MIT | Arts, Commerce & Science College

Alandi (D), Pune – 412 105

MASTER OF COMPUTER SCIENCE[M.Sc. COMPUTER SCIENCE]

PRACTICAL JOURNAL

DEPARTMENT OF SCIENCE AND COMPUTER SCIENCE

(ACEDEMIC YEAR 2023-2024)

Name of the student: …………………………………………… …..

Roll No.: …………….. University Exam No: …………………

Class: MSC (computer Science): F.Y. / S. Y.

Subject: ………………………………………………………..

MIT | Arts, Commerce & Science College

Alandi (D), Pune – 412 105

DEPARTMENT OF SCIENCE AND COMPUTER SCIENCE

CERTIFICATE

This is to certify that Mr./Miss ………………………………………

Of F.Y.MSC (Computer Science), Exam Seat No. …………

Has satisfactorily completed his/ her practical's in the

Subject………………………………………..

As laid down by the Savitribai Phule Pune University for the academic

Year…………

Date: ………..

Subject Teacher **Head of the Department**

External Examiner **Internal Examiner**

MIT | Arts, Commerce & Science College

Alandi (D), Pune – 412 105

INDEX

15	C program to prim’s algorithm.	09-02-2024		
16	C program to Dijkstra algorithm.	09-02-2024		
17	C program to min max algorithm.	09-02-2024		
18	C program to DFS.	16-02-2024		
19	C program to BFS.	16-02-2024		
20	C program to topological sorting.	16-02-2024		
21	C program to Knapsack with greedy method.	23-02-2024		
22	C program to optimal binary search.	23-02-2024		
23	C program to 0/1 knapsack.	15-03-2024		
24	C program to matrix chain multiplication.	15-03-2024		
25	C program to find longest common subsequence.	15-03-2024		
26	C program to N Queen problem.	22-03-2024		
27	C program to Travelling salesman problem.	22-03-2024		
28	C program to sum of subset.	08-04-2024		
29	C program to Graph coloring algorithm	08-04-2024		
30	C program to Hamilton cycle.	08-04-2024		
31	C program to Huffman coding	08-04-2024		

1) Addition of two arrays

```
#include <stdio.h>
int main() {
    int arr1[100], arr2[100], sum[100], n;
    printf("Enter the size of the arrays: ");
    scanf("%d", &n);
    printf("Enter elements of first array: ");
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr1[i]);
    }
    printf("Enter elements of second array: ");
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr2[i]);
    }
   printf("Sum of the arrays: \n");
    for (int i = 0; i < n; i++) {
        sum[i] = arr1[i] + arr2[i];
        printf("%d ", sum[i]);
    }
    printf("\n");
return 0;
}
```

Output:

```
Enter the size of the arrays: 5
Enter elements of first array: 2
6
1
9
5
Enter elements of second array: 1
3
7
2
8
Sum of the arrays:
3 9 8 11 13
```

2) find element in array

```
#include <stdio.h>
int main() {
    int arr[100], n, element;

    printf("Enter the size of the array: ");
    scanf("%d", &n);

    printf("Enter elements of array: ");
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }

    printf("Enter element to search: ");
    scanf("%d", &element);

    int found = 0;
    for (int i = 0; i < n; i++) {
        if (arr[i] == element) {
            found = 1;
            break;
        }
    }

    if (found) {
        printf("Element found in the array.\n");
    } else {
        printf("Element not found in the array.\n");
    }

    return 0;
}
```

Output:

```
Enter the size of the array: 5
Enter elements of array: 2
9
1
5
3
Enter element to search: 1
Element found in the array
```

3) Find minimum and maximum element from array

```
#include <stdio.h>
int main() {
    int arr[100], n, min, max;

    printf("Enter the size of the array: ");
    scanf("%d", &n);

    printf("Enter elements of array: ");
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }

    min = max = arr[0]; // Initialize with first element

    for (int i = 0; i < n; i++) {
        if (arr[i] < min) {
            min = arr[i];
        } else if (arr[i] > max) {
            max = arr[i];
        }
    }

    printf("Minimum element: %d\n", min);
    printf("Maximum element: %d\n", max);

    return 0;
}
```

Output:

```
Enter the size of the array: 4
Enter elements of array: 2
12
7
3
Minimum element: 2
Maximum element: 12
```

4) write c program for bubble sort

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

int main() {
    int arr[100], n;

    printf("Enter the size of the array: ");
    scanf("%d", &n);

    printf("Enter elements of array: ");
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }

    bubble_sort(arr, n);

    printf("Sorted array: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
printf("\n");
    return 0;
}
```

Output:

```
Enter the size of the array: 4
Enter elements of array: 23
45
7
2
Sorted array: 2 7 23 45
```

5)write c program for factorial of number

```
#include <stdio.h>
int factorial(int n) {
    if (n == 0 || n == 1) {
        return 1;
    } else {
        int fact = 1;
        for (int i = 2; i <= n; i++) {
            fact *= i;
        }
        return fact;
    }
}

int main() {
    int num;

    printf("Enter a non-negative number: ");
    scanf("%d", &num);

    int result = factorial(num);

    printf("Factorial of %d is %d\n", num, result);

    return 0;
}
```

Output:

```
Enter a non-negative number: 5
Factorial of 5 is 120
```

6) write c program for recursive factorial of number

```
#include <stdio.h>
int factorial(int n) {
    if (n == 0 || n == 1) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}

int main() {
    int num;

    printf("Enter a non-negative number: ");
    scanf("%d", &num);

    int result = factorial(num);

    printf("Factorial of %d is %d\n", num, result);

    return 0;
}
```

Output:

