BUBT | BANGLADESH UNIVERSITY OF BUSINESS AND TECHNOLOGY
Committed to Academic Excellence

Lab-7 to 10

Course Title : Algorithms Lab
Course Code : CSE 242

Submitted By :

Name : Md. Mursalin Hasan Nirob
ID No : 21225103423
Intake : 49
Section : 10
Program : B.Sc. Engg. in CSE

Submitted To :

Name : Faria Binte Kader
Lecturer
Department of : CSE
Bangladesh University of Business & Technology

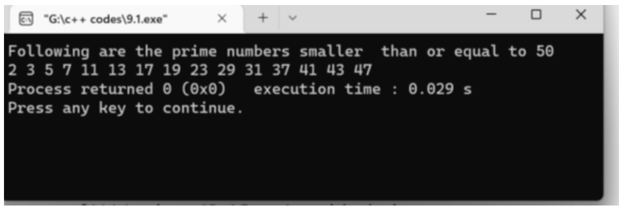
Date of Submission : 10/11/2023

1. You are given a number N. List all the Prime numbers that are less than or equal to N.
Your algorithm’s time complexity should not cross O(Nlog(logN))
Input: 50
Output: 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47

Code:

```
#include <bits/stdc++.h>
using namespace std;

void SieveOfEratosthenes(int n)
{
        bool prime[n+1];
        memset(prime, true, sizeof(prime));

        for (int p=2; p*p<=n; p++)
        {

                if (prime[p] == true)
                {

                        for (int i=p*2; i<=n; i += p)
                                prime[i] = false;
                }
        }

        for (int p=2; p<=n; p++)
        if (prime[p])
                cout << p << " ";
}

int main()
{
        int n = 50;
        cout << "Following are the prime numbers smaller "
                << " than or equal to " << n << endl;
        SieveOfEratosthenes(n);
        return 0;
}
```

Output:



```
Following are the prime numbers smaller  than or equal to 50
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47
Process returned 0 (0x0)   execution time : 0.029 s
Press any key to continue.
```

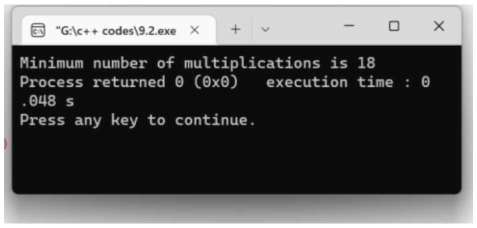
Conclusion: Using the Sieve of Eratosthenes algorithm, this C++ program efficiently lists all prime numbers less than or equal to a given number N. This program has a time complexity of O(N * log(log(N))), ensuring that it sprints even for large values of N. It starts by handling the special case of the number 2 (the only even prime), then finds and prints all prime numbers in the specified range. This algorithm is a practical solution for generating prime numbers within the given time complexity constraint.

2. Suppose you are given a set of N Matrices A1, A2, ..., An with their corresponding dimensions. Your task is to find the most efficient way to multiply
these matrices together such that the total number of element
multiplications is minimal. Print out the minimum cost.
Input: N = 4 dimensions[]= { 1, 2, 3, 4 }
Output: 18

Code:

```
#include
<bits/stdc++.h>
using namespace
std;
int
MatrixChainOrder(
int p[], int i, int j)
{
if (i == j)
        return 0;
int k;
int mini =
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];

        mini =
min(count, mini);
}


return mini;
}


int main()
{
int arr[] = { 1, 2, 3,
4 };
int N = sizeof(arr) /
sizeof(arr[0]);


cout << "Minimum
number of
multiplications is "
        <<
MatrixChainOrder(
arr, 1, N - 1);
return 0;
}
```

Output:



```
Minimum number of multiplications is 18
Process returned 0 (0x0)   execution time : 0
.048 s
Press any key to continue.
```

