# Broadcasting a message within a Social Network

## Overview

This project investigates the problem of making an announcement which I want to reach everybody within a social network. How many people do I need to announce that? I just want to find a set of people to essentially announce to the social network that something's true, and everyone will hear it. Ideally I would like to find the minimum set of people.

## Data

First I will use some tiny data sample in order to test the algorithm and check if it will be working fine, afterward I will test the algorithm with a huge data sample, the provided USCD Facebook data. The Facebook data represents friendships between students at USCD in 2005. Every line in the file specifies a particular friendship relationship between two users, just numbers, which are the user IDs.
So, for example, the first line in this file says that user 0 is friends with user 22. It would be an undirected graph, because the friendship has to exist in both directions, but the algorithm will create a directed graph.

## Questions

I want to find the group of people highly connected, so with this group I can reach everyone in the network. This will be a dominating set, afterward I would like to find the mínimum group of people in order to reach everyone in the network "The mínimum dominating set in the network".

**Easier**. I want to find the group of people highly connected, so with this group I can reach everyone in the network. This will be the dominating set.

**Harder**. I want to know if the group of people I found in the easier question is the minimum group of people in order to reach the whole network. This will be the mínimum dominating set.

## Algorithms, Data Structures and answer to your question

### Data Structures

Main Data Structure: The network has been laid out as a classic graph using an adjacency list. Each individual in the graph is a vertex and an edge between vertices represents a relationship. In order to represent the Graph I use a map, each entry in the map is a vertex(key) and the value is a list with its neighbors.

**Easier Question.** I tried a greedy algorithm that is described in (Johnson, 1973). Basically, at each stage, the greedy algorithm chooses the set which contains the largest number of uncovered elements. Starting from an empty set, if the current sub-set of vertices is not the dominating set, a new vertex which has the most number of the ad-jacent vertices that are not adjacent to any vertex in the current set will be added.

The greedy algorithm would work as follow:

- Mark all vértices as “uncovered”
- Initialize an empty dominant set
- While there are uncovered vértices:
    - Find the vertex, v, which would cover the most uncovered vértices
    - Add v to the dominant set
    - Mark that vertex and all of its uncovered neighbors as covered

**Harder Question.** Finding a minimum set seems like it might be problematic. The algorithm depicted in the easier question is an approximation algorithm for the Set Cover problem. We will always find out the dominating set and sometimes the mínimum dominating set, but not always. I will tried the “Big Step Greedy Set Cover Algorithm” depicted in: https://arxiv.org/pdf/1506.04220.pdf

The Big step greedy set cover algorithm starts with empty set cover, in each step selects p(in my case p == 2) sets from F such that the union of selected p sets contains greatest number of uncovered elements and adds the selected p sets to partial set cover. The process of adding p sets is repeated until all elements of X are covered by partial set cover. In the last step it may add less than p sets to avoid redundant sets .

# Algorithm Analysis

**Easier Question**. The Greedy approximate algorithm takes a polynomial time, the greedy algorithm provides a Log(n) approximate algorithm (Source: http://math.mit.edu/~goemans/18434S06/setcover-tamara.pdf). In my particular example:

Getting a set with all the vértices in the graph: O(1)

The algorithm iterates over the set of vértices. In the first iteration it takes the node that has more neighbors, adds this node to the dominant set, and delete from the set of vértices the node and its neighbors. The algorithm repeats the process until the set of vértices is empty.

The algorithm takes O(Log(n)) time complexity.

**Harder Question.** The Big step greedy set cover algorithm is an improvement of the Greedy approximate algorithm, but if step size p(in my particular case p==2) is small enough Bigstep greedy set cover algorithm runs in polynomial time. With (p) bigger than 2 can lead to a no polynomial time due to combinatorial explosion.

# Correctness verification

In order to test my implementation of the algorithm I developed a JUnit class to test the basic cases, toy examples of graphs with a few vértices and edges, and in all cases it worked as I expected.

I developed a static main method in order to check the behavior of the algorithm with a big data example (Facebook data).

# Reflection

The Greedy algorithm worked in each case as I expected, it is a very well known and tested algorithm. In some cases it found out the mínimum dominant set, but in other cases only a dominant set.

Experimental results show that proposed Big step greedy set cover algorithm with p=2 computes smaller set cover than the set cove rcomputed by the classical greedy algorithm in many cases. When step size p is small enough Bigstep greedy set cover algorithm runs in polynomia ltime. Big step greedy set cover algorithm is preferable than the classical greedy algorithm in scenarios where small improvement in the solution is valuable and some increment in running time isacceptable. The proposed Big step greedy method can be used for other combinatorial optimization problems.