Graph Description

Each of the node of the graphs denote a certain circle in the road map of a particular region. An edge denotes a road between them. Each edge is associated with two values, number of garbage bin located on a particular road and cost of visiting that road in the form: (garbage_bin_number, cost_for_a_road).

Input Graph

The input for the problem will be of the form:

Number_of_nodes/points_in_the_road_map Number_of_roads
Each of the next lines will be of the form,

> ***Name_of_the_first_endpoint_of_a_road***
> ***Name_of_the_second_endpoint_of_a_road***
> ***Number_of_garbage_bins_on_that_road Cost_of_operating_on_that_road***

So, according to the graphs drawn, inputs for our program are given below:

Input Graph #1:

9 10
Hatirpul Circle
Katabon Circle
2 13
Katabon Circle
Nilkhet Circle
1 9
Nilkhet Circle
Nilkhet Bus Stop
0 5
Nilkhet Bus Stop
Azimpur Bus Stand
1 11
Azimpur Bus Stand
Palashi Circle
1 6
Nilkhet Circle
Palashi Circle
1 7
Palashi Circle
Shahid Minar Circle

0 8
Shahid Minar Circle
Bakshi Bazar
1 6
Palashi CIrcle
Bakshi Bazar
1 10
Palashi Circle
Lalbagh Circle
3 15

Input Graph #2:

7 8
Moti Jhorna Lane
WASA Circle
2 8
Moti Jhorna Lane
Ispahani Circle
1 7
Ispahani Circle
WASA Circle
1 5
WASA Circle
Almas Cinema
0 8
Almas Cinema
Kazir Dewri Bazar
0 6
Kazir Dewri Bazar
Jamal Khan Circle
2 15
Jamal Khan Circle
Chittagong College
1 10
Ispahani Circle
Kazir Dewri Bazar
1 4

Input Graph #3:

9 10

2
3
1 19
2
1
2 13
1
4
1 5
4
5
2 18
4
6
0 3
5
8
1 9
8
7
3 21
7
9
0 14
2
7
1 20
6
5
2 11

DP Approach

Garbage trucks must pass through all the roads that have one or more garbage bins located on it. Now there can be roads in our map which do not contain any garbage bin but important for the traveling garbage trucks. For the sake of simplicity, we assume those roads containing no garbage bin are necessary for shorter routes. On the other hand, even if they are not needed for shorter routes, we can exclude those roads easily reconstruct our input graphs anytime.

So, we have some input graphs with nodes, edges, costs and values on edges, we need to visit some particular edges always to operate on them. We can think of Euler Circuit here.

How? Firstly, we assumed that all of our given roads are important for the map (if they are not, we can exclude them anytime and reconstruct the graph as mentioned before). Now, since all of them are important, we are sure to visit all the roads at least once.

Now if each of the nodes of our input graph has an even degree, the Euler Circuit will be the answer for our approach. It's because in an Euler Circuit, we leave our starting node, visit each edge once and then return back to our starting point. But there will be cases where a node has odd number of edges incident on it. If there are nodes who do not have even degree, in that case, like before, we will start from our starting point, visit the roads in our location map and return back to where we started. But in this case, we will have to visit some roads more than once since not all edges have even degree. To think this scenario in a simpler way, we can assume that we are adding some imaginary edges which will make the degrees of every edge even. And these imaginary edges will be added between two nodes which do not have even number of edges incident on them.

This is the part where we need Dynamic Programming. What should be the combination of nodes with odd degrees on which we will add imaginary edges to complete an Euler Circuit? If we can choose a particular combination which provides the optimum cost for adding extra edges, adding this optimum cost with total edges cost of the graph will give us the solution for the problem. This way we find the optimal cost for our problem. To find route, we can just do the Euler Circuit based on the solution.

Below goes the ***pseudocode*** for the DP approach:

```
Input

Calculate degree of each node

Run all pairs shortest path algorithm on the input graph, preferably
Floyd-Warshall Algorithm

Run bitmask DP to calculate the preferred combination of nodes to
add imaginary edges

In bitmask DP function:
      Each bit of the mask denotes a node, 1 means this node has
      even number of edges, 0 means it has odd number of edges
      In each step of DP, we find two nodes with odd degree, add
      their cost and go for the next step
      We do it for each combination and take the minimum

      Return minimum cost
```

To calculate the path, we remember which combination of nodes we got from our bitmask DP approach and make an Euler Circuit

This way we can easily calculate our shortest tour along with total cost for optimal garbage truck routing problem.

