a_dijkstra.cpp

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

#define INF 0x3f3f3f3f
#define FOR(i,n) for(int i = 0; i < n; ++i)
#define println(v) cout << v << endl

using namespace std;

void shortestPath(vector< vector <pair<int,int> > > adj, int v, int src, int finish)
{
    priority_queue < pair<int,int> , vector < pair <int, int> > , greater<pair < int, int >
 > > pq;
    vector<int> dist(v, INF);

    pq.push(make_pair(0, src));
    dist[src] = 0;
    while (!pq.empty()) {
        int u = pq.top().second;

        pq.pop();
        vector< pair <int,int> >::iterator x;
        for (x = adj[u].begin(); x != adj[u].end(); ++x) {
            int vertex = x->first;
            int weight = x->second;

            if (dist[vertex] > dist[u] + weight) {
                dist[vertex] = dist[u] + weight;
                pq.push(make_pair(dist[vertex], vertex));
            }
        }
    }
    println(dist[finish-1]);
}

int main()
{
    int n;
    int finish,value;

    cin >> n >> finish;
    //adj list
    vector < vector < pair < int, int > > > adj(n);
    //adj matrix
    vector < vector < pair < int, int > > > mtx(n,vector< pair < int, int > > (n));
    FOR(i,n){
        FOR(j,n+1){
            cin >> value;
            if(j!=0 && value != -1){
                adj[i].push_back(make_pair(j-1, value));
            }
        }
    }
    shortestPath(adj, n, 0,finish);
    return 0;
}
```

b_articulation_points.cpp

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

#define FOR(i, n) for(int i = 0; i < n; ++i)

using namespace std;



void art_points_utility(int u,  vector < int > &disc, vector < int > &low, vector < int > &
parent, vector < bool > &art_points, vector < vector < int > > adj, vector < bool > visited
) {
    static int time = 0;
    int children = 0;
    disc[u] = low[u] = ++time;
    vector<int>::iterator i;
    for (i = adj[u].begin(); i != adj[u].end(); ++i) {
        int v = *i;
        if (!visited[v]) {
            children++;
            parent[v] = u;
            art_points_utility(v, disc, low, parent, art_points,adj,visited);
            low[u] = min(low[u], low[v]);
            if (parent[u] == -1 && children > 1)
                art_points[u] = true;

            if (parent[u] != -1 && low[v] >= disc[u])
                art_points[u] = true;
        } else if (v != parent[u])
            low[u] = min(low[u], disc[v]);
    }
}

void APUtil(int u,  vector < int > &disc, vector < int > &low, vector < int > &parent, vect
or < bool > &art_points, vector < vector < int > > &adj, vector < bool > &visited) {
    static int time = 0;
    int children = 0;
    visited[u] = true;
    disc[u] = low[u] = ++time;
    vector<int>::iterator i;
    for (i = adj[u].begin(); i != adj[u].end(); ++i) {
        int v = *i;
        if (!visited[v]) {
            children++;
            parent[v] = u;
            APUtil(v, disc, low, parent, art_points, adj,visited);
            low[u] = min(low[u], low[v]);
            if (parent[u] == -1 && children > 1)
                art_points[u] = true;

            if (parent[u] != -1 && low[v] >= disc[u])
                art_points[u] = true;
        } else if (v != parent[u])
            low[u] = min(low[u], disc[v]);
    }
}


void articulation_points(int number_vertices, vector < vector < int > > adj) {

    bool is_ap_none = true;
    vector < int > parent(number_vertices,-1);
    vector < bool > art_points(number_vertices,false);
    vector < int > disc(number_vertices);
    vector < int > low(number_vertices);
    vector < bool > visited(number_vertices, false);

    vector < pair < int, int > > solutions;
```

b_articulation_points.cpp

```
    for (int i = 0; i < number_vertices; i++)
        if (!visited[i])
            APUtil(i,disc, low, parent,art_points,adj,visited);

    for (int i = 0; i < number_vertices; i++) {
        if (art_points[i]) {
            is_ap_none = false;
            vector<int>::iterator j;
            for (j = adj[i].begin(); j != adj[i].end(); ++j) {

                int pos = j - adj[i].begin();
                if (low[i] > disc[adj[i][pos]]) {
                    if (i < adj[i][pos])
                    {
                        solutions.push_back(make_pair(i,adj[i][pos]));
                    }

                    else
                    {
                        solutions.push_back(make_pair(adj[i][pos],i));

                    }

                }
            }
        }
    }

    if (is_ap_none) {
        printf("No road\n");
    }else{
        sort(solutions.begin(),solutions.end()/*,greater< pair < int, int > > ()*/);
        vector< pair < int, int > >::iterator itr;
        for(itr = solutions.begin(); itr != solutions.end(); ++itr){
            cout << itr->first << " " << itr->second << endl;
        }
    }
}




int main() {
    int number_positions, number_edges, vertex_a, vertex_b;

    while (cin >> number_positions >> number_edges) {
        vector < vector < int > > adj(number_positions);
        FOR(i, number_edges) {
            cin >> vertex_a >> vertex_b;
            adj[vertex_a].push_back(vertex_b);
            adj[vertex_b].push_back(vertex_a);
        }
        articulation_points(number_positions,adj);
    }
    return 0;
}
```

