"Heaven's Light is Our Guide"

Department of Computer Science & Engineering

RAJSHAHI UNIVERSITY OF ENGINEERING & TECHNOLOG

Lab Report-02

Submitted By:
Name: Khandoker Sefayet Alam
Roll:2003121
Department: Computer Science & Engineering
Section-C

Course code: CSE 2202
Course name: Algorithm Analysis and Design Sessional

Submitted To:

A. F. M. Minhazur Rahman
Lecturer, Department of CSE,RUET

Problem-01: Breadth First Search: Shortest Reach

Problem Statement:

Consider an undirected graph where each edge weighs 6 units. Each of the nodes is labeled consecutively from 1 to n.

You will be given a number of queries. For each query, you will be given a list of edges describing an undirected graph. After you create a representation of the graph, you must determine and report the shortest distance to each of the other nodes from a given starting position using the *breadth-first search* algorithm (BFS). Return an array of distances from the start node in node number order. If a node is unreachable, return -1 for that node.

Algorithm:

Using Breadth First Search or BFS , we can calculate the shortest path distance from the source for each node. If the node is disconnected from the source, the result will be -1. To implement this, we at first take input of the graph as an adjacency list and using BFS, we get shortest distance from the source for each node. If the node is connected, we multiply the distance by 6 as each node's weight is 6. Finally, we'll print all other nodes distance except the source.

Code:

```
#include<bits/stdc++.h>
#include<ext/pb_ds/assoc_container.hpp>
#include<ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

//VVI
#define fast ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);
#define pb                  push_back
#define ll                  long long
```

```cpp
#define nn '\n'
#define mem(a,b) memset(a,b,sizeof(a))
#define all(x) x.begin(), x.end()

//CONSTANTS
#define md                  10000007
#define PI 3.14159265358979323846
const double EPS = 1e-9;
const ll N = 2e3+10;
const ll M = 1e9+7;




vector<ll>g[N];
bool vis[N];
ll level[N];


void bfs(ll source){
    queue<ll> q;
    q.push(source);
    vis[source]=1;
    level[source]=0;
    while(!q.empty()){
        ll curr_v=q.front();
        q.pop();
        // cout<<curr_v<<" ";
        for(ll child: g[curr_v]){
            if(!vis[child]){
                q.push(child);
                vis[child]=1;
                level[child]=level[curr_v]+1;
            }
        }
    }
    // cout<<endl;
}
int main()
{
    fast;
     ll t;
    //setIO();
     //ll tno=1;;
```

```
    t=1;
    cin>>t;

    while(t--){
      ll n,m;
      cin>>n>>m;
      for(ll i=0;i<=n+10;i++){
        g[i].clear();
        level[i]=LLONG_MAX;
        vis[i]=0;
      }
      ll x,y;
      for(ll i=0;i<m;i++){
        cin>>x>>y;
        g[x].push_back(y);
        g[y].push_back(x);
      }
      ll src;
      cin>>src;
      bfs(src);
      for(ll i=1;i<=n;i++){
        if(i==src) continue;
        if(level[i]==LLONG_MAX) cout<<-1<<" ";
        else cout<<(level[i])*6<<" ";
      }
      cout<<nn;

    }


    return 0;
}
```

Input and outputs:

Input-01:

2

4 2

1 2

1 3

1

3 1

2 3

2

Output-01:

6 6 -1

-1 6

Input-02:

1

5 3

1 2

1 3

3 4

1

Output-02:

6 6 12 -1

Problem-02: Journey to the Moon

Problem Statement:

The member states of the UN are planning to send 2 people to the moon. They want them to be from different countries. You will be given a list of pairs of astronaut ID's. Each pair is made of astronauts from the same country. Determine how many pairs of astronauts from different countries they can choose from.

Example

$n = 4$

$astronaut = [1, 2], [2, 3]$

There are 4 astronauts numbered 0 through 3. Astronauts grouped by country are $[0]$ and $[1, 2, 3]$. There are 3 pairs to choose from: $[0, 1]$, $[0, 2]$ and $[0, 3]$.

Algorithm:

To solve this problem, we consider all members are nodes of a graph and members from the same country are connected. we need to find out the total number of connected components in the graph and the number of nodes in each graph. Now to make a team, we can take two nodes from two different components. To find out how many pairs, for each component we can select one member from that and other member from rest of the components. As same pair will be picked twice i.e (1,2) and (2,1) in this process, we have to print the half of the total such pairs formed. To implement this, from node 1 to n, we apply dfs on the node if it's not visited (As visited means it already is part of a connected

component) and the dfs function returns the number of nodes in the connected compontent. We store these values in a vector and using combinatorics, we get the answer.

