdio.h>
int binarySearch(int arr[],int n,int key)
{
        int low =0,high =n-1;
        while(low<=high)
        {
                int mid=(low+high)/2;
                if(arr[mid] == key)
                {
                        return mid;
                }
                else if(arr[mid]<key)
                {
                        low=mid+1;
                }
                else
                {
                        high=mid-1;
                }
        }
        return -1;
}
int main()
{
        int i,n,key;
        printf("Enter value of n : ");
        scanf("%d",&n);
        int arr[n];
        for( i=0;i<n;i++)
        {
                printf("Enter element for a[%d] : ",i);
                scanf("%d",&arr[i]);
        }
        printf("Enter key element : ");
        scanf("%d", &key);
        for(i=0;i<n-1;i++)
        {
                int j,temp;
                for(j=i+1;j<n;j++)
                {
                        if(arr[i]>arr[j])
                        {
                                temp=arr[i];
                                arr[i]=arr[j];
                                arr[j]=temp;
                        }
                }
        }
        printf("After sorting the elements in the array are\n");
        for(i=0;i<n;i++)
        {
                printf("Value of a[%d] = %d\n",i,arr[i]);
        }
    int result = binarySearch(arr,n,key);
        if(result == -1)
```

```
        }
        else
        {
        printf("The key element %d is found at the position %d\n",key,result);
        }
        return 0;
}
```


Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter value of n :
5
Enter element for a[0] :
4
Enter element for a[1] :
8
Enter element for a[2] :
6
Enter element for a[3] :
2
Enter element for a[4] :
1
Enter key element :
8
After sorting the elements in the array are
Value of a[0] = 1
Value of a[1] = 2
Value of a[2] = 4
Value of a[3] = 6
Value of a[4] = 8
The key element 8 is found at the position 4

Test Case - 2
User Output
Enter value of n :
7
Enter element for a[0] :
56
Enter element for a[1] :
89
Enter element for a[2] :
63
Enter element for a[3] :
215

```
325
Enter element for a[5] :
156
Enter element for a[6] :
256
Enter key element :
458
After sorting the elements in the array are
Value of a[0] = 56
Value of a[1] = 63
Value of a[2] = 89
Value of a[3] = 156
Value of a[4] = 215
Value of a[5] = 256
Value of a[6] = 325
The Key element 458 is not found in the array
```

S.No: 3	Exp. Name: ***Write a C program to Sort given elements using Insertion sort***	**Date: 2023-02-28**

ID: 21SCSE1010303

E2UC403B-2023-Section-14

Galgotias University Greater Noida

Aim:

Write a program to `sort` (`Ascending order`) the given elements using `insertion sort technique`.

At the time of execution, the program should print the message on the console as:

```
Enter value of n :
```

For example, if the user gives the **input** as:

```
Enter value of n : 3
```

Next, the program should print the messages one by one on the console as:

```
Enter element for a[0] :
Enter element for a[1] :
Enter element for a[2] :
```

if the user gives the **input** as:

```
Enter element for a[0] : 22
Enter element for a[1] : 33
Enter element for a[2] : 12
```

then the program should **print** the result as:

```
Before sorting the elements in the array are
Value of a[0] = 22
Value of a[1] = 33
Value of a[2] = 12
After sorting the elements in the array are
Value of a[0] = 12
Value of a[1] = 22
Value of a[2] = 33
```

Note: Do use the **printf()** function with a **newline** character (`\n`).

Source Code:

Program505.c

```
#include<stdio.h>

int main()
{
        int n, i, j ,temp;
        printf("Enter value of n : ");
        scanf("%d", &n);
        int arr[n];

        for(i=0; i<n; i++){
                printf("Enter element for a[%d] : ", i);
                scanf("%d", &arr[i]);
        }
        printf("Before sorting the elements in the array are\n");
        for(i=0; i<n; i++){
                printf("Value of a[%d] = %d\n",i,arr[i]);
        }
        for (i=1; i<n; i++){
                temp = arr[i];
                j= i-1;

                while(j>=0 && arr[j] > temp){
                        arr[j+1] = arr[j];
                        j=j-1;
                }
                arr[j+1]= temp;
        }
printf("After sorting the elements in the array are\n");
for(i = 0; i<n; i++)
{
        printf("Value of a[%d] = %d\n",i,arr[i]);
}
return 0;
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter value of n :
5
Enter element for a[0] :
7
Enter element for a[1] :
33
Enter element for a[2] :
12
Enter element for a[3] :
56
Enter element for a[4] :
9

```
Before sorting the elements in the array are
Value of a[0] = 7
Value of a[1] = 33
Value of a[2] = 12
Value of a[3] = 56
Value of a[4] = 9
After sorting the elements in the array are
Value of a[0] = 7
Value of a[1] = 9
Value of a[2] = 12
Value of a[3] = 33
Value of a[4] = 56
```

S.No: 4	Exp. Name: ***Write a C program to Search an element using Linear Search process***	Date: 2023-02-28

Aim:

Write a program to `search` a key element with in the given array of elements using `linear search` process.

At the time of execution, the program should print the message on the console as:

```
Enter value of n :
```

For example, if the user gives the **input** as:

```
Enter value of n : 3
```

Next, the program should print the messages one by one on the console as:

```
Enter element for a[0] :
Enter element for a[1] :
Enter element for a[2] :
```

if the user gives the **input** as:

```
Enter element for a[0] : 89
Enter element for a[1] : 33
Enter element for a[2] : 56
```

Next, the program should print the message on the console as:

```
Enter key element :
```

if the user gives the **input** as:

```
Enter key element : 56
```

then the program should **print** the result as:

```
The key element 56 is found at the position 2
```

Similarly if the key element is given as **25** for the above one dimensional array elements then the program should print the output as "**The Key element 25 is not found in the array**".

Note: Do use the **printf()** function with a **newline** character (`\n`) at the end.

Source Code:

Program509.c

```
#include<stdio.h>

int search(int arr[], int N, int x)
{
        int i;
        for(i=0; i<N; i++)
        if (arr[i]==x)
        return i;
 return -1;
}
int main(void)
{
        int arr[10], n, i;
        int x;
        printf("Enter value of n : ");
        scanf("%d",&n);
        for (i=0; i<n; i++)
        {
                printf("Enter element for a[%d] : " , i);
                scanf("%d", &arr[i]);
        }
        printf("Enter key element : ");
        scanf("%d",&x);
        int result = search(arr, n, x);
        if (result == -1){
        printf("The key element %d is not found in the array", x);
        }
        else{
        printf("The key element %d is found at the position %d", x, result);
        }
        printf("\n");
        return 0;
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter value of n :
5
Enter element for a[0] :
45
Enter element for a[1] :
67
Enter element for a[2] :
35
Enter element for a[3] :
28
Enter element for a[4] :
16
Enter key element :

28
The key element 28 is found at the position 3

Test Case - 2
User Output
Enter value of n :
5
Enter element for a[0] :
2
Enter element for a[1] :
7
Enter element for a[2] :
5
Enter element for a[3] :
1
Enter element for a[4] :
4
Enter key element :
2
The key element 2 is found at the position 0

Test Case - 3
User Output
Enter value of n :
4
Enter element for a[0] :
452
Enter element for a[1] :
356
Enter element for a[2] :
754
Enter element for a[3] :
127
Enter key element :
127
The key element 127 is found at the position 3

Test Case - 4
User Output
Enter value of n :
3
Enter element for a[0] :
5
Enter element for a[1] :

3
Enter key element :
4
The key element 4 is not found in the array

Test Case - 5
User Output
Enter value of n :
3
Enter element for a[0] :
11
Enter element for a[1] :
45
Enter element for a[2] :
37
Enter key element :
25
The key element 25 is not found in the array

S.No: 5	Exp. Name: ***Write a C program to Sort given elements using Quick sort***	Date: 2023-02-28

Aim:

Write a program to `sort` (`Ascending order`) the given elements using `quick sort` technique.

Note: Pick the last element as pivot. You will not be awarded marks if you do not follow this instruction.

At the time of execution, the program should print the message on the console as:

```
Enter array size :
```

For example, if the user gives the **input** as:

```
Enter array size : 5
```

Next, the program should print the following message on the console as:

```
Enter 5 elements :
```

if the user gives the **input** as:

```
Enter 5 elements : 34 67 12 45 22
```

then the program should **print** the result as:

```
Before sorting the elements are : 34 67 12 45 22
After sorting the elements are : 12 22 34 45 67
```

Note: Do use the **printf()** function with a **newline** character (`\n`).

Source Code:

QuickSortMain.c

```
#include <stdio.h>
#include "QuickSortFunctions.c"
void main() {
        int arr[15], i, n;
        printf("Enter array size : ");
        scanf("%d", &n);
        printf("Enter %d elements : ", n);
        for (i = 0; i < n; i++) {
                scanf("%d", &arr[i]);
        }
        printf("Before sorting the elements are : ");
        display(arr, n);
        quickSort(arr, 0, n - 1);
        printf("After sorting the elements are : ");
        display(arr, n);
}
```

QuickSortFunctions.c

```
#include<stdio.h>
void display(int arr[15], int n)
{
        int i;
        for(i=0;i<n;i++)
        printf("%d ",arr[i]);
        printf("\n");
}
void quickSort(int arr[15], int lb, int ub)
{
        int i, j, temp, pivot;
        if(lb<ub)
        {
                pivot=lb;
                i=lb;
                j=ub;
                while(i<j)
                {
                        while(arr[i]<arr[pivot] && i<=ub)
                        i++;
                        while(arr[j]>arr[pivot])
                        j--;
                        if(i<j)
                        {
                                temp=arr[i];
                                arr[i]=arr[j];
                                arr[j]=temp;
                        }
                }
                temp=arr[pivot];
                arr[pivot]=arr[j];
                arr[j]=temp;
                quickSort(arr,lb,j-1);
                quickSort(arr,j+1,ub);
        }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter array size :
5
Enter 5 elements :
34 67 12 45 22
Before sorting the elements are : 34 67 12 45 22
After sorting the elements are : 12 22 34 45 67

Test Case - 2
User Output

Enter array size :
8
Enter 8 elements :
77 55 22 44 99 33 11 66
Before sorting the elements are : 77 55 22 44 99 33 11 66
After sorting the elements are : 11 22 33 44 55 66 77 99

Test Case - 3
User Output
Enter array size :
5
Enter 5 elements :
-32 -45 -67 -46 -14
Before sorting the elements are : -32 -45 -67 -46 -14
After sorting the elements are : -67 -46 -45 -32 -14

S.No: 6	Exp. Name: ***Write a C program to sort the given elements using Heap sort***	Date: 2023-02-28

Aim:

Write a program to sort (ascending order) the given elements using heap sort technique.