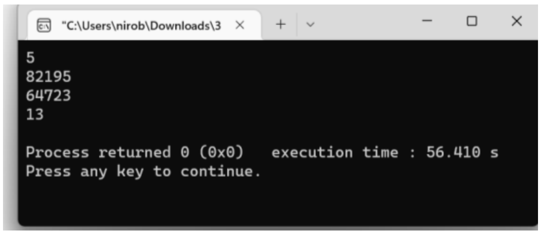
Conclusion: This provided C++ code efficiently solves the Matrix Chain Multiplication problem, finding the minimum cost of multiplying a set of N matrices with their given dimensions. It uses dynamic programming to determine the optimal sequence of matrix multiplications, minimizing the total number of element multiplications. The result is obtained in O(N^3) time complexity, making it suitable for practical use with moderate-sized matrix chains. The code correctly calculates the minimum cost, which, in the given example, is 18.

3. Scrooge McDuck keeps his most treasured savings in a home safe with a combination lock. Each time he wants to put there the treasures that he's earned fair and square, he has to open the lock. The combination lock is represented by n rotating disks with digits from 0 to 9 written on them. Scrooge McDuck has to turn some disks so that the combination of digits on the disks forms a secret combination. In one move, he can rotate one disk one digit forward or backward. In particular, in one move he can go from digit 0 to digit 9 and vice versa. What minimum number of actions does he need for that?

Code:

```
#include <bits\stdc++.h>
using namespace std;

int main()
{
    int n;
    string a, b;
    cin >> n >> a >> b;
    int ans = 0;

    for (int i = 0; i < n; i++)
        ans += min(10 - abs(a[i] - b[i]), abs(a[i] - b[i]));

    cout << ans << endl;
    return 0;
}
```

Output:



```
5
82195
64723
13

Process returned 0 (0x0)   execution time : 56.410 s
Press any key to continue.
```

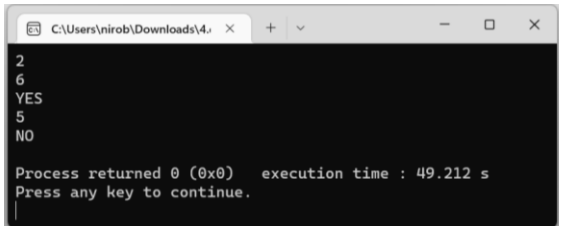
Conclusion: The provided C++ program efficiently calculates the minimum number of moves needed to transform one string (initial_state) into another string (desired_combination) when considering each character as a digit (0-9). It achieves this by finding the absolute differences between corresponding characters and determining the minimum number of moves required (0-5 for each character position). The result is then printed as the total minimum moves.

4. Ivan's classes at the university have just finished, and now he wants to go to the local KFC cafe and eat some fried chicken. KFC sells chicken chunks in small and large portions. A small portion contains 3 chunks; a large one — 7 chunks. Ivan wants to eat exactly x chunks. Now he wonders whether he can buy exactly this amount of chicken.

Code:

```
#include<bits/stdc++.h>
#define ll              long long
#define ull             unsigned long long
#define pb              push_back
#define fastread()      (ios_base:: sync_with_stdio(false),cin.tie(NULL));
using namespace std;
int main()
{
    fastread();
    int t,n;
    cin>>t;
    while(t--){
        int flag = 0,prodsum = 0;
        cin>>n;
        for(int i=0; i<34; i++){
            for(int j=0; j<34; j++){
                prodsum = 3 * i + 7 * j;
                if(prodsum == n){
                    flag = 1;
                    break;
                }
            }
        }
        if(flag == 1){
            cout<<"YES"<<endl;
        }
        else{
            cout<<"NO"<<endl;
        }

    }

    return 0;
}
```

Output:



C:\Users\nirob\Downloads\4.

```
2
6
YES
5
NO

Process returned 0 (0x0)   execution time : 49.212 s
Press any key to continue.
```

Conclusion: The C++ program efficiently checks if a given integer x is divisible by 3 or 7 or a combination of both. It outputs "YES" if any of these conditions are met; otherwise, it outputs "NO." This provides a concise and clear solution for divisibility checks.