c_dfs_bfs.cpp

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

#define FOR(i,n) for(int i = 0; i < n; ++i)
#define FOR_ITERATOR(i,n) for(vector<int>::iterator i = adj[n].begin(); i!= adj[n].end();i+
+)
#define println(v) cout << v << endl

using namespace std;
vector < vector < int > > adj;
vector < bool > visited;
int depthh;


pair<int,int> bfs(int u, vector < int > &depth){

    queue<int> q;
    q.push(u);
    depth[u] = 0;
    int highest_depth = 0;
    visited[u] = true;
    while (!q.empty()) {
        int f = q.front();
        q.pop();
        FOR_ITERATOR(i,f) {
            if (!visited[*i]) {
                depth[*i] = depth[f] + 1;
                if(depth[*i]> highest_depth)
                    highest_depth = depth[*i];
                q.push(*i);
                visited[*i] = true;
            }

        }

    }
    return make_pair( highest_depth,u);
}

pair < int, int > dfs(int u, int depth){
    visited[u] = true;
    //depth_array[u] = depth;
    if(depthh < depth)
        depthh = depth;
    FOR_ITERATOR(i,u){
        if (!visited[*i]){

            dfs(*i,depth+1);

        }
    }
    //printf("vertex: %d depth: %d\n",u+1,depth);
    return make_pair(depthh,u);
}


int main(){
    int number_vertices,a,b;
    cin >> number_vertices;
    adj.resize(number_vertices);
    vector < pair < int, int > > shortest(number_vertices);
    vector < pair < int, int > > tallest(number_vertices);

    FOR(i,number_vertices-1){
        cin >> a >> b;
        adj[a-1].push_back(b-1);
        adj[b-1].push_back(a-1);
    }
```

c_dfs_bfs.cpp

```
    FOR(j,number_vertices){
        visited.clear();
        visited.resize(number_vertices,false);
        vector < int > depth(number_vertices,0);
        depthh = 1;
        tallest.push_back(dfs(j,0));
        visited.clear();
        visited.resize(number_vertices,false);
        depth.clear();
        depth.resize(number_vertices,0);
        shortest.push_back(bfs(j,depth));
    }

    sort(tallest.begin(),tallest.end(),greater<pair <int, int > >());
    sort(shortest.begin(),shortest.end());


    vector < int > result_shortest;
    vector < int > result_tallest;

    vector < pair < int, int > >::iterator i;
    int aux=0;
    for(i = shortest.begin();i!=shortest.end();i++){
        if(i->first!=0 && aux==0){
            result_shortest.push_back(i->second+1);
            aux = i->first;
        }
        else if(i->first!=0 && i->first == aux){
            result_shortest.push_back(i->second+1);
        }
    }

    vector < pair < int, int > >::iterator j;
    aux = 0;
    for(j = tallest.begin();j!=tallest.end();j++){
        if(j->first!=0 && aux==0){
            result_tallest.push_back(j->second+1);
            aux = j->first;
        }
        else if(j->first!=0 && j->first == aux){
            result_tallest.push_back(j->second+1);
        }
    }

    sort(result_shortest.begin(),result_shortest.end());
    sort(result_tallest.begin(),result_tallest.end());


    printf("Shortest: ");
    FOR(k,(int)result_shortest.size()){
        if(k!=(int)result_shortest.size()-1)
            printf("%d ",result_shortest[k]);        else
            printf("%d\n",result_shortest[k]);
    }

    printf("Tallest: ");
    FOR(l,(int)result_tallest.size()){
        if(l!=(int)result_tallest.size()-1)
            printf("%d ",result_tallest[l]);
        else
            printf("%d\n",result_tallest[l]);
    }

}
```

e_floyd_warshall.cpp

```
#include <bits/stdc++.h>
using namespace std;

#define FOR(i,n) for(int i = 0; i < n; ++i)
#define INF 0x3f3f3f3f
#define println(v) cout << v << endl

void floyd_warshall(vector< vector<int> > &graph, int number_vertices) {
    int i, j, k;

    for (i = 0; i < number_vertices; i++){
        for (j = 0; j < number_vertices; j++)
        {
            if(graph[i][j])
                graph[i][j] = graph[i][j];
            else if(i == j){
                graph[i][j] = 0;
            }
            else
                graph[i][j] = INF;
        }
    }

    for (k = 0; k < number_vertices; k++) {
        for (i = 0; i < number_vertices; i++) {
            for (j = 0; j < number_vertices; j++) {
                graph[i][j] = min(graph[i][j],graph[i][k]+graph[k][j]);
            }
        }
    }
/*
    FOR(l,26){
        FOR(h,26) {
            if(graph[l][h] != INF)
                printf("%d\t", graph[l][h]);
            else
                printf("INF\t");
        }
        printf("\n");
    }
    printf("\n\n");
    */
}

bool is_sub_path(vector < vector < int > > &graph1, vector < vector < int > > &graph2) {
    FOR(i,26) {
        FOR(j,26){
            if(graph1[i][j] != INF && graph2[i][j] == INF) {
                return false;
            }
            else if(graph1[i][j] == INF && graph2[i][j] != INF) {
                return false;
            }
        }
    }
    return true;
}

int main(){
    int number_test_cases, n_1,n_2;
    char v_a,v_b;

    cin >> number_test_cases;
    FOR(i,number_test_cases){

        cin >> n_1;
        vector< vector<int> > f1(26, vector<int>(26));
        FOR(j,n_1){
```

e_floyd_warshall.cpp