```
 Enter a non-negative number: 3
 Factorial of 3 is 6
```

7) write c program to insertion sort

```
#include <stdio.h>
void insertion_sort(int arr[], int n) {
   int key, j;
   for (int i = 1; i < n; i++) {
      key = arr[i];
      j = i - 1;

      // Move elements of arr[0..i-1], that are greater than key, to one position
ahead of their current position
      while (j >= 0 && arr[j] > key) {
         arr[j + 1] = arr[j];
         j--;
      }
      arr[j + 1] = key;
   }
}
int main() {
   int arr[100], n;
   printf("Enter the size of the array: ");
   scanf("%d", &n);

   printf("Enter elements of array: ");
   for (int i = 0; i < n; i++) {
      scanf("%d", &arr[i]);
   }
   insertion_sort(arr, n);

   printf("Sorted array: ");
   for (int i = 0; i < n; i++) {
      printf("%d ", arr[i]);
   }
   printf("\n");
   return 0;
}
```

Output:

```
Enter the size of the array: 4
Enter elements of array: 8
14
3
7
Sorted array: 3 7 8 14
```

8) write c program for selection sort

```
#include <stdio.h>
void selection_sort(int arr[], int n) {
    int min_idx, temp;
    for (int i = 0; i < n - 1; i++) {
        min_idx = i;
        for (int j = i + 1; j < n; j++) {
            if (arr[j] < arr[min_idx]) {
                min_idx = j;
            }
        }
        // Swap the found minimum element with the first element
        if (min_idx != i) {
            temp = arr[min_idx];
            arr[min_idx] = arr[i];
            arr[i] = temp;
        }
    }
}
int main() {
    int arr[100], n;
    printf("Enter the size of the array: ");
    scanf("%d", &n);
    printf("Enter elements of array: ");
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }
    selection_sort(arr, n);
    printf("Sorted array: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }

    printf("\n");
    return 0;
}
```

Output:

```
Enter the size of the array: 5
Enter elements of array: 2 8 1 5 3
Sorted array: 1 2 3 5 8
```

9) c program for Quick sort

```
#include<stdio.h>

int main()
  {
   int a[10]={6,12,0,18,11,99,55,45,34,2};
   int n=10;
   void quicksort(int a[10],int lb,int ub)
     {
       int l,h,pivot,temp;
       if(lb<ub)
       {
         pivot=lb;
          l=lb;
          h=ub;
       while(l<h)
         {
           while(a[l]<=a[pivot] && l<ub )
             l++;
           while(a[h]>a[pivot])
             h--;
           if(l<h){
           temp=a[l];
             a[l]=a[h];
             a[h]=temp;
             }
         }
       temp=a[pivot];
       a[pivot]=a[h];
       a[h]=temp;
       quicksort(a,lb,h-1);
       quicksort(a,h+1,ub);
     }
   }
quicksort(a,0,n-1);
   for (int i = 0; i < n; i++)
      {
       printf("%d\t", a[i]);
      }
   return 0;  }
```

Output: 0   2   6   11   12   18   34   45   55   99

10) C program for Binary search

```
#include <stdio.h>
int binary_search(int arr[], int low, int high, int key) {
    if (low > high) {
        return -1;
    }

    int mid = low + (high - low) / 2;

    if (arr[mid] == key) {
        return mid;
    } else if (arr[mid] < key) {
        return binary_search(arr, mid + 1, high, key);
    } else {
        return binary_search(arr, low, mid - 1, key);
    }
}
int main() {
    int arr[100], n, key;
    printf("Enter the size of the array: ");
    scanf("%d", &n);
    printf("Enter elements of array (sorted in ascending order): ");
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }
    printf("Enter the element to search: ");
    scanf("%d", &key);
    int result = binary_search(arr, 0, n - 1, key);
    if (result == -1) {
        printf("Element is not present in array\n");
    } else {
        printf("Element is found at index %d\n", result);
    }
  return 0;
}
```

Output:

```
Enter the size of the array: 3
Enter elements of array (sorted in ascending order): 2
4
6
Enter the element to search: 4
Element is found at index 1
```

11) C program for merge sort

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

void merge(int arr[], int l, int m, int r)
{
    int i, j, k;
    int n1 = m - l + 1;
    int n2 = r - m;
    int L[n1], R[n2];

    for (i = 0; i < n1; i++)
        L[i] = arr[l + i];
    for (j = 0; j < n2; j++)
        R[j] = arr[m + 1 + j];

    i = 0;
    j = 0;
    k = l;
    while (i < n1 && j < n2)
     {
        if (L[i] <= R[j])
        {
            arr[k] = L[i];
            i++;
        }
        else
        {
            arr[k] = R[j];
            j++;
        }
        k++;
    }
      while (i < n1)
     {
        arr[k] = L[i];
        i++;
        k++;
    }
      while (j < n2)
     {
        arr[k] = R[j];
        j++;
        k++;
    }
}
void mergeSort(int arr[], int l, int r)
{
    if (l < r)
    {
           int m = l + (r - l) / 2;
        mergeSort(arr, l, m);
        mergeSort(arr, m + 1, r);
        merge(arr, l, m, r);
    }
}
  void printArray(int A[], int size)
{
    int i;
    for (i = 0; i < size; i++)
        printf("%d ", A[i]);
    printf("\n");
}

int main()
{
    int arr[] = { 12, 11, 13, 5, 6, 7 };
    int arr_size = sizeof(arr) / sizeof(arr[0]);

    printf("Given array is \n");
    printArray(arr, arr_size);

    mergeSort(arr, 0, arr_size - 1);

    printf("\nSorted array is \n");
    printArray(arr, arr_size);
    return 0;
}
```