Code:

```
#include<bits/stdc++.h>
#include<ext/pb_ds/assoc_container.hpp>
#include<ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

//VVI
#define fast ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);
#define pb                  push_back
#define ll                  long long

#define nn '\n'

#define mem(a,b) memset(a,b,sizeof(a))
#define all(x) x.begin(), x.end()

const double EPS = 1e-9;
const ll N = 2e5+10;
const ll M = 1e9+7;


vector<ll>g[N];
bool vis[N];


ll dfs(ll vertex){
    //take action on vertex after entering the vertex
    vis[vertex]=true;
    // cout<<vertex<<endl;
    ll ans=1;
    for(ll child:g[vertex]){
        // cout<<"par:"<<vertex<<" "<<"child:"<<child<<endl;
```

```
        if(vis[child]) continue;
        //take action on the node before entering the child
        ans=ans+dfs(child);
        //take action on the node after exiting the child
    }
    //take action on the vertex after exiting the node
    return ans;
}
int main()
{
    fast;
     ll t;
    //setIO();
     //ll tno=1;;
     t=1;
    //cin>>t;

    while(t--){
      ll n,p;
      cin>>n>>p;
      ll x,y;
      for(ll i=0;i<p;i++){
        cin>>x>>y;
        g[x].push_back(y);
        g[y].push_back(x);
      }
      vector<ll>cnts;
      ll tot=0;
      for(ll i=0;i<n;i++){
        if(!vis[i]){
            ll g=dfs(i);
            // cout<<i<<" "<<g<<nn;
            cnts.push_back(g);
            tot+=g;
        }
      }
      ll ans=0;
      for(ll i=0;i<cnts.size();i++){
        ans+=(tot-cnts[i])*cnts[i];
      }
      cout<<ans/2<<nn;


    }
```

```
    return 0;
}
```

Input and outputs:

Input-01:

5 3

0 1

2 3

0 4

Output-01:
6

Input-02:

4 1

0 2

Output-02:
5

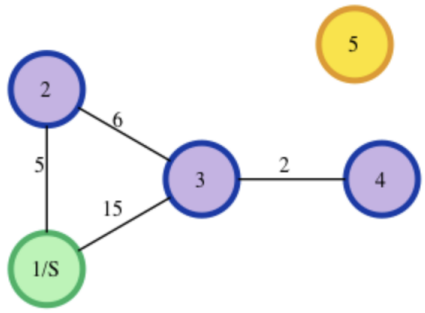
Problem-03: Dijkstra: Shortest Reach 2

Problem Statement:

Given an undirected graph and a starting node, determine the lengths of the shortest paths from the starting node to all other nodes in the graph. If a node is unreachable, its distance is -1. Nodes will be numbered consecutively from 1 to n , and edges will have varying distances or lengths.

For example, consider the following graph of 5 nodes:

Begin	End	Weight
1	2	5
2	3	6
3	4	2
1	3	15



Starting at node 1, the shortest path to 2 is direct and distance 5 . Going from 1 to 3 , there are two paths: 1->2->3 at a distance of 5+6=11 or 1->3 at a distance of 15 . Choose the shortest path, 11. From 1 to 4 , choose the shortest path through and extend it: 1->2->3->4 for a distance of 11 +2=13. There is no route to node 5 , so the distance is -1 .

The distances to all nodes in increasing node order, omitting the starting node, are 5 11 13 -1.

Algorithm:

This problem can easily be solved by Dijkstra algorithm as it determines the shortest distance for all nodes in a weighted graph. At first we take input of the graph as adjacency list and then take input of the source and apply Dijkstra on the source. In Dijkstra function, we maintain a min priority queue that keeps the minimum first element at the top position. We push the source node to the queue as paired with 0 which indicates it's distance from the source. While there are elements in the priority queue, we get the top element and check if relaxation is possible for the node or not and if possible, we traverse the childs of the nodes and make relaxation if possible. Finally, we print the shortest distance from the source for all elements except the source.