```
            cin >> v_a >> v_b;
            f1[v_a-65][v_b-65] =  1;
            /*printf("A entrada %c tem valor %d e liga ao %c com valor %d (total = %d)\n"
                    ,v_a,v_a,v_b,v_b,v_a+v_b);*/
            }

        cin >> n_2;
        //alloc memory for matrix
        vector< vector<int> > f2(26, vector<int>(26));
        FOR(j,n_2){
            cin >> v_a >> v_b;
            f2[v_a-65][v_b-65] =  1;
            /*printf("A entrada %c tem valor %d e liga ao %c com valor %d (total = %d)\n"
                    ,v_a,v_a,v_b,v_b,v_a+v_b);*/
        }
        floyd_warshall(f1,26);
        floyd_warshall(f2,26);
        if(is_sub_path(f1,f2))
        {
            println("YES");
            if(i!= number_test_cases-1)
            if(i!= number_test_cases-1)
                println("");
        }
        else
        {
            println("NO");
            if(i!= number_test_cases-1)
                println("");
        }

    }

}
```

f_bipartite.cpp

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

#define FOR(i,n) for(int i = 0; i < n; ++i)
#define FOR_ITERATOR(i,n) for(vector<int>::iterator i = adj[n].begin(); i!= adj[n].end();i+
+)
#define println(v) cout << v << endl

using namespace std;

vector < vector < int > > adj;
vector < int > color_array;

bool bfs(int u){

    queue<int> q;

    q.push(u);
    color_array[u] = 1;
    while (!q.empty()) {
        int f = q.front();
        q.pop();

        FOR_ITERATOR(i,f) {
            if (color_array[*i] == -1) {
                q.push(*i);
                color_array[*i] = 1 - color_array[f];
            }
            else if(color_array[*i] == color_array[f]){
                return false;
            }
        }
    }
    return true;
}

bool is_bipartite(int number_vertices) {
    FOR(j, number_vertices) {
        if (color_array[j] == -1) {
            if (!bfs(j)) {
                return false;
            }
        }
    }
    return true;
}

int main(){
   int number_vertices, number_edges, a, b;


    while(cin >> number_vertices >> number_edges){
        adj.clear();
        adj.resize(number_vertices);
        color_array.clear();
        color_array.resize(number_vertices,-1);
        FOR(i,number_edges){
            cin >> a >> b;
            adj[a-1].push_back(b-1);
            adj[b-1].push_back(a-1);
        }

        if(is_bipartite(number_vertices))
            println("NOT SURE");
        else
            println("NO");
    }
}
```

g_bfs.cpp

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

#define FOR(i,n) for(int i = 0; i < n; ++i)
#define FOR_ITERATOR(i,n) for(vector<int>::iterator i = adj[n].begin(); i!= adj[n].end();i+
+)
#define println(v) cout << v << endl

using namespace std;

int bfs(vector < vector < int > > adj, vector < bool > &visited, int u){

    queue<int> q;

    q.push(u);
    visited[u] = true;
    while (!q.empty()) {
        int f = q.front();
        q.pop();
        FOR_ITERATOR(i,f) {
            if (!visited[*i]) {
                q.push(*i);
                visited[*i] = true;
            }
            else if(visited[*i] && *i == u){
                return u;
            }
        }
    }
    return -1;
}

int main(){
    int number_vertices, number_functions,function_number;

    cin >> number_vertices;
    vector< vector <int > > adj(number_vertices);
    FOR(i,number_vertices){
        cin >> number_functions;
        FOR(j,number_functions) {
            cin >> function_number;
            adj[i].push_back(function_number);
        }
    }

    vector < int > result;

    FOR(i,number_vertices){
        vector < bool > visited(number_vertices,false);
        int vertex = bfs(adj,visited,i);
        if(vertex!=-1){
            result.push_back(vertex);
        }
    }

    sort(result.begin(),result.end());

    vector<int>::iterator i;
    for(i = result.begin(); i!= result.end(); i++){
        int t = *i;
        println(t);
    }
}
```

h_dfs.cpp

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

#define FOR(i,n) for(int i = 0; i < n; ++i)
#define FOR_ITERATOR(i,n) for(vector<int>::iterator i = adj[n].begin(); i!= adj[n].end();i+
+)
#define println(v) cout << v << endl

using namespace std;

int number_places = 0;
int counter = 0;

bool dfs(vector < vector <int> > &adj, vector < bool> &visited,  int u, int number_vertices
){
    visited[u] = true;
    counter++;
    FOR_ITERATOR(i,u){
        if (!visited[*i]){
            dfs(adj,visited,*i,number_vertices);
        }
    }
    return counter == number_vertices;
}

int trying(vector < int> &teste,int number_vertices){

    FOR(i,number_vertices){
        if(teste[i] == 0){
            return 0;
        }
    }
    return 1;

}

int main(){
    int number_vertices, number_edges;
    int edge_a, edge_b;

    cin >> number_vertices >> number_edges;

    vector< vector <int > > adj(number_vertices);
    vector< int > teste(number_vertices,0);

    FOR(i,number_edges){
        cin >> edge_a >> edge_b;
        adj[edge_b-1].push_back(edge_a-1);
        teste[edge_a-1] = 1;
        teste[edge_b-1] = 1;
    }

    if(trying(teste,number_vertices) == 0){
        println(0);
    }
    else{
        FOR(i,number_vertices){
            counter = 0;
            vector< bool > visited(number_vertices, false);
            if(dfs(adj,visited,i,number_vertices)){
                number_places++;
            }
        }
        println(number_places);
    }
}
```

i_bipartite.cpp

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

using namespace std;