Note: Do use the **printf()** function with a **newline** character (\n).

Source Code:

HeapSortMain.c

```
#include <stdio.h>
#include "HeapSortFunctions.c"
void main() {
        int arr[15], i, n;
        printf("Enter array size : ");
        scanf("%d", &n);
        printf("Enter %d elements : ", n);
        for (i = 0; i < n; i++) {
                scanf("%d", &arr[i]);
        }
        printf("Before sorting the elements are : ");
        display(arr, n);
        heapsort(arr,n);
        printf("After sorting the elements are : ");
        display(arr, n);
}
```

HeapSortFunctions.c

```
void display(int arr[15], int n)
{
        int i;
        for(i=0;i<n;i++)
        printf("%d ",arr[i]);
        printf("\n");
}
void swap(int*a, int*b)
{
        int temp= *a;
        *a = *b;
        *b = temp;
}
void heapify(int i, int n, int arr[])
{
        int largest = i;
        int left = 2*i+1;
        int right = 2*i+2;
        if((left<n)&&(arr[left]>arr[largest]))
        largest=left;
        if((right<n)&&(arr[right]>arr[largest]))
        largest=right;
        if(largest!=i)
        {
                swap(&arr[i], & arr[largest]);
                heapify(largest,n,arr);
        }
}
void heapsort(int arr[], int n)
{
        for(int i=n/2-1;i>=0;i--)
        heapify(i,n,arr);
        for(int i=n-1; i>=0;i--)
        {
                swap(&arr[0],&arr[i]);
                heapify(0,i,arr);
        }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter array size :
5
Enter 5 elements :
23 54 22 44 12
Before sorting the elements are : 23 54 22 44 12
After sorting the elements are : 12 22 23 44 54

Test Case - 2

User Output
Enter array size :
6
Enter 6 elements :
12 65 23 98 35 98
Before sorting the elements are : 12 65 23 98 35 98
After sorting the elements are : 12 23 35 65 98 98

Test Case - 3
User Output
Enter array size :
4
Enter 4 elements :
-23 -45 -12 -36
Before sorting the elements are : -23 -45 -12 -36
After sorting the elements are : -45 -36 -23 -12

Test Case - 4
User Output
Enter array size :
6
Enter 6 elements :
1 -3 8 -4 -2 5
Before sorting the elements are : 1 -3 8 -4 -2 5
After sorting the elements are : -4 -3 -2 1 5 8

S.No: 7	Exp. Name: ***Write a C program to Sort given elements using Merge sort***	Date: 2023-03-01

ID: 21SCSE1010303

E2UC403B-2023-Section-14

Galgotias University Greater Noida

Aim:

Write a program to `sort` (`Ascending order`) the given elements using `merge sort` technique.

At the time of execution, the program should print the message on the console as:

```
Enter array size :
```

For example, if the user gives the **input** as:

```
Enter array size : 5
```

Next, the program should print the following message on the console as:

```
Enter 5 elements :
```

if the user gives the **input** as:

```
Enter 5 elements : 34 67 12 45 22
```

then the program should **print** the result as:

```
Before sorting the elements are : 34 67 12 45 22
After sorting the elements are : 12 22 34 45 67
```

Note: Do use the **printf()** function with a **newline** character (`\n`).

Source Code:

MergeSortMain.c

```
#include <stdio.h>
#include "MergeSortFunctions.c"
void main() {
        int arr[15], i, n;
        printf("Enter array size : ");
        scanf("%d", &n);
        printf("Enter %d elements : ", n);
        for (i = 0; i < n; i++) {
                scanf("%d", &arr[i]);
        }
        printf("Before sorting the elements are : ");
        display(arr, n);
        splitAndMerge(arr, 0, n - 1);
        printf("After sorting the elements are : ");
        display(arr, n);
}
```

MergeSortFunctions.c

```
int i, j ,k , temp[30];
void display(int arr[15], int n)
{
        int i;
        for(i=0;i<n;i++)
        printf("%d ",arr[i]);
        printf("\n");
}
void merge(int arr[15], int low, int mid, int high)
{
        i= low;
        j= mid+1;
        k= low;
        while((i<=mid) && (j<=high))
        {
                if(arr[i]>=arr[j])
                temp[k++]=arr[j++];
                else
                temp[k++]=arr[i++];
        }
        while(i<=mid)
        temp[k++] = arr[i++];
        while(j<=high)
        temp[k++] = arr[j++];
        for(i=low;i<=high;i++)
        arr[i]=temp[i];
}
void splitAndMerge(int arr[15], int low, int high)
{
        int mid;
        if(low!=high)
        {
                mid =((low+high)/2);
                splitAndMerge(arr,low,mid);
                splitAndMerge(arr,mid+1,high);
                merge(arr,low,mid,high);
        }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter array size :
5
Enter 5 elements :
34 67 12 45 22
Before sorting the elements are : 34 67 12 45 22
After sorting the elements are : 12 22 34 45 67

Test Case - 2

User Output
Enter array size :
8
Enter 8 elements :
77 55 22 44 99 33 11 66
Before sorting the elements are : 77 55 22 44 99 33 11 66
After sorting the elements are : 11 22 33 44 55 66 77 99

Test Case - 3
User Output
Enter array size :
5
Enter 5 elements :
-32 -45 -67 -46 -14
Before sorting the elements are : -32 -45 -67 -46 -14
After sorting the elements are : -67 -46 -45 -32 -14

S.No: 8	Exp. Name: ***Implementation of Counting sort***	Date: 2023-03-03

Aim:

Write a program to sort given set of numbers Counting Sort

Source Code:

driverMain.c

```
#include<stdio.h>
void countingSort(int arr[], int size, int k){
        int c[k+1];
        int max=arr[0];
        if(size==0 || k<=0){
                for(int i=0;i<size;i++)
                printf("%d ",0);
                return;
        }
        int i,j;
        for(i=1;i<size;i++)
        if(arr[i]>max);
         max=arr[i];
         for(i=0;i<k+1;i++)
         c[i]=0;
        int sorted_arr[size];
        for(i=0;i<size;i++){
                j=arr[i];
                c[j]=c[j]+1;
        }
        for(i=1;i<k+1;i++){
                c[i]=c[i]+c[i-1];
        }
        for(i=size-1;i>=0;i--){
                j=arr[i];
                c[j]=c[j]-1;
                sorted_arr[c[j]]=arr[i];
        }
        for(i=0;i<size;i++){
                printf("%d ",sorted_arr[i]);
        }
}
int main(){
        int n;
        printf("Enter no of elements: ");
        scanf("%d",&n);
        printf("Enter value k (maximum range) of input data: ");
        int k;
        scanf("%d",&k);
        int i;
        int arr[n];
        int curele;
        printf("Enter elements of array:\n");
        while(i<n){
                scanf("%d",&curele);
                if(curele>k || curele<0){
                        printf("Invalid data please scan this value again\n");
                }
                else{
                        arr[i]=curele;
                        i++;
                }
        }
        printf("Elements in the array after performing count sort:\n");
        countingSort(arr,n,k);
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter no of elements:
6
Enter value k (maximum range) of input data:
10
Enter elements of array:
4
9
12
Invalid data please scan this value again
15
Invalid data please scan this value again
4
6
8
10
Elements in the array after performing count sort:
4 4 6 8 9 10

Test Case - 2
User Output
Enter no of elements:
5
Enter value k (maximum range) of input data:
6
Enter elements of array:
5
0
6
3
2
Elements in the array after performing count sort:
0 2 3 5 6

S.No: 9	Exp. Name: ***Matrix chain multiplication using dynamic programming***	Date: 2023-03-03

Aim:

Write a program to implement Matrix Chain Multiplication

Source Code:

MatrixChainMul.c

```
#include<stdio.h>
#include<limits.h>

int matrixChainMulOrder(int A[], int n){
        int mat[n][n];
        int i, j, cl, k,smc;


        for(i=1;i<n;i++){
                mat[i][i]=0;

        }
        for(cl=2;cl<n;cl++){
                for(i=1;i<n-cl+1;i++){
                        j=i+cl-1;
                        mat[i][j]=INT_MAX;
                        for(k=i;k<j;k++){
                                smc=mat[i][k]+mat[k+1][j]+A[i-1]*A[k]*A[j];
                                if(smc<mat[i][j]){
                                        mat[i][j]=smc;
                                }
                        }
                }
        }
        return mat[1][n-1];
}
int main(){
        int n,i;
        printf("Enter the size of input array: ");
        scanf("%d",&n);

        int A[n];
        printf("Enter the dimensions for MCM:\n");

        for(i=0;i<n;i++){
                scanf("%d",&A[i]);

        }

        int m= matrixChainMulOrder(A,n);
        printf("Minimum number of multiplication operations is = %d\n",m);

        return 0;

}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter the size of input array:
3

10 20 30
Minimum number of multiplication operations is = 6000

Test Case - 2
User Output
Enter the size of input array:
5
Enter the dimensions for MCM:
1 6 4 8 2
Minimum number of multiplication operations is = 72

S.No: 10	Exp. Name: ***Fractional Knapsack Problem***	Date: 2023-03-03

Aim:

The below program has a method void knapsack(). Which takes four parameters **number of objects**, the **weight of each object**, the **profit** corresponding to each one and the **capacity of the knapsack**. Write a program using a fractional knapsack algorithm to get the maximum profit.