Complexity Analysis of Dynamic Programming Approach

The complexity of the approach described above is rather frustrating. It is because we are using a bitmask which itself can support at best 20 bits to work on. Unfortunately, we need to find a pair of nodes which have odd degree in each step of DP call. So, even with tweaking with bits to reduce loop calculation, we don't have a better complexity than $O(n^2 2^n)$

$n = 19$ is the best we can do using DP approach where n is the number of nodes.

Code Implemented in C++

```
#include <bits/stdc++.h>

#define MAX 20
#define INF 1e8

using namespace std;

int n, m, u, v, c, g[MAX][MAX];
string a, b;
int cnt=0, d[MAX][MAX], deg[MAX];
map<string,int> mapped;
int dp[(1<<MAX)+2], path[(1<<MAX)+2];
vector<string> rev;
vector<int> graph[MAX];
bool gone[MAX][MAX];
int extra[MAX]; // to calculate extra travel

void floydWarshall()
{
    for(int i=0; i<n; i++)
    {
        for(int j=0; j<n; j++)
        {
            for(int k=0; k<n; k++)
            {
                d[j][k]=min(d[j][k],d[j][i]+d[i][k]);
            }
        }
    }
}
```

```
int go(int mask)
{
    if(mask==(1<<n)-1) return 0;
    if(dp[mask]!=-1) return dp[mask];

    int ret=INF;

    for(int i=0; i<n; i++)
    {
        for(int j=i+1; j<n; j++)
        {
            if(!(mask&(1<<i)) && !(mask&(1<<j)))
            {
                int out=d[i][j]+go(mask|(1<<i)|(1<<j));

                if(ret>out)
                {
                    ret=out;
                    path[mask]=(mask|(1<<i)|(1<<j));
                }
            }
        }
    }

    return dp[mask]=ret;
}

void generate_combination(int mask)
{
      if(mask==(1<<n)-1) return;

      int nxt=path[mask];

      vector<int> edge;

      for(int i=0; i<n; i++)
      {
            if((mask&(1<<i))!=(nxt&(1<<i)))
            {
                  // cout<<i<<endl;
                  edge.push_back(i);
            }
      }

      cout<<rev[edge[0]]<<"---"<<rev[edge[1]]<<endl;
      // cout<<edge[0]<<" "<<edge[1]<<endl;

      extra[edge[0]]=edge[1];
      extra[edge[1]]=edge[0];
```

```
        generate_combination(path[mask]);
}

int main()
{
        // freopen("in1.txt","r",stdin);
        // freopen("in2.txt","r",stdin);
        freopen("in3.txt","r",stdin);

        cin>>n>>m; // number of nodes and edges

        rev.resize(n); // to hold road point names

        cin.ignore();

        memset(d,63,sizeof(d)); // initializing distance array
        memset(dp,-1,sizeof(dp)); // initializing dp array

        int opt_cost=0;

        for(int i=0; i<m; i++)
        {
                getline(cin,a);
                getline(cin,b);

                // we map the names to integer values for the sake of simplicity

                if(mapped.find(a)==mapped.end())
                {
                        mapped[a]=cnt;
                        rev[cnt]=a;
                        cnt++;
                }

                if(mapped.find(b)==mapped.end())
                {
                        mapped[b]=cnt;
                        rev[cnt]=b;
                        cnt++;
                }

                int u=mapped[a], v=mapped[b];


                cin>>g[u][v]>>c; // number of grabage bins and cost

                opt_cost+=c;
                g[v][u]=g[u][v];
                d[u][v]=d[v][u]=c;
                deg[u]++; deg[v]++; // calculating degree of each node
```

```
		graph[u].push_back(v);
		graph[v].push_back(u);

		cin.ignore();
	}

	floydWarshall();

	int mask=0;

	for(int i=0; i<n; i++)
	{
		if(deg[i]%2==0)
		{
			mask|=(1<<i);
			// cout<<i<<endl;
		}
	}

	int out=go(mask);

	cout<<out<<endl;
	cout<<"Optimal cost: "<<opt_cost<<endl;
	cout<<"Extra Travel Between Nodes:"<<endl;

	generate_combination(mask);

	return 0;
}
```