#define FOR(i,n) for(int i = 0; i < n; ++i)
#define FOR_ITERATOR(i,n) for(vector<int>::iterator i = adj[n].begin(); i!= adj[n].end();i+
+)
#define println(v) cout << v << endl

bool bfs(int u, vector < vector < int > > &adj, vector < int > &color_array){

    queue<int> q;

    q.push(u);
    color_array[u] = 1;
    while (!q.empty()) {
        int f = q.front();
        q.pop();

        FOR_ITERATOR(i,f) {
            if (color_array[*i] == -1) {
                q.push(*i);
                color_array[*i] = 1 - color_array[f];
            }
            else if(color_array[*i] == color_array[f]){
                return false;
            }
        }
    }
    return true;
}

bool is_bipartite(int number_vertices, vector < vector < int > > &adj, vector < int > color
_array) {
    FOR(j, number_vertices) {
        if (color_array[j] == -1) {
            if (!bfs(j,adj,color_array)) {
                return false;
            }
        }
    }
    return true;
}

int main(){

    int number_vertices, number_edges, a, b;

    while( cin >> number_vertices >> number_edges){
        vector < vector < int > > adj(number_vertices);
        vector < int > color_array(number_vertices,-1);
        FOR(i,number_edges){
            cin >> a >> b;
            adj[a-1].push_back(b-1);
            adj[b-1].push_back(a-1);
        }
        if(is_bipartite(number_vertices,adj,color_array)){
            println("No");
        }else{
            println("Yes");
        }
        adj.clear();
    }
}
```

j_floyd_warshall.cpp

```
#include <bits/stdc++.h>
using namespace std;

#define FOR(i,n) for(int i = 0; i < n; ++i)
#define INF 0x3f3f3f3f
#define println(v) cout << v << endl

void floyd_warshall(vector< vector<int> > &graph, int number_vertices) {
    int i, j, k;

    for (i = 0; i < number_vertices; i++){
        for (j = 0; j < number_vertices; j++)
        {
            if(graph[i][j])
                graph[i][j] = graph[i][j];
            else if(i == j){
                graph[i][j] = 0;
            }
            else
                graph[i][j] = INF;
        }
    }

    for (k = 0; k < number_vertices; k++) {
        for (i = 0; i < number_vertices; i++) {
            for (j = 0; j < number_vertices; j++) {
                graph[i][j] = min(graph[i][j],graph[i][k]+graph[k][j]);
            }
        }
    }
/*
    FOR(l,26){
        FOR(h,26) {
            if(graph[l][h] != INF)
                printf("%d\t", graph[l][h]);
            else
                printf("INF\t");
        }
        printf("\n");
    }
    printf("\n\n");
    */
}

bool is_sub_path(vector < vector < int > > &graph1, vector < vector < int > > &graph2) {
    FOR(i,26) {
        FOR(j,26){
            if(graph1[i][j] != INF && graph2[i][j] == INF) {
                return false;
            }
            else if(graph1[i][j] == INF && graph2[i][j] != INF) {
                return false;
            }
        }
    }
    return true;
}

int main(){
    int number_test_cases, n_1,n_2;
    char v_a,v_b;

    cin >> number_test_cases;
    FOR(i,number_test_cases){

        cin >> n_1;
        vector< vector<int> > f1(26, vector<int>(26));
        FOR(j,n_1){
```

j_floyd_warshall.cpp

```
            cin >> v_a >> v_b;
            f1[v_a-65][v_b-65] =  1;
            /*printf("A entrada %c tem valor %d e liga ao %c com valor %d (total = %d)\n"
                    ,v_a,v_a,v_b,v_b,v_a+v_b);*/
        }

        cin >> n_2;
        //alloc memory for matrix
        vector< vector<int> > f2(26, vector<int>(26));
        FOR(j,n_2){
            cin >> v_a >> v_b;
            f2[v_a-65][v_b-65] =  1;
            /*printf("A entrada %c tem valor %d e liga ao %c com valor %d (total = %d)\n"
                    ,v_a,v_a,v_b,v_b,v_a+v_b);*/
        }
        floyd_warshall(f1,26);
        floyd_warshall(f2,26);
        if(is_sub_path(f1,f2))
        {
            println("YES");
            if(i!= number_test_cases-1)
                if(i!= number_test_cases-1)
                    println("");
        }
        else
        {
            println("NO");
            if(i!= number_test_cases-1)
                println("");
        }