Print the output as follows:

```
Sample Input and Output:
Enter the no. of objects: 6
Enter the weights and profits of each object:
1 2
4 5
8 9
4 6
5 2
3 5
Enter the capacity of knapsack:10
Maximum profit is:- 15.500000
```

Source Code:

knapsack.c

```
# include<stdio.h>
void knapsack(int n, float weight[], float profit[], float capacity) {
  // write your code here
float x[n],tp=0;
int i,j,u;
u=capacity;
for(i=0;i<n;i++)
x[i]=0.0;
for(i=0;i<n;i++){
        if(weight[i]>u)
        break;

        else{
                x[i]=1.0;
                tp=tp+profit[i];
                u=u-weight[i];
        }
}
if(i<n)
x[i]=u/weight[i];
tp=tp+(x[i]*profit[i]);
printf("Maximum profit is:- %f\n",tp);
}

int main() {
        float weight[20], profit[20], capacity;
        int num, i, j;
        float ratio[20], temp;
        printf("Enter the no. of objects: ");
        scanf("%d", &num);
        printf("Enter the weights and profits of each object:\n");
        for (i = 0; i < num; i++) {
                scanf("%f %f", &weight[i], &profit[i]);
        }
        printf("Enter the capacity of knapsack:");
        scanf("%f", &capacity);
        for (i = 0; i < num; i++) {
                ratio[i] = profit[i] / weight[i];
        }

        for (i = 0; i < num; i++) {
                for (j = i + 1; j < num; j++) {
                        if (ratio[i] < ratio[j]) {
                                temp = ratio[j];
                                ratio[j] = ratio[i];
                                ratio[i] = temp;
                                temp = weight[j];
                                weight[j] = weight[i];
                                weight[i] = temp;
                                temp = profit[j];
                                profit[j] = profit[i];
                                profit[i] = temp;
                        }
                }
        }
        knapsack(num, weight, profit, capacity);
```

```
        return(0);
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter the no. of objects:
6
Enter the weights and profits of each object:
1 2
4 5
8 9
4 6
5 2
3 5
Enter the capacity of knapsack:
10
Maximum profit is:- 15.500000

Test Case - 2
User Output
Enter the no. of objects:
5
Enter the weights and profits of each object:
4 6
1 3
7 5
5 3
3 4
Enter the capacity of knapsack:
10
Maximum profit is:- 14.428572

S.No: 11	Exp. Name: ***0/1 knapsack problem using Dynamic programming***	Date: 2023-03-31

Aim:

Write a program to implement Knapsack using Dynamic programming

Source Code:

DynamicKnapsack.c

```
#include<stdio.h>
#include<stdlib.h>
int max(int a, int b) {
        return (a > b)? a : b;
}
int knapsack(int w[], int p[], int n, int W) {
        int i, wt;
        int **dpTable;
        dpTable = (int**)malloc((n + 1) * sizeof(int*));
        for (i = 0; i < n + 1; i++)
        dpTable[i] = (int*)malloc((W + 1) * sizeof(int));
        for (i = 0; i <= n; i++){
                for (wt = 0; wt <= W; wt++){
                        if (i == 0 || wt == 0)
                        dpTable[i][wt] = 0;
                        else if (w[i - 1] <= wt)
                        dpTable[i][wt] = max(p[i - 1] + dpTable[i - 1][wt - w[i - 1]],
dpTable[i - 1]
                        [wt]);
                        else
                        dpTable[i][wt] = dpTable[i - 1][wt];
                        }
                        }
                        return dpTable[n][W];
                        }
                        int main() {
                                int i, j, n;
                                int W;
                                int Max;
                                int *w;
                                int *p;
                                printf("Enter the no. of items: ");
                                scanf("%d",&n);
                                w = (int *)malloc(n*sizeof(int));
                                p = (int *)malloc(n*sizeof(int));
                                printf("Enter the weight and price of all items\n");
                                for(i = 0;i < n;i++)
                                scanf("%d%d",&w[i],&p[i]);
                                printf("enter the capacity of knapsack: ");
                                scanf("%d",&W);
                                Max = knapsack(w,p,n,W);
                                printf("The maximum value of items that can be put into
knapsack is: %d ", Max);
                        }
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter the no. of items:
3

10 60
20 100
30 120
enter the capacity of knapsack:
50
The maximum value of items that can be put into knapsack is: 220

Test Case - 2
User Output
Enter the no. of items:
3
Enter the weight and price of all items
4 1
5 2
1 3
enter the capacity of knapsack:
4
The maximum value of items that can be put into knapsack is: 3

S.No: 12	Exp. Name: ***Dijkstra's Shortest path algorithm***	Date: 2023-04-01

Aim:

Given a graph **G** and source vertex **S**, Dijkstra's shortest path algorithm is used to find shortest paths from source **S** to all vertices in the given graph.
Dijkstra algorithm is also called single source shortest path algorithm. It is based on greedy technique. Little variation in the algorithm can find the shortest path from the source nodes to all the other nodes in the graph.

Sample Input and Output:

```
Enter the number of vertices : 4
Enter the number of edges : 5
Enter source : 1
Enter destination : 2
Enter weight : 4
Enter source : 1
Enter destination : 4
Enter weight : 10
Enter source : 1
Enter destination : 3
Enter weight : 6
Enter source : 2
Enter destination : 4
Enter weight : 5
Enter source : 3
Enter destination : 4
Enter weight : 2
Enter the source :1
Node    Distance    Path
2   4   2<-1
3   6   3<-1
4   8   4<-3<-1
```

Source Code:

Dijkstras.c

```
#include <limits.h>
#include <stdio.h>
#define MAX 20
int V,E;
int graph[MAX][MAX];
#define INFINITY 99999
void dijkstra(int G[MAX][MAX],int n,int startnode) {
        int cost[MAX][MAX],distance[MAX],pred[MAX];
        int visited[MAX],count,mindistance,nextnode,i,j;
        for(i=1;i<=n;i++)
        for(j=1;j<=n;j++) {
                if(G[i][j]==0)
                cost[i][j]=INFINITY;
                else
                cost[i][j]=G[i][j];
                }
                for(i=1;i<=n;i++)
                {
                        distance[i]=cost[startnode][i];
                        pred[i]=startnode;
                        visited[i]=0;
                        }
                        distance[startnode]=0;
                        visited[startnode]=1;
                        count=1;
                        while(count<n-1)
                        {
                                mindistance=INFINITY;
                                for(i=1;i<=n;i++)
                                if(distance[i]<mindistance&&!visited[i])        {
                                        mindistance=distance[i];
                                        nextnode=i;
                                        }
                                        visited[nextnode]=1;
                                        for(i=1;i<=n;i++)
                                        if(!visited[i])
                                        if(mindistance+cost[nextnode][i]<distance[i])   {
                                                distance[i]=mindistance+cost[nextnode][i];
                                                pred[i]=nextnode;
                                                }
                                                count++;
                                                }
                                                printf("Node\tDistance\tPath\n");
                                                for(i=1;i<=n;i++)
                                                if(i!=startnode)
                                                {
                                                        if(distance[i]==INFINITY){
                                                                printf("%4d\t%8s",i,"INF");
                                                                printf("\tNO PATH\n");
                                                                }
                                                                else{

printf("%4d\t%8d",i,distance[i]);
                                                                        printf("\t%d",i);
                                                                        j=i;
```

```
                                                                        j=pred[j];
                                                                        printf("<-
%d",j);
                                                                        }

while(j!=startnode);

printf("\n");
                                                                        }
                                                                        }

}
int main()
{       int s,d,w,i,j;
printf("Enter the number of vertices : ");
scanf("%d",&V);
printf("Enter the number of edges : ");
scanf("%d",&E); for(i = 1 ; i <= V; i++) {
        for(j=1; j <= V; j++ ) {
                graph[i][i] = 0;
                }
                }
                for(i=1;i<=E;i++)
                {
                        printf("Enter source : ");
                        scanf("%d",&s);
                        printf("Enter destination : ");
                        scanf("%d",&d);
                        printf("Enter weight : ");
                        scanf("%d",&w);
                        if(s > V || d > V || s<=0 || d<=0) {
                                printf("Invalid index. Try again.\n");
                                i--;
                                continue;
                                }
                                else {
                                        graph[s][d] = w;
                                        }
                                        }
                                        printf("Enter the source :");
                                        scanf("%d",&s);
                                        dijkstra(graph, V,s);
                                        return 0;
                                        }
```


Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter the number of vertices :
4
Enter the number of edges :

```
5
Enter source :
1
Enter destination :
2
Enter weight :
4
Enter source :
1
Enter destination :
4
Enter weight :
10
Enter source :
1
Enter destination :
3
Enter weight :
6
Enter source :
2
Enter destination :
4
Enter weight :
5
Enter source :
3
Enter destination :
4
Enter weight :
2
Enter the source :
1
Node    Distance        Path
   2          4        2<-1
   3          6        3<-1
   4          8        4<-3<-1
```

Test Case - 2

User Output

```
Enter the number of vertices :
5
Enter the number of edges :
6
Enter source :
1
Enter destination :
```

2
Enter source :
1
Enter destination :
5
Enter weight :
3
Enter source :
2
Enter destination :
4
Enter weight :
4
Enter source :
2
Enter destination :
3
Enter weight :
7
Enter source :
4
Enter destination :
3
Enter weight :
2
Enter source :
5
Enter destination :
4
Enter weight :
1
Enter the source :
2

Node	Distance	Path
1	INF	NO PATH
3	6	3<-4<-2
4	4	4<-2
5	INF	NO PATH

Test Case - 3
User Output
Enter the number of vertices :
4
Enter the number of edges :
5
Enter source :
1

```
Enter weight :
4
Enter source :
3
Enter destination :
2
Enter weight :
5
Enter source :
4
Enter destination :
1
Enter weight :
1
Enter source :
4
Enter destination :
2
Enter weight :
3
Enter source :
4
Enter destination :
3
Enter weight :
8
Enter the source :
1
Node    Distance        Path
   2           4        2<-1
   3         INF        NO PATH
   4         INF        NO PATH
```

S.No: 13	Exp. Name: ***Bellmen ford algorithm***	Date: 2023-04-01

Aim:

The Bellman-Ford algorithm is an algorithm that computes the shortest paths from a single source vertex to all of the other vertices in a weighted digraph. It is slower than Dijkstra's algorithm but capable of handling graphs having the negative edge cycles

Procedural steps are

1. initializes distances from the source to all vertices as infinite and distance to the source itself as 0.
2. Calculates the shortest distances.
3. Do following |V|-1 times where |V| is the number of vertices in the given graph.
4. Do following for each edge u-v. Let us say weight of edge u-v as **w**
5. If dist[v] > dist[u] +w then update dist[v] as follows
6. dist[v] = dist[u] + w
7. If there is a negative weight cycle in the graph
8. If dist[v] > dist[u] + w, then Graph contains negative weight cycle

Worst case time complexity of this algorithm is $O(|V| * |E|)$ Write the missing code in the below program and print the output as shown in the sample tes cases