Output:

```
Given array is
12 11 13 5 6 7

Sorted array is
5 6 7 11 12 13
```

12) C program to preform Bucket sort

```
#include <stdio.h>
void bucketsort(int a[], int n){ // function to implement bucket sort
   int max = a[0]; // get the maximum element in the array
   for (int i = 1; i < n; i++)
      if (a[i] > max)
         max = a[i];
   int b[max], i;
   for (int i = 0; i <= max; i++) {
      b[i] = 0;
   }
   for (int i = 0; i < n; i++) {
      b[a[i]]++;
   }
   for (int i = 0, j = 0; i <= max; i++) {
      while (b[i] > 0) {
         a[j++] = i;
         b[i]--;
      }
   }
}
int main(){
   int a[] = {12, 45, 33, 87, 56, 9, 11, 7, 67};
   int n = sizeof(a) / sizeof(a[0]); // n is the size of array
   printf("Before sorting array elements are: \n");
   for (int i = 0; i < n; ++i)
      printf("%d ", a[i]);
   bucketsort(a, n);
   printf("\nAfter sorting array elements are: \n");
   for (int i = 0; i < n; ++i)
      printf("%d ", a[i]);
}
```

Output:

```
Before sorting array elements are:
12 45 33 87 56 9 11 7 67
After sorting array elements are:
7 9 11 12 33 45 56 67 87
```

13) C program to preform counting sort

```
#include<stdio.h>

void counting_sort(int a[],int n,int max)
{
     int count[50]={0},i,j;

     for(i=0;i<n;++i)
      count[a[i]]=count[a[i]]+1;
     printf("\nSorted elements are:");

     for(i=0;i<=max;++i)
      for(j=1;j<=count[i];++j)

       printf("%d ",i);
}
int main()
{
    int a[50],n,i,max=0;
    printf("Enter number of elements:");
    scanf("%d",&n);
    printf("\nEnter elements:");

    for(i=0;i<n;++i)
    {
     scanf("%d",&a[i]);
     if(a[i]>max)
      max=a[i];
    }
    counting_sort(a,n,max);
    return 0;
}
```

Output:

```
Enter number of elements:5
Enter elements:21
23
33
11
45
Sorted elements are:11 21 23 33 45
```

14) C program to preform Strassen's matrix

```
#include<stdio.h>
int main()
             {
               int n;
               printf("Enter number of elements you want to enter into matrics:\n");
               scanf("%d",&n);
               int a[n][n],b[n][n],c[n][n],i,j;
               int p,q,r,s,t,u,v;
               printf("Enter the elements of first matrix: ");
      for(i=0;i<n;i++)
        {
          for(j=0;j<n;j++)
           {
             scanf("%d",&a[i][j]);
            }
        }
        printf("\n");

      printf("Enter the elements of second matrix: ");
      for(i=0;i<n;i++)
        {
          for(j=0;j<n;j++)
           {
             scanf("%d",&b[i][j]);
            }
        }
        printf("\n");
       printf("first matrix\n");

        for(i=0;i<n;i++)
         {
           printf("\n");
           for(j=0;j<n;j++)
            {
              printf("%d\t",a[i][j]);
            }
          }
        printf("\n");
       printf("Second matrix\n");

        for(i=0;i<n;i++)
         {
           printf("\n");
           for(j=0;j<n;j++)
            {
              printf("%d\t",b[i][j]);
            }
          }
       printf("\n");
       p=(a[0][0]+a[1][1])*(b[0][0]+b[1][1]);
       q=(a[1][0]+a[1][1])*b[0][0];
       r=a[0][0]*(b[0][1]-b[1][1]);
       s=a[1][1]*(b[1][0]-b[0][0]);
       t=(a[0][0]+a[0][1])*b[1][1];
       u=(a[1][0]-a[0][0])*(b[0][0]+b[0][1]);
       v=(a[0][1]-a[1][1])*(b[1][0]+b[1][1]);

       c[0][0]=p+s-t+v;
       c[0][1]=r+t;
       c[1][0]=q+s;
       c[1][1]=p+r-q+u;

       printf("Result matrix\n");
        for(i=0;i<n;i++)
         {
           printf("\n");
           for(j=0;j<n;j++)
            {
              printf("%d\t",c[i][j]);
            }
          }
                         return 0;
             }
```

Output:

```
Enter number of elements you want to enter into matrics:
2
Enter the elements of first matrix:
2
3
4
5
```

```
Enter the elements of second matrix:
1
2
3
4
first matrix
2       3
4       5
Second matrix
1       2
3       4
Result matrix
11      16
19      28
```

15)C program to preform prim's algorithm

```
#include<stdio.h>

int a,b,u,v,n,i,j,ne=1;
int visited[10]= {0},min,mincost=0,cost[10][10];

void main()
{
printf("\n Enter the number of nodes:");
scanf("%d",&n);
printf("\n Enter the adjacency matrix of weight:\n");
for (i=1;i<=n;i++)
{
 for (j=1;j<=n;j++)
 {
      printf("enter the weight of %d and %d\n",i,j);
   scanf("%d",&cost[i][j]);
   if(cost[i][j]==0)
      cost[i][j]=999;
 }
}
visited[1]=1;
printf("\n");
while(ne<n)
{
for (i=1,min=999;i<=n;i++)
  for (j=1;j<=n;j++)
   if(cost[i][j]<min)
    if(visited[i]!=0)
    {
    min=cost[i][j];
    a=u=i;
   b=v=j;
   }
if(visited[u]==0 || visited[v]==0)
{
printf("\n Edge %d:(%d %d) cost:%d",ne++,a,b,min);
mincost+=min;
visited[b]=1;
}cost[a][b]=cost[b][a]=999;
}printf("\n Minimun cost=%d",mincost);
```