    }

}
```

k_articulation_points.cpp

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

#define FOR(i, n) for(int i = 0; i < n; ++i)
#define println(v) cout << v << endl

using namespace std;

int number_critical_points;
int dfsTime = 0;

void articulation_points(int v, vector< vector < int > > &adj,vector<int> &dfs, vector <int
> &low, vector<int> &parent, vector <bool> &is_ap ) {
    dfsTime++;
    dfs[v] = low[v] = dfsTime;
    vector<int>::iterator i;
    for (i = adj[v].begin(); i != adj[v].end(); ++i) {
        int w = *i;
        if (dfs[w] == -1) {
            parent[w] = v;
            articulation_points(w,adj,dfs,low,parent,is_ap);
            low[v] = min(low[v], low[w]);
            if (dfs[v] == 1 && dfs[w] != 2) {
                if(!is_ap[v]){
                    is_ap[v] = true;
                    number_critical_points++;
                }
            }
            if (dfs[v] != 1 && low[w] >= dfs[v]) {
                if(!is_ap[v]){
                    is_ap[v] = true;
                    number_critical_points++;
                }
            }
        } else if (parent[v] != w) {
            low[v] = min(low[v], dfs[w]);
        }
    }
}

int main() {
    string line;
    int number_vertices;
    int tt;
    int vertex;

    while(getline(cin,line)){
        istringstream iss(line);

        iss >> number_vertices;
        vector < vector < int > > adj(number_vertices);
        vector < int > dfs(number_vertices,-1);
        vector < int > low(number_vertices);
        vector < int > parent(number_vertices);
        vector < bool > is_ap(number_vertices,false);
        number_critical_points = 0;
        dfsTime = 0;

        if(number_vertices == 0){
            break;
        }
        while(getline(cin,line)){
            istringstream iss(line);
            iss>>vertex;
            if(vertex == 0){
                break;
            }
            while (iss >> tt) {
                adj[vertex-1].push_back(tt-1);
```

k_articulation_points.cpp

```
                adj[tt-1].push_back(vertex-1);

            }
        }

        FOR(j,number_vertices){
            if (dfs[j]==-1){
                articulation_points(j,adj,dfs,low,parent,is_ap);
            }
        }
        println(number_critical_points);

    }
    return 0;
}
```

l_kruskal.cpp

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

#define FOR(i,n) for(int i = 0; i < n; ++i)

using namespace std;

double **adj;
vector < pair < int, int > > vertices;
vector < pair < double, pair < int, int > >  > edges;
int v_set[750], v_rank[750];

void make_set(int number_vertices) {
    FOR(i, number_vertices) {
        v_set[i] = i;
        v_rank[i] = 0;
    }
}

int find_set(int a) {
    if(v_set[a] != a) {
        v_set[a] = find_set(v_set[a]);
    }
    return v_set[a];
}

void link(int a, int b) {
    if (v_rank[a] > v_rank[b]) {
        v_set[b] = a;
    }
    else {
        v_set[a] = b;
        if(v_rank[a]==v_rank[b]) {
            v_rank[b]++;
        }
    }
}

void union_set(int a, int b) {
    link(find_set(a), find_set(b));
}



double calculate_distance(int x1, int x2, int y1, int y2){
    return (double) sqrt( fabs(x1 - x2) * fabs(x1 - x2) + fabs(y1 - y2) * fabs(y1 - y2));
}

double kruskal(int number_vertices){

    double minimum_path = 0;
    make_set(number_vertices);
    sort(edges.begin(),edges.end());

    for(int i=0; i < (int)edges.size(); i++){
        int u = edges[i].second.first;
        int v = edges[i].second.second;
        double w = edges[i].first;

        if(find_set(u)!= find_set(v)){
            union_set(u,v);
            minimum_path += w;
        }
    }
    return minimum_path;
}

int main(){
```

l_kruskal.cpp

```
    int number_vertices, number_connections, x, y, a,b;


    while(cin >> number_vertices){

        adj = (double**) malloc(number_vertices * sizeof(double*));
        for(int i = 0; i < number_vertices; ++i){
            adj[i] = (double*) malloc(number_vertices * sizeof(double));
        }

        for(int i = 0 ; i < number_vertices; ++i){
            cin >> x >> y;
            vertices.push_back(make_pair(x,y));
        }
        int x1, x2, y1, y2;
        for(int i=0; i < number_vertices; ++i){
            for(int j=i+1; j < number_vertices; ++j){

                x1 = vertices[i].first;
                y1 = vertices[i].second;
                x2 = vertices[j].first;
                y2 = vertices[j].second;
                double distance = calculate_distance(x1,x2,y1,y2);
                adj[i][j] = distance;
                adj[j][i] = distance;
            }
        }

        cin >> number_connections;

        for(int i=0 ; i < number_connections; ++i){
            cin >> a >> b;
            adj[a-1][b-1] = 0.0;
            adj[b-1][a-1] = 0.0;
        }

        for(int i=0; i < number_vertices; ++i){
            for(int j=i+1; j <number_vertices; ++j){
                if(i!=j){
                    edges.push_back(make_pair(adj[i][j],make_pair(i,j)));
                }
            }
        }

        printf("%.2lf\n",kruskal(number_vertices));

        for(int i = 0; i < number_vertices; ++i){
            free(adj[i]);
        }
        free(adj);
        vertices.clear();
        edges.clear();

