Abelardo Riojas
ISC 4221C Lab 4 Report
Professor Quaife Fall 2019

1. Graph Representation
   i. Construct a GRF for a graph that represents the countries of South America. Each node should represent one of the 12 countries (excluding the French colony north of Brazil), and two nodes have an edge between them if they share a land border. The (x, y) -coordinates are not important for this problem, and you can set them somewhat arbitrarily. Explicitly state the country that corresponds to each node.

```
file = 'southamerica.grf'

1 4.0 3.0 2 3 4 8 11
2 4.0 5.5 1 3 4 8 9
3 6.0 6.0 1 2 5 7 8 9 10 11 12
4 3.0 3.0 1 2 9
5 3.0 8.5 3 6 9 12
6 2.2 8.0 5 9
7 5.0 9.0 3 10 11
8 5.0 4.5 1 2 3
9 2.5 8.0 2 3 4 6
10 6.5 8.5 3 7
11 5.5 4.5 1 3
12 4.0 9.0 3 5 7
```

Countries are sorted alphabetically with Argentina corresponding to 1 and Venezuela corresponding to 12.

1 Argentina, 2 Bolivia, 3 Brazil, 4 Chile, 5 Colombia, 6 Ecuador, 7 Guyana, 8 Paraguay, 9 Peru, 10 Suriname, 11 Uruguay and 12 Venezuela.

   ii. Use your code to convert the GRF format of your graph to each of the other four formats. Include the output in your writeup.

(Given code for reading .grf files)
```
fid = fopen('southamerica.grf');
str_line = 1;
line_count = 1;
while 1
    str_line = fgetl(fid); % read next line from the file
    if str_line ~= -1 % if str_line is not empty
```

```
        num_line = str2num(str_line);  % convert string to an array
        data(line_count) = {num_line};  % save the array into data
        line_count = line_count + 1;
    else % if str_line is empty, stop reading
        break
    end
end
fclose(fid);
```

Displays output to the command line

```
adj = adjMatrix(data)
edgelist = edgeList(data)
adjstructure = adjStruct(data,1);
incidence = incidenceMatrix(data)
```

Functions

```
function matrix = adjMatrix(data)
    n = size(data,2);
    matrix = zeros(n);
    for x = 1:n
        node = data{x};
        connections = node(4:end);
        for connection = connections
            matrix(x,connection) = 1;
        end
    end
end

function edgelist = edgeList(data)
    n = size(data,2);
    edgelist = [];
    for x = 1:n
        node = data{x};
        connections = node(4:end);
        for connection = connections
            edgelist = [edgelist;[x connection]];
        end
    end
    n = size(edgelist,1);
    for x = n:-1:1
        if (ismember(flip(edgelist(x,:)), edgelist, 'rows') == 1)
            edgelist(x,:) = [];
        end
    end
end
```

