Find bridges

```cpp
#include <bits/stdc++.h>
#define ll long long
#define R return
#define pb push_back
#define F first
#define S second
#define B break
using namespace std;
int timein[1000];
int timeout[1000];
int vis[1000];
int low[1000];
int timer=0;
vector <int>adj[1000];
void dfs(int node,int p){
    vis[node]=1;
    timein[node]=timer;
    low[node]=timer;
    timer++;
    for(auto it:adj[node]){
        if(it==p)C;
        if(vis[it]){
            low[node]=min(low[node],low[it]);
        }
        else{
            dfs(it,node);
            low[node]=min(low[node],low[it]);
            if(low[node]<low[it]){
                cout<<node<<" "<<it<<endl;
            }
        }
    }
     timeout[node]=timer;
            timer++;
}


int main(){
    int n,m;
    cin>>n>>m;
    for(int i=0;i<m;i++){
        int x,y;
        cin>>x>>y;
        adj[x].pb(y);
        adj[y].pb(x);
    }
    for(int i=1;i<=n;i++){
        if(!vis[i])dfs(i,-1);
    }
  return 0;}
```

articulation points

```cpp
#include <bits/stdc++.h>
#define ll long long
#define R return
#define pb push_back
#define F first
#define S second
#define B break
#define C continue
#define SI size()
#define En "\n"
using namespace std;
int timein[1000];
int timeout[1000];
int vis[1000];
int low[1000];
int timer=0;
int isap[1000];
vector <int>adj[1000];

void dfs(int node,int p){
    vis[node]=1;
    timein[node]=timer;
    low[node]=timer;
    timer++;
    int child=0;
    for(auto it:adj[node]){
        if(it==p)C;
        if(vis[it]){
            low[node]=min(low[node],timein[it]);
        }
        else{
            child++;
            dfs(it,node);
            low[node]=min(low[node],low[it]);
            if(low[it]>=timein[node]&&p!=-1){
                isap[node]=1;
            }

        }
    }
    if(p==-1&&child>1){isap[node]=1;}
}


int main(){
    int n,m;
    cin>>n>>m;
    for(int i=0;i<m;i++){
        int x,y;
        cin>>x>>y;
        adj[x].pb(y);
        adj[y].pb(x);
    }
    for(int i=1;i<=n;i++){
        if(!vis[i])dfs(i,-1);
    }
```

```
    for(int i=1;i<=n;i++){
        if(isap[i])cout<<i<<endl;
    }
  return 0;}
```

Dijkstra

```
#include <bits/stdc++.h>
#define ll long long
#define R return
#define pb push_back
#define F first
#define S second
#define B break
#define C continue
#define SI size()
#define os4 10005
#define os5 100005
#define os6 1000005
#define os7 10000005
#define inf 1000000007
#define En "\n"
using namespace std;
vector <pair<int,int>>adj[os5];
int d[os5];
int p[os5];
int vis[os5];
int n;
//number of nodes
void dijkstra(int node){
    fill(d,d+n+5,inf);
    fill(p,p+n+5,-1);
    d[node]=0;
    for(int i=0;i<n;i++){
        int v=-1;
        for(int j=0;j<n;j++){
            if(!vis[j]&&(v==-1||d[j]<d[v])){
                v=j;
            }
        }
    if(d[v]==inf)B;
    vis[v]=1;
    for(auto it:adj[v]){
        int to=it.F;
        int len=it.S;
        if(d[v]+len<d[to]){
            d[to]=d[v]+len;
            p[to]=v;
        }
    }
    }
}
vector<int> restore_path(int node, int to) {
    vector<int> path;
    for (int v = to; v != node; v = p[v]){
            path.push_back(v);
    }
    path.push_back(node);
```

```
52
53      reverse(path.begin(), path.end());
54      return path;
55  }
56  int main(){
57      int m;
58      cin>>n>>m;
59      for(int i=0;i<m;i++){
60          int x,y,z;
61          cin>>x>>y>>z;
62          adj[x].pb({y,z});
63          adj[y].pb({x,z});
64      }
65      dijkstra(0);
66      for(int i=0;i<n;i++){
67          cout<<i<<" "<<d[i]<<endl;
68      }
69      vector <int>a=restore_path(0,5);
70      for(auto it:a){
71          cout<<it<<" ";
72      }
73    return 0;
74  }
```

