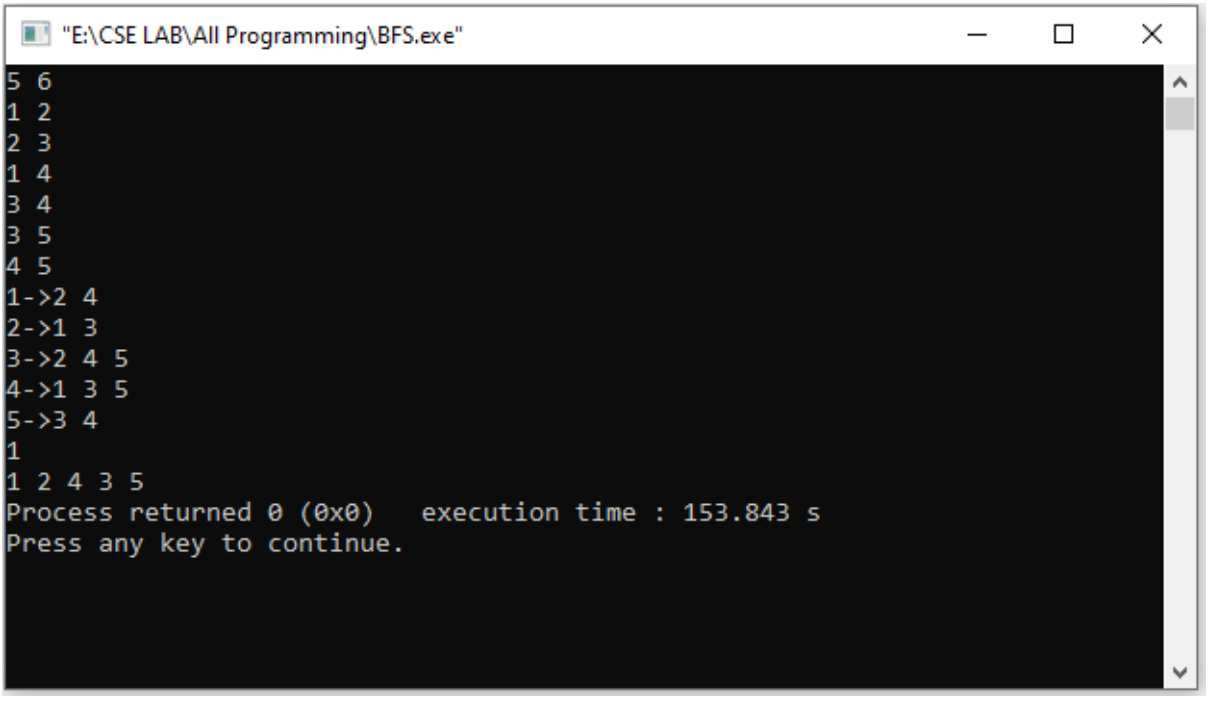
INDEX

Topic Page No

1.BFS IMPLEMENTATION:

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
    int n,e;
    cin>>n>>e;
    int visit[1010]= {0};
    vector<int>graph[1010];
     int u,v;
    for(int i=0; i<e; i++)
    {
        cin>>u>>v;
        graph[u].push_back(v);
        graph[v].push_back(u);
    }
    for(int i=1; i<=n; i++)
    {
        cout<<i<<"->";
        for(int j=0; j<graph[i].size(); j++)
            cout<<graph[i][j]<<" ";
        cout<<endl;
    }
   int start;
    cin>>start;
 queue<int>q;
    q.push(start);
visit[start]=1;
```

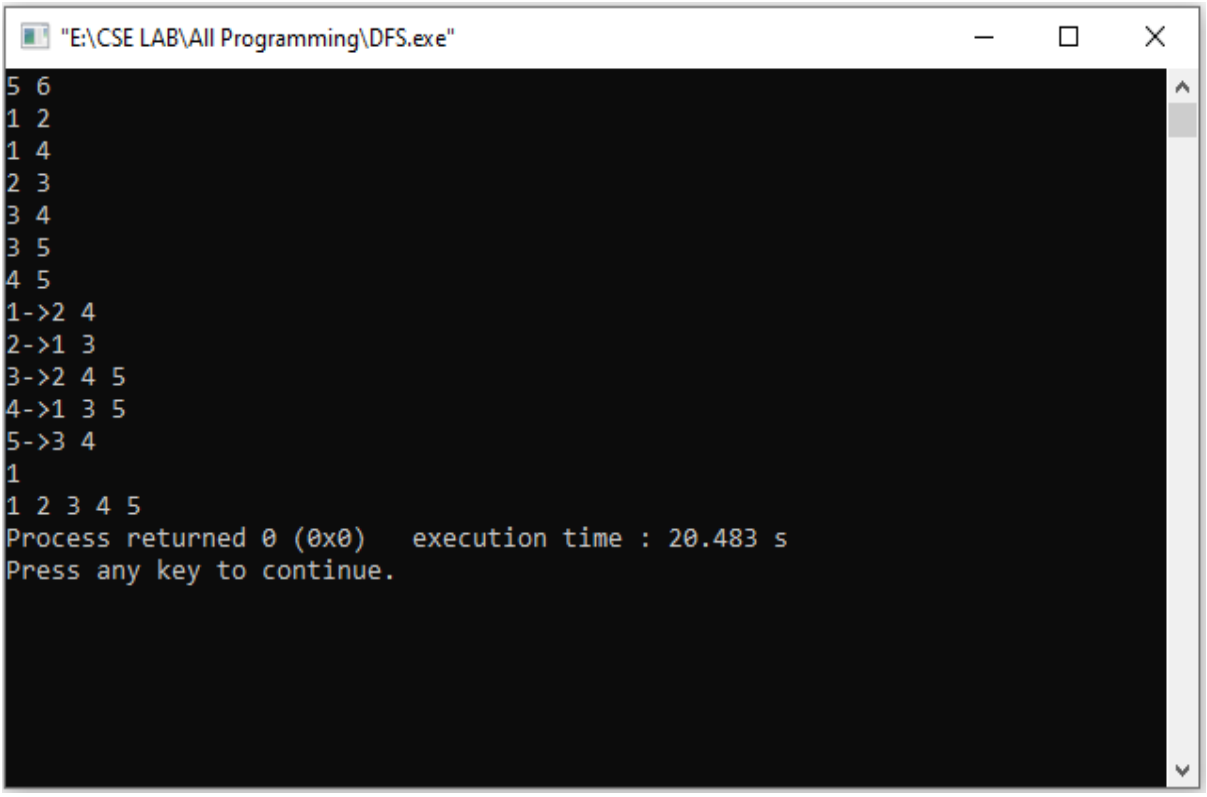
```
while(!q.empty())
   {