Code:

```
#include<bits/stdc++.h>
#include<ext/pb_ds/assoc_container.hpp>
#include<ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

//VVI
#define fast ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);
#define pb                  push_back
#define ll                  long long

#define nn '\n'

#define mem(a,b) memset(a,b,sizeof(a))
#define all(x) x.begin(), x.end()
#define rev(x) reverse(all(x))

//CONSTANTS
#define md                  10000007
#define PI 3.14159265358979323846
const double EPS = 1e-9;
const ll N = 2e5+10;
const ll M = 1e9+7;
```

```
/// Data structures
typedef unsigned long long ull;
typedef pair<ll, ll>    pll;
typedef vector<ll>      vl;
typedef vector<pll>     vpll;
typedef vector<vl>      vvl;
template <typename T> using PQ = priority_queue<T>;
template <typename T> using QP = priority_queue<T,vector<T>,greater<T>>;

template <typename T> using ordered_set = tree<T, null_type, less<T>,
rb_tree_tag, tree_order_statistics_node_update>;
template <typename T,typename R> using ordered_map = tree<T, R , less<T>,
rb_tree_tag, tree_order_statistics_node_update>;
;



vpll g[N];
vl dist(N,LLONG_MAX);
vl par(N,-1);
ll n,m,x,y,k;
void dijkstra(ll source){
    QP<pll> pq;
    pq.push({0,source});
    dist[source]=0;
    while(pq.size()){
        ll v=pq.top().second;
        ll v_dist=pq.top().first;
        pq.pop();
        if(dist[v]<v_dist) continue;
        for(auto &child:g[v]){
            ll child_v=child.first;
            ll wt=child.second;
            if(dist[v]+wt<dist[child_v]){
                dist[child_v]=dist[v]+wt;
                par[child_v]=v;
                pq.push({dist[child_v],child_v});
            }
        }
    }
}
void reset(){
        for(ll i=0;i<=n+10;i++){
            g[i].clear();
            par[i]=-1;
```

```
            dist[i]=LLONG_MAX;
        }
}
int main()
{
    fast;
     ll t;
    //setIO();
     //ll tno=1;;
     t=1;
    cin>>t;

    while(t--){

        cin>>n>>m;
        reset();
        for(ll i=0;i<m;i++){
            cin>>x>>y>>k;
            g[x].push_back({y,k});
            g[y].push_back({x,k});
        }
        ll src;
        cin>>src;
        dijkstra(src);
        for(ll i=1;i<=n;i++){
            if(i==src) continue;
            if(dist[i]==LLONG_MAX) cout<<-1<<" ";
            else cout<<dist[i]<<" ";
        }
        cout<<nn;
    }


    return 0;
}
```

Input and outputs:

Input-01:

1

4 4

1 2 24

1 4 20

3 1 3

4 3 12

1

Output-01:

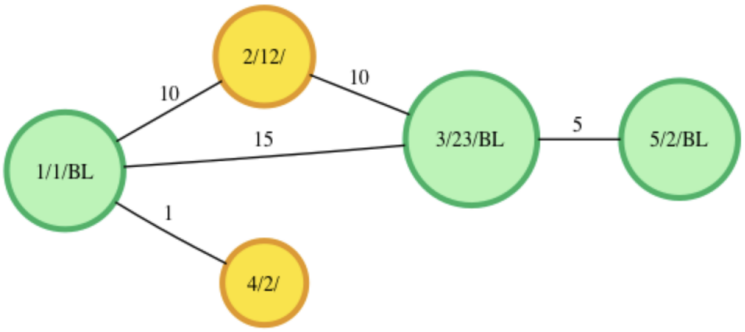
24 3 15

Problem-04: Synchronous Shopping

Problem Statement:

Bitville is a seaside city that has a number of shopping centers connected by bidirectional roads, each of which has a travel time associated with it. Each of the shopping centers may have a fishmonger who sells one or more kinds of fish. Two cats, ***Big Cat*** and ***Little Cat***, are at shopping center 1 (each of the centers is numbered consecutively from 1 to n). They have a list of fish they want to purchase, and to save time, they will divide the list between them. Determine the total travel time for the cats to purchase all of the types of fish, finally meeting at shopping center n . Their paths may intersect, they may backtrack through shopping center n , and one may arrive at a different time than the other. The minimum time to determine is when both have arrived at the destination.

For example, there are $n = 5$ shopping centers selling $k = 3$ types of fish. The following is a graph that shows a possible layout of the shopping centers connected by $m = 4$ paths. Each of the centers is labeled `center number/fish types offered/cat(s) that visit(s)`. Here B and L represent Big Cat and Little Cat, respectively. In this example, both cats take the same path, i.e. $1 \rightarrow 3 \rightarrow 5$ and arrive at time $15 + 5 = 20$ having purchased all three types of fish they want. Neither cat visits shopping centers 2 or 4.