1. Implement DFS to traverse a graph and print the nodes in the order that each node is visited. Your program will take input the number of vertices and edges, then the edges sequentially and print out the order that the nodes were visited.

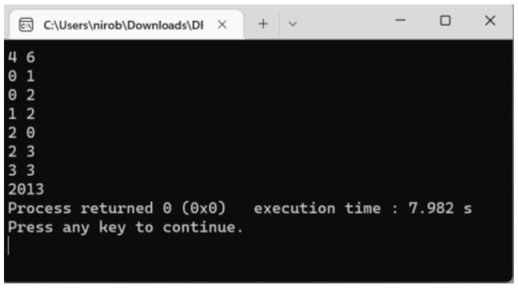
For the above graph,
Input:
4 6 //vertex number, edge number
0 1
0 2

1 2
2 0
2 3
3 3
Output: If we call DFS from vertex 2, the output will be,
2 0 1 3

Code:

```
#include<bits/stdc++.h>
using namespace std;
int nodeCount,edgeCount;
bool vis[1005];
void dfs(int src,vector<int>G[1005])
{
    vis[src]=true;
    cout<<src;
    for(auto v:G[src])
    {
        if(vis[v]==false)


        {
            vis[v]=true;
            dfs(v,G);
        }
    }
}
int main()
{
    cin>>nodeCount>>edgeCount;
    memset(vis,false,sizeof(vis));
    vector <int>G[1005];
    for(int i=1;i<=edgeCount;i++)
    {
        int a,b;
        cin>>a>>b;
        G[a].push_back(b);
    }
    dfs(2,G);
}
```

Output:


C:\Users\nirob\Downloads\DI

```
4 6
0 1
0 2
1 2
2 0
2 3
3 3
2013
Process returned 0 (0x0)   execution time : 7.982 s
Press any key to continue.
```

Conclusion: The C++ program efficiently performs Depth-First Search (DFS) on a graph. It takes the number of vertices, edges, and edge connections as input, and then starting from a specified vertex, it prints the order in which nodes are visited. This allows you to explore a graph in a systematic manner.

2. Implement BFS to traverse a graph and print the nodes in the order that each node is visited. Your program will take input the number of vertices and edges, then the edges sequentially and print out the order that the nodes were visited.

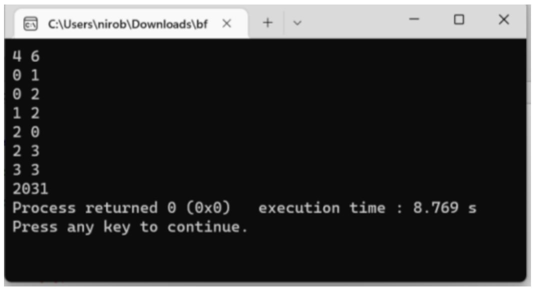
For the above graph,
Input:
4 6 //vertex number, edge number
0 1
0 2
1 2
2 0
2 3
3 3

Output: If we call BFS from vertex 2, the output will be,
2 0 3 1

Code:

```
#include<bits/stdc++.h>
using namespace std;

int nodeCount, edgeCount;
bool vis[1005];

void bfs (int src, vector <int>G[1005])
{
    queue <int>q;
    q.push (src);
    vis [src]=true;

    while (!q.empty())
    {
        int u=q.front();
        cout<< u ;
        q.pop();
        for (auto v : G[u])
        {
            if (! vis [v])
            {
                vis [v]=true;
                q.push (v);
            }
        }
    }
}

int main()
{
    cin>>nodeCount>>edgeCount;
        memset(vis,false,sizeof(vis));
        vector <int>G[1005];

    for (int i=1; i<=edgeCount; i++)
    {
        int a,b;
        cin>>a>>b;
        G[a].push_back (b);
    }
    bfs(2,G);
}
```

Output:



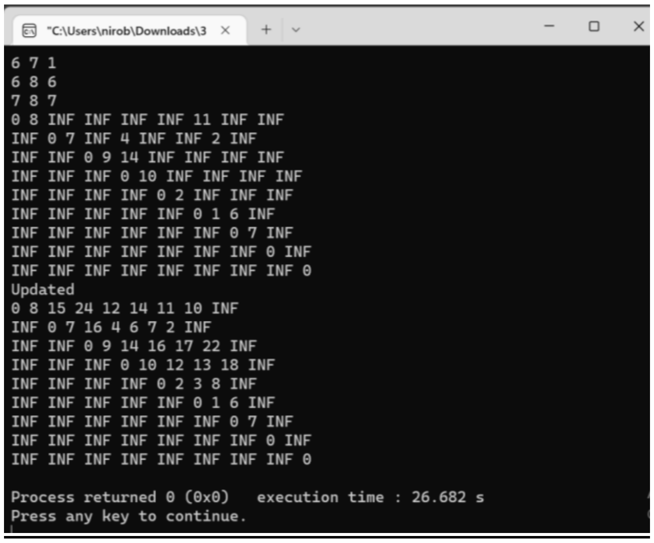
```
C:\Users\nirob\Downloads\bf
4 6
0 1
0 2
1 2
2 0
2 3
3 3
2031
Process returned 0 (0x0)   execution time : 8.769 s
Press any key to continue.
```

Conclusion: In this C++ program, we implemented Breadth-First Search (BFS) to traverse a graph and print the nodes in the order they are visited. The program takes the number of vertices, the number of edges, and the edge connections as input. It efficiently performs BFS starting from a specified vertex, marking visited nodes to avoid revisiting them.

3. Given a directed graph, find out the shortest path between all the pairs of vertices.

Code:
```
#include <bits/stdc++.h>
using namespace std;
const int INF = 1e7;
int main()
{
    int n, e;
    cin >> n >> e;
    int dis[n + 1][n + 1];
    for (int i = 1; i <= n; i++)
    {
        for (int j = 1; j <= n; j++)
        {
            dis[i][j] = INF;
            if (i == j)
                dis[i][j] = 0;
        }
    }
    while (e--)
    {
        int a, b, w;
        cin >> a >> b >> w;
        dis[a][b] = w;
    }
    for (int i = 1; i <= n; i++)
    {
        for (int j = 1; j <= n; j++)
        {
            if (dis[i][j] == INF)
                cout << "INF"
                     << " ";
            else
                cout << dis[i][j] << " ";
        }
        cout << endl;
    }
    for (int k = 1; k <= n; k++)
    {
        for (int i = 1; i <= n; i++)
        {
            for (int j = 1; j <= n; j++)
            {
                if (dis[i][k] + dis[k][j] < dis[i][j])
                {
                    dis[i][j] = dis[i][k] + dis[k][j];
                }
            }
        }
    }
    cout << "Updated" << endl;
    for (int i = 1; i <= n; i++)
    {
        for (int j = 1; j <= n; j++)
        {
            if (dis[i][j] == INF)
                cout << "INF"
                     << " ";
            else
                cout << dis[i][j] << " ";
        }
        cout << endl;
    }
    return 0;
}
```

<u>Output:</u>



```
"C:\Users\nirob\Downloads\3
6 7 1
6 8 6
7 8 7
0 8 INF INF INF INF 11 INF INF
INF 0 7 INF 4 INF INF 2 INF
INF INF 0 9 14 INF INF INF INF
INF INF INF 0 10 INF INF INF INF
INF INF INF INF 0 2 INF INF INF
INF INF INF INF INF 0 1 6 INF
INF INF INF INF INF INF 0 7 INF
INF INF INF INF INF INF INF 0 INF
INF INF INF INF INF INF INF INF 0
Updated
0 8 15 24 12 14 11 10 INF
INF 0 7 16 4 6 7 2 INF
INF INF 0 9 14 16 17 22 INF
INF INF INF 0 10 12 13 18 INF
INF INF INF INF 0 2 3 8 INF
INF INF INF INF INF 0 1 6 INF
INF INF INF INF INF INF 0 7 INF
INF INF INF INF INF INF INF 0 INF
INF INF INF INF INF INF INF INF 0

Process returned 0 (0x0)   execution time : 26.682 s
Press any key to continue.
```

Conclusion: The C++ program efficiently checks whether an undirected graph is bipartite. It takes the number of vertices, edges, and edge connections as input, and then uses a BFS-based approach to determine if the graph can be divided into two disjoint sets. If it's bipartite, the program outputs "Yes"; otherwise, it outputs "No." This program is a valuable tool for graph analysis and decision-making.