int x,y;
       x=q.front();
       cout<<x<<" ";
       q.pop();


       for(int i=0; i<graph[x].size(); i++)
       {
          y=graph[x][i];


          if(!visit[y])
          {
             visit[y]=1;
             q.push(y) ;
          }
       }
   }
}
```



2.DFS IMPLEMENTATION:

```
#include<bits/stdc++.h>
using namespace std;
vector<int>graph[1000];
int visit[1000]={0};
int n,e,s;
int dfs(int u)
{
      int i,v;
      visit[u]=1;
      cout<<u<<" ";
     for(i=0;i<graph[u].size();i++)
     {
           v=graph[u][i];
           if(visit[v]==0)
           {
                 dfs(v);
           }
     }
}
int main()
{
   scanf("%d%d",&n,&e);
   int u,v;
   for(int i=1; i<=e; i++)
   {
      cin>>u>>v;              ///udirected graph
      graph[u].push_back(v);
      graph[v].push_back(u);
```

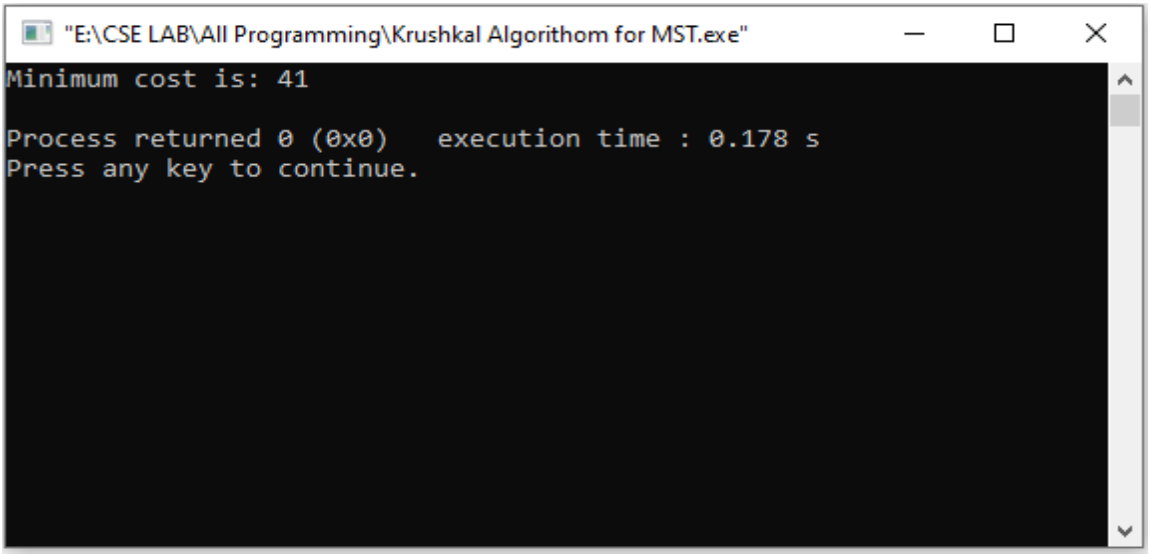
```
   }//print graph adjacency list
  for(int i=1;i<=n;i++)
    {
    cout<<i<<"->";
          for(int j=0;j<graph[i].size();j++)
            cout<<graph[i][j]<<" ";
          cout<<endl;
    }
   cin>>s;
    dfs(s);
}
```



3. KRUSHKAL'S IMPLEMENTATION

```
#include<bits/stdc++.h>
using namespace std;
const int MAX = 1e6-1;
int root[MAX];
const int nodes = 4, edges = 5;
pair <long long, pair<int, int> > p[MAX];
int parent(int a)                    //find the parent of the given node
{
   while(root[a] != a)
   {
      root[a] = root[root[a]];
      a = root[a];
   }
   return a;
}
void union_find(int a, int b)        //check if the given two vertices are in the same
"union" or not
{
   int d = parent(a);
   int e = parent(b);
   root[d] = root[e];
}
long long kruskal()
{
   int a, b;
   long long cost, minCost = 0;
   for(int i = 0 ; i < edges ; ++i)
   {
```