Source Code:

bellmenFord.c

```
#include <limits.h>
#include <stdio.h>
#define MAX 20
int V, E, isNegtiveWeights=0;
int graph[MAX][MAX];
#define INFINITY 99999
void bellmenFord(int G[MAX][MAX],int n,int startnode) {
        int cost[MAX][MAX],distance[MAX],pred[MAX];
        int visited[MAX],count,mindistance,nextnode,i,j;
        for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
        if(G[i][j]==0)
        cost[i][j]=INFINITY;
        else
        cost[i][j]=G[i][j];
        for(i=1;i<=n;i++) {
                distance[i]=cost[startnode][i];
                pred[i]=startnode;
                visited[i]=0;

        }
        distance[startnode]=0;
        visited[startnode]=1;
        count=1;
        while(count<n-1)
        {mindistance=INFINITY;
        for(i=1;i<=n;i++)
        if(distance[i]<mindistance&&!visited[i]) {
                mindistance=distance[i];
                nextnode=i;

        }
        visited[nextnode]=1;
        for(i=1;i<=n;i++)
        if(!visited[i])
        if (mindistance+cost[nextnode][i]<distance[i]) {
                distance[i]=mindistance+cost[nextnode][i];
                pred[i]=nextnode;

        }
        if (mindistance+cost[nextnode][i] < distance[i]) {
                isNegtiveWeights =1;

        }
        count++;

        }
        printf("Node\tDistance\tPath\n");
        for(i=1;i<=n;i++)
        if(i!=startnode)
        {
                if(distance[i] == INFINITY) {
                        printf("%4d\t%8s",i,"INF");
                        printf("\tNO PATH\n");
```

```
                        printf("\t%d",i);
                        j=i;
                        do
                        {
                                j=pred[j];
                                printf("<-%d",j);
                        }
                        while(j!=startnode);
                        printf("\n");

                }

        }

}
int main() {
        int s,d,w,i,j;
        printf("Enter the number of vertices : ");
        scanf("%d",&V);
        printf("Enter the number of edges : ");
        scanf("%d",&E);
        for(i = 1 ; i <= V; i++) {
                for(j=1; j <= V; j++ ) {
                        graph[i][i] = 0;

                }

        }
        for(i=1;i<=E;i++) {
                printf("Enter source : ");
                scanf("%d",&s);
                printf("Enter destination : ");
                scanf("%d",&d);
                printf("Enter weight : ");
                scanf("%d",&w);
                if(s > V || d > V || s<=0 || d<=0) {
                        printf("Invalid index. Try again.\n");
                        i--;
                        continue;

                } else {
                        graph[s][d] = w;

                }

        }
        printf("Enter the source :");
        scanf("%d",&s);
        bellmenFord(graph, V,s);
        if(isNegtiveWeights == 1) {
                printf("Graph contains negitive loops");

        }
        return 0;
```

```
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter the number of vertices :
5
Enter the number of edges :
7
Enter source :
1
Enter destination :
2
Enter weight :
17
Enter source :
2
Enter destination :
3
Enter weight :
25
Enter source :
3
Enter destination :
4
Enter weight :
1
Enter source :
4
Enter destination :
2
Enter weight :
-24
Enter source :
2
Enter destination :
5
Enter weight :
15
Enter source :
1
Enter destination :
4
Enter weight :

```
Enter source :
5
Enter destination :
3
Enter weight :
9
Enter the source :
1
Node    Distance        Path
   2          17        2<-1
   3          41        3<-5<-2<-1
   4          40        4<-1
   5          32        5<-2<-1
```

Test Case - 2

User Output

```
Enter the number of vertices :
5
Enter the number of edges :
6
Enter source :
1
Enter destination :
2
Enter weight :
6
Enter source :
2
Enter destination :
5
Enter weight :
1
Enter source :
2
Enter destination :
4
Enter weight :
5
Enter source :
1
Enter destination :
4
Enter weight :
4
Enter source :
3
Enter destination :
```

```
-4
Enter source :
5
Enter destination :
3
Enter weight :
2
Enter the source :
1
Node    Distance        Path
   2            6        2<-1
   3            9        3<-5<-2<-1
   4            4        4<-1
   5            7        5<-2<-1
Graph contains negitive loops
```

S.No: 14	Exp. Name: ***Problem on N Queens***	Date: 2023-04-01

Aim:

Write a program to arrange N queens on a **N * N** chessboard using backtracking algorithm, such that no queen can attack any other queens on the board, and print the result as shown in the example.

Sample Input and Output:

```
Enter the value of N for NxN chessboard: 4
0   1   0   0
0   0   0   1
1   0   0   0
0   0   1   0
```

Source Code:

backtrack.c

```
#include <stdio.h>
int N;
int board[100][100];


int is_attack(int i,int j) {
        //This is the function to check if the cell is attacked or not

    //check if there is any queen in row or column

    //check for diagonals if found return 1 else return 0

int k,l;
for(k = 0; k < N; k++) {
        if((board[i][k] == 1) || (board[k][j] == 1))
        return 1;

}
        //checking for diagonals
        for(k = 0; k < N; k++) {
                for(l = 0; l < N; l++){
                        if(((k+l) == (i+j)) || ((k-l) == (i-j))) {
                                if(board[k][l] == 1)
                                return 1;

                        }

                }

        }
        return 0;}

int N_queen(int n) {
    int i,j;
    if(n == 0)
        return 1;

     // check whether we can put queen to this posotion or not
     // the condition for this is the queen should not be placed if it is in attacking mode
     // or if the place is already occupied
     // continue this wiht the remaining queens also
     for(i = 0; i < N; i++) {
        for(j = 0; j < N; j++) {
        //checking if we can place a queen here or not
        //queen will not be placed if the place is being attacked
        //or already occupied
        if((!is_attack(i,j)) && (board[i][j]!=1)) {
                board[i][j] = 1;
                //recursion
                //wether we can put the next queen with this arrangment or not
                if(N_queen(n-1) == 1){
                        return 1;

                }
                board[i][j] = 0;
```

```
        }
}
}
int main() {
    printf("Enter the value of N for NxN chessboard: ");
    scanf("%d",&N);

    int i,j;
    for(i = 0 ;i < N; i++) {
        for(j = 0; j < N; j++) {
            board[i][j] = 0;
        }
    }
    N_queen(N);
    for(i = 0 ;i < N; i++) {
        for(j = 0; j < N; j++)
            printf("%d\t",board[i][j]);
        printf("\n");
    }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Enter the value of N for NxN chessboard:
4
0       1       0       0
0       0       0       1
1       0       0       0
0       0       1       0
```

Test Case - 2

User Output

```
Enter the value of N for NxN chessboard:
8
1       0       0       0       0       0       0       0
0       0       0       0       1       0       0       0
0       0       0       0       0       0       0       1
0       0       0       0       0       1       0       0
0       0       1       0       0       0       0       0
0       0       0       0       0       0       1       0
0       1       0       0       0       0       0       0
0       0       0       1       0       0       0       0
```

S.No: 15	Exp. Name: ***String matching algorithm***	Date: 2023-03-31

Aim:

Write a program to implement Naive string matching algorithm

Source Code:

stringMatching.c

```
#include <stdio.h>
#include <string.h>
void search(char* pat, char* txt)
{ int M = strlen(pat);
int N = strlen(txt);
int count = 0;
for (int i = 0; i <= N - M; i++) {
        int j;
        for (j = 0; j < M; j++)
        if (txt[i + j] != pat[j])
        break;
        if (j == M)
        { printf("Pattern found at index %d\n", i);
        count = count + 1; }
}
if(count == 0)
printf("Not Found");
}
int main()
{ char txt[50], pat[50];
scanf("%s", txt);
scanf("%s", pat);
search(pat, txt);
return 0;
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
AABAACAADAABAAABAA
AABA
Pattern found at index 0
Pattern found at index 9
Pattern found at index 13

Test Case - 2
User Output

IamSoLuckyToLearnProgramming
lucky
Not Found

Test Case - 3
User Output
hjyhfdrtiufgghfvrdvgfbhgcgvvjbhgvbygfvbhg
fvbhg
Pattern found at index 36

S.No: 16	Exp. Name: ***Rabin - karp algorithm***	Date: 2023-03-31

Aim:

Write a program to implement String Matching using Rabin-Karp algorithm

Source Code:

robinKarp.c

```
#include<stdio.h>
#include<string.h>
int main (){
        char txt[80], pat[80];
        int q;
        printf("Enter the container string: ");
        scanf("%s", txt);
        printf("Enter the pattern to be searched: ");
        scanf("%s", pat);
        int d = 256;
        printf("Enter a prime number: ");
        scanf("%d", &q);
        int M = strlen(pat);
        int N = strlen(txt);
        int i, j, c = 0;
        int p = 0;
        int t = 0;
        int h = 1;
        for(i = 0; i < M - 1; i++)
        h = (h * d) % q;
        for (i = 0; i < M; i++){
                p = (d * p + pat[i]) % q;
                t = (d * t + txt[i]) % q;
        }
        for (i = 0; i <= N - M; i++){
                if (p == t){
                        for (j = 0; j < M; j++){
                                if (txt[i + j] != pat[j])
                                break;
                        }
                        if (j == M)
                        { printf ("Pattern found at index %d\n", i);
                        c += 1; }
                }
                if (i < N - M){
                        t = (d * (t - txt[i] * h) + txt[i + M]) % q;
                        if (t < 0)
                        t = (t + q);
                }
        }
        if (c == 0)
        printf("Not found");
        return 0;
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter the container string:
Heisthebestfighteramongallthesoldersinthekingdom
Enter the pattern to be searched:
the
Enter a prime number:
2
Pattern found at index 4
Pattern found at index 26
Pattern found at index 38

Test Case - 2
User Output
Enter the container string:
Heisthebestfighteramongallthesoldersinthekingdom
Enter the pattern to be searched:
of
Enter a prime number:
3
Not found

Test Case - 3
User Output
Enter the container string:
Heisthebestfighteramongallthesoldersinthekingdom
Enter the pattern to be searched:
t
Enter a prime number:
2
Pattern found at index 4
Pattern found at index 10
Pattern found at index 15
Pattern found at index 26
Pattern found at index 38

S.No: 17	Exp. Name: ***Write a C program to implement topological sort on a graph***	Date: 2023-03-31

Aim:

Write a program to implement topological sort on a given graph.

You should write the program to match the sample input outputs shown below:

Sample input and output:

```
Enter the number of vertices : 4
Enter the number of edges : 4
Enter source : 1
Enter destination : 2
Enter source : 2
Enter destination : 3
Enter source : 3
Enter destination : 4
Enter source : 1
Enter destination : 4
The topological order is:1 2 3 4
```

Source Code:

TopologicalSort2.c