Output:

```
Enter the number of nodes:6

 Enter the adjacency matrix of weight:
enter the weight of 1 and 1
0
enter the weight of 1 and 2
4
enter the weight of 1 and 3
0
enter the weight of 1 and 4
0
enter the weight of 1 and 5
0
enter the weight of 1 and 6
8
enter the weight of 2 and 1
4
enter the weight of 2 and 2
0
enter the weight of 2 and 3
20
enter the weight of 2 and 4
0
enter the weight of 2 and 5
0
enter the weight of 2 and 6
16
enter the weight of 3 and 1
0
enter the weight of 3 and 2
20
enter the weight of 3 and 3
0
enter the weight of 3 and 4
10
enter the weight of 3 and 5
6
enter the weight of 3 and 6
5
enter the weight of 4 and 1
0
enter the weight of 4 and 2
```

```
0
enter the weight of 4 and 3
10
enter the weight of 4 and 4
0
enter the weight of 4 and 5
25
enter the weight of 4 and 6
0
enter the weight of 5 and 1
0
enter the weight of 5 and 2
0
enter the weight of 5 and 3
6
enter the weight of 5 and 4
25
enter the weight of 5 and 5
0
enter the weight of 5 and 6

7
enter the weight of 6 and 1
8
enter the weight of 6 and 2
16
enter the weight of 6 and 3
5
enter the weight of 6 and 4
0
enter the weight of 6 and 5
7
enter the weight of 6 and 6
0


 Edge 1:(1 2) cost:4
 Edge 2:(1 6) cost:8
 Edge 3:(6 3) cost:5
 Edge 4:(3 5) cost:6
 Edge 5:(3 4) cost:10
 Minimun cost=33
```

16) C program to preform Dijkstra algorithm

```
#include<stdio.h>
#define INFINITY 9999
#define MAX 10

void dijkstra(int G[MAX][MAX],int n,int startnode);

int main()
{
int G[MAX][MAX],i,j,n,u;
printf("Enter no. of vertices:");
scanf("%d",&n);
printf("\nEnter the adjacency matrix:\n");

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

printf("\nEnter the starting node:");
scanf("%d",&u);
dijkstra(G,n,u);

return 0;
}

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=0;i<n;i++)
for(j=0;j<n;j++)
if(G[i][j]==0)
cost[i][j]=INFINITY;
else
cost[i][j]=G[i][j];

for(i=0;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=0;i<n;i++)
if(distance[i]<mindistance&&!visited[i])
{
mindistance=distance[i];
nextnode=i;
}

visited[nextnode]=1;
for(i=0;i<n;i++)
if(!visited[i])
if(mindistance+cost[nextnode][i]<distance[i])
{
distance[i]=mindistance+cost[nextnode][i];
pred[i]=nextnode;
}
count++;
}

for(i=0;i<n;i++)
if(i!=startnode)
{
printf("\nDistance of node%d=%d",i,distance[i]);
printf("\nPath=%d",i);
j=i;
do
{
j=pred[j];
printf("<-%d",j);
}
while(j!=startnode);
}
```

Output:

```
Enter no. of vertices:6

Enter the adjacency matrix:
0 4 0 0 0 8
4 0 20 0 0 16
0 20 0 10 6 5
0 0 10 0 25 0
0 0 6 25 0 7
8 16 5 0 7 0

Enter the starting node:1

Distance of node0=4
Path=0<-1
Distance of node2=17
Path=2<-5<-0<-1
Distance of node3=27
Path=3<-2<-5<-0<-1
Distance of node4=19
Path=4<-5<-0<-1
Distance of node5=12
Path=5<-0<-1
```

17) C program for Min max algorithm

```
 #include<stdio.h>
#include<stdio.h>
int max, min;
int a[100];
void maxmin(int i, int j)
{
            int max1, min1, mid;
            if(i==j){
                        max = min = a[i];
            }
            else
            {
              if(i == j-1)
            {
               if(a[i] <a[j])
                {
                          max = a[j];
                          min = a[i];
               }
                        else
                {
                        max = a[i];
                        min = a[j];
                }
             }
             else
                        {
                          mid = (i+j)/2;
                        maxmin(i, mid);
                        max1 = max; min1 = min;
                        maxmin(mid+1, j);
                        if(max <max1)
                         max = max1;
                        if(min > min1)
                        min = min1;
                                   }
                         }
            }
int main ()

 {
```

```
            int i, num;
            printf ("\nEnter the total number of elements : ");
            scanf ("%d",&num);
            printf ("Enter the numbers : \n");
            for (i=1;i<=num;i++)
            scanf ("%d",&a[i]);

            max = a[0];
                min = a[0];
                    maxmin(1, num);
                    printf ("Minimum number : %d\n", min);
                      printf ("Maximum number: %d\n", max);
                              return 0;
                    }
```

Output:

```
Enter the total number of elements : 6
Enter the numbers :
45
2
34
21
78
45
Minimum number : 2
Maximum number: 78
```

