CMSC 15100 Introduction to Computer Science I

Homework 11

Your answers should be saved in a .rkt file and submitted via Canvas before the start of the next lecture. At the top of the file for this assignment and all future assignments include a comment,

```
;; Homework 11
;;
;; your name
;; your CNET id
;; your collaborators
```

If your code doesn't compile, it may not be graded. Please double check to make sure your code runs before submitting!

Problems

Before you start this assignment, copy in the type of a `Graph` from lecture,

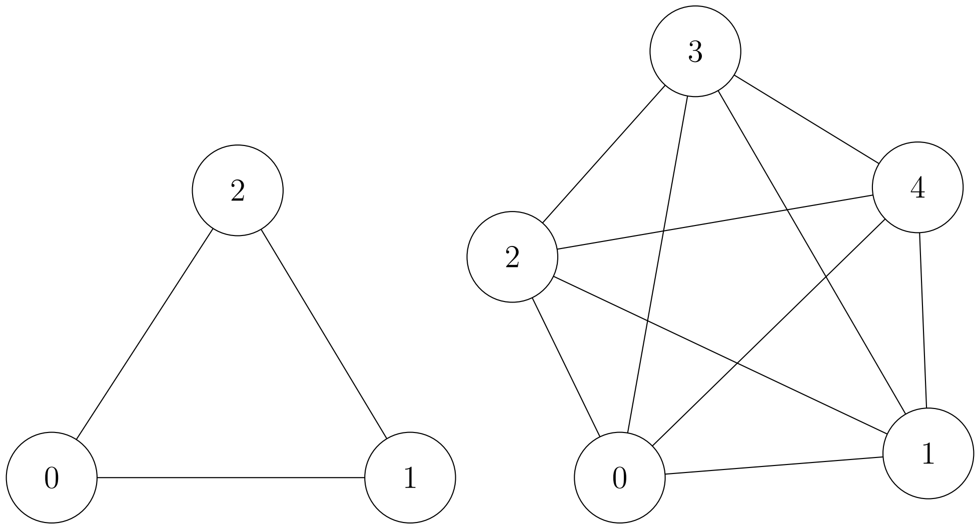
```
(define-type Vertex Integer)
(define-struct Graph
  ([n : Integer]
   [adj : (Vectorof (Listof Vertex))]))
```

You can also find and use any functions from lecture on Piazza. You can assume that every input `Graph` is simple and undirected.

1. (11 points)

   (a) Implement the function `count-degree` (Hint: Use vector-count),

   ```
   (: count-degree : Graph Integer -> Integer)
   ;; on input g and d, outputs the number of vertices in g with degree d
   ```



(b) A clique is a graph where every two vertices are adjacent. For example, the previous page shows two cliques with three and five vertices respectively.

Make a function `clique?` that checks whether or not a graph is a clique.

(c) In terms of n and m, what is the runtime of your functions `count-degree` and `clique?`? Explain your answers.

2. (7 points) Make a function that adds a vertex to a `Graph` that is connected to every other vertex. The type of this function should be

```
(: add-vertex : Graph -> Graph)
```

To keep the graph undirected, make sure to add the new vertex to the adjacency lists of every other vertex currently in the graph.(Hint use: vector-map and build-list)

To test this problem, define a variable for a `Graph`, define a variable for the result of `add-vertex`, then run `check-expect` on the result of `adjacent?` on a couple of pairs of vertices. The code for `adjacent?` can be found on Piazza.

3. (13 points) RaCalendar (the calendar app from Homework 4) is improving its service! Some very busy users were reporting that the app was slow, so RaCalendar is upgrading its underlying data structure from a list to a binary search tree. Furthermore, RaCalendar wants to be more flexible than just using `day-of-week` by upgrading this field to `day-of-year`, a number between 1 and 365 representing which day of the year the `Activity` occurs on (1 for January 1 and 365 for December 31). The new definition of an `Activity` is

```
(define-struct Activity
  ([desc : String]
   [day-of-year : Integer]
   [loc : String]))
```

The binary search tree will be sorted by `day-of-year`.

(a) Create a type definition for `Calendar` and define a comparison function for comparing `Activitys` that can be used as part of a binary search tree.

(b) Create a `Calendar` variable with at least 5,000 scheduled events in it, and make sure that most of the events are not the same. Your code for this problem should not take more than 100 lines.

There are many ways to accomplish this task. One approach uses `random` to choose a random location, description, and day. The code `(random n)` produces a random number from 0, 1, 2, ..., $n-1$. Keep in mind Calendar is not a list in this problem but a BST.

(c) Make a function `go-on-vacation` which takes in a `Calendar`, a start date, and an end date, and removes all the events that occur between the start date and the end date (both inclusive).

If the start date is after the end date, raise an error.