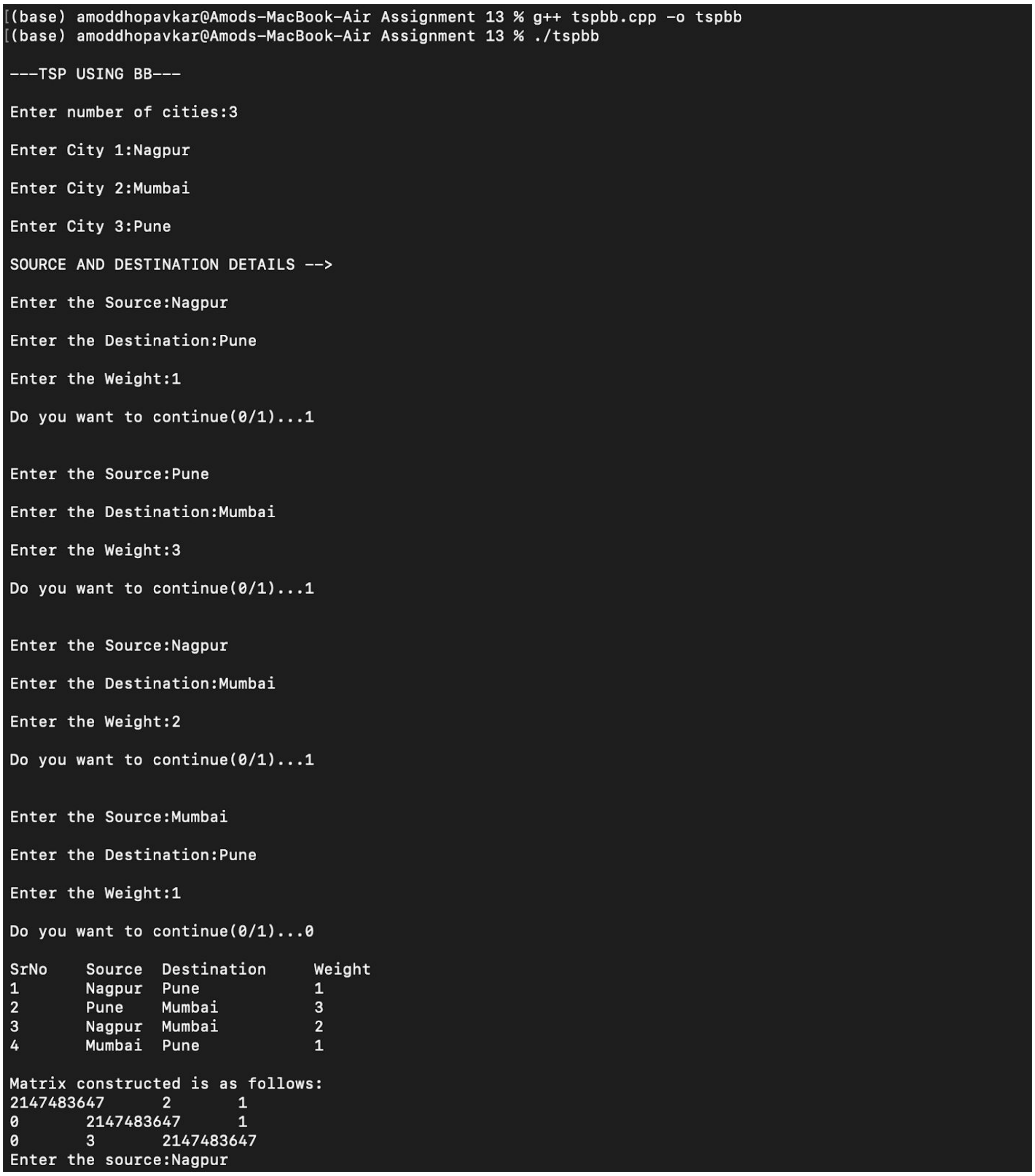
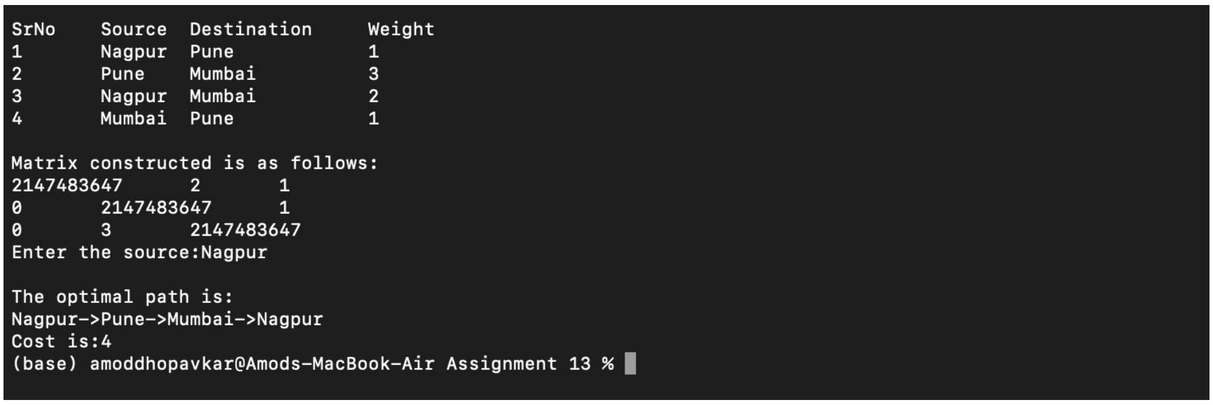
```
/*
Name            : Amod Dhopavkar
Roll No         : 33304
Batch       : K-11
Problem     : TSP using DP
*/

#include<iostream>
#include <bits/stdc++.h>
#include<vector>
#define SIZE 100
using namespace std;
vector<int>a;

typedef struct Graph{
    int u;
    int v;
    int weight;
    string city;
    string s1;
    int reduced[100][100];
}Graph;

int rowreduction(Graph G[100],int n,int r) {                    //Row reduction
    int i,j,row[n];
    fill_n(row,n,INT_MAX);
    for(i=0;i<n;i++) {
        for(j=0;j<n;j++) {
            if(G[r].reduced[i][j]<row[i]) {
                row[i]=G[r].reduced[i][j];
            }
        }
    }

    for(i=0;i<n;i++) {
        for(j=0;j<n;j++) {
            if(G[r].reduced[i][j]!=INT_MAX && row[i]!=0) {      //Checking for 0 and Infinite
                G[r].reduced[i][j]-=row[i];
            }
        }
    }

    int rowsum=0;
    for(i=0;i<n;i++) {
        rowsum+=row[i];
    }
    return rowsum;
}

int columnreduction(Graph G[100],int n,int r) {                //Column reduction
    int i,j,col[n];
    fill_n(col,n,INT_MAX);
    for(i=0;i<n;i++) {
        for(j=0;j<n;j++) {
            if(G[r].reduced[j][i]<col[i]) {
                col[i]=G[r].reduced[j][i];
            }
        }
    }

    for(i=0;i<n;i++) {
        for(j=0;j<n;j++) {
            if(G[r].reduced[j][i]!=INT_MAX && col[i]!=0) {      //Checking for 0 and Infinite
                G[r].reduced[j][i]-=col[i];
            }
        }
    }

    int colsum=0;
    for(i=0;i<n;i++) {
        colsum+=col[i];
    }
    return colsum;
}

void initialize(Graph G[100],int j1,int n,int k) {
    int i,j;
    for(i=0;i<n;i++) {
        for(j=0;j<n;j++) {

            G[a[j1]].reduced[i][j]=G[k].reduced[i][j];
        }
    }

    for(i=0;i<n;i++)                                    //Set source and destination to INT_MAX
    {
        G[a[j1]].reduced[k][i]=INT_MAX;
        G[a[j1]].reduced[i][j1]=INT_MAX;
    }
    G[a[j1]].reduced[k][j1]=INT_MAX;
}
```

```
int main() {
    int n,i,j;
    Graph G[SIZE];
    map<string,int>m;
    map<int,string>m1;

    string city;

    cout<<"\n---TSP USING BB---\n";
    cout<<"\nEnter number of cities:";
    cin>>n;

    int M[100][100];
    int count=1;
    for(i=0;i<n;i++) {
        cout<<"\nEnter City "<<i+1<<":";
        cin>>city;
        m.insert(pair<string,int>(city,i));
        m1.insert(pair<int,string>(i,city));
    }

    for(i=0;i<n;i++) {                                          //Initialize the main matrix