18) C program to preform DFS

```
#include<stdio.h>
#include<stdlib.h>
#define MAX 100
#define initial 1
#define visited 2

int n;
int adj[MAX][MAX];
int state[MAX];

void DF_Traversal();
void DFS(int v);
void create_graph();

int stack[MAX];
int top = -1;
void push(int v);
int pop();
int isEmpty_stack();

int main()
{
   int i,j,ind=0,outd=0,tot=0;

   printf("Enter Number of Vertices\n");
   scanf("%d",&n);

   for(i=0;i<n;i++)
   {
      for(j=0;j<n;j++)
      {
         printf("Enter Edges %d is Connected with %d(1 - Yes & 0 - No)\n",i,j);
         scanf("%d",&adj[i][j]);
      }
   }

   printf("Adjacency Matrix:\n\t");
   for(i=0;i<n;i++)
   {
      printf("V%d\t",i);
   }
   printf("\n");
   for(i=0;i<n;i++)
   {
      printf("V%d\t",i);
      for(j=0;j<n;j++)
      {
         printf("%d\t",adj[i][j]);
         if(adj[i][j] ==1)
         {
            outd++;
         }



      }
      printf("\n");
   }

    DF_Traversal();
}

void DF_Traversal()
{
      int v;

      for(v=0; v<n; v++)
            state[v]=initial;

      printf("\nEnter starting node for Depth First Search : ");
      scanf("%d",&v);
      DFS(v);
      printf("\n");
}

void DFS(int v)
{
      int i;
      push(v);
      while(!isEmpty_stack())
      {
            v = pop();
            if(state[v]==initial)
            {
```

```
                        printf("%d ",v);
                        state[v]=visited;
                }
                for(i=n-1; i>=0; i--)
                {
                        if(adj[v][i]==1 && state[i]==initial)
                                push(i);
                }
        }
}

void push(int v)
{
        if(top == (MAX-1))
        {
                printf("\nStack Overflow\n");
                return;
        }
        top=top+1;
        stack[top] = v;
}
int pop()
{
        int v;
        if(top == -1)
        {
                printf("\nStack Underflow\n");
                exit(1);
        }
        else
        {
                v = stack[top];
                top=top-1;
                return v;
        }
}
int isEmpty_stack( )
{
  if(top == -1)
          return 1;
  else
          return 0;
}
```

Output:

```
Enter Number of Vertices
4
Enter Edges 0 is Connected with 0(1 - Yes & 0 - No)
0
Enter Edges 0 is Connected with 1(1 - Yes & 0 - No)
1
Enter Edges 0 is Connected with 2(1 - Yes & 0 - No)
1
Enter Edges 0 is Connected with 3(1 - Yes & 0 - No)
0
Enter Edges 1 is Connected with 0(1 - Yes & 0 - No)
1
Enter Edges 1 is Connected with 1(1 - Yes & 0 - No)
0
Enter Edges 1 is Connected with 2(1 - Yes & 0 - No)
0
Enter Edges 1 is Connected with 3(1 - Yes & 0 - No)
1
Enter Edges 2 is Connected with 0(1 - Yes & 0 - No)
1
Enter Edges 2 is Connected with 1(1 - Yes & 0 - No)
0
Enter Edges 2 is Connected with 2(1 - Yes & 0 - No)
0
Enter Edges 2 is Connected with 3(1 - Yes & 0 - No)
0
Enter Edges 3 is Connected with 0(1 - Yes & 0 - No)
0
Enter Edges 3 is Connected with 1(1 - Yes & 0 - No)
1
Enter Edges 3 is Connected with 2(1 - Yes & 0 - No)
0
Enter Edges 3 is Connected with 3(1 - Yes & 0 - No)
0
Adjacency Matrix:
        V0      V1      V2      V3
V0      0       1       1       0
V1      1       0       0       1
V2      1       0       0       0
V3      0       1       0       0
Enter starting node for Depth First Search : 0
0 1 3 2
```

19) C program to preform BFS

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

#define MAX 100

int n;
int adj[MAX][MAX];
int visited[MAX];
void create_graph();
void BF_Traversal();
void BFS();

int queue[MAX];
int front = -1,rear = -1;
void push_queue(int vertex);
int pop_queue();
int isEmpty_queue();

int main()
{
  create_graph();
  BFS();
  return 0;
}


void BFS()
{
   int v;

  for(v=0; v<n; v++)
      visited[v] = 0;



  printf("Enter Start Vertex for BFS: \n");
  scanf("%d", &v);


  int i;
  push_queue(v);


  while(!isEmpty_queue())
  {

    v = pop_queue( );
     if(visited[v])
        continue;

    printf("%d ",v);
    visited[v] = 1;

    for(i=0; i<n; i++)
    {
      if(adj[v][i] == 1 && visited[i] == 0)
      {
       push_queue(i);
      }
    }
  }

}

void push_queue(int vertex)
{
  if(rear == MAX-1)
    printf("Queue Overflow\n");
  else
  {
    if(front == -1)
      front = 0;
    rear = rear+1;
    queue[rear] = vertex ;

  }
}

int isEmpty_queue()
{
  if(front == -1 || front > rear)
    return 1;
  else
    return 0;
}

int pop_queue()
{
  int delete_item;
  if(front == -1 || front > rear)
  {
    printf("Queue Underflow\n");
    exit(1);
  }

  delete_item = queue[front];
  front = front+1;
  return delete_item;
}

void create_graph()
{
  int i,j,count,max_edge , origin,destin;

  printf("Enter number of vertices : ");
  scanf("%d",&n);

  for(i=0; i<n; i++)
  {
    for(j=0;j<n;j++)
    {
       scanf("%d",&adj[i][j]);
    }
      break;


  }

  printf("Display Adjuency Matrix\n");
  for(i=0;i<n;i++)
  {
     printf("V%d\t",i);
     for(j=0;j<n;j++)
     {
        printf("%d  ",adj[i][j]);
    }
    printf("\n");
  }
}
```

Output:

```
Enter number of vertices : 7
0
1
1
0
0
0
0
1
0
0
1
0
0
0
1
0
0
1
0
0
1
0
1
1
0
1
1
0
0
0
0
1
0
0
0
0
0
```