```
        a = p[i].second.first;
        b = p[i].second.second;
        cost = p[i].first;
        if(parent(a) != parent(b))
        {
            minCost += cost;
            union_find(a, b);
        }
    }
    return minCost;
}
int main()
{
    int x, y;
    long long weight, cost, minCost;
    for(int i = 0; i < MAX;i++)
    {
        root[i] = i;
    }
    p[0] = make_pair(10, make_pair(0, 1));
    p[1] = make_pair(18, make_pair(1, 2));
    p[2] = make_pair(13, make_pair(2, 3));
    p[3] = make_pair(21, make_pair(0, 2));
    p[4] = make_pair(22, make_pair(1, 3));
 sort(p, p + edges);
    minCost = kruskal();
    cout << "Minimum cost is: "<< minCost << endl;
    return 0;}
```

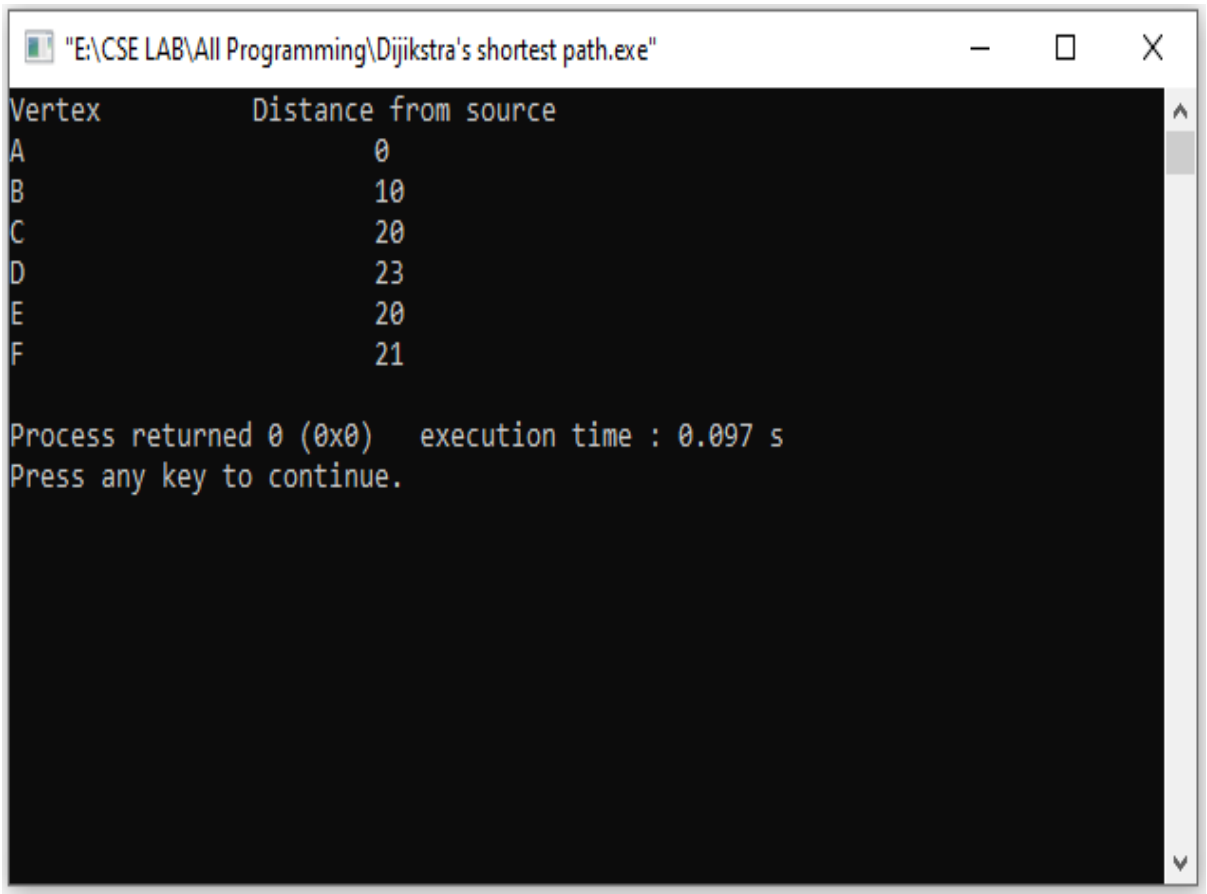


4.DIJKSTRA IMPLEMENTATION

```
#include<bits/stdc++.h>
using namespace std;
int minDist(int dist[], int visit[])
{
   int min=INT_MAX,idx;
  for(int i=0; i<6; i++)
   {
      if(visit[i]==0 && dist[i]<=min)
      {
         min=dist[i];
         idx=i;
      }
   }
   return idx;
}
void Dijkstra(int graph[6][6],int s)
{
   int dist[6],visit[6];
```