        for(j=0;j<n;j++) {
            if(i==j) {
                M[i][j]=INT_MAX;
            }

            else {
                M[i][j]=0;
            }
        }
    }

    int c=0;
    cout<<"\nSOURCE AND DESTINATION DETAILS -->\n";
    while(count==1) {
        cout<<"\nEnter the Source:";
        cin>>G[c].city;
        cout<<"\nEnter the Destination:";
        cin>>G[c].s1;
        cout<<"\nEnter the Weight:";
        cin>>G[c].weight;
        G[c].u=m[G[c].city];
        G[c].v=m[G[c].s1];
        M[G[c].u][G[c].v]=G[c].weight;
        c++;
        cout<<"\nDo you want to continue(0/1)...";
        cin>>count;

        if(count==0) {
            break;
        }
        cout<<endl;
    }

    cout<<endl;
    cout<<"SrNo"<<"\t"<<"Source"<<"\t"<<"Destination"<<"\t"<<"Weight"<<endl;
    for(i=0;i<c;i++) {
        cout<<i+1<<"\t"<<G[i].city<<"\t"<<G[i].s1<<"\t\t"<<G[i].weight<<endl;
    }
    cout<<endl;
    cout<<"Matrix constructed is as follows:\n";
    for(i=0;i<n;i++) {
        for(j=0;j<n;j++) {
            cout<<M[i][j]<<"\t";
        }
        cout<<endl;
    }

    c=0;
    for(i=0;i<n;i++) {
        for(j=0;j<n;j++) {
            G[c].reduced[i][j]=M[i][j];
        }
    }

    cout<<"Enter the source:";
    cin>>city;
    cout<<endl;
    int r=m[city];

    vector<int>result;
    vector<int>cos_bound;
    int rowsum=rowreduction(G,n,r);
    int colsum=columnreduction(G,n,r);
    int l_bound=rowsum+colsum;
    result.push_back(r);
    cos_bound.push_back(l_bound);                                   //Source vertex cost

    for(j=0;j<n;j++) {
        if(j==r) {
            continue;
        }

        else {
            a.push_back(j);                                    //unvisited vertex
        }
    }

    int k=r;                                               //index of the visited node
    int i1,j1;
    int min_cost=INT_MAX;                                      //minimum cost for one level
    i=0;
    int count1=0;
    int cost=0;
    int l=a.size();
    while(a.size()>0) {
        int index=0;
        int bound=-1;
        for(j=0;j<a.size();j++) {
            initialize(G,j,n,k);                          //initialize with latest matrix whose index
has the min cost
            rowsum=rowreduction(G,n,a[j]);                        //row reduction
            colsum=columnreduction(G,n,a[j]);                      //column reduction
            l_bound=rowsum+colsum;

            if(min_cost>l_bound+M[k][a[j]]+cos_bound[count1]) {        //Min cost