```
0
1
0
0
0
0
0
1
0
0
0
0
Display Adjuency Matrix
V0      0  1  1  0  0  0  0
V1      1  0  0  1  0  0  0
V2      1  0  0  1  0  0  1
V3      0  1  1  0  1  1  0
V4      0  0  0  1  0  0  0
V5      0  0  0  1  0  0  0
V6      0  0  1  0  0  0  0
Enter Start Vertex for BFS:
0
0 1 2 3 6 4 5
```

20) C program to preform Topological sorting

```
#include<stdio.h>
int main()
{

   int i,j,k,n,a[10][10],indeg[10],flag[10],count=0;

   printf("Enter the no of vertices:\n");

   scanf("%d",&n);

   printf("Enter the adjacency matrix:\n");

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

      printf("Enter row %d\n",i+1);

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

         scanf("%d",&a[i][j]);

   }

   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]+a[j][i];
      }
      printf("Indegree of Node %d is %d\n",i+1,indeg[i]);
   }
```

```
printf("\nThe topological order is: = ");
while(count<n)
{


    for(k=0;k<n;k++)
    {

        if((indeg[k]==0) && (flag[k]==0))
        {

            printf("%d \n",(k+1));

            flag [k]=1;



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

                if(a[k][i]==1)

                    a[k][i] = 0;


            }

        for(i=0;i<n;i++)
        {
            indeg[i] = 0;
            for(j=0;j<n;j++)
            {
                indeg[i]=indeg[i]+a[j][i];
            }
          //  printf("Indegree of Node %d is %d\n",i+1,indeg[i]);
        }
        }
    }

 count++;

}
```

```
    return 0;

}
```

Output:

```
Enter the no of vertices:
6
Enter the adjacency matrix:
Enter row 1
0
0
1
0
0
1
Enter row 2
1
0
0
0
1
0
Enter row 3
0
0
0
1
0
0
Enter row 4
0
0
0
0
0
0
Enter row 5
1
0
0
1
0
1
```

```
Enter row 6
0
0
1
0
0
0
Indegree of Node 1 is 2
Indegree of Node 2 is 0
Indegree of Node 3 is 2
Indegree of Node 4 is 2
Indegree of Node 5 is 1
Indegree of Node 6 is 2

The topological order is: =
 2
5
1
6
3
4
------Outgoing edges-----
  A  B  C  D  E  F
A 0  0  1  0  0  1
B 1  0  0  0  1  0
C 0  0  0  1  0  0
D 0  0  0  0  0  0
E 1  0  0  1  0  1
F 0  0  1  0  0  0
```

21) C program to preform Knapsack with greedy method

```
#include <stdio.h>

// Structure to represent an item
typedef struct {
    int weight;
    int profit;
    double ratio; // Ratio of profit to weight (profit/weight)
} Item;

// Function to compare items based on their profit/weight ratio (descending order)
int compare(const void *a, const void *b) {
    const Item *item1 = (const Item *)a;
    const Item *item2 = (const Item *)b;
    return item2->ratio - item1->ratio;
}

int main() {
    int n, capacity;

    printf("Enter the number of items: ");
    scanf("%d", &n);

    printf("Enter the capacity of the knapsack: ");
    scanf("%d", &capacity);

    Item items[n];

    printf("Enter weight and profit for each item:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d %d", &items[i].weight, &items[i].profit);
        // Calculate profit/weight ratio for each item
        items[i].ratio = (double)items[i].profit / items[i].weight;
    }

    // Sort items in descending order of profit/weight ratio
    qsort(items, n, sizeof(Item), compare);

    int total_weight = 0, total_profit = 0;

    printf("Selected Items:\n");
    for (int i = 0; i < n; i++) {
        // Check if the remaining capacity can accommodate the whole item
        if (total_weight + items[i].weight <= capacity) {
            total_weight += items[i].weight;
            total_profit += items[i].profit;
            printf("Item %d (weight: %d, profit: %d)\n", i + 1, items[i].weight,
items[i].profit);
        } else {
            // If remaining capacity is less, take a fraction of the item
            double fraction = (double)(capacity - total_weight) / items[i].weight;
            total_weight += capacity - total_weight;
            total_profit += fraction * items[i].profit;
            printf("Item %d (fraction: %.2f, profit: %.2f)\n", i + 1, fraction, fraction
* items[i].profit);
            break; // No need to consider remaining items, knapsack is full
        }
    }

    printf("Total Profit: %d\n", total_profit);

    return 0;
}
```

22)C program to preform Optimal binary search

```
#include<stdio.h>

int main()
  {
        int low, f,l,mid, n, no, a[10]={2,45,56,57,58,60,67,78,89,90};//array
should be in ascending or decending order
        n=10;
        printf("Enter value to find number in array:");
        scanf("%d", &no);
        f = 0;
        l= n - 1;
        mid = (f+l)/2;
        while (f <= l)
                {
                        if(a[mid] < no)
                                f = mid + 1;
                        else if (a[mid] == no)
                        {
                                printf("%d found at location %d\n", no,
mid+1);
                                break;
                        }
                        else
                                l= mid - 1;
                        mid = (l+f)/2;
                }
        if(f>l)
                printf("Not found! %d not present in the array\n", no);

        return 0;
  }
```

Output:

```
Enter value to find number in array:60
60 found at location 6
```

23) C program to preform 0/1 Knapsack

```
#include<stdio.h>

// Function to find maximum of two integers
int max(int a, int b) {
   return (a > b) ? a : b;
}

// Function to solve 0/1 Knapsack using dynamic programming
int knapSack(int W, int wt[], int val[], int n) {
   int i, w;
   int K[n + 1][W + 1];