```
#include <stdio.h>
int main() {
        int i,j,s,d,k,E,N,graph[10][10],indeg[10],flag[10],count=0;
        printf("Enter the number of vertices : ");
        scanf("%d",&N);
        printf("Enter the number of edges : ");
        scanf("%d",&E);
        for (i = 0 ; i<= E; i++) {
                for (j = 0; j<=E; j++) {
                        graph[i][j] = 0;
                }
        }
        for(i=1;i<=E;i++) {
                printf("Enter source : ");
                scanf("%d",&s);
                printf("Enter destination : ");
                scanf("%d",&d);
                if(s > N || d > N || s<=0 || d<=0) {
                        printf("Invalid index. Try again.\n");
                        i--;
                        continue;
                } else {
                        graph[s][d] = 1;
                }
        }
        for(i=0;i<N;i++){
                indeg[i]=0;
                flag[i]=0;
        }
        for(i=0;i<N;i++)
        for(j=0;j<N;j++)
        indeg[i]=indeg[i]+graph[j][i];
        printf("The topological order is:");
        while(count<N){
                for(k=0;k<N;k++){
                        if((indeg[k]==0) && (flag[k]==0)){
                                printf("%d ",(k+1));
                                flag [k]=1;
                        }
                        for(i=0;i<N;i++){
                                if(graph[i][k]==1)
                                indeg[k]--;
                        }
                }
                count++;
        }
        printf("\n");
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output
Enter the number of vertices :
4
Enter the number of edges :
4
Enter source :
1
Enter destination :
2
Enter source :
1
Enter destination :
3
Enter source :
2
Enter destination :
4
Enter source :
3
Enter destination :
4
The topological order is:1 2 3 4

S.No: 18	Exp. Name: ***Write a C program for constructing a minimum cost spanning tree of a graph using Prim's Algorithm***	Date: 2023-03-31

Aim:

Write a C program for constructing a minimum cost spanning tree of a graph using Prim's Algorithm

Source Code:

PrimsMinimumSpanningTree.c

```
#include<stdio.h>
#include<conio.h>
int a,b,u,v,n,i,j,ne=1,e,s,d,w;
int visited[10]={0},min,mincost=0,cost[10][10];
void prims() {
        visited[1]=1;
        while(ne < n) {
                min = 999;
                for(i=1;i<=n;i++) {
                        if(visited[i] == 1) {
                                for(j=1;j<=n;j++) {
                                        if(visited[j] == 0 && cost[i][j] < min ) {
                                                min = cost[i][j];
                                                a = i;
                                                b = j;
                                        }
                                }
                        }
                }
                printf("Edge cost from %d to %d : %d\n",a,b,cost[a][b]);
                ne++;
                mincost+=cost[a][b];
                visited[b]=1;
                cost[a][b]=cost[b][a]=999;
        }
        printf("Minimum cost of spanning tree = %d\n",mincost);
}
void main() {
        printf("Enter the number of vertices : ");
        scanf("%d",&n);
        printf("Enter the number of edges : ");
        scanf("%d",&e);
        for(i=1;i<=e;i++) {
                printf("Enter source : ");
                scanf("%d",&s);
                printf("Enter destination : ");
                scanf("%d",&d);
                printf("Enter weight : ");
                scanf("%d",&w);
                if(s<=0 || d<=0 || s> n || d > n || w < 0 ) {
                        printf("Invalid data.Try again.\n");
                        i--;
                        continue;
                }
                cost[s][d]=w;
                cost[d][s]=w;
        }
        for(i=1;i<=n;i++) {
                for(j=1;j<=n;j++) {
                        if(cost[i][j]==0)
                        cost[i][j]=999;
                }
        }
        printf("The edges of Minimum Cost Spanning Tree are : \n");
        prims();
```

```
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter the number of vertices :
7
Enter the number of edges :
11
Enter source :
1
Enter destination :
2
Enter weight :
7
Enter source :
2
Enter destination :
3
Enter weight :
8
Enter source :
1
Enter destination :
4
Enter weight :
5
Enter source :
2
Enter destination :
4
Enter weight :
9
Enter source :
2
Enter destination :
5
Enter weight :
7
Enter source :
3
Enter destination :
5
Enter weight :

Enter source :
4
Enter destination :
5
Enter weight :
15
Enter source :
4
Enter destination :
6
Enter weight :
6
Enter source :
5
Enter destination :
6
Enter weight :
8
Enter source :
5
Enter destination :
7
Enter weight :
9
Enter source :
6
Enter destination :
7
Enter weight :
11
The edges of Minimum Cost Spanning Tree are :
Edge cost from 1 to 4 : 5
Edge cost from 4 to 6 : 6
Edge cost from 1 to 2 : 7
Edge cost from 2 to 5 : 7
Edge cost from 5 to 3 : 5
Edge cost from 5 to 7 : 9
Minimum cost of spanning tree = 39

Test Case - 2
User Output
Enter the number of vertices :
7
Enter the number of edges :
9
Enter source :
2
Enter destination :

```
Enter weight :
14
Enter source :
1
Enter destination :
2
Enter weight :
28
Enter source :
1
Enter destination :
6
Enter weight :
10
Enter source :
5
Enter destination :
6
Enter weight :
25
Enter source :
4
Enter destination :
5
Enter weight :
22
Enter source :
4
Enter destination :
7
Enter weight :
18
Enter source :
2
Enter destination :
3
Enter weight :
16
Enter source :
3
Enter destination :
4
Enter weight :
12
Enter source :
5
Enter destination :
7
```

The edges of Minimum Cost Spanning Tree are :
Edge cost from 1 to 6 : 10
Edge cost from 6 to 5 : 25
Edge cost from 5 to 4 : 22
Edge cost from 4 to 3 : 12
Edge cost from 3 to 2 : 16
Edge cost from 2 to 7 : 14
Minimum cost of spanning tree = 99

Test Case - 3
User Output
Enter the number of vertices :
5
Enter the number of edges :
7
Enter source :
1
Enter destination :
2
Enter weight :
2
Enter source :
1
Enter destination :
3
Enter weight :
5
Enter source :
2
Enter destination :
3
Enter weight :
8
Enter source :
2
Enter destination :
4
Enter weight :
4
Enter source :
3
Enter destination :
4
Enter weight :
5
Enter source :
3
Enter destination :

```
Enter weight :
6
Enter source :
4
Enter destination :
5
Enter weight :
4
The edges of Minimum Cost Spanning Tree are :
Edge cost from 1 to 2 : 2
Edge cost from 2 to 4 : 4
Edge cost from 4 to 5 : 4
Edge cost from 1 to 3 : 5
Minimum cost of spanning tree = 15
```

S.No: 19	Exp. Name: ***Sum Of Subset Problem***	Date: 2023-03-31

Aim:

In sum of the subset problem, we have to find a subset such that the sum of elements in the subset is equal a given number. The backtracking approach generates all permutations in the worst case but in general, performs better than the recursive approach towards subset sum problem.

Procedural Steps:

9. Start with an empty set
10. Add the next element from the list to the set
11. If the subset is having sum equal to the given number, then stop with that subset as a solution.
12. If the subset is not possible or if we have reached the end of the set, then backtrack through the subset until we find the most suitable value.
13. If the subset is possible and the sum of subset < the given number then go to step 2.
14. If we all the elements are visited without finding a suitable subset and if no backtracking is possible then stop without a solution.

Write a program to find possible subsets for the given number using backtracking

Source Code:

sumOfSets.c

```
#include<stdio.h>
#define TRUE 1
#define FALSE 0
void sumset(int i,int wt,int total);
int inc[50],w[50],sum,n;
int subset(int i,int wt,int total)
{
        return(((wt+total)>=sum)&&((wt==sum)||(wt+w[i+1]<=sum)));
}
void main()
{
        int i,j,n,temp,total=0;
        printf("Enter number of elements in the subset: ");
        scanf("%d",&n);
        printf("Enter %d numbers to the set: ", n);
        for (i
        = 0;
        i
        < n; i++) {
                scanf("%d",&w[i]);
                total+=w[i];
        }
        printf("Enter the input number to find subsets: ");
        scanf("%d",&sum);
        for (i=0;i<=n;i++)
        for (j=0;j<n
        -1;j++)
        if(w[j]>w[j+1])
        {
                temp=w[j];
                w[j]=w[j+1];
                w[j+1]=temp;
        }
        //printf("The given %d numbers in ascending order: ", n);
        //for (i=0;i<n;i++)
        //printf("%d ",w[i]);
        if((total<sum))
        printf("Subset construction is not possible");
        else
        {
                for (i=0;i<n;i++)
                inc[i]=0;
                printf("The solution is: ");
                sumset(
                        -1,0,total);
                        }
                        }
                        void sumset(int i,int wt,int total) {
                                int j;
                                if(subset(i,wt,total))
                                {
                                        if(wt==sum)
                                        {
                                                printf("\n{");
                                                for (j=0;j<=i;j++)
```

```
                                        printf("}");
                                        } else {
                                                inc[i+1]=TRUE;
                                                sumset(i+1,wt+w[i+1],total-w[i+1]);
                                                inc[i+1]=FALSE;
                                                sumset(i+1,wt,total-w[i+1]);
                                                }
                                                }
                                                }
```


Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter number of elements in the subset:
3
Enter 3 numbers to the set:
12 3 2
Enter the input number to find subsets:
5
The solution is:
{2 3 }

Test Case - 2
User Output
Enter number of elements in the subset:
4
Enter 4 numbers to the set:
100 25 36 54
Enter the input number to find subsets:
500
Subset construction is not possible

Test Case - 3
User Output
Enter number of elements in the subset:
4
Enter 4 numbers to the set:
1 2 1 5
Enter the input number to find subsets:
3
The solution is:
{1 2 }
{1 2 }

S.No: 20	Exp. Name: ***Floyd - Warshall's Algorithm***	Date: 2023-04-01

Aim:

Write a program to implement All Pairs Shortest Paths using Floyd’s Algorithm

Source Code:

Warshall.c

```
#include<stdio.h>
#define INF 99999
int graph[20][20];
int N,E;
void printSolution(int dist[][N]);
void floydWarshall () {
        int dist[N][N], i, j, k;
        for (i = 1; i <= N; i++)
        for (j = 1; j <= N; j++) {
                if (i == j) {
                        dist[i][j] = 0;
                        } else {
                                dist[i][j] = graph[i][j];
                                }
                        }
                                for (k = 1; k <= N; k++) {
                                        for (i = 1; i <= N; i++) {
                                                for (j = 1; j <= N; j++) {
                                                        if (dist[i][k] + dist[k][j] <
dist[i][j])
                                                        dist[i][j] = dist[i][k] + dist[k]
[j];
                                                        }