    }
}
```

m_dfs.cpp

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

using namespace std;

#define FOR(i,n) for(int i=0; i < n; i++)
#define println(v) cout << v << endl;


int max_depth = 0;
int dfsTimer =0;
vector < vector < int > > adj;
vector < bool > visited;

void depth_first_search(int u){
    visited[u] = true;
    dfsTimer ++;
    if(dfsTimer > max_depth){
        max_depth = dfsTimer;
    }
    vector<int>::iterator i;
    for(i = adj[u].begin(); i!= adj[u].end();++i){
        if(!visited[*i])
            depth_first_search(*i);
    }
}

int main(){
    int number_vertices;
    int number_edges;
    string user_name, user1, user2;
    map<string,int> database;
    cin >> number_vertices >> number_edges;
    adj.resize(number_vertices);
    visited.resize(number_vertices,false);
    FOR(i,number_vertices){
        cin >> user_name;
        database[user_name] = i;
    }

    FOR(j,number_edges){
        cin >> user1 >> user2;
        adj[database[user1]].push_back(database[user2]);
        adj[database[user2]].push_back(database[user1]);

    }

    FOR(i,number_vertices){
        if(!visited[i]){
            dfsTimer =0;
            depth_first_search(i);
        }
    }

    println(max_depth);


}
```

n_floyd_warshall.cpp

```
#include <bits/stdc++.h>
using namespace std;

#define FOR(i,n) for(int i = 0; i < n; ++i)
#define INF 0x3f3f3f3f
#define println(v) cout << v << endl


void floyd_warshall(int **matrix, int number_vertices) {
    int i, j, k;

    for (k = 0; k < number_vertices; k++) {
        for (i = 0; i < number_vertices; i++) {
            for (j = 0; j < number_vertices; j++) {
                if (matrix[i][k] + matrix[k][j] < matrix[i][j])
                    matrix[i][j] = matrix[i][k] + matrix[k][j];
            }
        }
    }
}

int main(){
    int number_vertices, number_edges, number_values, a, b, cost, v_a,v_b;

    cin >> number_vertices;

    int **matrix = (int **)calloc(number_vertices,sizeof(int*));
    for(int i = 0; i < number_vertices; i++)
        matrix[i] = (int*)calloc(number_vertices,sizeof(int));



    cin >> number_edges;

    FOR(i,number_edges){
        cin >> a >> b >> cost;
        matrix[a-1][b-1] = cost;
    }

    for (int i = 0; i < number_vertices; i++){
        for (int j = 0; j < number_vertices; j++)
        {
            if(matrix[i][j])
                matrix[i][j] = matrix[i][j];
            else if(i == j){
                matrix[i][j] = 0;
            }
            else
                matrix[i][j] = INF;
        }
    }



    floyd_warshall(matrix, number_vertices);

    cin >> number_values;
    FOR(i, number_values){

        cin >> v_a >> v_b;
        if(matrix[v_a-1][v_b-1]!= INF)
            println(matrix[v_a-1][v_b-1]);
        else
            println("Impossible!");
    }

}
```

o_bellman_ford.cpp

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

using namespace std;


#define INF 0x3f3f3f3f
#define FOR(i,n) for(int i = 0; i < n; ++i)
#define FOR_ITERATOR(i,n) for(vector< pair < int, int > >::iterator i = adj[n].begin(); i!=
 adj[n].end();i++)
#define println(v) cout << v << endl


bool bellman_ford( vector< vector < pair<int, int> > > &adj,int number_vertices,vector< pai
r<int, int> > &shortestDistances) {

    FOR(i, number_vertices) {
        shortestDistances[i].first = INF;
        shortestDistances[i].second = -1;
    }
    shortestDistances[0].first = 0;
    shortestDistances[0].second = 0;

    FOR(i, number_vertices) {
        FOR(j, number_vertices) {

            FOR_ITERATOR(k, j) {
                if (k->second + shortestDistances[j].first < shortestDistances[k->first].fi
rst) {
                    shortestDistances[k->first].first = k->second + shortestDistances[j].fi
rst;
                    shortestDistances[k->first].second = j;
                }
            }
        }
    }

    FOR(j, number_vertices) {
        FOR_ITERATOR(k, j) {
            if (k->second + shortestDistances[j].first < shortestDistances[k->first].first)
 {
                return true;
            }
        }
    }
    return false;
}

int main(){
        int number_teste_cases, number_vertices, number_edges,a,b,value;

        cin >> number_teste_cases;

        FOR(i,number_teste_cases){
            cin >> number_vertices >> number_edges;
            vector < vector < pair < int, int >  > > adj(number_vertices);
            vector< pair<int, int> > shortestDistances(number_vertices);

            FOR(j,number_edges){
                cin >> a >> b >> value;
                adj[a].push_back(make_pair(b,value));
            }
            if(bellman_ford(adj,number_vertices,shortestDistances)){
                println("possible");
            } else{
                println("not possible");
            }
        }
}
```

p_dijkstra.cpp

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

using namespace std;

#define INF 0x3f3f3f3f
#define FOR(i,n) for(int i = 0; i < n; ++i)
#define println(v) cout << v << endl

int main(){
    int number_elevators, destination, elevator_travel_speed, floor;

    while(cin >> number_elevators >> destination){
        vector < int > time(number_elevators);
        vector < vector < int > > all_stops(number_elevators);
        vector < vector < pair < int, int > > > adj(number_elevators*100);
        FOR(i,number_elevators){
            cin >> elevator_travel_speed;
            time[i] = elevator_travel_speed;
        }
        FOR(i,number_elevators){
            string line;
            getline(cin, line);
            istringstream iss(line);

            for(int j = i ; j >= 0; --j){
                while (iss >> floor){
                    println(floor);
                    if(i==0){
                        all_stops[i-j].push_back(floor);
                    }else{
                        all_stops[i-j].push_back(floor+100*(j+1));
                    }

                }
            }
        }
        FOR(i,(int)all_stops.size()){

            FOR(j,(int)all_stops[i].size()){
                //println(j);
                //printf("o elevador %d estÃ¡ ligado aos pisos %d com valor %d\n",i,all_sto
ps[i][j],0);
            }

        }

    }
}
```

