

# CS51 Assignment 2: Faculty Cliques

Due: Friday, February 20th, 2009, 5:00PM

*Total Points: 40 (including 5 style points)*

## 1 Style

On this and most future problem sets (unless otherwise specified), we will be grading style on a more holistic 5-point scale. Rather than deducting a certain number of style points per questionable stylistic choice, we will make style comments throughout your problem set and then decide at the end how good it was overall. This will make it easier to ensure that we aren't docking you multiple times for the same type of style error, since after grading the whole problem set we will have a better sense of how many unique style errors there were. The grading is (roughly) as follows:

1 point: No attention paid to style.

2 points: Many different style mistakes.

3 points: A few different style mistakes.

4 points: Very good style.

5 points: Perfect style. This will be hard to get.

E-mail the Head TFs if you have any questions about the new system.

## 2 Graphs

*(30 points total)*

In the last problem set we focused on trees. We now turn our attention to a different data structure: the *graph*.

As discussed in lecture, a graph is built up from nodes (also referred to as vertices or points) and edges (often represented as lines or arrows) that connect these nodes. A graph can represent many different relationship topologies, ranging from cycles to trees. These abstract concepts from mathematics and computer science have many concrete applications. In CS51 we use graphs as a means for modeling human relationships, but they are also used

in other contexts including modeling nerve cells in biology and understanding the impact of air-traffic control choices.

**Exercise 1.**

7 points

Let's warm up by defining a Scheme representation of a directed graph. One way of representing a graph in Scheme would be to store a list of edges, where each edge is simply a pair of nodes, the first representing the source of an edge and the second representing the destination of the edge. In scheme this looks like:

```
(define-struct edge (source dest))
```

Each expression typed into the DrScheme *REPL* (i.e., read-eval-print-loop) contains a blank (\_\_\_\_). What could go in the blanks to produce the results shown?

(a) [2 points]

```
(edge? ____)  
true
```

(b) [2 points]

```
(not (edge? ____))  
true
```

(c) [2 points]

```
(____ (make-edge "Greg" "Ramin"))  
"Ramin"
```

(d) [1 point]

```
____  
"I_<3_CS51"
```

**Exercise 2.**

23 points

(a) [3 points] What if we want to represent an undirected graph with this struct? Well, one simple approach would be to use our `make-edge` twice for every edge in the undirected graph. If `a` and `b` are connected by an edge in a simple, undirected graph, then we would represent it with the following scheme:

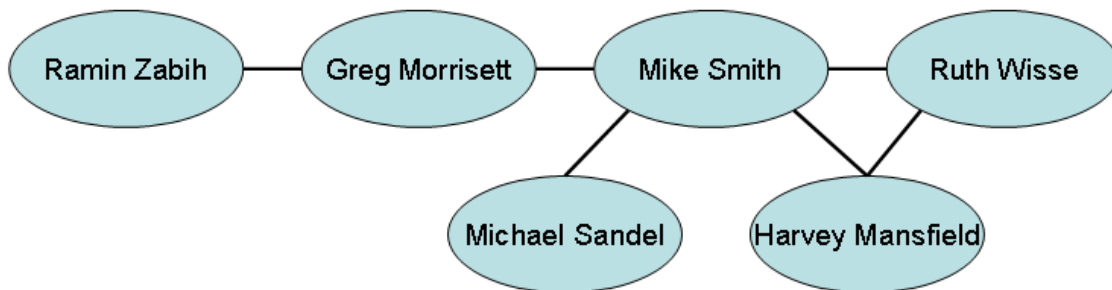
```
(list (make-edge "a" "b") (make-edge "b" "a"))
```

For large graphs, manually doubling each edge can be cumbersome. Write a function, `undirect-edge`, that takes an edge and returns a list containing the provided edge and the corresponding edge pointing in the opposite direction.

For example:

```
> (undirect-edge (make-edge "a" "b"))
(list (make-edge "a" "b") (make-edge "b" "a"))
```

(b) [2 points] Represent the following undirected graph in Scheme as a list of edges and bind the variable `faculty` to it. You must use `undirect-edge` in your solution. Make sure that your answer produces a list of edges and not a list of lists of edges:



(c) [8 points]

In the previous part we exposed the true inner workings of the faculty and found that they are strikingly similar to the relationships you would find in an elementary school! Like groupings of people, graphs can contain one or more cliques - a group of nodes that share connections to every other node in the group.

Implement `clique?`, a function that takes a graph, as represented by a list of `edge` structures, and a list of nodes and returns `true` if edges exist in both directions between every pair of nodes in the list and `false` otherwise. Note, you will likely find it extremely helpful to write one or more helper function in defining `clique?`.

(d) [10 points] Clique-checking is all well and good. But what about clique-finding? We'd like to be able to find all groups of three faculty members who form a triangle of friends.

Please write a function `find-triangles` that takes as its only input a graph as we've been representing them in this problem set, and returns a list of all 3-cliques (groups of 3 nodes all connected to each other). You need not deal with cliques of larger sizes.

### 3 Testing

(4 points total)

#### Exercise 3.

4 points

Here are a couple of tests. You'll want to also write some of your own! There are 4 points on this assignment that will be given based on the thoroughness of your test cases.

## 4 Finishing Up

*(1 point total)*

### Exercise 4.

*1 point*

Please tell us how much time you spent on this problem set! You may use this space to add any thoughts or questions you want to send to your TF as well, if you so desire.