```
% Tony Hyun Kim
% 2013 10 08
% CS 224w: PS 1, Problem 3(c):
%   Generate networks to a file as a function of alpha
clear all; close all;

% Tree parameters
h = 10; % Height of tree
b = 2;  % Branching factor
k = 5;  % Degree per node

N = b^h; % Total number of nodes (leaves of tree)
M = k*N; % Total number of edges

% Load the tree distance matrix.
% Note: Once I realized how inefficient my treedist calculation was, I
%       decided to store the distance matrix for the h=10, b=2 case.
load('D_h10_b2.mat');

nodelist = 0:(N-1);    % Use 0-based index
edgelist = zeros(M,2); % Format: [source target]

% Populate the edges according to the given edge probability model
alphas = 0.1:0.1:10;
for alpha = alphas
    for i = 1:N % Select source node
        ni = nodelist(i);

        % Probability distribution to other nodes
        p = b.^(-alpha*D(i,:)); % Use stored distances
        p(i) = 0; % No self loops
        for l = 1:k % Obtain edges
            j = randsample(N, 1, true, p/sum(p));
            edgelist(k*(i-1)+l,:) = [ni nodelist(j)];

            p(j) = 0; % Do not redraw the same edge
        end
    end

    savename = sprintf('alpha%02d_%1d.mat', ...
                       floor(alpha), mod(floor(10*alpha),10));
    save(savename, 'h', 'b', 'k', 'N', 'M',...
                   'alpha', 'nodelist', 'edgelist');
end
```

```
% Tony Hyun Kim
% 2013 10 08
% CS 224w: PS 1, Problem 3(c):
%   Run search simulations on synthesized graphs
clear all; close all;

sources = dir('alpha*.mat');
Nsources = length(sources);

% Need to load the distance metric
load('D_h10_b2.mat');

for f = 1:Nsources
    load(sources(f).name);

    % Perform the random (s,t) searchs
    Nsearch = 1000;

    % Format: [ns nt pathlen]
    search = zeros(Nsearch,3);
    for i = 1:Nsearch
        r = randsample(N, 2, false); % Sample without replacement

        ns = nodelist(r(1)); % Basically conversion for Matlab 1-index
        nt = nodelist(r(2));
        search(i,:) = [ns nt inf];

        len = 0;

        % Set current position u to s
        nu = ns;
        h_ut = D(nu+1, nt+1);
        nu_neighbors = edgelist((1+k*nu):(k+k*nu),2);
        hs = D(nu_neighbors+1, nt+1);
        [min_hs, min_ind] = min(hs);

        while( min_hs < h_ut )
            len = len + 1;

            % Update current position
            nu   = nu_neighbors(min_ind);
            h_ut = min_hs;

            % Did we reach the target? Then terminate
            if (h_ut == 0)
                search(i,3) = len;
                break;
            end

            % Update neighbors
            nu_neighbors = edgelist((1+k*nu):(k+k*nu),2);
            hs = D(nu_neighbors+1, nt+1);
            [min_hs, min_ind] = min(hs);
        end
    end
```

```
    sieve = search(:,3)<inf;
    Nsuccess = sum(sieve);
    avglen = mean(search(logical(sieve),3));
    fprintf('%2.1f %.4f %2.4f\n', alpha, Nsuccess/Nsearch, avglen);
end
```

```
% Compute the tree distance between node u and list of nodes vs
% in a binary tree with height h
function ds = treedist(u, vs, h)

ds = zeros(size(vs));

for j = 1:length(vs)
    ub = dec2bin(u,h);
    vb = dec2bin(vs(j),h);

    d = 0;
    while (~isempty(ub))
        if (all(ub == vb))
            break;
        else
            d = d + 1;
            ub = ub(1:end-1);
            vb = vb(1:end-1);
        end
    end

    ds(j) = d;
end
```