r_articulation_points.cpp

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

#define FOR(i, n) for(int i = 0; i < n; ++i)
#define println(v) cout << v << endl

using namespace std;

int max_number_associates = 0;
int dfsTime = 0;

void articulation_points(int v, vector< vector < int > > &adj,vector<int> &dfs, vector <int
> &low, vector<int> &parent, vector <bool> &is_ap) {
    dfsTime++;
    dfs[v] = low[v] = dfsTime;
    vector< int >::iterator i;
    for (i = adj[v].begin(); i != adj[v].end(); ++i) {
        int w = *i;
        if (dfs[w] == -1) {
            parent[w] = v;
            articulation_points(w,adj,dfs,low,parent,is_ap);
            low[v] = min(low[v], low[w]);
            if (dfs[v] == 1 && dfs[w] != 2) {
                if(!is_ap[v]){
                    is_ap[v] = true;
                    if((int) adj[v].size() > max_number_associates){
                        max_number_associates = (int)adj[v].size();
                    }
                }
            }
            if (dfs[v] != 1 && low[w] >= dfs[v]) {
                if(!is_ap[v]){
                    is_ap[v] = true;
                    if((int) adj[v].size() > max_number_associates){
                        max_number_associates = (int)adj[v].size();
                    }
                }
            }
        } else if (parent[v] != w) {
            low[v] = min(low[v], dfs[w]);
        }
    }
}

int main(){

    int number_vertices, number_associates,associate_id;

    while(cin >> number_vertices){
        dfsTime = 0;
        max_number_associates = 0;
        vector < vector < int  > > adj(number_vertices);
        vector < int > dfs(number_vertices,-1);
        vector < int > low(number_vertices);
        vector < int > parent(number_vertices);
        vector < bool > is_ap(number_vertices,false);
        FOR(i,number_vertices){
            cin >> number_associates;

            FOR(j,number_associates){
                cin >> associate_id;
                adj[i].push_back(associate_id-1);
                //adj[associate_id-1].push_back(make_pair(i,weight));
            }
        }
        FOR(j,number_vertices){
            if (dfs[j]==-1){
                articulation_points(j,adj,dfs,low,parent,is_ap);
            }
```

r_articulation_points.cpp

```
        }
        if(max_number_associates ==0){
            println("Strong network");
        }else{
            println(max_number_associates);
        }
    }


}
```

t_bfs_cycles.cpp

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

#define FOR(i,n) for(int i = 0; i < n; ++i)
#define FOR_ITERATOR(i,n) for(vector<int>::iterator i = adj[n].begin(); i!= adj[n].end();i+
+)
#define println(v) cout << v << endl
#define INF 0x3f3f3f3f

using namespace std;

int bfs(vector < vector < int > > &adj, vector < int > &depth, int u){

    queue<int> q;
    q.push(u);
    depth[u] = 1;
    while (!q.empty()) {
        int f = q.front();
        q.pop();
        FOR_ITERATOR(i,f) {

            if(depth[*i] == 0) {
                depth[*i] = depth[f] + 1;
                q.push(*i);
            }
            else if(*i == u){
                return depth[f];
            }
        }
    }
    return INF;
}

int main() {
    int number_vertices, value;
    cin >> number_vertices;
    vector < vector < int > > adj(number_vertices);

    FOR(i, number_vertices) {
        FOR(j, number_vertices) {
            cin >> value;
            if (value != 0) {
                adj[i].push_back(j);
            }
        }
    }
    int result = 0;
    int shortest_cycle_number = INF;
    FOR(i,number_vertices){

        vector<int> depth(number_vertices, 0);
        result = bfs(adj,depth,i);
        if(result < shortest_cycle_number)
            shortest_cycle_number = result;
    }
    if(shortest_cycle_number == INF){
        println("0");
    }
    else{
        println(shortest_cycle_number);
    }
    return 0;
}
```

u_bipartite.cpp

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

#define FOR(i,n) for(int i = 0; i < n; ++i)
#define FOR_ITERATOR(i,n) for(vector<int>::iterator i = adj[n].begin(); i!= adj[n].end();i+
+)
#define println(v) cout << v << endl

using namespace std;

vector < vector < int > > adj;

bool bfs(vector < int > &color_array, int u){

    queue<int> q;

    q.push(u);
    color_array[u] = 1;
    while (!q.empty()) {
        int f = q.front();
        q.pop();

        FOR_ITERATOR(i,f) {
            if (color_array[*i] == -1) {
                q.push(*i);
                color_array[*i] = 1 - color_array[f];
            }
            else if(color_array[*i] == color_array[f]){
                return false;
            }
        }
    }
    return true;
}

bool is_bipartite(vector < int > &color_array, int number_vertices) {
    FOR(j, number_vertices) {
        if (color_array[j] == -1) {
            if (!bfs(color_array, j)) {
                return false;
            }
        }
    }
    return true;
}

int main(){
    int number_vertices,value;
    cin >> number_vertices;
    adj.resize(number_vertices);
    vector < int > color_array(number_vertices,-1);
    FOR(i,number_vertices){
        FOR(j,number_vertices){
            cin >> value;
            if(value!=0){
                adj[i].push_back(j);
            }
        }
    }

    if(is_bipartite(color_array,number_vertices))
        println("True");
    else
        println("False");
}
```

v_floyd_warshall.cpp