```
  for(int i = 0; i<6; i++)
    {
       dist[i] = INT_MAX;
       visit[i] = 0;
    }
    dist[s] = 0;
    for(int i = 0; i<6; i++)
    {
       int x=minDist(dist,visit);
       visit[x]=1;
       for(int i = 0; i<6; i++)
       {
          // Updating the minimum distance for the particular node.
          if(!visit[i] && graph[x][i] &&  dist[x]!=INT_MAX  &&
dist[x]+graph[x][i]<dist[i])
              dist[i]=dist[x]+graph[x][i];
       }
    }
     cout<<"Vertex          Distance from source"<<endl;
    for(int i = 0; i<6; i++)
    {
       char str=65+i;
       cout<<str<<"\t\t\t"<<dist[i]<<endl;
    }
}
```

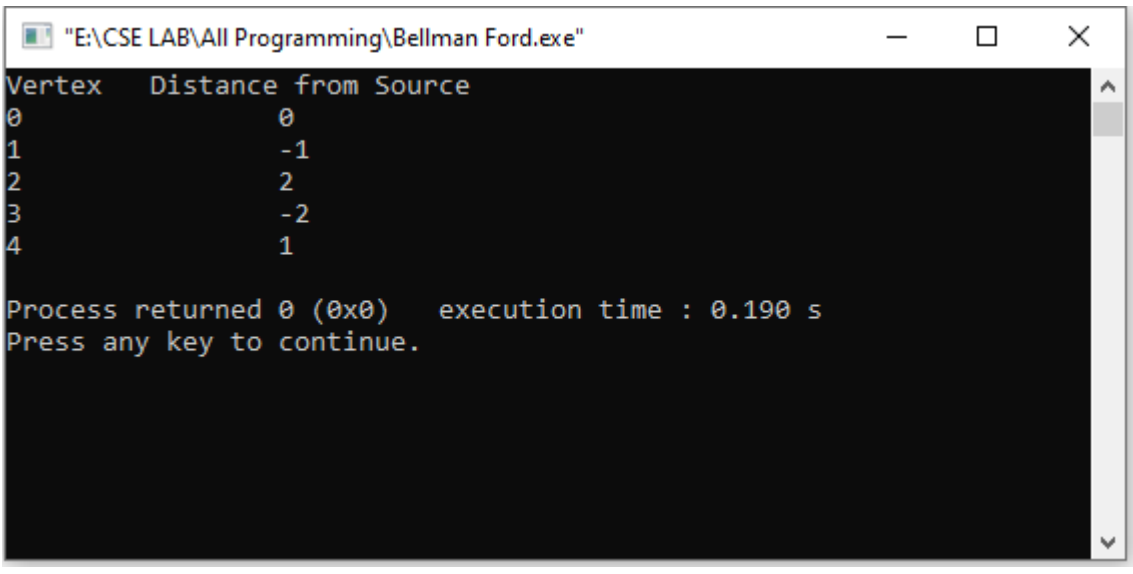
```
int main()
{
   int graph[6][6]=
   {
      {0, 10, 20, 0, 0, 0},
      {10, 0, 0, 50, 10, 0},
      {20, 0, 0, 20, 33, 0},
      {0, 50, 20, 0, 20, 2},
      {0, 10, 33, 20, 0, 1},
      {0, 0, 0, 2, 1, 0}
   };
   Dijkstra(graph,0);
   return 0;
}
```



5.BELLMAN-FORD

```
#include<bits/stdc++.h>
using namespace std;
void print(int dist[], int n)
{
   cout<<"Vertex   Distance from Source\n";
   for (int i = 0; i < n; ++i)
      printf("%d \t\t %d\n", i, dist[i]);
}
void BellmanFord(int V, int E, int start, int E1[],int E2[],int W[])
{
   int dist[V];

   for (int i = 0; i < V; i++)
      dist[i]   = INT_MAX;
   dist[start] = 0;

   for (int i = 1; i <= V-1; i++)
   {
      for (int j = 0; j < E; j++)
      {
         int u = E1[j];
         int v = E2[j];
         int weight = W[j];
         if (dist[u] != INT_MAX && dist[u] + weight < dist[v])
            dist[v] = dist[u] + weight;
      }
   }
for (int i = 0; i < E; i++)
```

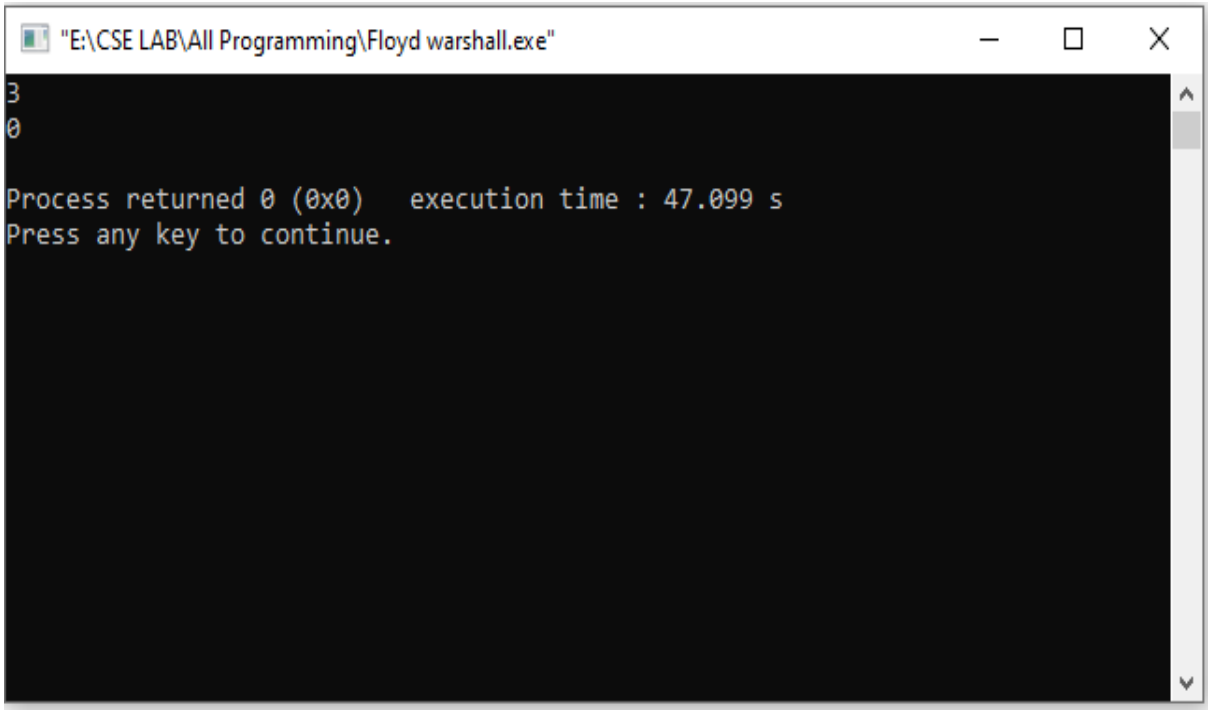
```
    {
        int u = E1[i];
        int v = E2[i];
        int weight = W[i];
        if (dist[u] != INT_MAX && dist[u] + weight < dist[v])
            cout<<"Graph contains negative weight cycle";
    }
    print(dist, V);
    return;
}
int main ()
{
    int v=5;
    int e=8;
    int E1[8]= {0,0,1,1,1,3,3,4};
    int E2[8]= {1,2,2,3,4,2,1,3};
    int W[8]= {-1,4,3,2,2,5,1,-3};
    BellmanFord(v, e, 0,E1,E2,W);
    return 0;
}
```



6.FLOYED-WARSHALL

