IT300 Assignment 4

NAME: SUYASH CHINTAWAR
ROLL NO.: 191IT109
TOPIC: SHORTEST PATH
ALGORITHMS

Q1. Write a program to solve the currency conversion problem using shortest path algorithms(consider 10 different currencies) and draw the graph.

SOLUTION:

The currencies that have been taken are INR,MYR,HKD,EUR,GBP,AUD,BRL,SGD,JPY,USD and the last one is gold.

Output:

```
ubuntu@suyash-18-04:~/Desktop/Sem 5/IT300/Assignment4$ g++ currency_conversion.cpp
ubuntu@suyash-18-04:~/Desktop/Sem 5/IT300/Assignment4$ ./a.out
Enter the currency code (3 letter word. All caps): USD
INR: 129290
MYR: 7295.05
HKD: 13567
EUR: 1501.16
GBP: 1296.74
AUD: 2416.91
BRL: 9437.97
SGD: 2371.73
JPY: 195036
USD: 1742.61 (Final Answer)
GOLD: 1.00673
ubuntu@suyash-18-04:~/Desktop/Sem 5/IT300/Assignment4$
```

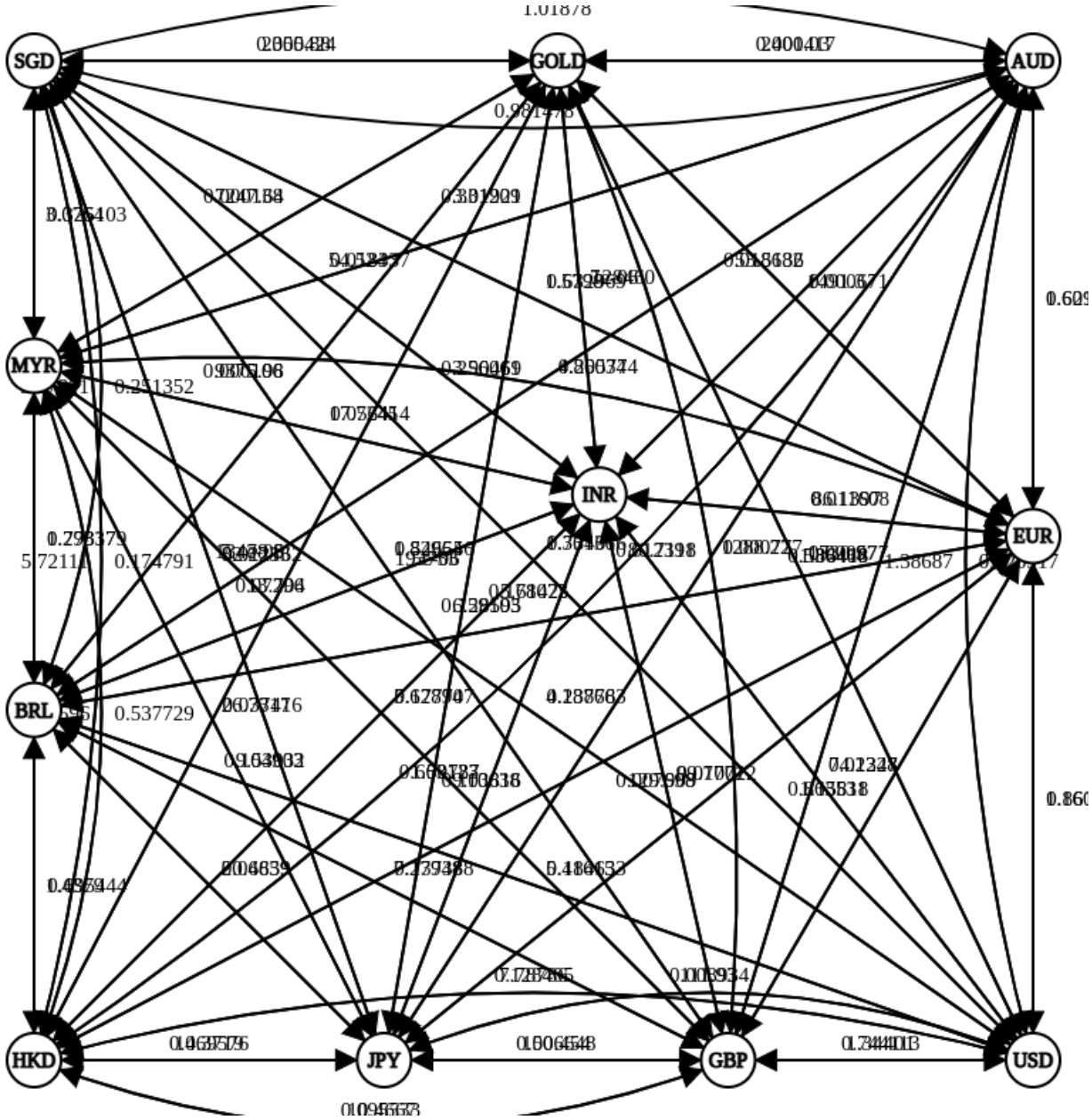
Code:

```cpp
/*
This program implements currency conversion problem using Bellman Ford Algorithm.
The exchange rates have been fixed in the main program.
The user will ask for required currency. (3 capital-letter word)
*/
#include<bits/stdc++.h>
using namespace std;
#define f first
#define s second
typedef vector<double> vd;
typedef pair<int,int> pii;

vd distances(11,INT_MAX); // stores the final distances or costs in this case

void bellman_ford(int n, vector<pair<pii,double>> edges,int src)
{
    distances[src]=0; // distance of source is zero frm itself
    int num_edges = edges.size();
    for(int i=0;i<n-1;i++) //run |v|-1 times
    {
        for(int j=0;j<num_edges;j++) // check for updates in each edge
        {
            int s,e;
            double cost;
            s=edges[j].f.f; //starting of edge
            e=edges[j].f.s; // end of edge
            cost=edges[j].s;
            if(distances[s]<INT_MAX) //update min cost distance found
            {
                distances[e] = min(distances[e],distances[s]+cost);
            }
        }
    }
}

int main()
{
    vector<vd> exchange_rates(11,vd(11,0.0)); // 10 currencies and one for gold

    //INR
    exchange_rates[0] = {1.000000, 17.724539, 9.530317, 86.139734, 99.777132, 53.513617, 13.703974, 54.523953, 0.663123, 74.222804, 128459.86};
    //MYR
    exchange_rates[1] = {0.056414, 1.000000, 0.537729, 4.860339, 5.628941, 3.019212, 0.773179, 3.076402, 0.037416, 4.187656, 7247.64};
    //HKD
    exchange_rates[2] = {0.104903, 1.859603, 1.000000, 9.038162, 10.466745, 5.614247, 1.437896, 5.721106, 0.069576, 7.787864, 13478.81};