                min_cost=l_bound+M[k][a[j]]+cos_bound[count1];
                bound=l_bound;
                index=j;
            }
        }

        cos_bound.push_back(min_cost);                           //push the lower bound cost
        result.push_back(a[index]);                           //push the visited vertex
        cost+=M[k][a[index]];                              //add the cost of the vertex
        k=a[index];                                     //Next live node
        a.erase(a.begin()+index);                             //remove it from the unvisited nodes
        count1++;

    }

    cost+=M[result[result.size()-1]][m[city]];
    result.push_back(m[city]);                                  //Insert the source in the path
    cout<<"The optimal path is:\n";
    for(i=0;i<result.size();i++) {
        if(i==result.size()-1) {
            cout<<m1[result[i]];
        }
        else {
            cout<<m1[result[i]]<<"->";
        }
    }
    cout<<endl<<"Cost is:"<<cost<<endl;                                //Cost of the path
    return 0;
}
```

Output→



```
[(base) amoddhopavkar@Amods-MacBook-Air Assignment 13 % g++ tspbb.cpp -o tspbb
[(base) amoddhopavkar@Amods-MacBook-Air Assignment 13 % ./tspbb

---TSP USING BB---

Enter number of cities:3

Enter City 1:Nagpur

Enter City 2:Mumbai

Enter City 3:Pune

SOURCE AND DESTINATION DETAILS -->

Enter the Source:Nagpur

Enter the Destination:Pune

Enter the Weight:1

Do you want to continue(0/1)...1


Enter the Source:Pune

Enter the Destination:Mumbai

Enter the Weight:3

Do you want to continue(0/1)...1


Enter the Source:Nagpur

Enter the Destination:Mumbai

Enter the Weight:2

Do you want to continue(0/1)...1


Enter the Source:Mumbai

Enter the Destination:Pune

Enter the Weight:1

Do you want to continue(0/1)...0

SrNo    Source  Destination     Weight
1       Nagpur  Pune            1
2       Pune    Mumbai          3
3       Nagpur  Mumbai          2
4       Mumbai  Pune            1

Matrix constructed is as follows:
2147483647      2       1
0       2147483647      1
0       3       2147483647
Enter the source:Nagpur
```



```
SrNo    Source  Destination     Weight
1       Nagpur  Pune            1
2       Pune    Mumbai          3
3       Nagpur  Mumbai          2
4       Mumbai  Pune            1

Matrix constructed is as follows:
2147483647      2       1
0       2147483647      1
0       3       2147483647
Enter the source:Nagpur
```

```

The optimal path is:
Nagpur->Pune->Mumbai->Nagpur
Cost is:4
(base) amoddhopavkar@Amods-MacBook-Air Assignment 13 % 
```