# CSE 4304:Data Structure Lab-08
# Graph Basic

Namisa Najah Raisa 210042112

November 2023

## 1 Problem A:Back to underworld

This problem is asking to find out the maximum number of whichever raced individual that participated in a duel. As input it takes information of the two individuals but not their races.So for each test case I have to find out the maximum count of an individual.

```cpp
#include <iostream>
#include <vector>
using namespace std;

vector<int> g[20009];
bool vis[20009];
int xx, yy;

void dfs(int source, bool p)
{
    if (vis[source]) return;
    vis[source] = true;
    if (p)
        xx++;
    else
        yy++;
    int sz = g[source].size();
    for (int i = 0; i < sz; i++)
        dfs(g[source][i], !p);
}

int main()
{
    int t, n, a, b;
    cin >> t;
```

```
    for (int i = 1; i <= t; i++)
    {
        for (int j = 0; j < 20009; j++)
            g[j].clear();

        for (int j = 0; j < 20009; j++)
            vis[j] = false;

        cin >> n;

        for (int j = 1; j <= n; j++)
        {
            cin >> a >> b;
            g[a].push_back(b);
            g[b].push_back(a);
        }

        int ans = 0;
        for (int j = 1; j < 20009; j++)
        {
            if (!vis[j] && !g[j].empty())
            {
                xx = yy = 0;
                dfs(j, true);
                ans += max(xx, yy);
            }
        }
        cout << "Case " << i << ": " << ans << endl;
    }
    return 0;
}
```

I solved this using Depth-First Search to traverse the graph.

Here g[] is an array of vectors which is used to represent an adjacency list of a graph.Each g[i] is a vector containing neighbours of vertex 'i'. bool vis[] declares an array booleans.It marks whether a vertex was visited during DFS por not.The 'xx' and "yy' variables are for counting the number of members for the two different races.

The **'void dfs(int source,bool p)'** takes two parameters.'source'—>The vertex from which the traversal will start.'p'—>A boolean variable to keep track of the current race(vampire or lykan)being assigned to the members during the traversal.

**'if(vis[source]) return'**—>if vis[source] is true,it means that vertex has been visited and it returns early.This is to avoid infinite loops in the DFS.

**'vis[source]=true'**—>this marks the current 'source' vertex as visited so that it's not visited again during DFS.

**'if(p)xx++;else yy++'** —> refers that if p is true then it is one of the races and it increments that member. If it's not true then the other member gets incremented.

**'int sz=g[source].size()'**—>counts the number of neighbours of the current 'source' vertex by finding out the number of elements of 'g[source]' which contains the neighbours of the 'source' vertex.

The FOR loop after that iterates through all the neighbours of the current 'source' vertex.For each neighbour it calls the dfs() function recursively,passing the neighbour as the new 'source' and switching the 'p' by '!p'.It means that if the current vertex is associated with one race (vampire), the neighbor will be associated with the other race (lykan), ensuring that the two races alternate as the DFS progresses through the graph.

In the main function t,n,a,b refers to the number of test cases,number of dual fights,and the pairs of individuals involved in dual fights respectively.

The first FOR loop iterates for the number of test cases.

**'for (int j =0; j < 20009; j++) g[j].clear();'**—> This loop clears the adjacency list represented by the g array for the current test case. It ensures that the graph is reset before working for each new test case.

**'for (int j = 0; j < 20009; j++) vis[j] = false;'**—> This loop initializes the vis array, marking all vertices as unvisited for the current test case. This is important to ensure that the DFS traversal starts from an unvisited vertex.

**'for (int j = 1; j <= n; j++)'**—> This is a loop that iterates through each dual fight for the current test case. It starts from j = 1 and continues until j reaches the value of n.Takes a,b inputs.

**'g[a].**`push_back(b)`**;g[b].push**`push_back(a)`**'**→These lines add the individuals a and b to each other's list of neighbors in the adjacency list g. This is done because dual fights are represented as edges in an undirected graph, and these lines ensure that the graph reflects the connections between individuals involved in the fights.

**'int ans = 0;'**→ This line initializes the ans variable to zero.This will store the maximum possible number of members for one of the races within a

connected component of the graph.

**'for (int j = 1; j < 20009; j++)'**→ This is a loop that iterates through each vertex in the range from 1 to the size of the array. It aims to find connected components and calculate the maximum possible number of members for one of the races in each connected component.

**'if (!vis[j] && !g[j].empty())'**→ This conditional checks if the vertex j has not been visited (!vis[j]) and if it has at least one neighbor, meaning it is part of a connected component (!g[j].empty()). If these conditions are met, it means a new connected component is encountered.

In that IF condition **'xx = yy = 0;dfs(j, true);ans+=max(xx, yy);'**→initializes xx(vampire) and yy(lykan) to zero.Then calls the dfs() function to start the traversal from 'j'. The 'true' indicates that the initial race is vampire.After the DFS traversal 'ans' calculates the maximum possible number of members.

At last it just prints the output.

_____X_____