Bellman-Ford Algorithm

Single source shortest path with negative weight edges

```
1  #include <bits/stdc++.h>
2  #define ll long long
3  #define R return
4  #define pb push_back
5  #define F first
6  #define S second
7  #define B break
8  #define C continue
9  #define SI size()
10 #define os4 10005
11 #define os5 100005
12 #define os6 1000005
13 #define os7 10000005
14 #define inf 1000000007
15 #define En "\n"
16 using namespace std;
17 vector <pair<int,int>>adj[os5];
18
19 struct edge
20 {
21     int a, b, cost;
22 };
23
24 int  m, v;
25 int d[os5];
26 int p[os5];
27 int n;
28 vector<edge> e;
29 const int INF = 1000000000;
```

```
30
31 void solve()
32 {
33     v=0;//source
34     fill(d,d+n+5,inf);
35     fill(p,p+n+5,-1);
36     d[v] = 0;
37
38     for (;;)
39     {
40         bool any = false;
41         for (int j = 0; j < m; ++j)
42             if (d[e[j].a] < INF)
43                 if (d[e[j].b] > d[e[j].a] + e[j].cost)
44                 {
45                     d[e[j].b] = d[e[j].a] + e[j].cost;
46                     p[e[j].b] = e[j].a;
47                     any = true;
48                 }
49         if (!any)  break;
50     }
51     int t=4;//end of path
52     if (d[t] == inf)
53         cout << "No path from " << v << " to " << t << ".";
54     else
55     {
56         vector<int> path;
57         for (int cur = t; cur != -1; cur = p[cur])
58             path.push_back (cur);
59         reverse (path.begin(), path.end());
60
61         cout << "Path from " << v << " to " << t << ": ";
62         for (size_t i=0; i<path.size(); ++i)
63             cout << path[i] << ' ';
64     }
65     cout<<endl;
66 }
67 int main(){
68     cin>>n>>m;
69     for(int i=0;i<m;i++){
70         int x,y,z;
71         cin>>x>>y>>z;
72         e.pb({x,y,z});
73     }
74
75     solve();
76     for(int i=0;i<n;i++){
77         cout<<i<<" "<<d[i]<<endl;
78     }
79   return 0;
80 }
```

improved Bellman-Ford Algorithm with path restore

```
#include <bits/stdc++.h>
#define ll long long
#define R return
#define pb push_back
#define F first
#define S second
#define B break
#define C continue
#define SI size()
#define os4 10005
#define os5 100005
#define os6 1000005
#define os7 10000005
#define inf 1000000007
#define En "\n"
using namespace std;
vector <pair<int,int>>adj[os5];

struct edge
{
    int a, b, cost;
};

int  m, v;
int d[os5];
int p[os5];
int n;
vector<edge> e;
const int INF = 1000000000;

void solve()
{
    v=0;//source
    fill(d,d+n+5,inf);
    fill(p,p+n+5,-1);
    d[v] = 0;

    for (;;)
    {
        bool any = false;
        for (int j = 0; j < m; ++j)
            if (d[e[j].a] < INF)
                if (d[e[j].b] > d[e[j].a] + e[j].cost)
                {
                    d[e[j].b] = d[e[j].a] + e[j].cost;
                    p[e[j].b] = e[j].a;
                    any = true;
                }
        if (!any)  break;
    }
    int t=4;//end of path
    if (d[t] == inf)
        cout << "No path from " << v << " to " << t << ".";
    else
    {
        vector<int> path;
```

```
        for (int cur = t; cur != -1; cur = p[cur])
            path.push_back (cur);
        reverse (path.begin(), path.end());

        cout << "Path from " << v << " to " << t << ": ";
        for (size_t i=0; i<path.size(); ++i)
            cout << path[i] << ' ';
    }
    cout<<endl;
}
int main(){
    cin>>n>>m;
    for(int i=0;i<m;i++){
        int x,y,z;
        cin>>x>>y>>z;
        e.pb({x,y,z});
    }

    solve();
    for(int i=0;i<n;i++){
        cout<<i<<" "<<d[i]<<endl;
    }
  return 0;
}
```

Floyd-Warshall Algorithm

```
for (int i = 0; i < n; ++i) {
    for (int j = 0; j < n; ++j) {
        for (int t = 0; t < n; ++t) {
            if (d[i][t] < INF && d[t][t] < 0 && d[t][j] < INF)
                d[i][j] = - INF;
        }
    }
}
```

