CS100R Assignment 3

out: Sep 22nd, 2007
due: **Section 4: 1-3, Sep 28th, 2007**
due: **Section 4: 4-7, Oct 05th, 2007**

1 Introduction

In the last assignment we made the robots follow the moving light stick. This assignment will be similar, but we will use better algorithms to find the lightstick's center, shape and orientation. In particular, we will look at:

- Finding the largest region of connected red pixels, using connected components, and
- Finding the center of this region, using modified quicksort.

2 Quicksort

Quicksort involves the following steps:

- Pick an element as the pivot value, x.
- Partition the array into two groups: elements $\leq x$ and elements $> x$.
- Recurse: sort these two smaller arrays separately, then join them together.

Our **modified quicksort**, as discussed in lecture, can find the k^{th} largest element in an array. This involves the following steps:

- Pick an element in array A as the pivot, call it x.
- Divide A into A_1 (elements $< x$), A_2 (elements $= x$), A_3 (elements $> x$).
- If $k \leq \texttt{length}(A_3)$:
  - Recurse: find the k^{th} largest element in A_3.
- If $k > \texttt{length}(A_2) + \texttt{length}(A_3)$:
  - Find, in A_1, the right-most element.
  - Let $j = k - (\texttt{length}(A_2) + \texttt{length}(A_3))$.
  - Recurse: find the j^{th} largest element of A_1.
- Otherwise, return x.

3 Connected components

A connected component of an undirected graph is a maximal set of vertices which has the property that you can go from any vertex to every other by traveling through the edges in the graph. A connected component in the graph can be found using depth-first search (DFS) or breadth-first search (BFS), as described in lecture. DFS is implemented using stacks and BFS using queues.

We have provided functions which emulate stacks and queues using Matlab arrays. For queues, these are `enqueue`, `dequeue`, `queueCreate`; for stacks, `stackPush`, `stackPop`, `stackCreate`. Example usage is shown below, but you can read more about how to use these functions using Matlab's `help` command.

```
myQueue = queueCreate();
myQueue = enqueue(myQueue, 5);
myQueue = enqueue(myQueue, -1);
[value, myQueue] = dequeue(myQueue);  % value should be 5
[value, myQueue] = dequeue(myQueue);  % value should be -1

myStack = stackCreate();
myStack = stackPush(myStack, 5);
myStack = stackPush(myStack, -1);
[value, myStack] = stackPop(myStack); % value should be -1
[value, myStack] = stackPop(myStack); % value should be 5
```

We encourage you to use the builtin Matlab connected components function `bwlabel` to debug your code. It takes two parameters, as follows:

```
L = bwlabel(image, 4)
```

The number 4 indicates the neighborhood system (left, down, right, up). The size of the output matrix `L` is same as the input image, but each connected component is marked with a unique number, starting from 1 and going until the number of connected components in the graph is reached. In other words, all points in blob 1 will have a value of 1, all points in blob 2 will have a value of 2, and so on and so forth. You can use another matlab function called `label2rgb` to visualize different connected components:

```
L = bwlabel(image, 4);

RGB = label2rgb(L, 'jet', 'blue');
```

You can use Matlab's `help` command to see more information about these functions.

1	1	1	1	0
0	0	0	0	0
0	1	1	1	0
0	1	1	0	1
0	0	0	0	1

Figure 1: Use this example to test your implementation of connected-component-finding using BFS and DFS.

4 What To Do

1. Implement **modified quicksort** and use it to find the median of a given array. Your implementation should have the following prototype:

   ```
   function m = median_student(A)
   ```

   Place your code in `median_student.m`.

2. Implement and test **connected components** on images, using the 4-connected neighborhood system. You must use both DFS and BFS. In your DFS implementation, use the code for stacks given above, and in your BFS implementation use the code for queues. The output of your function will be a labeled image, where every pixel is labeled with the number of the blob it is in. Test your code by using the example input in figure 1.

   (a) Your **DFS** implementation should have the following prototype:

   ```
   function labeled = CC_DFS_student(img)
   ```

   Place your code in `CC_DFS_student.m`. Test your code by using the example input in figure 1.

   (b) Your **BFS** implementation should have the following prototype:

   ```
   function labeled = CC_BFS_student(img)
   ```

   Place your code in `CC_BFS_student.m`. Test your code by using the example input in figure 1.

3. Using your DFS implementation, compute the connected components in an input image and return only the **largest one**. Your implementation should have the following prototype:

   ```
   function labeled = big_red_student(img)
   ```

Place your code in `big_red_student.m`.

4. Using your big red region code and your code based on modified quicksort, find the **median center** of the biggest red region. Your implementation should have the following prototype:

   ```
   function [r,c] = big_red_ctr_student(img)
   ```

   Place your code in `big_red_ctr_student.m`.

5. Using the connected components code which you have developed, find the **size** (number of pixels) of each connected component and plot a histogram using matlab's `hist` command. As usual, consult Matlab `help` if you are unsure about how to use a function. Place your code in `CC_sizes_student.m`.

6. Using the centroid code from the previous assignment, find the centroid of **every** connected component and plot its **distance** from the centroid of the biggest connected component (again, use a histogram). The distance can be found using the formula:

   ```
   distance = sqrt((r1 - r2) * (r1 - r2) + (c1 - c2) * (c1 - c2))
   ```

   Place your code in `CC_centroids_student.m`.

7. Write a function which drives a robot based on the position of the largest connected component and its area. More specifically, if the area of the blob increases, move the robot forward; if it decreases, move the robot backward. If the centroid moves to the left, rotate the robot to the left; if the centroid moves to the right, rotate the robot to the right. You will need to use the following robot functions `robotConnect`, `robotLock`, `robotDrive`, `robotQuery` and `robotUnlock`. Look them up using Matlab's `help` command for information about how they're used. Place your code in `CC_robots_student.m`.