```
#include <bits/stdc++.h>
#define MAX_N 300
#define INF 987654321
using namespace std;
typedef long long lld;
int n;
int dist[MAX_N][MAX_N];
int flojd[MAX_N][MAX_N];
void FloydWarshall()
{
   for (int i=1; i<=n; i++)
   {
      for (int j=1; j<=n; j++)
      {
         flojd[i][j] = dist[i][j];
      }
      flojd[i][i] = 0;
   }
for (int k=1; k<=n; k++)
   {
      for (int i=1; i<=n; i++)
      {
         for (int j=1; j<=n; j++)
         {
            if (flojd[i][k] + flojd[k][j] < flojd[i][j])
            {
               flojd[i][j] = flojd[i][k] + flojd[k][j];
```

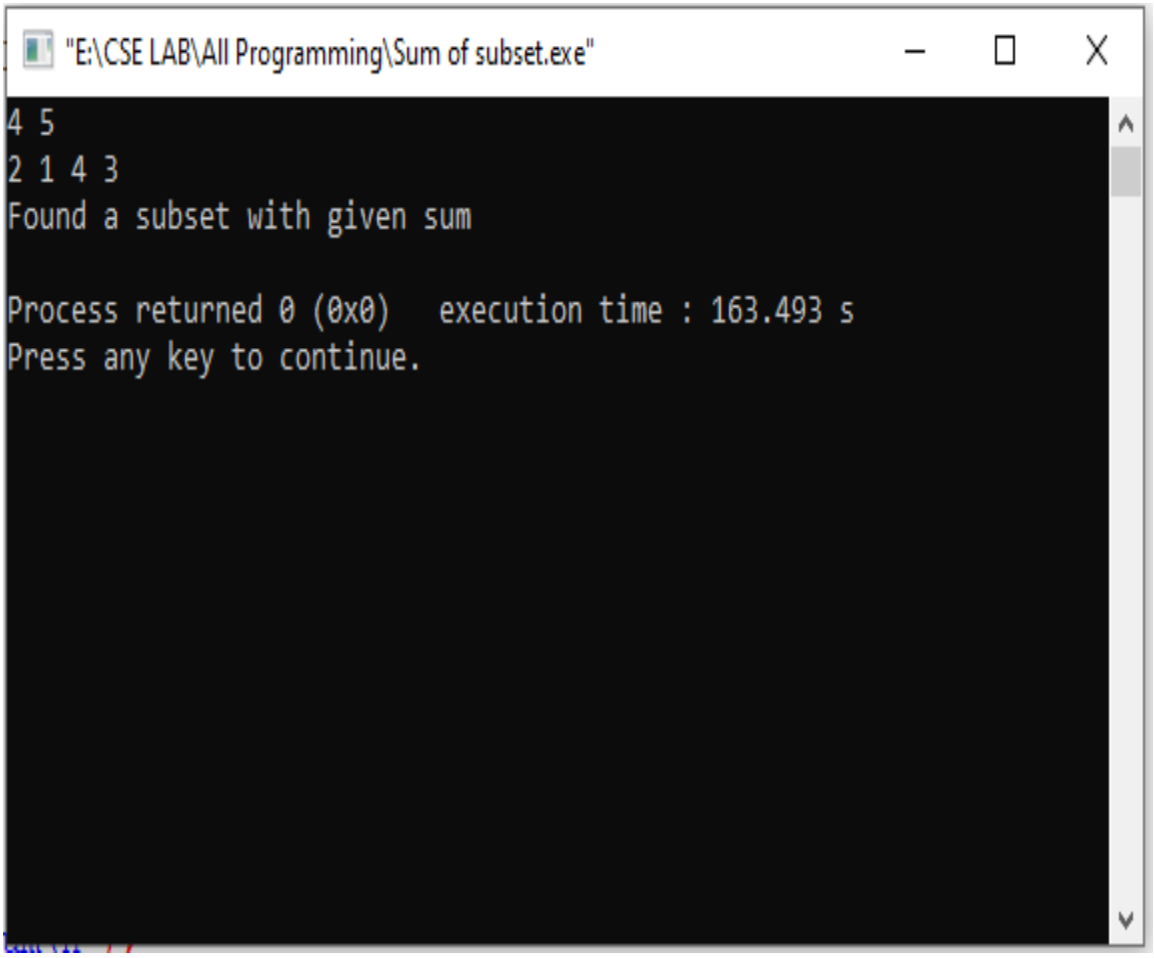
```
            }
        }
    }
  }
}
int main()
{
   int n;cin>>n;
   dist[1][1] = 0, dist[1][2] = 3, dist[1][3] = INF;
   dist[2][1] = INF, dist[2][2] = 0, dist[2][3] = 4;
   dist[3][1] = INF, dist[3][2] = 1, dist[3][3] = 0;
   FloydWarshall();
   printf("%d\n",flojd[1][3]);
   return 0;
}
```



7.SUM OF SUBSET