1. Given a weighted graph with the following adjacency matrix, apply Prim's algorithm to find the Minimum Spanning Tree (MST). Provide the MST edges and their weights.

Input:

3 3 //vertext number, edge number

0 1 5 //u, v, weight

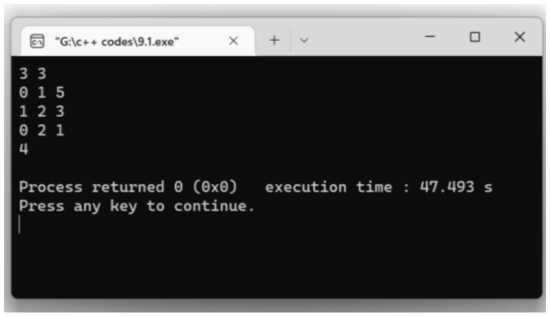
1 2 3

0 2 1

Output:

4

Code:



```
3 3
0 1 5
1 2 3
0 2 1
4

Process returned 0 (0x0)   execution time : 47.493 s
Press any key to continue.
```

3. Given an undirected weighted graph, write a program which outputs “YES” if the graph has a

cycle and outputs “NO” if it does not. Choose appropriate algorithm among Kruskal’s or Prim’s

algorithm to solve this problem.

Code:

```
#include <iostream>
```

```
#include <vector>

#include <list>

#include <algorithm>


using namespace std;


class Graph {

    int vertices;

    vector<list<int>> adjList;


public:

    Graph(int vertices) : vertices(vertices), adjList(vertices) {}


    void addEdge(int u, int v) {

        adjList[u].push_back(v);

        adjList[v].push_back(u);

    }


    bool isCyclicUtil(int v, vector<bool>& visited, int parent) {

        visited[v] = true;
        for (int neighbor : adjList[v]) {
            if (!visited[neighbor]) {
                if (isCyclicUtil(neighbor, visited, v)) {
                    return true;
                }
            } else if (neighbor != parent) {
                return true;
            }
        }

        return false;
    }

    bool isCyclic() {
        vector<bool> visited(vertices, false);

        for (int v = 0; v < vertices; ++v) {
            if (!visited[v]) {
                if (isCyclicUtil(v, visited, -1)) {
                    return true;
                }
            }
        }

        return false;
    }
};

int main() {
    int vertices, edges;
    cout << "Enter the number of vertices and edges: ";
    cin >> vertices >> edges;

    Graph graph(vertices);

    cout << "Enter the edges (u, v): ";
    for (int i = 0; i < edges; ++i) {
        int u, v;
        cin >> u >> v;
        graph.addEdge(u, v);

    }



    if (graph.isCyclic()) {

        cout << "YES - The graph has a cycle.\n";

    } else {

        cout << "NO - The graph does not have a cycle.\n";

    }



    return 0;

}
```

1. Implement Dijkstra Algorithm to find out the shortest path from a source to all the vertices in the graph.
Input: //vertex number, edge number

```
9 14
0 1 4
0 7 8
1 2 8
1 7 11
2 3 7
2 8 2
2 5 4
3 4 9
3 5 14
4 5 10
5 6 2
6 7 1
6 8 6
```

```
7 8 7
```

Output:

```
0 0 //Vertex, Distance from source
1 4
2 12
3 19
4 21
5 11
6 9
7 8
8 14
```