    //EUR
    exchange_rates[3] = {0.011608, 0.205774, 0.110638, 1.000000, 1.158182, 0.621268, 0.159103, 0.632969, 0.007698, 0.861727, 1491.30};
    //GBP
    exchange_rates[4] = {0.010022, 0.177707, 0.095533, 0.863531, 1.000000, 0.536466, 0.137388, 0.546556, 0.006648, 0.744113, 1288.22};
    //AUD
    exchange_rates[5] = {0.018682, 0.331209, 0.178073, 1.609861, 1.864179, 1.000000, 0.256061, 1.018777, 0.012391, 1.386866, 2401.03};
    //BRL
    exchange_rates[6] = {0.072960, 1.293367, 0.695444, 6.285954, 7.279477, 3.904688, 1.000000, 3.978708, 0.048390, 5.416152, 9375.98};
    //SGD
    exchange_rates[7] = {0.018337, 0.325103, 0.174791, 1.579799, 1.829643, 0.981478, 0.251352, 1.000000, 0.012162, 1.361363, 2355.88};
```

```
    //SGD
    exchange_rates[7] = {0.018337, 0.325103, 0.174791, 1.579799, 1.829643, 0.981478, 0.251352, 1.000000, 0.012162, 1.361363, 2355.88};
    //JPY
    exchange_rates[8] = {1.507872, 26.731656, 14.371931, 129.908580, 150.454451, 80.711814, 20.665924, 82.223337, 1.000000, 111.929783, 193733.04};
    //USD
    exchange_rates[9] = {0.013470, 0.238783, 0.128405, 1.160558, 1.344010, 0.720917, 0.184633, 0.734616, 0.008934, 1.000000, 1730.97};
    //GOLD
    exchange_rates[10] = {0.000007, 0.000138, 0.000074, 0.000671, 0.000777, 0.000417, 0.000106, 0.000424, 0.000005, 0.000577, 1.00000};

    int n=11; //num vertices
    string target;
    cout<<"Enter the currency code (3 letter word. All caps): ";
    cin>>target;

    vector<pair<pii,double>> edges(n*n); //stores our graph
    unordered_map<string,int> s_id; //map from string to id
    unordered_map<int,string> id_s; // map from id to string
    vector<string> currencies = {"INR","MYR","HKD","EUR","GBP","AUD","BRL","SGD","JPY","USD","GOLD"};

    for(int i=0;i<currencies.size();i++) s_id[currencies[i]]=i;
    for(int i=0;i<currencies.size();i++) id_s[i]=currencies[i];

    //generate graph edges
    for(int i=0;i<n;i++)
    {
        for(int j=i+1;j<n;j++)
        {
            edges.push_back({{j,i},-1*log10(exchange_rates[i][j])});
            edges.push_back({{i,j},-1*log10(double(1)/exchange_rates[i][j])});
        }
    }

    //call bellman ford function with gold as source
    bellman_ford(n,edges,s_id["GOLD"]);

    //print all values
    for(int i=0;i<n;i++)
    {
        cout<<id_s[i]<<": "<<pow(10,-1*distances[i]);
        if(i==s_id[target]) cout<<" (Final Answer)";
        cout<<endl;
    }

}
```

