**Design Document for PS16 - [Friend Circle]**

**Data Structure being used:**

We are using Graph data structure with adjacency list representation to solve the given problem.

Considering an undirected graph where the students are the nodes/vertices in the graph, friendship between the two students are edges.

**Problem Solving Approach:**

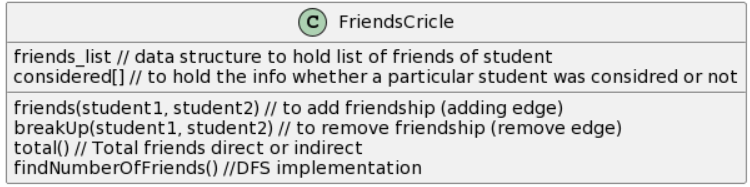
The task at hand is to determine the number of students a particular student 0 can speak to.

Student 0 is friends with Student 1. Student 1 can speak to his direct friends, also with all the friends of his friends.

We are supposed find all the nodes that node 1 is connected to, be it directly or via his/her friends.

Using Depth First Search (DFS) algorithm to traverse through the graph to find the required total number of students that student 0 can speak to.

**Implementation details:**



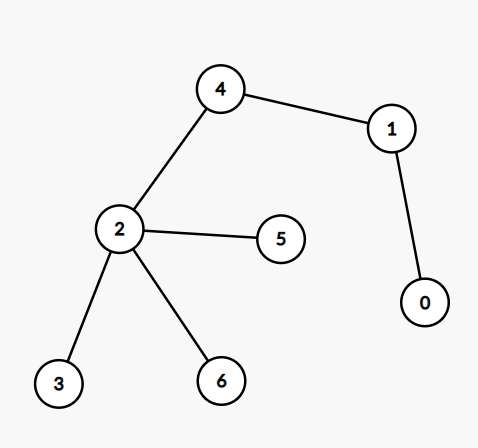
We have a class FriendsCircle. A List students to hold all the student Ids, a data structure friends\_list which holds data of the students and their corresponding friends. A list name considered which holds Boolean values to indicate if the student was considered in count of friends or not. Parsing the input file happens ad part of the main function.

Visualizing the implementation:

1. Adding Friendships between nodes.

friends 0 1, friends 2 3, friends 2 4, friends 2 5, friends 2 6, friends 1 4

The graph would look like:



From the graph in the picture above we could say that student can speak to 6 other students.

1. Total Number of students student 0 can talk can be determined by traversing through the graph.

Put in graph terms we could say that student 0 can talk to all the students who he/she can connect directly or indirectly.

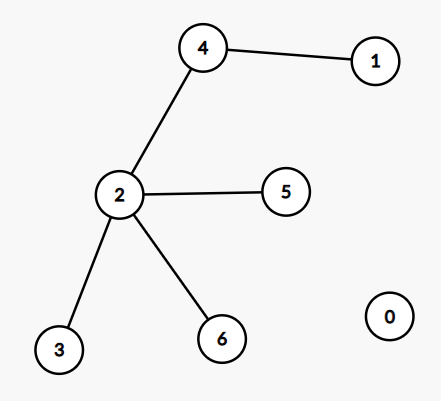
From the above graph we could see student 0 is directly connected to only student 1 but he can still talk to others because they are all connected to student 1.

The traversal approach used in the solution is Depth First Search (DFS).

1. Breaking up friends:

Breakup 0 1

This would lead to removal of link between the students 0 and 1



Post this operation the number of students student 0 can talk to is 0.

**Details Of Operations**:

1. Friends: Operation to befriend 2 students.

In terms of graph, it is equivalent to adding an edge.

The complexity of this operation would be O(n).

1. breakUp: Operation to remove the friendship between 2 students.

In terms of graph, it is equivalent to deleting an edge.

The Complexity of this operation would be O(n).

1. total: Operation to find the total number of students student0 can speak to.

In terms of graph is to traverse the graph to find the total connections of a node.

Performing the DFS:

The complexity of this operation would be O(e) where e is the number of edges(friendships)

Resetting all the considered array entries to false once the calculation is over:

The complexity of this operation would be O(n) where n is the number of students.

So Total Time Complexity of this operation would be O(n + e) where n is number of students and e is number of friendships

1. Adding up the complexities of operations O(n) + O(n) + O(n+e) = O(n + e)

We also have outer loop to parse through the input file with i number of lines.

Therefore, Overall complexity of the program would O(i(n+e)) where i is number of lines in input file, n is number of students and e is number of edges(friendships).

**Alternate way of modelling the problem:**

1. Adjacency list implementation with list of lists:

The number of nodes should be predetermined in this case.

We could use list of list to hold the student and their corresponding friends list but that would impose input restrictions. Like the student ids must be contagious in manner. If the inputs are not contagious then we end up creating list with number of elements equal to the largest entered student id. All the array entries may not be utilised then.

Also, it is difficult to manage this approach with increase in the array bounds.

1. Adjacency Matrix representation of graph:

We could have used the adjacency matrix instead of adjacency list to represent the graph nodes and edges.

Adjacency matrices need n \* n data types.

If the number of edges in graph are less than the memory allocated to most of nodes is marked false or 0. It turns out to be a sparse matrix. Instead of having such huge data types we could simply save the list of adjacent nodes in the adjacency lists.