```
#include <bits/stdc++.h>
using namespace std;
bool isSubsetSum(int set[], int n, int sum)
{
   bool subset[n+1][sum+1];

   for (int i = 0; i <= n; i++)
      subset[i][0] = true;

   for (int i = 1; i <= sum; i++)
      subset[0][i] = false;
   for (int i = 1; i <= n; i++)
   {
      for (int j = 1; j <= sum; j++)
      {
         if(j<set[i-1])
            subset[i][j] = subset[i-1][j];
         if (j >= set[i-1])
            subset[i][j] = subset[i-1][j] || subset[i - 1][j-set[i-1]];
      }
   }
  return subset[n][sum];
}
```

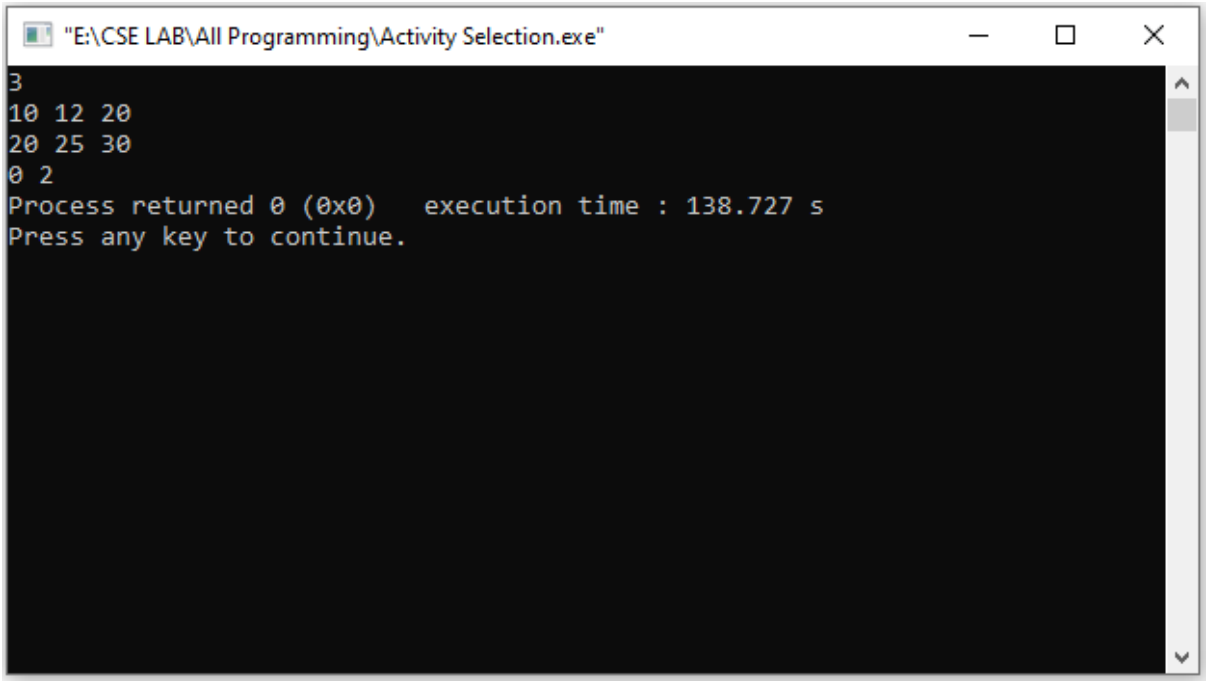
```
int main()
{
    int n,sum;
    cin>>n>>sum;
    int set[n+6];
   for(int i=0; i<n; i++) cin>>set[i];
   if (isSubsetSum(set, n, sum) == true)
        printf("Found a subset with given sum\n");
    else
        printf("No subset with given sum\n");

    return 0;
}
```



8.ACTIVITY SELECTION

```
#include <bits/stdc++.h>
using namespace std;
void AS(int s[], int f[], int n)
{
   int j=0;
 cout << j << " ";
for (int i= 1; i < n; i++)
   {
      if (s[i] >= f[j])
      {
         cout << i << " ";
         j = i;
      }
   }
}
int main()
{
  int n;cin>>n;
  int s[n+6],f[n+6];
 for(int i=0;i<n;i++)cin>>s[i];
  for(int i=0;i<n;i++)cin>>f[i];
  vector<pair<int,int>>vp;
  for(int i=0;i<n;i++)
  {
        vp.push_back(make_pair(f[i],s[i]));
  }
  sort(vp.begin(),vp.end());
   AS(s, f, n);
```