The exchange rates have been fixed for all of these currencies. The output is 1742.61 USD for gold with the direct conversion of 1730.97 USD. this says there is a better path from 1 oz of gold to USD which yields a higher value

Graph:

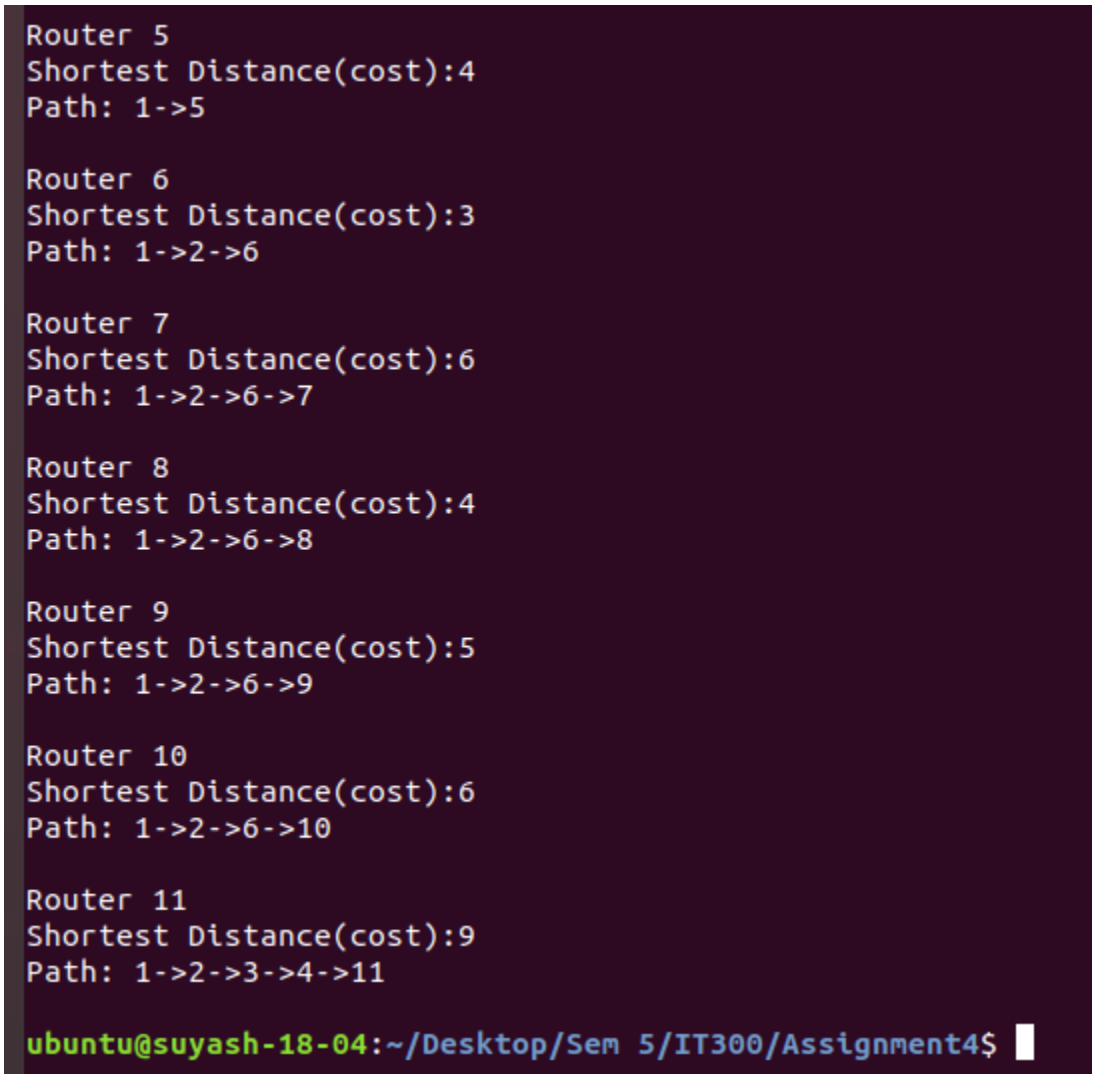


Q2. Write a program that should find the shortest cost path from the source router to other routers in the network(IP Routing problem).

SOLUTION:

Output:

```
ubuntu@suyash-18-04:~/Desktop/Sem 5/IT300/Assignment4$ g++ ip-routing.cpp
ubuntu@suyash-18-04:~/Desktop/Sem 5/IT300/Assignment4$ ./a.out
Enter number of routers(vertices) in the network:11
Enter number of edges:21
Enter edges(21 lines, 3 space separated integers on each line,
format: router1 router2 edge_weight):
NOTE: enter edges assuming 1-based vertices
1 2 1
1 5 4
1 8 6
2 3 2
2 6 2
5 2 4
5 6 3
8 5 2
8 6 1