" "

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

Checking a graph for acyclicity and finding a cycle in

```
int n;
vector<vector<int>> adj;
vector<char> color;
vector<int> parent;
int cycle_start, cycle_end;

bool dfs(int v) {
    color[v] = 1;
    for (int u : adj[v]) {
        if (color[u] == 0) {
            parent[u] = v;
            if (dfs(u))
                return true;
        } else if (color[u] == 1) {
            cycle_end = v;
            cycle_start = u;
            return true;
        }
    }
    color[v] = 2;
    return false;
}

void find_cycle() {
    color.assign(n, 0);
    parent.assign(n, -1);
    cycle_start = -1;

    for (int v = 0; v < n; v++) {
        if (color[v] == 0 && dfs(v))
            break;
    }

    if (cycle_start == -1) {
        cout << "Acyclic" << endl;
    } else {
        vector<int> cycle;
        cycle.push_back(cycle_start);
        for (int v = cycle_end; v != cycle_start; v = parent[v])
            cycle.push_back(v);
        cycle.push_back(cycle_start);
        reverse(cycle.begin(), cycle.end());

        cout << "Cycle found: ";
        for (int v : cycle)
            cout << v << " ";
        cout << endl;
    }
}
```

" "

```cpp
int n;
vector<vector<int>> adj;
vector<bool> visited;
vector<int> parent;
int cycle_start, cycle_end;

bool dfs(int v, int par) { // passing vertex and its parent vertex
    visited[v] = true;
    for (int u : adj[v]) {
        if(u == par) continue; // skipping edge to parent vertex
        if (visited[u]) {
            cycle_end = v;
            cycle_start = u;
            return true;
        }
        parent[u] = v;
        if (dfs(u, parent[u]))
            return true;
    }
    return false;
}

void find_cycle() {
    visited.assign(n, false);
    parent.assign(n, -1);
    cycle_start = -1;

    for (int v = 0; v < n; v++) {
        if (!visited[v] && dfs(v, parent[v]))
            break;
    }

    if (cycle_start == -1) {
        cout << "Acyclic" << endl;
    } else {
        vector<int> cycle;
        cycle.push_back(cycle_start);
        for (int v = cycle_end; v != cycle_start; v = parent[v])
            cycle.push_back(v);
        cycle.push_back(cycle_start);
        reverse(cycle.begin(), cycle.end());

        cout << "Cycle found: ";
        for (int v : cycle)
            cout << v << " ";
        cout << endl;
    }
}
```

Topological sort

```
int n; // number of vertices
vector<vector<int>> adj; // adjacency list of graph
vector<bool> visited;
vector<int> ans;

void dfs(int v) {
    visited[v] = true;
    for (int u : adj[v]) {
        if (!visited[u])
            dfs(u);
    }
    ans.push_back(v);
}

void topological_sort() {
    visited.assign(n, false);
    ans.clear();
    for (int i = 0; i < n; ++i) {
        if (!visited[i])
            dfs(i);
    }
    reverse(ans.begin(), ans.end());
}
```

Coloring graph problem(Foarming teams)

```
#include <bits/stdc++.h>
#define ll long long
#define fast ios::sync_with_stdio(0),cin.tie(),cout.tie();
#define pb push_back
#define mp make_pair
#define Mx 1e9
#define F first
#define S second
#define con continue
#define Si size()
using namespace std;
ll a[200005];
int vis[1003];
int dfss[1003];
int team[1003];
vector <int>adj[2005];
int ans;
int t1,t2;
void dfs(int node,int t,int p){
  //  cout<<node<<" "<<team[node]<<" "<<t<<endl;
    if(vis[node]&&t!=team[node]){ans++;/*cout<<"ans\n";*/return;}
    if(vis[node])return;
```

```
    vis[node]=1;
    team[node]=t;
    for(auto v:adj[node]){
        int newt=(t==1)?2:1;
        if(v==p)con;
        dfs(v,newt,node);
    }
}
int main()
{
    int cnt=0;
    int n,m;
    cin>>n>>m;
    for(int i=0;i<m;i++){
        int u,v;
        cin>>u>>v;
        dfss[cnt]=u;
        cnt++;
        adj[u].pb(v);
        adj[v].pb(u);
        }
    for(int i=1;i<=cnt;i++){
        if(vis[dfss[i]])con;
        dfs(dfss[i],1,dfss[i]);
    }
    int left=0;
    for(int i=1;i<=n;i++){
        if(!vis[i])left++;
    }
    int res=ans/2;
    n-=res;
    if(n%2)res++;
    cout<<res;
    return 0;
}
```