Code:

```
#include<bits/stdc++.h>
using namespace std;

typedef pair<int,int>pii;
int nodecount ,edgecount;
void dijkstra(int src,vector<vector<int>>G[1005],int dist[])
{
priority_queue <pii,vector<pii>,greater<pii> >pq;
pq.push({0,src});
dist[src]=0;
while(!pq.empty())
{
    int u = pq.top().second;
    pq.pop();
     int sz = G[u].size();
     for(auto v : G[u])
     {
         int cost = dist[u]+ v[1];
         if(dist[v[0]]>cost)
         {

             dist[v[0]]=cost;
             pq.push({cost,v[0]});
         }
     }
}


}
int main (){
cin>> nodecount>>edgecount;
int dist[nodecount];
vector<vector<int>>G[1005];
for (int i=1;i<=nodecount;i++)
{

    dist[i]=INT_MAX /2;
}
for(int i=1;i<= edgecount;i++)
{
     int u,v,weight;
     cin>> v>>weight;
     G[u].push_back({v,weight});
     G[v].push_back({u,weight});
}
dijkstra(0,G,dist);
for(int i=0;i<= nodecount;i++)
    cout<<dist[i]<<endl;
return 0;
}
```

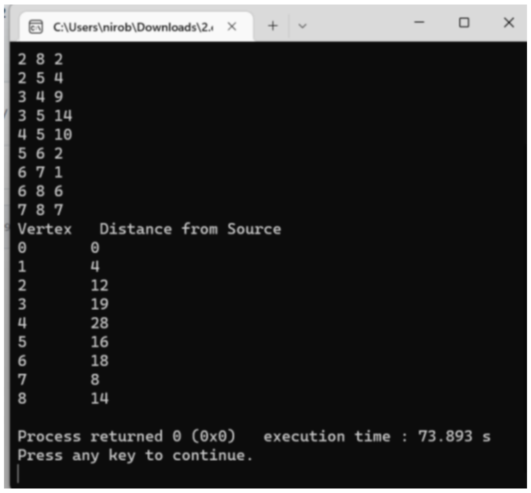
Output:

```
0 0
1 4
2 12
3 19
4 21
5 11
6 9
7 8
8 14
```

2. Suppose, there is a possibility that there might be a negative cycle in your given graph.
To detect negative cycles or to find the shortest path, what algorithm would you choose?
Implement the algorithm for a given graph. Use the same input format as the previous problem.

Code:
```
#include <iostream>
#include <vector>
#include <climits>

using namespace std;

struct Edge {
    int src, dest, weight;
};



void bellmanFord(vector<Edge>& edges, int src, int V) {
    vector<int> dist(V, INT_MAX);
    dist[src] = 0;

    for (int i = 1; i < V; i++) {
        for (const Edge& edge : edges) {
            int u = edge.src;
            int v = edge.dest;
            int weight = edge.weight;

            if (dist[u] != INT_MAX && dist[u] + weight < dist[v]) {
                dist[v] = dist[u] + weight;

            }
        }
    }

    // Check for negative cycles
    for (const Edge& edge : edges) {
        int u = edge.src;
        int v = edge.dest;
        int weight = edge.weight;
        if (dist[u] != INT_MAX && dist[u] + weight < dist[v]) {
            cout << "Negative cycle detected!" << endl;
            return;
        }
    }

    cout << "Vertex \t Distance from Source" << endl;
    for (int i = 0; i < V; i++) {
        cout << i << "\t" << dist[i] << endl;
    }
}

int main() {
    int vertices, edges;
    cin >> vertices >> edges;
    vector<Edge> graphEdges;

    for (int i = 0; i < edges; i++) {
        int u, v, weight;
        cin >> u >> v >> weight;
        graphEdges.push_back({u, v, weight});
    }

    bellmanFord(graphEdges, 0, vertices);

    return 0;
}
```

Output:

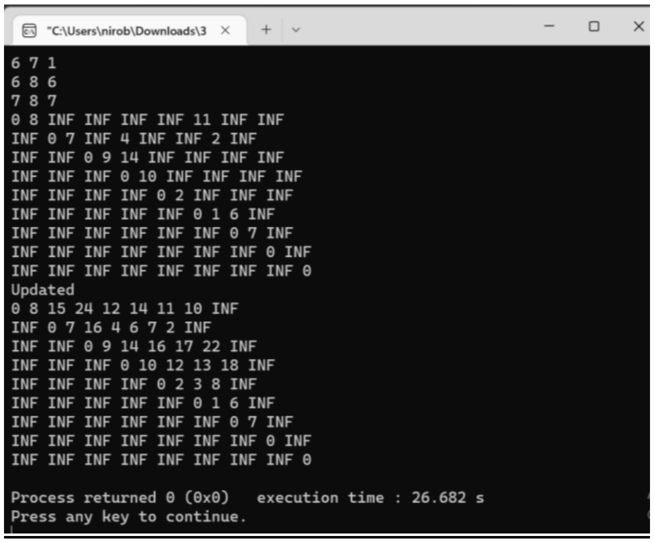


```
2 8 2
2 5 4
3 4 9
3 5 14
4 5 10
5 6 2
6 7 1
6 8 6
7 8 7
Vertex   Distance from Source
0       0
1       4
2       12
3       19
4       28
5       16
6       18
7       8
8       14

Process returned 0 (0x0)   execution time : 73.893 s
Press any key to continue.
```

3. Given a directed graph, find out the shortest path between all the pairs of vertices.

Code:
```
#include <bits/stdc++.h>
using namespace std;
const int INF = 1e7;
int main()
{
    int n, e;
    cin >> n >> e;
    int dis[n + 1][n + 1];
    for (int i = 1; i <= n; i++)
    {
        for (int j = 1; j <= n; j++)
        {
            dis[i][j] = INF;
            if (i == j)
                dis[i][j] = 0;
        }
    }
    while (e--)
    {
        int a, b, w;
        cin >> a >> b >> w;
        dis[a][b] = w;
    }
    for (int i = 1; i <= n; i++)
    {
        for (int j = 1; j <= n; j++)
        {
            if (dis[i][j] == INF)
                cout << "INF"
                     << " ";
            else
                cout << dis[i][j] << " ";
        }
        cout << endl;
    }
    for (int k = 1; k <= n; k++)
    {
        for (int i = 1; i <= n; i++)
        {
            for (int j = 1; j <= n; j++)
            {
                if (dis[i][k] + dis[k][j] < dis[i][j])
                {
                    dis[i][j] = dis[i][k] + dis[k][j];
                }
            }
        }
    }
    cout << "Updated" << endl;
    for (int i = 1; i <= n; i++)
    {
        for (int j = 1; j <= n; j++)
        {
            if (dis[i][j] == INF)
                cout << "INF"
                     << " ";
            else
                cout << dis[i][j] << " ";
        }
        cout << endl;
    }
    return 0;
}
```

Output:


"C:\Users\nirob\Downloads\3

```
6 7 1
6 8 6
7 8 7
0 8 INF INF INF INF 11 INF INF
INF 0 7 INF 4 INF INF 2 INF
INF INF 0 9 14 INF INF INF INF
INF INF INF 0 10 INF INF INF INF
INF INF INF INF 0 2 INF INF INF
INF INF INF INF INF 0 1 6 INF
INF INF INF INF INF INF 0 7 INF
INF INF INF INF INF INF INF 0 INF
INF INF INF INF INF INF INF INF 0
Updated
0 8 15 24 12 14 11 10 INF
INF 0 7 16 4 6 7 2 INF
INF INF 0 9 14 16 17 22 INF
INF INF INF 0 10 12 13 18 INF
INF INF INF INF 0 2 3 8 INF
INF INF INF INF INF 0 1 6 INF
INF INF INF INF INF INF 0 7 INF
INF INF INF INF INF INF INF 0 INF
INF INF INF INF INF INF INF INF 0

Process returned 0 (0x0)   execution time : 26.682 s
Press any key to continue.
```