   // Build table K[][] in bottom-up manner
   for (i = 0; i <= n; i++) {
      for (w = 0; w <= W; w++) {
         if (i == 0 || w == 0)
            K[i][w] = 0;
         else if (wt[i - 1] <= w)
            K[i][w] = max(val[i - 1] + K[i - 1][w - wt[i - 1]], K[i - 1][w]);
         else
            K[i][w] = K[i - 1][w];
      }
   }
   return K[n][W];
}

int main() {
   int val[] = {2,3,1,4};
   int wt[] = {3,4,6,5};
   int W = 8;
   int n = sizeof(val) / sizeof(val[0]);

   printf("Maximum value that can be obtained: %d\n", knapSack(W, wt, val, n));
   return 0;
}
```

Output:

```
Maximum value that can be obtained: 6
```

24) C program to preform Matrix chain multiplication

```
#include <stdio.h>
#define INT_MAX 5000
int MatrixChainOrder(int p[], int i, int j)
{
	if (i==j)
		return 0;
	int k;
	int min=INT_MAX;
	int count;
	for (k=i;k<j;k++)
	{
		count=MatrixChainOrder(p,i,k)
				+ MatrixChainOrder(p,k+1,j)
				+ p[i-1]*p[k]*p[j];

		if (count<min)
			min=count;
	}
	return min;
}
int main()
{
	int arr[]={ 13,5,89,5,34};
	int N=sizeof(arr)/sizeof(arr[0]);
	printf("Minimum number of multiplications is %d ",
	MatrixChainOrder(arr,1,N-1));

	return 0;
}
```

Output:

```
Minimum number of multiplications is 4760
```

25) C program to preform to find longest common subsequence

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

int i,j,m,n,LCB[20][20];
char S1[20]="ABCBDAB",S2[20]="BDCABA";

void lcb() {
  m =strlen(S1);
  n =strlen(S2);

  // Filling 0's in the matrix
  for(i=0;i<=m;i++)
    LCB[i][0] = 0;
  for (i = 0; i <= n; i++)
    LCB[0][i] = 0;

  // Building the mtrix in bottom-up way
  for (i=1; i<=m; i++)
    for (j=1; j<=n; j++) {
      if(S1[i-1]==S2[j-1])
       {
        LCB[i][j]=LCB[i-1][j-1]+1;
      }
else if (LCB[i-1][j]>=LCB[i][j-1]) {
        LCB[i][j]=LCB[i - 1][j];
      }
 Else
 {
        LCB[i][j]=LCB[i][j-1];
      }
    }

  int index=LCB[m][n];
  char lcb[index + 1];
  lcb[index] = '\0';

  int i=m,j=n;
  while (i>0&&j>0)
   {
    if(S1[i-1]==S2[j-1])
     {
      lcb[index-1]=S1[i-1];
      i--;
      j--;
      index--;
    }

    else if (LCB[i-1][j]>LCB[i][j-1])
      i--;
    else
      j--;
  }

  // Printing the sub sequences
  printf("S1 : %s \nS2 : %s \n", S1, S2);
  printf("LCB: %s", lcb);
}

int main() {
  lcb();
  printf("\n");
}
```

Output:

```
S1 : ABCBDAB
S2 : BDCABA
LCB: BDAB
```

```
}
```

26) C program to preform N Queen problem

```
#include <stdio.h>
#define N 4
int board[N][N] = {0};

int Safe(int row, int col) {
    int i, j;

    // Check left side of the row
    for (i=0; i<col; i++)
        if (board[row][i])
            return 0;
    // Check upper diagonal on left side
    for (i=row, j=col; i>=0 && j>=0; i--,j--)
        if (board[i][j])
            return 0;

    // Check lower diagonal on left side
    for (i=row, j=col; j>=0 &&i<N;i++,j--)
        if (board[i][j])
            return 0;

    return 1;
}

int solveNQueens(int col) {
    if(col>=N)
        return 1;

    for(int i=0;i<N;i++) {
        if(Safe(i,col)) {
            board[i][col]=1;
            if(solveNQueens(col+1))
                return 1;
            board[i][col]=0;
        }
    }

    return 0;
}

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

int main() {
    if (solveNQueens(0))
        printSolution();
    else
        printf("Solution does not exist.");

    return 0;
}
```

Output:

```
0 0 1 0
1 0 0 0
0 0 0 1
0 1 0 0
```

27) C program to preform Travelling Salesman problem

```
 #include <stdio.h>
int matrix[10][10] = {
    {0, 3, 4, 5},
    {2, 0, 5, 6},
    {1, 5, 0, 3},
    {4, 3, 2, 0}
};
int visited[10], n, cost = 0;


void travellingsalesman(int c)
{
   int k, adj_vertex = 99;
   int min = 99;


   visited[c] = 1;
   printf("%d ", c + 1);

   for(k = 0; k < n; k++) {
      if((matrix[c][k] != 0) && (visited[k] == 0))
       {
         if(matrix[c][k] < min)
         {
            min = matrix[c][k];
         }
         adj_vertex = k;
      }
   }
   if(min != 99)
   {
      cost = cost + min;
   }
   if(adj_vertex == 99)
   {
      adj_vertex = 0;
      printf("%d", adj_vertex + 1);
      cost = cost + matrix[c][adj_vertex];
      return;
   }
   travellingsalesman(adj_vertex);
}
int main()
{
   int i, j;
   n = 5;
   printf("Enter number vertices:");
   scanf("%d",&n);
     for(i = 0; i < n; i++)
      {
       visited[i] = 0;
      }
             printf("Shortest Path: ");
             travellingsalesman(0);
             printf("\nMinimum Cost: ");
             printf("%d\n", cost);
   return 0;
}
```