                                        }

                                }
                                printSolution(dist);

}
void printSolution(int dist[][N]) {
        printf ("The following matrix shows the shortest distances between all pairs of the
vertices.\n");
        for (int i = 1; i <= N; i++) {
                for (int j = 1; j <= N; j++) {
                        if (dist[i][j] == INF)
                        printf("%5s", "INF");
                        else
                        printf ("%5d", dist[i][j]);
                        }
                        printf("\n");
                        }

}
int main() {
        int s,d,w,i,j;
        printf("Enter the number of vertices : ");
        scanf("%d",&N);
        printf("Enter the number of edges : ");
        scanf("%d",&E);         for(i = 1 ; i <= N;i++)
        {
                for(j = 1 ; j <= N ;j++ ) {
                        graph[i][j] = INF;
                        }
                        }
```

```
                        scanf("%d",&s);
                        printf("Enter destination : ");
                        scanf("%d",&d);
                        printf("Enter weight : ");
                        scanf("%d",&w);
                        if(s > N || d > N || s<=0 || d<=0)
                        {
                                printf("Invalid index. Try again.\n");
                                i--;
                                continue;
                                } else {
                                        graph[s][d] = w;
                                        }

                }
                floydWarshall();
                return 0;

}
```


Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter the number of vertices :
4
Enter the number of edges :
5
Enter source :
1
Enter destination :
2
Enter weight :
4
Enter source :
1
Enter destination :
4
Enter weight :
10
Enter source :
1
Enter destination :
3
Enter weight :
6
Enter source :
2

4
Enter weight :
5
Enter source :
3
Enter destination :
4
Enter weight :
2
The following matrix shows the shortest distances between all pairs of the vertices.
0 4 6 8
INF 0 INF 5
INF INF 0 2
INF INF INF 0

Test Case - 2
User Output
Enter the number of vertices :
5
Enter the number of edges :
6
Enter source :
1
Enter destination :
2
Enter weight :
2
Enter source :
1
Enter destination :
5
Enter weight :
3
Enter source :
2
Enter destination :
4
Enter weight :
4
Enter source :
2
Enter destination :
3
Enter weight :
7
Enter source :
4

```
Enter weight :
2
Enter source :
5
Enter destination :
4
Enter weight :
1
The following matrix shows the shortest distances between all pairs of the vertices.
    0    2    6    4    3
  INF    0    6    4  INF
  INF  INF    0  INF  INF
  INF  INF    2    0  INF
  INF  INF    3    1    0
```

Test Case - 3

User Output

```
Enter the number of vertices :
4
Enter the number of edges :
5
Enter source :
1
Enter destination :
2
Enter weight :
4
Enter source :
3
Enter destination :
2
Enter weight :
5
Enter source :
4
Enter destination :
1
Enter weight :
1
Enter source :
4
Enter destination :
2
Enter weight :
3
Enter source :
4
```

Enter weight :
8
The following matrix shows the shortest distances between all pairs of the vertices.
0 4 INF INF
INF 0 INF INF
INF 5 0 INF
1 3 8 0

Test Case - 4
User Output
Enter the number of vertices :
4
Enter the number of edges :
6
Enter source :
1
Enter destination :
2
Enter weight :
1
Enter source :
1
Enter destination :
4
Enter weight :
3
Enter source :
2
Enter destination :
3
Enter weight :
6
Enter source :
3
Enter destination :
1
Enter weight :
-2
Enter source :
4
Enter destination :
2
Enter weight :
5
Enter source :
4
Enter destination :

```
10
The following matrix shows the shortest distances between all pairs of the vertices.
    0    1    7    3
    4    0    6    7
   -2   -1    0    1
    8    5   10    0
```

S.No: 21	Exp. Name: ***Implementation of Knuth-Morris-Pratt algorithm***	Date: 2023-03-31

Aim:

Implement the Knuth-Morris-Pratt algorithm is used to find if a pattern is present in the text or not.

Sample input and output.

```
Enter the text: Today is a good day for programming
Enter the pattern to be searched: day
Found the pattern at index : 2
Found the pattern at index : 16
```

Note: Do not display anything if the pattern is not found in the text.

Source Code:

KMPAlgorithm.c

```
#include<stdio.h>
#include<string.h>
#include<malloc.h>
#define NO_OF_CHARS 256
void KMPSearch(char* txt, char* pat) {
int pat_len = strlen(pat);
int text_len = strlen(txt);
int LPS[pat_len];
computeLongestPrefixSuffixArray(pat,pat_len,LPS);
int text_index = 0;
int pat_index = 0;
int endl = 0;
while (text_index < text_len) {
        if (pat[pat_index] == txt[text_index]) {
                pat_index++;
                text_index++;
        }
        if (pat_index == pat_len) {
                printf("Found the pattern at index : %d\n", text_index - pat_len);
                pat_index = LPS[pat_index - 1];
        } else if (text_index < text_len && pat[pat_index] != txt[text_index]) {
                if (pat_index != 0)
                pat_index = LPS[pat_index - 1];
                else
                text_index = text_index + 1;
        }
}
}
void computeLongestPrefixSuffixArray (char* pat, int pat_len, int* LPS) {
        int len = 0;
        LPS[0] = 0;
        int i = 1;
        while (i < pat_len) {
                if (pat[i] == pat[len]) {
                        len++;
                        LPS[i] = len;
                        i++;
                }
                else {
                        if (len != 0) {
                                len = LPS[len - 1];
                        }
                        else {
                                LPS[i] = 0;
                                i++;
                        }
                }
        }
}
int main() {
        char *txt,*pat;
        txt = (char*) malloc(NO_OF_CHARS*sizeof(char));
        pat = (char*) malloc(NO_OF_CHARS*sizeof(char));
        printf("Enter the text: ");
        gets(txt);
```

```
        KMPSearch(txt, pat);
        return 0;
}
```


Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter the text:
Today is a good day for programming
Enter the pattern to be searched:
day
Found the pattern at index : 2
Found the pattern at index : 16

Test Case - 2
User Output
Enter the text:
I love coding, Coding loves me.
Enter the pattern to be searched:
coding
Found the pattern at index : 7

Test Case - 3
User Output
Enter the text:
I love coding, coding loves me, so on.
Enter the pattern to be searched:
coding
Found the pattern at index : 7
Found the pattern at index : 15

S.No: 22	Exp. Name: ***Job Sequencing Problem with Deadline***	Date: 2023-03-31

Aim:

Write a program to implement Task scheduling problem using greedy approach

Source Code:

jobDeadline.c

```
#include<stdio.h>
int n, i, j, k, t;
int checkSlot(int s[], int p) {
        int ptr = 0;
        for (int i = 0; i < n; i++) {
                if (s[i] == p)
                ptr++;
        }
        if (ptr == 0)
        return 1; else
        return 0;
}
int main() {
        printf("Enter the no of jobs: ");
        scanf("%d", & n);
        int slot[n], profit, p[n], d[n], maxprofit;
        for (i = 0; i < n; i++) {
                printf("Enter the profit of job %d: ", i + 1);
                scanf("%d", & p[i]);
                printf("Enter the deadline of job %d: ", i + 1);
                scanf("%d", & d[i]);
        }
        for (i = 0; i < n; i++)
        for (j = i + 1; j < n; j++)
        if (p[i] < p[j]) {
                t = p[i];
                p[i] = p[j];
                p[j] = t;
                t = d[i];
                d[i] = d[j];
                d[j] = t;
        }
        for (i = 0; i < n; i++)
        slot[i] = 0;
        for (i = 0; i < n; i++)
        for (j = d[i]; j > 0; j--) {
                if (checkSlot(slot, j) == 1) {
                        slot[i] = j;
                        break;
                }
        }
        printf("Index   Profit  Deadline  Slot Alloted ");
        for (i = 0; i < n; i++) {
                if (slot[i] > 0)
                printf("\n%d\t%d \t%d \t[%d - %d]", i + 1, p[i], d[i], (slot[i] - 1),
                slot[i]); else
                printf("\n%d\t%d\t%d \tRejected", i + 1, p[i], d[i]);

        }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output			
Enter the no of jobs:			
3			
Enter the profit of job 1:			
190			
Enter the deadline of job 1:			
3			
Enter the profit of job 2:			
110			
Enter the deadline of job 2:			
4			
Enter the profit of job 3:			
50			
Enter the deadline of job 3:			
2			
Index	Profit	Deadline	Slot Alloted
1	190	3	[2 - 3]
2	110	4	[3 - 4]
3	50	2	[1 - 2]

S.No: 23	Exp. Name: ***Activity Selection Problem***	Date: 2023-03-31

Aim:

Write a C program to implement Greedy algorithm using **Activity Selection Problem**.

Note:

- The activities are numbered from 0 to n-1, where **n** is the number of activities.
- Sort the activities according to their finishing time.
- If the activity has start time greater than or equal to the finish time of previously selected activity, then select it.

Source Code:

ASP.c

```
#include <stdio.h>
#include <stdlib.h>
struct Activity {
        int start;
        int finish;
};
int compare(const void *a, const void *b) {
        return (((struct Activity *)a)->finish - ((struct Activity *)b)->finish);
}
void printMaxActivities(struct Activity arr[], int n) {
        qsort(arr, n, sizeof(arr[0]), compare);
        int i = 0;
        printf("The following activities are selected: \n");
        printf("%d %d\n", arr[i].start, arr[i].finish);
        for (int j = 1; j < n; j++) {
                if (arr[j].start >= arr[i].finish) {
                        printf("%d %d\n", arr[j].start, arr[j].finish);
                        i = j;
                }
        }
}
int main() {
        int n;
        printf("Enter the number of activities: ");
        scanf("%d", &n);
        struct Activity arr[n];
        printf("Enter the start and finish times of each activity: \n");
        for (int i = 0; i < n; i++) {
                scanf("%d%d", &arr[i].start, &arr[i].finish);
        }
        printMaxActivities(arr, n);
        return 0;
}
```

Execution Results - All test cases have succeeded!

User Output
Enter the number of activities:
4
Enter the start and finish times of each activity:
1 10
9 6
8 5
6 3
The following activities are selected:
6 3
8 5
9 6

S.No: 24	Exp. Name: ***Hamiltonian Cycles in a connected undirected Graph of vertices using backtracking principle.***	Date: 2023-04-01

Aim:

Write a C program to find all **Hamiltonian Cycles** in a connected **undirected Graph** G of **n** vertices using **backtracking** principle.

Note: Input **1** if there's an edge and **0** otherwise.