```
#include <bits/stdc++.h>
using namespace std;

#define FOR(i,n) for(int i = 0; i < n; ++i)
#define INF 0x3f3f3f3f
#define println(v) cout << v << endl
int number_vertices;

void floyd_warshall(int **matrix) {
    int i, j, k;

    for (k = 0; k < number_vertices; k++) {
        for (i = 0; i < number_vertices; i++) {
            for (j = 0; j < number_vertices; j++) {
                if (matrix[i][k] + matrix[k][j] < matrix[i][j])
                    matrix[i][j] = matrix[i][k] + matrix[k][j];
            }
        }
    }

}

int main(){
    int value, a,b;

    cin >> number_vertices;

    int **matrix = (int **)malloc(number_vertices * sizeof(int *));
    for (int i=0; i<number_vertices; i++)
        matrix[i] = (int *)malloc(number_vertices * sizeof(int));

    FOR(i,number_vertices){
        FOR(j,number_vertices){
            cin >> value;
            if(value == -1){
                matrix[i][j] = INF;
            }else{
                matrix[i][j] = value;
            }
        }
    }

    floyd_warshall(matrix);
    while(cin >> a >> b){
        println(matrix[a-1][b-1]);
    }
}
```

w_articulation_points.cpp

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

#define FOR(i, n) for(int i = 0; i < n; ++i)
#define println(v) cout << v << endl

using namespace std;

int total_weight = 0;
int dfsTime = 0;

void articulation_points(int v, vector< vector < pair < int, int > > > &adj,vector<int> &df
s, vector <int> &low, vector<int> &parent, vector <bool> &is_ap) {
    dfsTime++;
    dfs[v] = low[v] = dfsTime;
    vector<pair <int, int> >::iterator i;
    for (i = adj[v].begin(); i != adj[v].end(); ++i) {
        int w = i->first;
        int weight = 0;
        if (dfs[w] == -1) {
            parent[w] = v;
            articulation_points(w,adj,dfs,low,parent,is_ap);
            low[v] = min(low[v], low[w]);
            if (dfs[v] == 1 && dfs[w] != 2) {
                if(!is_ap[v]){
                    is_ap[v] = true;
                    for(int k = 0;  k < (int)adj[v].size(); k++){
                        weight += adj[v][k].second;
                    }
                    if(weight > total_weight){
                        total_weight = weight;
                    }
                }
            }
            if (dfs[v] != 1 && low[w] >= dfs[v]) {
                if(!is_ap[v]){
                    is_ap[v] = true;
                    for(int k = 0; k < (int)adj[v].size(); k++){
                        weight += adj[v][k].second;
                    }
                    if(weight > total_weight){
                        total_weight= weight;
                    }
                }
            }
        } else if (parent[v] != w) {
            low[v] = min(low[v], dfs[w]);
        }
    }
}

int main(){

    int number_vertices, number_crossings,weight,destination;

    while(cin >> number_vertices){
        dfsTime = 0;
        total_weight = 0;
        vector < vector < pair < int, int > > > adj(number_vertices);
        vector < int > dfs(number_vertices,-1);
        vector < int > low(number_vertices);
        vector < int > parent(number_vertices);
        vector < bool > is_ap(number_vertices,false);
        FOR(i,number_vertices){
            cin >> number_crossings;

            FOR(j,number_crossings){
                cin >> destination >> weight;
                adj[i].push_back(make_pair(destination-1,weight));
```

w_articulation_points.cpp

```
                //adj[destination-1].push_back(make_pair(i,weight));
            }
        }
        FOR(j,number_vertices){
            if (dfs[j]==-1){
                articulation_points(j,adj,dfs,low,parent,is_ap);
            }
        }
        if(total_weight ==0){
            println("Well designed city!");
        }else{
            println(total_weight);
        }
    }


}
```

x_disjoint_set.cpp

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

using namespace std;

#include <bits/stdc++.h>

using namespace std;

#define FOR(i,n) for(int i = 0; i < n; ++i)
#define FOR_ITERATOR(i,n) for(vector<int>::iterator i = adj[n].begin(); i!= adj[n].end();i+
+)
#define println(v) cout << v << endl

double distance(int x1, int x2, int y1, int y2){
    return sqrt( pow(x1-x2,2) + pow(y1-y2,2));
}

int main(){
    int number_tests, number_swarms, number_robots,number_technicians, w,z;
    int x_tech, x_robot, y_tech, y_robot;

    FOR(i,number_tests){
        vector < pair < int, int > > tech_location(number_technicians);
        vector < vector < pair <  int, int > > > swarm_location(number_swarms);
        cin >> number_swarms >> number_technicians >> w >> z;

        FOR(j,number_technicians){
            cin >> x_tech >> y_tech;
            tech_location.push_back(make_pair(x_tech,y_tech));
        }
        FOR(j,number_swarms){
            cin >> number_robots;
            FOR(k,number_robots){
                cin >> x_robot >> y_robot;
                swarm_location[j].push_back(make_pair(x_robot,y_robot));
            }
        }

    }
}
```