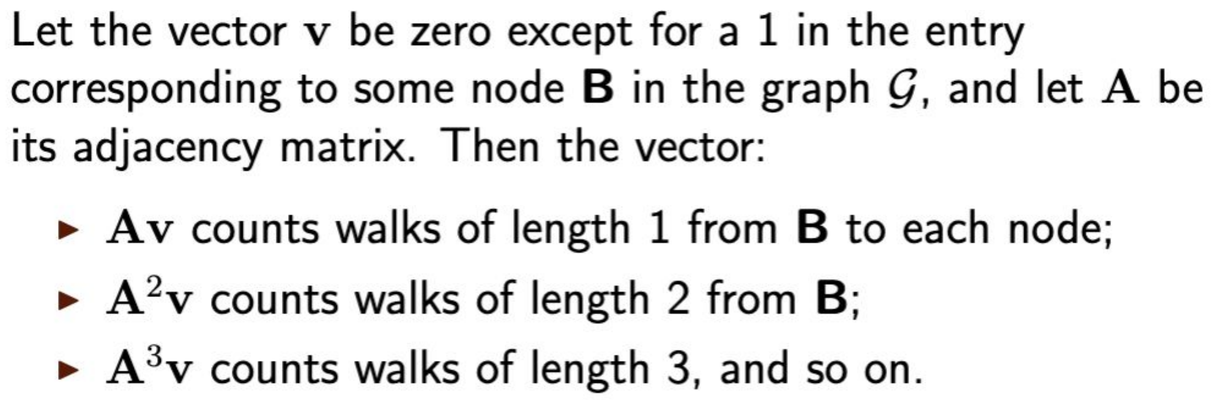
```
function structure = adjStruct(data,display)
    n = size(data,2);
    structure = cell(n,1);
    for x = 1:n
        node = data{x};
        structure{x} = node(4:end);
    end
    if display == 1
        disp("adjstructure =")
        for x = 1:n
            disp(structure{x})
        end
    end
end


function incidence = incidenceMatrix(data)
    n = size(data,2);
    edgelist = [];
    for x = 1:n
        node = data{x};
        connections = node(4:end);
        for connection = connections
            edgelist = [edgelist;[x connection]];
        end
    end
    n = size(edgelist,1);
    for x = n:-1:1
        if (ismember(flip(edgelist(x,:)), edgelist, 'rows') == 1)
            edgelist(x,:) = [];
        end
    end
    incidence = zeros(5);
    for x = 1:size(edgelist,1)
        incidence(x,edgelist(x,1)) = 1;
        incidence(x,edgelist(x,2)) = 1;
    end
end
```

Output:

```
adjmatrix =

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


edgelist =

     1     2
     1     3
     1     4
     1     8
     1    11
     2     3
     2     4
     2     8
     2     9
     3     5
     3     7
     3     8
     3     9
     3    10
     3    11
     3    12
     4     9
     5     6
     5     9
     5    12
     6     9
     7    10
     7    11
    12     7

adjstructure =
     2     3     4     8    11

     1     3     4     8     9

     1     2     5     7     8     9    10    11    12

     1     2     9
     3     6     9    12

     5     9

     3    10    11

     1     2     3

     2     3     4     6

     3     7

     1     3

     3     5     7


incidence =

     1     1     0     0     0     0     0     0     0     0     0     0
     1     0     1     0     0     0     0     0     0     0     0     0
     1     0     0     1     0     0     0     0     0     0     0     0
     1     0     0     0     0     0     0     1     0     0     0     0
     1     0     0     0     0     0     0     0     0     0     1     0
     0     1     1     0     0     0     0     0     0     0     0     0
     0     1     0     1     0     0     0     0     0     0     0     0
     0     1     0     0     0     0     0     1     0     0     0     0
     0     1     0     0     0     0     0     0     1     0     0     0
     0     0     1     0     1     0     0     0     0     0     0     0
     0     0     1     0     0     0     1     0     0     0     0     0
     0     0     1     0     0     0     0     1     0     0     0     0
     0     0     1     0     0     0     0     0     1     0     0     0
     0     0     1     0     0     0     0     0     0     1     0     0
     0     0     1     0     0     0     0     0     0     0     1     0
     0     0     1     0     0     0     0     0     0     0     0     1
     0     0     0     1     0     0     0     0     1     0     0     0
     0     0     0     0     1     1     0     0     0     0     0     0
     0     0     0     0     1     0     0     0     1     0     0     0
     0     0     0     0     1     0     0     0     0     0     0     1
     0     0     0     0     0     1     0     0     1     0     0     0
     0     0     0     0     0     0     1     0     0     1     0     0
     0     0     0     0     0     0     1     0     0     0     1     0
     0     0     0     0     0     0     1     0     0     0     0     1
```

iii. Suppose you are hitch-hiking from Chile to Venezuela. How would you use your adjacency matrix to compute walks between the two countries? What is the fewest number of countries that you must visit, including Chile and Venezuela? How many different shortest walks are possible? List all of them.

From the lecture notes:



Let the vector $\mathbf{v}$ be zero except for a 1 in the entry corresponding to some node $\mathbf{B}$ in the graph $\mathcal{G}$, and let $\mathbf{A}$ be its adjacency matrix. Then the vector:

- $\mathbf{Av}$ counts walks of length 1 from $\mathbf{B}$ to each node;
- $\mathbf{A}^2\mathbf{v}$ counts walks of length 2 from $\mathbf{B}$;
- $\mathbf{A}^3\mathbf{v}$ counts walks of length 3, and so on.