Source Code:

Hamiltonian.c

```
#include<stdio.h>
#include<stdlib.h>
#include <stdbool.h>
int V;
void printSolution(int path[]);
bool isSafe(int v, bool** graph, int path[], int pos) {
        if (graph[path[pos-1]][v] == 0)
                return false;
        for (int i = 0; i < pos; i++)
                if (path[i] == v)
                                return false;
        return true;
}
bool hamCycleUtil(bool** graph, int path[], int pos) {
                if (pos == V) {
                        if
                        ( graph[path[pos-1]][path[0]] ==1)
                        return true;
                        else
                        return false;
                        }
                for (int v= 1;v< V; v++)
                {
                                if (isSafe(v, graph, path, pos)) {
                                        path[pos]= v;
                                        if (hamCycleUtil (graph, path, pos+1) == true)
                                        return true;
                                        path[pos]=-1;
                                }
                }
                return false;
}
bool hamCycle(bool** graph) {
                int *path = (int*) malloc(V * sizeof(int));
                for (int i= 0;i< V; i++)
                        path[i]=-1;

                path[0]= 0;
                if( hamCycleUtil(graph, path, 1) == false)
                {
                        printf("Solution does not exist");
                        free(path);
                        return false;
                }
                else
                {
                        printSolution(path);
                        free(path);
                        return true;
                }
}
void printSolution(int path[])
                {
                        printf ("Following is one Hamiltonian Cycle: \n");
                        for (int i = 0; i < V; i++)
```

```
                        printf("\n");
}
int main()
{
        printf("Number of vertices: ");
        scanf("%d", &V);
        bool** graph = (bool**) malloc(V * sizeof(bool*));
        for (int i = 0; i < V; i++)
                        graph[i] = (bool*) malloc(V * sizeof(bool));
        printf("Enter the adjacency matrix:\n");
        for (int i = 0; i < V; i++)
                        for (int j = 0; j < V; j++)
                                scanf("%d", &graph[i][j]);

                        hamCycle(graph);

        for (int i = 0; i < V; i++)
                        free(graph[i]);
        free(graph);
        return 0;
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Number of vertices:
5
Enter the adjacency matrix:
0 1 0 1 0
1 0 1 1 1
0 1 0 0 1
1 1 0 0 1
0 1 1 1 0
Following is one Hamiltonian Cycle:
0 1 2 4 3 0

Test Case - 2
User Output
Number of vertices:
5
Enter the adjacency matrix:
0 1 0 1 0
1 0 1 1 1
0 1 0 0 1
1 1 0 0 1
0 1 0 0 0

S.No: 25	Exp. Name: ***Longest common subsequence using dynamic programming (Top Down)***	Date: 2023-03-31

Aim:

Longest common subsequence using dynamic programming (Top Down)

- In this approach we are using the same recursive implementation that we have used in the implementation of Bruteforce approach.
- But to avoid recomputing the results of overlapping subproblems we use a **table of size (m + 1) x (n + 1)** where m is the length of string 1 and n is the length of sting 2.
- Initially we **initialise the 2 dimentional table with -1 values**, And as we compute the solution to some subproblem, we store its result in the table, The reason to store the result is that, if we encounter the same subproblem again then we will just use that result from the table rather then wasting our time in recomputing the subproblem again.

Complete the function **int longestCommonSubsequence(char *str1, char *str2, int m, int n, int **Table)** given in the file **LcsFunction.c.**
Here str1 and str2 are the strings with m, n as their respective lengths. Table is a 2-D container of size (m + 1) x (n + 1). In the driver code you can see Table[m + 1][n + 1] is already initialised with -1 values**.**
(Refer pseudo-code given in the hint section to solve LCS using Top-Down dunamic programming).

Source Code:

LcsTopDownDPmain.c

```
/* implementation of LCS problem (Recursive - Brute force with exponential time */
#include<stdio.h>
#include<stdlib.h>
#include"LcsFunction.c"

/* main method*/
int main()
{
char *A, *B;
int m, n, i, j;
printf("Enter the length of string A and B: ");
scanf("%d%d",&m,&n);
printf("Enter string A followed by string B: ");
A = (char *)malloc(m * sizeof(char));
B = (char *)malloc(n * sizeof(char));
int** Table = (int**)malloc((m + 1) * sizeof(int*));
for (i = 0; i < (m + 1); i++)
        Table[i] = (int*)malloc((n + 1) * sizeof(int));
for(i = 0; i< (m + 1) ; i++) {
        for(j = 0; j < (n+ 1); j++)
                Table[i][j] = -1;
}
scanf("%s%s",A,B);
printf("LCS length is = %d", longestCommonSubsequence( A, B, m, n, Table ) );

return 0;
}
```

LcsFunction.c

```
/* Finding max out of two integer values (Utility function */
int maximum(int a, int b){
        return (a > b)? a : b;
}
/* Function computes and retun the length of Longest Common Subsequence
for A[0..m-1], B[0..n-1] */
int longestCommonSubsequence( char *A, char *B, int m, int n, int **Table ) {
        if (m == 0 || n == 0)
        return 0;
        if (A[m - 1] == B[n - 1])
        return Table[m][n] = 1 + longestCommonSubsequence(A, B, m - 1, n - 1,
        Table);
        if (Table[m][n] != -1) {
                return Table[m][n];
        }
        return Table[m][n] = maximum(longestCommonSubsequence(A, B, m, n - 1, Table),
        longestCommonSubsequence(A, B, m - 1, n, Table));
        }
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter the length of string A and B:
6 7
Enter string A followed by string B:
AGGTAB GXTXAYB
LCS length is = 4

Test Case - 2
User Output
Enter the length of string A and B:
5 4
Enter string A followed by string B:
APPLE CAPP
LCS length is = 3

S.No: 26	Exp. Name: ***Huffmann coding***	Date: 2023-04-01

Aim:

Huffman coding is a lossless data compression algorithm. In this algorithm, for the given input characters a variable-length code is generated.

The length of the code is related to the frequency of characters i.e most frequent characters in the input have the smallest code whare as least frequent characters have longer codes.

For example, consider "aaabccaabb", the frequency of character **a** is larger than **b** and the character **c** has the least frequency. So the length of the code for a is smaller than b, and code for b will be smaller than c.

There are mainly two parts in Huffman coding.

15. The first one is to create a Huffman tree
16. And another one is to traverse the tree to find codes.

Procedure steps for building a Huffman tree

17. Create a leaf node for each unique character and build a min-heap of all leaf nodes
18. Extract two nodes which are having minimum frequency from the min-heap
19. Create a new internal node with a frequency equal to the sum of the two node frequencies.
20. Make the first extracted node as its left child and the other extracted node as its right child.
21. Add this node to the min-heap.
22. Repeat these steps until the heap contains only one node. The remaining node is the root node and the tree is complete.

Source Code:

haffman.c

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_TREE_HT 100
struct MinHeapNode {
        char data;
        unsigned freq;
        struct MinHeapNode *left, *right;
};
struct MinHeap {
        unsigned size;
        unsigned capacity;
        struct MinHeapNode** array;
};
struct MinHeapNode* newNode(char data, unsigned freq) {
        struct MinHeapNode* temp = (struct MinHeapNode*)malloc(sizeof(struct MinHeapNode));
        temp->left = temp->right = NULL;
        temp->data = data;
        temp->freq = freq;
        return temp;
}
struct MinHeap* createMinHeap(unsigned capacity) {
        struct MinHeap* minHeap = (struct MinHeap*)malloc(sizeof(struct MinHeap));
        minHeap->size = 0;
        minHeap->capacity = capacity;
        minHeap->array = (struct MinHeapNode**)malloc(minHeap->capacity * sizeof(struct
        MinHeapNode*));
        return minHeap;
        }
        void swapMinHeapNode(struct MinHeapNode** a, struct MinHeapNode** b) {
                struct MinHeapNode* t = *a;
                *a = *b;
                *b = t;
        }
        void minHeapify(struct MinHeap* minHeap, int idx) {
                int smallest = idx;
                int left = 2 * idx + 1;
                int right = 2 * idx + 2;
                if (left < minHeap->size && minHeap->array[left]->freq < minHeap-
>array[smallest]->freq)
                smallest = left;
                if (right < minHeap->size && minHeap->array[right]->freq < minHeap-
>array[smallest]->freq)
                smallest = right;
                if (smallest != idx) {
                        swapMinHeapNode(&minHeap->array[smallest], &minHeap->array[idx]);
                        minHeapify(minHeap, smallest);
                }
        }
int isSizeOne(struct MinHeap* minHeap) {
        return (minHeap->size == 1);
}
struct MinHeapNode* extractMin(struct MinHeap* minHeap) {
        struct MinHeapNode* temp = minHeap->array[0];
        minHeap->array[0] = minHeap->array[minHeap->size - 1];
        --minHeap->size;
```

```
}
void insertMinHeap(struct MinHeap* minHeap, struct MinHeapNode* minHeapNode) {
        ++minHeap->size;
        int i = minHeap->size - 1;
        while (i && minHeapNode->freq < minHeap->array[(i - 1) / 2]->freq) {
                minHeap->array[i] = minHeap->array[(i - 1) / 2];
                i = (i - 1) / 2;
        }
        minHeap->array[i] = minHeapNode;
}
void buildMinHeap(struct MinHeap* minHeap) {
        int n = minHeap->size - 1;
        int i;
        for (i = (n - 1) / 2; i >= 0; --i)
        minHeapify(minHeap, i);
}
void printArr(int arr[], int n) {
        int i;
        for (i = 0; i < n; ++i)
        printf("%d", arr[i]);
        printf("\n");
}
int isLeaf(struct MinHeapNode* root) {
        return !(root->left) && !(root->right);
}
struct MinHeap* createAndBuildMinHeap(char data[], int freq[], int size) {
        struct MinHeap* minHeap = createMinHeap(size);
        for (int i = 0; i < size; ++i)
        minHeap->array[i] = newNode(data[i], freq[i]);
        minHeap->size = size;
        buildMinHeap(minHeap);
        return minHeap;
}
struct MinHeapNode* buildHuffmanTree(char data[], int freq[], int size) {
        struct MinHeapNode *left, *right, *top;
        struct MinHeap* minHeap = createAndBuildMinHeap(data, freq, size);
        while (!isSizeOne(minHeap)) {
                left = extractMin(minHeap);
                right=extractMin(minHeap);
                top=newNode('$',left->freq+right->freq);
                top->left = left;
                top->right = right;
                insertMinHeap(minHeap, top);
        }
        return extractMin(minHeap);
}
void printCodes(struct MinHeapNode* root, int arr[], int top) {
        if (root->left) {
                arr[top] = 0;
                printCodes(root->left, arr, top + 1);
        }
        if (root->right) {
                arr[top] = 1;
                printCodes(root->right, arr, top + 1);
        }
```

