```
function [J, center1, radius1] = Prob6c(I,color,reg_maxdist)

    xarr = [0 0 0];
    yarr = [0 0 0];

    for i = 1 : size(I,1)
        for j = 1 : size(I,2)
            if ( abs(color(1) - I(i,j,1)) <= .02 && ...
                 abs(color(2) - I(i,j,2)) <= .02 && ...
                 abs(color(3) - I(i,j,3)) <= .02 )
                xarr = [i i i];
                yarr = [j j j];
                break;
            end
        end
        if (xarr(1) ~= 0)
            break;
        end
    end


    J = zeros(size(I)); % output
    Isizes = size(I);   % sizes of image
    for i = 1 : size(I,3)
        x = xarr(i);
        y = yarr(i);
        neg_free = 10000;   % for neighbor list
        neg_pos = 0;        % position of neighbor, and also number of neighbors
        neg_list = zeros(neg_free,3);   % holds all the neighbor information
        neigb = [-1 0; 1 0; 0 -1;0 1];  % used for finding 4 direction neighbor
        pixdist = 0;
        reg_mean = I(x,y,i);
        reg_size = 1;

        % checks whether pixdist isn't bigger than the max region distance
        % checks whether the region isn't bigger than the image

        % if pixdist > reg_maxdist then no neihgbors are similar to region
        while (pixdist < reg_maxdist(i) && reg_size < numel(I(:,:,i)))

            % finds the 4 neighbors of pixel and adds to neighbor list
            for j = 1 : 4
                % j = 1, it goes to the left neighbor
                % j = 2, it goes to the right neighbor
                % j = 3, it goes to the up neighbor
                % j = 4, it goes to the down neighbor
                % (xn , yn) - neighbor pixel that we working with
                xn = x + neigb(j,1);
                yn = y + neigb(j,2);

                % is (xn,yn) within the image boundarys 1 < xn, yn < dim(image)
```

```
52                  ins = (xn >= 1) && (yn >= 1) && (xn <= Isizes(1)) && ...
53                        (yn <= Isizes(2));
54
55                  % checks if inside image, then checks if neighbor pixel wasn't
56                  % already counted as a neighbor
57                  if (ins && (J(xn,yn,i) == 0))
58                      neg_pos = neg_pos + 1;  % increment neighbor
59                      % saves neighbor location and data
60                      neg_list(neg_pos, :) = [xn yn I(xn, yn,i)];
61                      J(xn, yn,i) = 1; % notes as neighbor
62                  end
63              end
64
65              % Make neighbor list bigger if needed;
66              if (neg_pos + 10 > neg_free)
67                  neg_free = neg_free + 10000;
68                  neg_list( (neg_pos + 1) : neg_free, :) = 0;
69              end
70
71              % dist finds distance between the neighbor and the mean
72              dist = abs( neg_list(1: neg_pos,3) - reg_mean ) ;
73
74              % pixdist is the smallest distance from one neighbor to the mean
75              % index is the index of the neighbor that has the smallest distance
76              % from the mean
77              [pixdist, index] = min(dist);
78
79              % Path which the algorithm goes is the path of 2s
80              J(x,y,i) = 2;
81              % increments region size
82              reg_size = reg_size + 1;
83
84              % mean = (mean * reg_size + closest neighbor value) / (reg_size + 1)
85              reg_mean = (reg_mean * reg_size + neg_list(index,3))/(reg_size+1);
86
87              % restarts except starts with closest neighbor
88              x = neg_list(index,1);
89              y = neg_list(index,2);
90
91              % replaces the closest neighbor with the last neighbor
92              neg_list(index,:) = neg_list(neg_pos,:);
93
94              % decrements neg_pos so the last neighbor gets overwritten
95              neg_pos = neg_pos-1;
96          end
97
98          % converts path of 2's into path of 1's and everything else is 0
99          J(:,:,i) = J(:,:,i) > 1;
100     end
101 end
```