```
    return 0;
}
```



9.HUFFMAN CODE

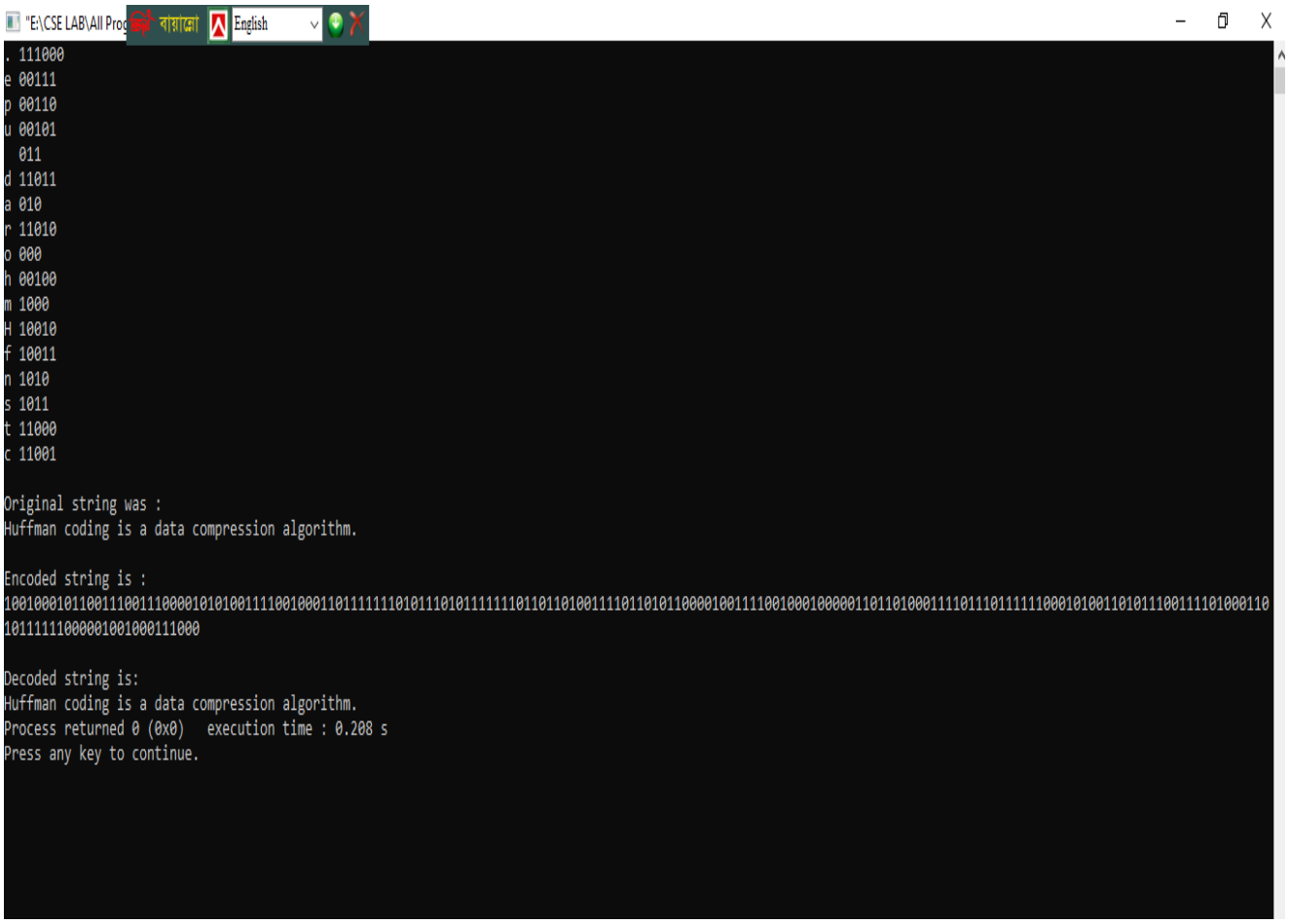
```
#include<bits/stdc++.h>
using namespace std;
struct Node
{
    char ch;
    int freq;
    Node *left, *right;
};
Node* getNode(char ch, int freq, Node* left, Node* right)
{
    Node* node = new Node();

    node->ch = ch;
```

```
    node->freq = freq;
    node->left = left;
    node->right = right;
    return node;
}
struct comp
{
    bool operator()(Node* l, Node* r)
    {
        return l->freq > r->freq;
    }
};
void encode(Node* root, string str,unordered_map<char, string>
&huffmanCode)
{
    if (root == nullptr)
        return;

    if (!root->left && !root->right)
    {
        huffmanCode[root->ch] = str;
    }
    encode(root->left, str + "0", huffmanCode);
    encode(root->right, str + "1", huffmanCode);
}
void decode(Node* root, int &index, string str)
{
    if (root == nullptr)
    {
        return;
```

```
    }
    if (!root->left && !root->right)
    {
        cout << root->ch;
        return;
    }
    index++;
  if (str[index] =='0')
        decode(root->left, index, str);
    else
        decode(root->right, index, str);
}
void huffman(string text)
{
    unordered_map<char, int> freq;
    for (char ch: text)
    {
        freq[ch]++;
    }
    priority_queue<Node*, vector<Node*>, comp> pq;
    for (auto pair: freq)
    {
        pq.push(getNode(pair.first, pair.second, nullptr, nullptr));
    }
    while (pq.size() != 1)
    {
        Node *left = pq.top();
        pq.pop();
        Node *right = pq.top();
```