BFS

```
void bfs(int x)
{
    queue <int> q;
    v[x]=1;
    q.push(x);
    while(!q.empty())
    {
        int u=q.front();
        cout<<u<<" ";
        q.pop();
        for(int i=0; i<adj[u].size(); i++)
        {
            int nod=adj[u][i];
            if(!v[nod])
            {
                v[nod]=1;
                q.push(nod);
            }
        }
    }}
```

Checking Cycle (RF)

```
bool cycle=0;
void dfs(int x)
{
    in[x]=1;
    v[x]=1;
    for(int i=0; i<adj[x].size(); i++)
    {
        int nod=adj[x][i];
        if(!v[nod])
            dfs(nod);
        else if(in[nod])
        {
            cycle=1;
            return ;
        }
    }
    in[x]=0;
}
```

Shortest Cycle in a Graph

```
int n;
int res=INT_MAX;
vector<int>adj[1000100];
vector<int> par(MX);
int shortest_cycle(int n,int st)
{
    int ans = INT_MAX;
    for (int i = 1; i <= st; i++)
    {
        vector<int> dist(n+1, (int)(1e9));
        dist[i] = 0;
        queue<int> q;
        q.push(i);
        while (!q.empty())
        {
            int x = q.front();
            q.pop();
            for(int j=0; j<adj[x].size(); j++)
            {
                int child=adj[x][j];
                if(child!=par[x])
                {
                    if (dist[child] == (int)(1e9))
                    {
                        dist[child] = 1 + dist[x];
                        par[child] = x;
```

```
                        q.push(child);
                    }
                    else
                        ans = min(ans, dist[x]+dist[child] + 1);
                }
            }
        }
    }
    res=min(res,ans);
}
```

DSU with max and min value

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

using namespace std;
int p[300007];
int siz[300007];
int mx[300007];
int mn[300007];
int get(int x){
    if(p[x]==x)return x;
        return p[x]=get(p[x]);
    }
    void unionn(int x,int y){
        x=get(x);
        y=get(y);
        if(x==y)return;
        if (siz[x]>siz[y])swap(x,y);
        p[x]=y;
        siz[y]+=siz[x];
        mx[y]=max(mx[x],mx[y]);
        mn[y]=min(mn[x],mn[y]);
    }
    int main()
    {
        ios::sync_with_stdio(0);
        cin.tie(0);
        cout.tie(0);
        int n,m;
        cin>>n>>m;
        for(int i=1;i<=n;i++){
            p[i]=i;
            siz[i]=1;
            mx[i]=i;
            mn[i]=i;
        }
            string s;
            int x,y;
        while(m--){
            cin>>s;
            if(s=="union"){
                cin>>x>>y;
                unionn(x,y);
            }
            else{
                cin>>x;
                x=get(x);
                cout<<mn[x]<<" "<<mx[x]<<" "<<siz[x]<<"\n";
            }

        }
```

```
50.          return 0;
51.      }
```

Priority Queue

```
1. priority_queue<int>q1;  //Descending Order
2. priority_queue<int,vector<int>,greater<int>>q2; //Ascending order
3. q.push(x); //add element
4. q.top();
5. q.pop();
```

Descending sort

```
1. sort(a,a+n,greater<>())
```

number of unique elements in string or vector

```
1. int m = unique(v.begin(), v.end())-v.begin();
```

Get unique elements

```
2. //for vector
3. vector<int> v = {1,1,1,1,1,1,1,2,3,3,3,10,1,2,3,7,7,8};
4. vector<int>::iterator ip;
5. ip = unique(v.begin(), v.end());
6. v.resize(distance(v.begin(), ip));
7. for (ip = v.begin(); ip != v.end(); ++ip) {
8. cout << *ip << " ";
9. //for string
10. string s;
11.     cin>>s;
12.     string::iterator sp;
13.     sp = unique(s.begin(), s.end());
14.     s.resize(distance(s.begin(),sp));
15.     for (sp = s.begin(); sp != s.end(); ++sp) {
16.         cout << *sp;
17.     }
```

Set presision

```
1. double a=3.324143124151524;
2. cout<<fixed<<setprecision(4)<<a;
```