8 9 6
3 4 2
9 6 2
9 10 6
6 4 2
6 7 3
6 10 3
4 11 4
10 7 1
10 11 4
7 11 3
7 4 1

*********SHORTEST PATHS AND THEIR DISTANCES (COSTS)*********
Router 1
Shortest Distance(cost):0
Path: 1

Router 2
Shortest Distance(cost):1
Path: 1->2

Router 3
Shortest Distance(cost):3
Path: 1->2->3

Router 4
Shortest Distance(cost):5
Path: 1->2->3->4

Router 5
Shortest Distance(cost):4
Path: 1->5

Router 6
Shortest Distance(cost):3
Path: 1->2->6
```


Router 5
Shortest Distance(cost):4
Path: 1->5
Router 6
Shortest Distance(cost):3
Path: 1->2->6
Router 7
Shortest Distance(cost):6
Path: 1->2->6->7
Router 8
Shortest Distance(cost):4
Path: 1->2->6->8
Router 9
Shortest Distance(cost):5
Path: 1->2->6->9
Router 10
Shortest Distance(cost):6
Path: 1->2->6->10
Router 11
Shortest Distance(cost):9
Path: 1->2->3->4->11
ubuntu@suyash-18-04:~/Desktop/Sem 5/IT300/Assignment4$