Fewest number of countries: 4, number of shortest walks 4.

```
A = adjmatrix;
v = [0 0 0 1 0 0 0 0 0 0 0 0]';
n = size(data,2);
for x = 1:n
    vn = (A^x)*v;
    if (vn(12) ~= 0)
        disp([x, vn(12)])
        break
    end
end
```

Output from the command line: `3        4`

List:

Chile → Argentina → Brazil → Venezuela
Chile → Bolivia → Brazil → Venezuela
Chile → Peru → Brazil → Venezuela
Chile → Peru → Colombia → Venezuela

iv. How many different walks are there of size 8? Describe how you determined this number.

```
vn = A^8 * v;
```

```
disp(vn(12))
```

Output from the command line: `8410.`

This number was taken by taking the 8th power of A and multiplying it by the vector `[0 0 0 1 0 0 0 0 0 0 0 0]'` Taking the 12th entry of that new vector (corresponding to the number of 8-step walks it takes to get from Chile to Venezuela) gives you gives you the number of walks.

2. Traveling salesman problem

   i. Read the edge list of a graph

```
fid = fopen('tsp.txt');
str_line = 1;
line_count = 1;
while 1
    str_line = fgetl(fid); % read next line from the file
    if str_line ~= -1 % if str_line is not empty
        num_line = str2num(str_line);  % convert string to an array
        tspdata(line_count) = {num_line};  % save the array into data
        line_count = line_count + 1;
    else % if str_line is empty, stop reading
        break
    end
end
fclose(fid);
```

tsp.txt

```
1 2 4.00
1 3 6.00
1 6 2.25
1 7 3.75
1 8 4.25
2 3 5.50
2 7 6.50
3 4 5.00
3 8 2.50
4 8 7.00
5 6 2.50
5 7 6.25
5 8 3.50
6 7 4.00
6 8 3.75
```

ii. Read the edge weights of a graph (if you prefer, this can be part of the same file containing the edge list);

   Edge weights are read in the tsp.txt file.

iii. Check all possible itineraries

```
tspadj = tspMatrix(tspdata);
fastest = [];
for i = 1:8
    [itin] = sortrows(itinerary(tspadj,8,i));
    fastest = [fastest; itin(1,:)];
end
disp(fastest(:,2:10))


function [permu] = itinerary(tspadj,nodes,home)
    myarray =  1:nodes;
    all_perms = perms(myarray);
    np = size(all_perms,1);
    all_perms = [ones(np,1).*home all_perms];
    permu = [];

    for x = 1:factorial(nodes)
        perm = all_perms(x,:);
        n = numel(perm);
        if perm(1) == perm(end)
            sum = 0;
            for i=1:n-1
                a = perm(i);
                b = perm(i+1);
                sum = sum + tspadj(a,b);
            end
            if sum < 10^10
            permu = [permu;sum perm];
            end
        end
    end
end
function matrix = tspMatrix(data)
    n = size(data,2);
    matrix = ones(n).*(10^10);
    for x = 1:n
        edge = data{x};
        matrix(edge(1),edge(2)) = edge(3);
        matrix(edge(2),edge(1)) = edge(3);
    end
```

```
end
```

iv. Print the sequence of nodes with the shortest itineraries.
(Using above code)

Output from the command line:

```
1    2    3    4    8    5    6    7    1
2    1    7    6    5    8    4    3    2
3    2    1    7    6    5    8    4    3
4    3    2    1    7    6    5    8    4
5    6    7    1    2    3    4    8    5
6    5    8    4    3    2    1    7    6
7    1    2    3    4    8    5    6    7
8    4    3    2    1    7    6    5    8
```

v. Print the total length of the shortest itineraries.

```
disp(fastest(:,1))
```

Output from the command line:

```
35.2500
35.2500
35.2500
35.2500
35.2500
35.2500
35.2500
35.2500
```