# Computational Problem Solving CSCI-603
# Holi Cow! Lab 9


## 1 Introduction

*Holi* is an annual celebration of color held in India. Reema, a famous artist, is planning to unveil her *pièce de résistance*, entitled *Field of Dreams*, at the festival. A local farmer has agreed to lend Reema a plot of land on which she plans to create her masterpiece. Scattered across the field are the farmer's prized cows who are all blissfully slumbering.

Reema has brought with her a variety of paint balls of various sizes and colors that she has already placed at random spots in the field. These paint balls are very large and fragile. Whenever one pops, it splatters paint emanating from its position, forming a colorful circle of a specific radius. If there are other paint balls within the radius, these paint balls also pop, which might set off other paint balls, and so forth.

Reema would like to create the most colorful scene possible, by popping just a single paint ball. Of course, she wants to maximize the amount of paint that lands on the sleeping cows. That is, for each paint ball, maximize:

$$\sum_{all\ cows} \#\ of\ all\ paint\ bombs\ that\ painted\ the\ cow$$

## 2 Implementation

### 2.1 Design

You may design your solution as you wish. However, make sure your solution is object-oriented and breaks the problem down between various classes and methods. Minimally, you should have a class to reresent the graph, another to represent the vertices as well as a class that runs the simulation.

You are free (and encouraged) to use the graph code from the lecture code. You can modify any of the code provided to suit your needs.

### 2.2 Main Program

The main program should be named `holicow.py`. The file of cows and paint balls is provided to it on the command line. The format will be identical to that in the problem solving. If it is not present, you should print the usage statement and exit:

```
Usage: python3 holicow.py {filename}
```

If the file name is provided, but does not exist, you should print the following message and exit:

```
File not found: {filename}
```

If the file exists, its contents are guaranteed to all be valid.

### 2.3 Output

There are three main parts to the output that your program must produce:

1. *Displaying the field.* Once the input file is read and the graph is built, the program must displayed the following information:
   - The total number of vertices in the graph, the number of cows and the number of paint balls.

     ```
     Number of vertices: #, cows: #, paint balls: #
     ```
   - The entire field as an adjacency list where each vertex indicates what neighboring vertices it is connected to. A typical line would follow the format (where `Vertex` is either the name of the cow, or the color of the paint ball):

     ```
     {Vertex} neighbors: [{Vertex}, {Vertex}, ...]
     ```

2. *Trigger each starting paint ball in any order.* The trigger message should be:

   ```
   Triggering {COLOR} paint ball...
   ```

   The chain reaction of triggering other paint balls should use a message that is tabbed in:

   ```
   {COLOR} paint ball is triggered by {COLOR} paint ball
   ```

   If a cow is painted by a triggered paint ball, the message should also be tabbed in:

   ```
   {COW} is painted {COLOR}!
   ```

3. *Displaying the optimal result.* After all starting paintballs have been triggered, you should display which starting paint ball color painted the cows the most. In the case of a tie, choose one. The message should be:

`Triggering the RED paint ball is the best choice with 4 total paint on the cows:`

After this, the cows that were painted by the best paint ball should be displayed, by name, one per line, along with the color/s they were painted, each tabbed in e.g.:

`{Cow} colors: {{Color1}, {Color2}, ...}`

In the case where no cows are painted, display the following message instead:

`No cows were painted by any starting paint ball!`

### 2.4 Sample Run

This is a sample run using the input from problem solving:

```
Field of Dreams
Number of vertices: 8, cows: 4, paint balls: 4
-----------------------------------------------------
Daisy neighbors: []
BLUE neighbors: ['Babe']
Fauntleroy neighbors: []
RED neighbors: ['Daisy', 'Milka', 'GREEN']
YELLOW neighbors: ['Fauntleroy']
Babe neighbors: []
Milka neighbors: []
GREEN neighbors: ['BLUE', 'Milka']

Beginning simulation...
Triggering BLUE paint ball...
        Babe is painted BLUE!
Triggering GREEN paint ball...
        BLUE paint ball is triggered by GREEN paint ball
        Milka is painted GREEN!
        Babe is painted BLUE!
Triggering YELLOW paint ball...
        Fauntleroy is painted YELLOW!
Triggering RED paint ball...
        Daisy is painted RED!
        Milka is painted RED!
        GREEN paint ball is triggered by RED paint ball
        BLUE paint ball is triggered by GREEN paint ball
        Milka is painted GREEN!
        Babe is painted BLUE!

Results:
Triggering the RED paint ball is the best choice with 4 total paint on the cows:
        Daisy's colors: {'RED'}
        Fauntleroy's colors: {}
        Babe's colors: {'BLUE'}
        Milka's colors: {'RED', 'GREEN'}
```

### 2.5 Testing

You must construct at least 3 non-trivial test cases and store them in files named `test1.txt`, `test2.txt` and `test3.txt`. Also include a file `test-description.txt` that explains what each of your test cases is supposed to test.

## 3 Grading

This assignment will be graded using the following rubric:

- 20%: Problem Solving
- 10% Design: The solution is object oriented and breaks the problem down between various classes and methods.
- 55% Functionality:
  - 20% Graph is built correctly (using the adjacency list output).
  - 20% Starting paint balls are triggered and the sequence and paintings are valid.
  - 10% Optimal results, if present, are correct.
  - 5% Error handling: (e.g. file not found, no cows painted)
- 10% Testing: Provided a good range of test cases with descriptions.
- 5% Style: Proper commenting of modules, classes and methods (e.g. using docstring's).

Good code design is a significant graded component. Your program should not be just a "monolitic" main function. The main program should be very small and pass control into classes/methods. You should have things like a method to read the graph file in and build the graph, a method to run the main simulation, etc.

Also, it is very hard to award functionality points if the graph is not built correctly.

## 4 Submission

Create a ZIP file named Lab9.zip that contains all your source code, your test files (test1.txt, test2.txt, test3.txt, and test-descriptions.txt). Submit the ZIP file to the MyCourses assignment before the due date (if you submit another format, such as 7-Zip, WinRAR, or Tar, you will not receive credit for this lab).