Output:

```
Enter number vertices:4
Shortest Path: 1 4 3 2 1
Minimum Cost: 12
```

28) C program to preform sum of subset

```
#include <stdio.h>

int main() {
    int n, i, j, num[10], sum = 0;

    printf("Enter the number of elements in the array: ");
    scanf("%d", &n);

    printf("Enter the elements of the array: ");
    for (i = 0; i < n; i++) {
        scanf("%d", &num[i]);
    }
    printf("Subsets and their sums:\n");
    for (i = 0; i < (1 << n); i++) {
        sum = 0;
        printf("{ ");
        for (j = 0; j < n; j++) {
            if (i & (1 << j)) {
                printf("%d ", num[j]);
                sum += num[j];
            }
        }
        printf("} Sum = %d\n", sum);
    }
    return 0;
}
```

Output:

```
Enter the number of elements in the array: 2
Enter the elements of the array: 1
6
Subsets and their sums:
{ } Sum = 0
{ 1 } Sum = 1
{ 6 } Sum = 6
{ 1 6 } Sum = 7
```

29) C program to preform Graph colouring algorithm

```
#include <stdio.h>

#define V 4 // Number of vertices

void printSolution(int color[]);
int isSafe(int v, int graph[V][V],
           int color[], int c)
{
    for (int i = 0; i < V; i++)
        if (graph[v][i] && c == color[i])
            return 0;
    return 1;
}
int graphColoringUtil(int graph[V][V],
                      int m, int color[],
                      int v)
{
    // base case: If all vertices are
    // assigned a color then return true
    if (v == V)
        return 1;

    // Consider this vertex v and
    // try different colors
    for (int c = 1; c <= m; c++)
    {
        // Check if assignment of
        // color c to v is fine
        if (isSafe(v, graph, color, c))
        {
            color[v] = c;

            // recur to assign colors to
            // rest of the vertices
            if (graphColoringUtil(
                    graph, m, color, v + 1) == 1)
                return 1;

            // If assigning color c doesn't
            // lead to a solution then remove it
            color[v] = 0;
        }
    }

    // If no color can be assigned to
    // this vertex then return false
    return 0;
}

int graphColoring(int graph[V][V], int m)
{
    // Initialize all color values as 0.
    // This initialization is needed
    // correct functioning of isSafe()
    int color[V];
    for (int i = 0; i < V; i++)
        color[i] = 0;

    // Call graphColoringUtil()
    // for vertex 0
    if (graphColoringUtil(graph, m, color, 0) == 0)
    {
        printf("Solution does not exist");
        return 0;
    }

    // Print the solution
    printSolution(color);
    return 1;
}

/* A utility function to print solution */
void printSolution(int color[])
{
    printf("Solution Exists:"
           " Following are the assigned colors \n");
    for (int i = 0; i < V; i++)
        printf(" %d ", color[i]);
    printf("\n");
}

// Driver code
int main()
{
    int graph[V][V] = {
        {0, 1, 1, 1},
        {1, 0, 1, 0},
        {1, 1, 0, 1},
        {1, 0, 1, 0},
    };
    int m = 3; // Number of colors
    graphColoring(graph, m);
    return 0;
}
```

Output:

```
Solution Exists: Following are the assigned colors
 1  2  3  2
```

30) C Program to find Hamilton cycle.

```
#include <stdio.h>
#define V 5

int graph[V][V] = {
    {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}
};

int path[V];

void printSolution() {
    printf("Solution Exists: Following is one Hamiltonian Cycle:\n");
    for (int i = 0; i < V; i++)
        printf("%d ", path[i]);
    printf("%d ", path[0]);
    printf("\n");
}

int isSafe(int v, int pos) {
    if (graph[path[pos - 1]][v] == 0)
        return 0;

    for (int i = 0; i < pos; i++)
        if (path[i] == v)
            return 0;

    return 1;
}

int hamCycleUtil(int pos) {
    if (pos == V) {
        if (graph[path[pos - 1]][path[0]] == 1)
            return 1;
        else
            return 0;
    }

    for (int v = 1; v < V; v++) {
        if (isSafe(v, pos)) {
            path[pos] = v;
            if (hamCycleUtil(pos + 1) == 1)
                return 1;
            path[pos] = -1;
        }
    }

    return 0;
}

int hamCycle() {
    for (int i = 0; i < V; i++)
        path[i] = -1;

    path[0] = 0;
    if (hamCycleUtil(1) == 0) {
        printf("Solution does not exist\n");
        return 0;
    }

    printSolution();
    return 1;
}

int main() {
    hamCycle();
    return 0;
}
```

Output:

```
Solution Exists: Following is one Hamiltonian Cycle:
0 1 2 4 3 0
```

31) C program to find Huffman coding.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.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;
   minHeapify(minHeap, 0);
   return temp;
}

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) {
```

```c
    if (root->left) {
        arr[top] = 0;
        printCodes(root->left, arr, top + 1);
    }
    if (root->right) {
        arr[top] = 1;
        printCodes(root->right, arr, top + 1);
    }
    if (isLeaf(root)) {
        printf("%c: ", root->data);
        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[] = {'a', 'b', 'c', 'd', 'e', 'f'};
    int freq[] = {5, 9, 12, 13, 16, 45};
    int size = sizeof(arr) / sizeof(arr[0]);
    printf("Huffman Codes:\n");
    HuffmanCodes(arr, freq, size);
    return 0;
}
```

Output:

```
Huffman Codes:
f: 0
c: 100
d: 101
a: 1100
b: 1101
e: 111
```