```
                printArr(arr, top);
        }
}
void HuffmanCodes(char data[], int freq[], int size) {
        struct MinHeapNode* root = buildHuffmanTree(data, freq, size);
        int arr[MAX_TREE_HT], top = 0;
        printCodes(root, arr, top);
}
int main()
{

        char arr[50];
        int frequency[50],i, n;

        printf("Enter number of characters: ");
        scanf("%d", &n);
        printf("Enter %d characters:\n", n);
        for(i = 0; i < n; i++)
        {
                scanf("%s", &arr[i]);
        }
        printf("Enter frequency of each character:\n");
        for(i = 0; i < n; i++)
        {
                scanf("%d", &frequency[i]);
        }
        int size = n / sizeof(arr[0]);
        printf("Char | Huffman code\n");
        HuffmanCodes(arr, frequency, size);
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter number of characters:
5
Enter 5 characters:
w
r
y
h
v
Enter frequency of each character:
3
4

56
8
9
Char | Huffman code
v --> 00
w --> 0100
r --> 0101
h --> 011
y --> 1

Test Case - 2
User Output
Enter number of characters:
6
Enter 6 characters:
a
b
c
d
e
f
Enter frequency of each character:
0
1
2
3
4
5
Char | Huffman code
d --> 00
a --> 0100
b --> 0101
c --> 011
e --> 10
f --> 11

Test Case - 3
User Output
Enter number of characters:
8
Enter 8 characters:
h
t
d
v

i
q
Enter frequency of each character:
90
5
6
45
11
2
34
78
Char | Huffman code
v --> 00
u --> 0100
d --> 01010
m --> 010110
t --> 010111
i --> 011
q --> 10
h --> 11

Test Case - 4
User Output
Enter number of characters:
2
Enter 2 characters:
1
5
Enter frequency of each character:
34
56
Char | Huffman code
1 --> 0
5 --> 1

Test Case - 5
User Output
Enter number of characters:
5
Enter 5 characters:
w
g
u
n
i
Enter frequency of each character:

6
4
56
89
Char | Huffman code
g --> 000
w --> 0010
u --> 0011
n --> 01
i --> 1

S.No: 27	Exp. Name: ***Minimum spanning tree - Kruskal's Algorithm***	Date: 2023-04-01

Aim:

Write a C program to implement Kruskal's algorithm to generate a minimum cost spanning tree.

Source Code:

KruskalSpanningTree.c

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
int i,j,k,a,b,u,v,n,e,s,d,w,ne=1;
int min,mincost=0,cost[9][9],parent[9];
int find(int i)
{
        while(parent[i])
        i=parent[i];
        return i;
}
int uni(int i,int j)
{
        if(i!=j)
        {
                parent[j]=i;
                return 1;
        }
        return 0;
}
void kruskal()
{
        while(ne
        < n)
        {
                for(i=1,min=999;i<=n;i++)
                {
                        for(j=1;j <= n;j++)
                        {
                                if(cost[i][j]
                                < min)
                                {
                                        min=cost[i][j];
                                        a=u=i;
                                        b=v=j;
                                }
                        }
                }
                u=find(u);
                v=find(v);
                if(uni(u,v))
                {
                        printf("Edge cost from %d to %d : %d\n",a,b,min);
                        mincost +=min;
                        ne++;
                        }
                        cost[a][b]=cost[b][a]=999;
                        }
                        printf("Minimum cost of spanning tree = %d\n",mincost);
                        }
                        void main()
                        {
                                printf("Enter the number of vertices : ");
                                scanf("%d",&n);
                                printf("Enter the number of edges : ");
```

```
                                {
                                        printf("Enter source : ");
                                        scanf("%d",&s);
                                        printf("Enter destination : ");
                                        scanf("%d",&d);
                                        printf("Enter weight : ");
                                        scanf("%d",&w);
                                        if(s<=0 || d<=0 || s> n || d > n || w < 0 ) {
                                                printf("Invalid data.Try again.\n");
                                                i--;
                                        }
                                                cost[s][d]=w;
                                        }
                                        for(i=1;i<=n;i++) {
                                                for(j=1;j<=n;j++) {
                                                        if(cost[i][j]==0)
                                                        cost[i][j]=999;
                                                }
                                        }
                                        printf("The edges of Minimum Cost Spanning Tree are
: \n");
                                        kruskal();
                                }
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter the number of vertices :
7
Enter the number of edges :
9
Enter source :
2
Enter destination :
7
Enter weight :
14
Enter source :
1
Enter destination :
2
Enter weight :
28
Enter source :
1
Enter destination :
6
Enter weight :

Enter source :
5
Enter destination :
6
Enter weight :
25
Enter source :
4
Enter destination :
5
Enter weight :
22
Enter source :
4
Enter destination :
7
Enter weight :
18
Enter source :
2
Enter destination :
3
Enter weight :
16
Enter source :
3
Enter destination :
4
Enter weight :
12
Enter source :
5
Enter destination :
7
Enter weight :
24
The edges of Minimum Cost Spanning Tree are :
Edge cost from 1 to 6 : 10
Edge cost from 3 to 4 : 12
Edge cost from 2 to 7 : 14
Edge cost from 2 to 3 : 16
Edge cost from 4 to 5 : 22
Edge cost from 5 to 6 : 25
Minimum cost of spanning tree = 99

Test Case - 2
User Output
Enter the number of vertices :

Enter the number of edges :
11
Enter source :
1
Enter destination :
2
Enter weight :
7
Enter source :
2
Enter destination :
3
Enter weight :
8
Enter source :
1
Enter destination :
4
Enter weight :
5
Enter source :
2
Enter destination :
4
Enter weight :
9
Enter source :
2
Enter destination :
5
Enter weight :
7
Enter source :
3
Enter destination :
5
Enter weight :
5
Enter source :
4
Enter destination :
5
Enter weight :
15
Enter source :
4
Enter destination :
6

```
Enter source :
5
Enter destination :
6
Enter weight :
8
Enter source :
5
Enter destination :
7
Enter weight :
9
Enter source :
6
Enter destination :
7
Enter weight :
11
The edges of Minimum Cost Spanning Tree are :
Edge cost from 1 to 4 : 5
Edge cost from 3 to 5 : 5
Edge cost from 4 to 6 : 6
Edge cost from 1 to 2 : 7
Edge cost from 2 to 5 : 7
Edge cost from 5 to 7 : 9
Minimum cost of spanning tree = 39
```

S.No: 28	Exp. Name: ***Travelling Sales Person problem using Dynamic programming***	Date: 2023-04-01

Aim:

Write a C program to implement **Travelling Sales Person** problem using **Dynamic programming**.

Source Code:

TSP.c

```
#include<stdio.h>
int ary[10][10], completed[10], n, cost= 0;
void takeInput(){
        int i, j;
        printf("Number of villages: ");
        scanf("%d",& n);
        for (i = 0; i < n; i++) {
                for (j= 0;j< n; j++)
                scanf("%d",& ary[i][j]);
                completed[i]= 0;
                }
                printf("The cost list is:");
                for (i= 0;i< n; i++){
                        printf("\n");
                        for (j= 0;j< n; j++)
                        printf("\t%d", ary[i][j]);
                        }

}
void mincost(int city) {
        int i, ncity;
        completed[city]= 1;
        printf("%d-->", city + 1);
        ncity= least(city);
        if (ncity == 999) {
                ncity= 0;
                printf("%d", ncity+ 1);
                cost += ary[city][ncity];
                return;
                }
                mincost(ncity);

}
int least(int c) {
        int i, nc= 999;
        int min= 999, kmin;
        for (i= 0;i< n; i++){
                if ((ary[c][i] != 0) && (completed[i] == 0))
                if (ary[c][i]+ ary[i][c]< min){
                        min = ary[i][0] + ary[c][i];
                        kmin= ary[c][i];
                        nc= i;
                        }
                        }
                        if (min != 999)
                        cost += kmin;
                        return nc;

}
int main()
{       takeInput();
printf("\nThe Path is:\n");
mincost(0);
printf("\nMinimum cost is %d", cost);
return 0;
```

```
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Number of villages:
3
0 10 15
10 0 35
15 35 0
The cost list is:
0 10 15
10 0 35
15 35 0
The Path is:
1-->2-->3-->1
Minimum cost is 60

S.No: 29	Exp. Name: ***Subset of a given set S of n positive integers whose sum is equal to a given positive integer.***	Date: 2023-04-01

Aim:

Design and implement a C program to find a subset of a given set S = {Sl, S2,.....,Sn} of n positive integers whose sum is equal to a given positive integer d. For example, if S ={1, 2, 5, 6, 8} and d= 9, there are two solutions {1,2,6}and {1,8}. Display a suitable message, if the given problem instance doesn't have a solution.

Source Code:

SubSets.c

```
#include <stdio.h>
int s[10], x[10], d;
void sumofsub(int, int, int);
void main()
{int n, sum = 0;
int i;
printf("Size of set : ");
scanf("%d", & n);
printf("Elements in increasing order:\n");
for (i = 1; i <= n; i++)
scanf("%d", & s[i]);
printf("Target Sum: ");
scanf("%d", &d);
for (i = 1; i <= n; i++)
sum = sum + s[i];
if (sum < d || s[1] > d)
printf("No subset possible");
else
sumofsub(0, 1, sum);
getchar();
}
void sumofsub(int m, int k, int r)
{
        int i = 1;
        x[k] = 1;
        if ((m + s[k]) == d)
        {
                printf("Subset:");
                for (i = 1; i <= k; i++)
                if (x[i] == 1)
                printf("\t%d", s[i]);
                printf("\n");

        }
        else
        if (m + s[k] + s[k + 1] <= d)
        sumofsub(m + s[k], k + 1, r - s[k]);
        if ((m + r - s[k] >= d) && (m + s[k + 1] <= d))
        {
                x[k] = 0;
                sumofsub(m, k + 1, r - s[k]);

        }

}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Size of set :
6

1 2 3 4 5 6
Target Sum:
6
Subset: 1 2 3
Subset: 1 5
Subset: 2 4
Subset: 6

Test Case - 2
User Output
Size of set :
5
Elements in increasing order:
4 5 6 7 8
Target Sum:
3
No subset possible