```
        pq.pop();
  int sum = left->freq + right->freq;
        pq.push(getNode('\0', sum, left, right));
    }
Node* root = pq.top();
 unordered_map<char, string> huffmanCode;
    encode(root, "", huffmanCode);
  cout << "Huffman Codes are :\n" << '\n';
    for (auto pair: huffmanCode)
    {
        cout << pair.first << " " << pair.second << '\n';
    }
 cout << "\nOriginal string was :\n" << text << '\n';
 string str = "";
    for (char ch: text)
    {
        str += huffmanCode[ch];
    }
 cout << "\nEncoded string is :\n" << str << '\n';
 int index = -1;
    cout << "\nDecoded string is: \n";
    while (index < (int)str.size() - 2)
    {
        decode(root, index, str);
    }
}
```

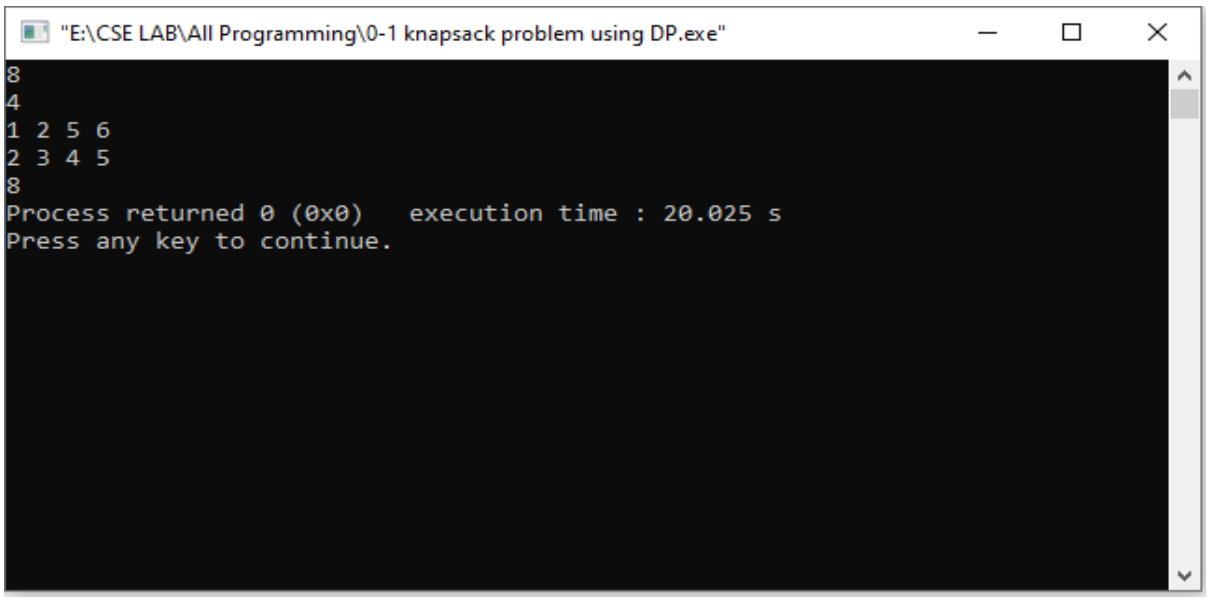
```
int main()
{
    string text = "Huffman coding is a data compression algorithm.";
    huffman(text);
    return 0;
}
```



10. 0-1 KNAPSACK:

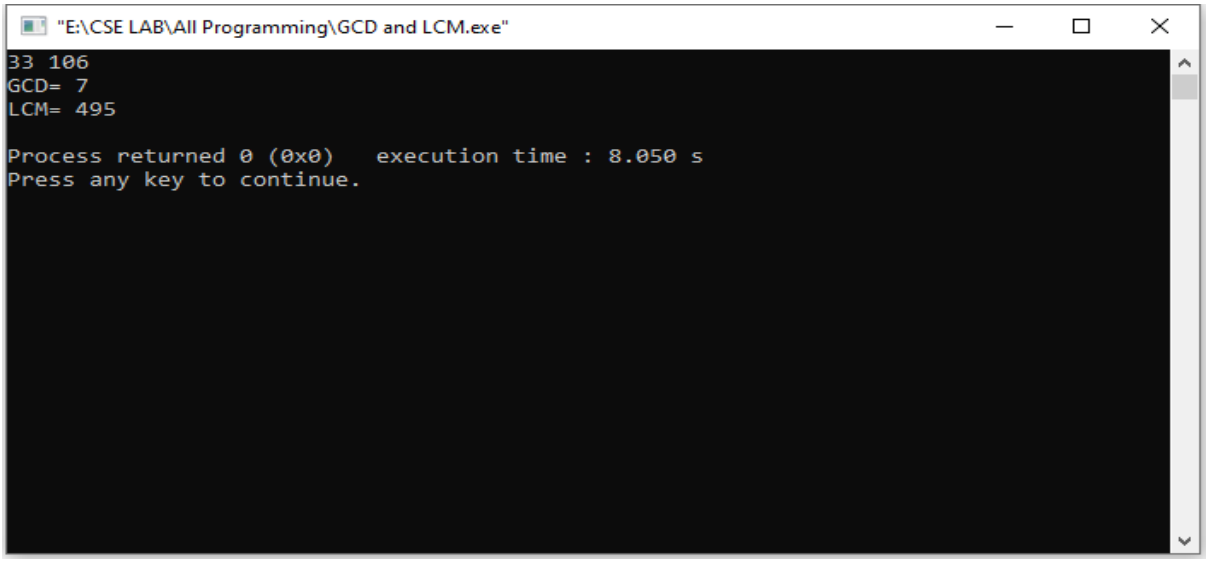
```
#include <bits/stdc++.h>
using namespace std;
int max(int a, int b)
{
    return (a > b) ? a : b;
```

```
}
int knapSack(int W, int wt[], int val[], int n)
{
 if (n == 0 || W == 0)
       return 0;
  if (wt[n - 1] > W)
 return knapSack(W, wt, val, n - 1);
 else
return max(val[n - 1]+ knapSack(W - wt[n - 1], wt, val, n -1),knapSack(W, wt, val, n - 1));
}
int main()
{
   int W,n;
   cin>>W>>n;
   int val[n+6],wt[n+6];
   for(int i=0; i<n; i++)cin>>val[i];
   for(int i=0; i<n; i++)cin>>wt[i];
   cout << knapSack(W, wt, val, n);
   return 0;
}
```



11.GCD & LCM:

```
#include<bits/stdc++.h>
using namespace std;
int main()
{  long long int a,b,r,n1,n2,LCM,GCD;
   cin>>n1>>n2;
   if(n1>n2)
      swap(n1,n2);
   a=n1;  b=n2;
 while(b!=0)
   {  r=b%a;
      a=b;
      b=r; }
    GCD=a;
    cout<<"GCD= "<<GCD<<endl;
   LCM=((n2/GCD)*n1);
   cout<<"LCM= "<<LCM<<endl;
}
```



------End------