Code:

```cpp
/*
1)This program implements the Dijkstra's Algorithm for ip routing
considering edges as 1-based,
i.e. 1,2,3..n where 'n' is no. of routers
2)The program uses priority_queue C++STL to implement min heap
3)Time Complexity is O(ElogV) where E->#edges,V->#vertices
4)Adjacency list representation has been used to represent graphs
*/
#include<bits/stdc++.h>
using namespace std;
typedef pair<int,int> pii;//for easier declaration of variables

//Compute adjacency list of graph
void adjacency_list(int n,int edges,int edge[][3],vector<pii> adj_list[])
{
    for(int i=0;i<edges;i++)
    {
        adj_list[edge[i][0]-1].push_back(make_pair(edge[i][1],edge[i][2]));
        adj_list[edge[i][1]-1].push_back(make_pair(edge[i][0],edge[i][2]));
    }
}

//Implement Dijkstra's Algorithm
void dijkstra(int src,int n, vector<pii> adj_list[])
{
    vector<int> d(n,INT_MAX);//distances/costs of all routers, initialized to infinte
    vector<int> parent(n+1,-1);//parents of vertices(routers) to display final paths
    vector<int> found(n,0);//to store shortest distances/costs
    d[src-1]=0;
    priority_queue<pii,vector<pii>,greater<pii>> q;//min-heap
    q.push(make_pair(0,src));//pushing starting vertex(router) as distance is zero from itself
    while(!q.empty())
    {
        int dv=q.top().first;//distance of shortest distanced vertex
        int v=q.top().second;//vertex with shortest distance
        q.pop();
        if(dv!=d[v-1]) continue;//occur when path of adjacent vertex already is shortest
        found[v-1]=dv;
        for(auto x:adj_list[v-1])//check and update adjacent vertices' distances
        {
            int adj_x=x.first;
            int weight=x.second;
            if(dv+weight < d[adj_x-1])//shorter path found
            {
                d[adj_x-1]=dv+weight;
                q.push(make_pair(d[adj_x-1],adj_x));
                parent[adj_x]=v;
            }
        }
    }
    cout<<"\n*********SHORTEST PATHS AND THEIR DISTANCES (COSTS)*********\n";
    for(int i=0;i<n;i++)
    {
        cout<<"Router "<<i+1<<"\nShortest Distance(cost):"<<found[i]<<"\nPath: ";
        vector<int> path;
```

```
    }
    cout<<"\n*********SHORTEST PATHS AND THEIR DISTANCES (COSTS)*********\n";
    for(int i=0;i<n;i++)
    {
        cout<<"Router "<<i+1<<"\nShortest Distance(cost):"<<found[i]<<"\nPath: ";
        vector<int> path;
        for(int k=i+1;k!=-1;k=parent[k])
        {
            path.push_back(k);
        }
        for(int i=path.size()-1;i>=1;i--)
        {
            cout<<path[i]<<"->";
        }
        cout<<path[0];
        cout<<endl<<endl;
    }

}

int main()
{
    int n,edges,type;
    int edge[1000][3];
    cout<<"Enter number of routers(vertices) in the network:";
    cin>>n;
    cout<<"Enter number of edges:";
    cin>>edges;
    cout<<"Enter edges("<<edges<<" lines, 3 space separated integers on each line";
    cout<<"\nformat: router1 router2 edge_weight):\nNOTE: enter edges assuming 1-based vertices\n";
    for(int i=0;i<edges;i++)
    {
        cin>>edge[i][0]>>edge[i][1]>>edge[i][2];
    }

    vector<pii> adj_list[n];
    adjacency_list(n,edges,edge,adj_list);
    dijkstra(1,n,adj_list);
}
```

THANK YOU