Algorithm:

This problem can be solved using a variation of Dijkstra algorithm. In Dijkstra algorithm we only find out the shortest distance. But in this variation we need to find out shortest distance from source to a node taking certain fishes. Let's keep a record of fishes that we took on the way using bitmasks. Let 0101 this binary value represent that we have taken the first and third fish as from left 1st and 3rd bits are set. So, dist[x][5] will represent the shortest distance from source to x taking 1st and 3rd fish [Since 5's binary equivalent 0101]. At first we take inputs of the graph and using an array we store the bitmasked value of the types of fishes it has. While using Dijkstra, we'll keep records for each paths, which fishes we took and for the node we reach, we'll measure the shortest distance with certain fishes. In the end, as there are two cats, we'll bruteforce over bits to figure out if the paths compliment each other with all required fishes and we find the shortest time. Finally, we'll print the shortest time required for two cats to get all the fishes.

Code:

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

// VVI
#define fast                        \
    ios_base::sync_with_stdio(0); \
    cin.tie(0);                   \
    cout.tie(0);
#define pb push_back
#define ll long long
#define mp make_pair


#define nn '\n'

#define mem(a, b) memset(a, b, sizeof(a))
#define all(x) x.begin(), x.end()
#define rev(x) reverse(all(x))
```

```cpp
// CONSTANTS
#define md 10000007
#define PI 3.14159265358979323846
const double EPS = 1e-9;
const ll N = 2e3 + 10;
const ll M = 1e9 + 7;



/// Data structures
typedef unsigned long long ull;
typedef pair<ll, ll> pll;
typedef vector<ll> vl;
typedef vector<pll> vpll;
typedef vector<vl> vvl;
template <typename T>
using PQ = priority_queue<T>;
template <typename T>
using QP = priority_queue<T, vector<T>, greater<T>>;

template <typename T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
tree_order_statistics_node_update>;
template <typename T, typename R>
using ordered_map = tree<T, R, less<T>, rb_tree_tag,
tree_order_statistics_node_update>;
;



vpll g[N];
ll dist[N][1024];
vl musk(N, 0);
ll n, m, x, y, k, w;

void dijkstra(ll source)
{
    QP<pair<ll, pll>> pq;
    ll srcmsk = musk[source];

    pq.push({0, {source, srcmsk}});

    dist[source][srcmsk] = 0;
    while (pq.size())
    {
```

```
        auto currr=pq.top();

        ll v = currr.second.first;
        ll v_dist = currr.first;
        ll currmsk = currr.second.second;
        // cout<<v_dist<<" "<<currmsk<<" "<<v<<nn;
        pq.pop();
        // if (dist[v][currmsk] < v_dist)
        //     continue;
        for (auto &child : g[v])
        {
            ll child_v = child.first;
            ll wt = child.second;
            ll childmsk = currmsk | musk[child_v];
            if (v_dist + wt < dist[child_v][childmsk])
            {
                dist[child_v][childmsk] = v_dist + wt;
                pq.push({dist[child_v][childmsk], {child_v, childmsk}});
            }
        }
    }
}

void reset()
{
    for (ll i = 0; i <= n + 10; i++)
    {
        g[i].clear();
        musk[i]=0;
        for (ll j = 0; j < 1024; j++)
            dist[i][j] = LLONG_MAX;
    }
}

int main()
{
    fast;
    ll t;
    // setIO();
    // ll tno=1;;
    t = 1;
    // cin>>t;

    while (t--)
```

```
    {
        cin >> n >> m >> k;
        ll totmsk = ((1LL << k) - 1);
        reset();
        for (ll i = 1; i <= n; i++)
        {
            cin >> x;
            for (ll j = 0; j < x; j++)
            {
                cin >> y;
                y--;
                musk[i] |= (1<<y);
            }
        }
        for (ll i = 0; i < m; i++)
        {
            cin >> x >> y >> w;
            g[x].push_back({y, w});
            g[y].push_back({x, w});
        }
        dist[1][musk[1]]=0;
        dijkstra(1);
        // for(ll i=1;i<=n;i++) cout<<g[i]<<nn;

        ll ans = LLONG_MAX;
        for(ll i=0;i<=totmsk;i++){
            for(ll j=0;j<=totmsk;j++){
                if( (i | j ) == totmsk){
                    ll curr=max(dist[n][i],dist[n][j]);
                    // cout<<curr<<nn;
                    ans=min(ans,curr);
                }
            }
        }
        // cout<<dist[n][totmsk]<<nn;
        cout << ans << nn;
    }

    return 0;
}
```

Input and outputs:

Input-01:

5 5 5

1 1

1 2

1 3

1 4

1 5

1 2 10

1 3 10

2 4 10

3 5 10

4 5 10

Output-01:

30

Input-02:

6 10 3

2 1 2

1 3

0

2 1 3

1 2

1 3

1 2 572

4 2 913

2 6 220

1 3 579

2 3 808

5 3 298

6 1 927

4 5 171

1 5 671

2 5 463

Output-02:

792