Advanced Topic
- Shortest Path Problem

Introduction

Shortest Path is a topic in mathematical optimization, which uses math to select the best(most efficient) sets of elements in all possible cases. These types of problems are usually given in a map with several nodes (with edges connecting each nodes and a measurement denoting the cost of each edge (weight), a starting node and a destination node). These types of problem are easy if there are only a few nodes, but if we have more nodes, the problem becomes insolvable without the use of a computer. Thankfully, Matlab makes it all easy; it provides a function that can easily solve this problem.

There are two functions that can solve this type of problem, graphshortestpath and shortestpath. In this tutorial, we will only be going over graphshortestpath because the other one requires knowledge of constructing a biograph object, whereas graphshortestpath only requires a sparse matrix.

First we will talk about the syntax of this function, which is:

[sum_of_distance, path, predecessor_nodes] = graphshortestpath(inputs)

Definitions

Definitions of Outputs

- sum_of_distance: the sum of the weights in the winning path.
- path: a sparse matrix that records all the nodes gone through in the winning path.
- predecessor_nodes: a matrix that finds last node gone through in the winning paths from the starting node to all the other nodes. If you are still unclear about this, see sample problem 1 at the bottom.

Now let's talk about the combination of inputs:

- graphshortestpath(map, starting node): finds a winning path that starts from the starting point to all other nodes.
- graphshortestpath(map, starting node, destination node): it finds a winning path that starts from the starting point to the destination node.

Definitions of inputs

- Map is a sparse matrix including all the edges, their direction and flow values. The declaration is slightly different than that of the normal matrix. When creating a sparse matrix, you must use sparse() function. It can be used in a lot of places and in a lot of different formats, but for the use of map, the format is:

  map = sparse([First row, the starting node of the edge], [Second row, the ending node of the edge], [Third row, the flow value of the edge], number of nodes, number of nodes)

  Because it is a matrix, the length of each row must be the same. The length of each row is the number of edges in the map.

  For example:

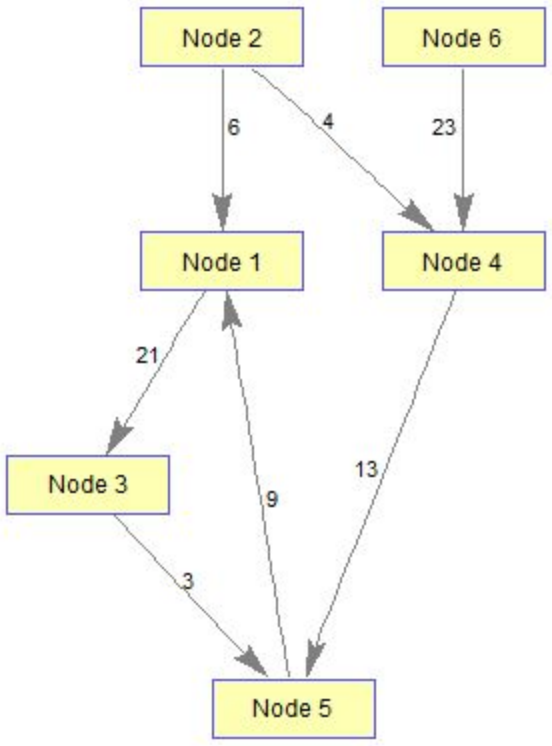
  map = sparse([1, 3, 2, 6, 2, 4, 5], [3, 5, 1, 4, 4, 5, 1], [21, 3, 6, 23, 4, 13, 9], 6, 6)

  which will generate 7 edges and 6 nodes with the following information

Edge	Starting node	Ending node	Flow value

Edge 1	1	3	21
Edge 2	3	5	3
Edge 3	2	1	6
Edge 4	6	4	23
Edge 5	2	4	4
Edge 6	4	5	13
Edge 7	5	1	9

The map will look like this:



Note that each edge has a direction. This means that going from node 1 to node 6 is impossible. However, going from node 6 to node 1 is possible with edges 4, 6 and 7. Also, the map declaration is poorly written because the edges are not in order. It is best to organize them in order so it is easy to understand. A more organized version of the map would be:

```
map = sparse( [1, 2, 2, 3, 4, 5, 6], ...
        [3, 1, 4, 5, 5, 1, 5], ...
        [21, 6, 4, 3, 13, 9, 23], 6, 6)
```

(Note: Those "..."'s at the end of each line mean that the next line of code is part of the command. If they were not added, matlab would think those were three different commands and therefore result in an error).

- Starting Node: This input only requires a single number, indicating which node is the starting node. For example, with the same map, if the input is 6, then the starting node will be Node 6.
- Ending Node: This input only requires a single number, indicating which node is the destination node. For example, with the same map, if the input is 1, then the destination node will be Node 1.

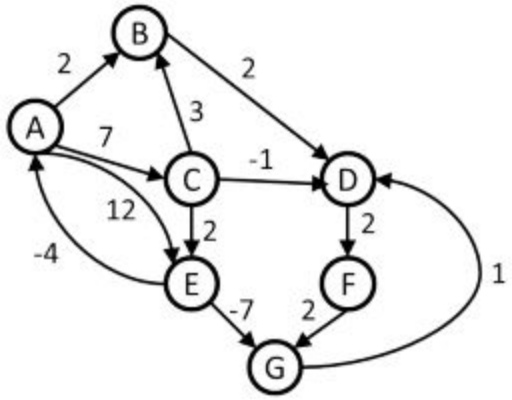
Name-Value Pairs

Setting	Value
'Directed'	Indicator of whether the map is directed or undirected. If the map is directed, each edge then has a certain direction, going from the opposite direction is impossible. If the map is undirected, it means going from either side of an edge is legal. The only inputs are 'true' and 'false'
'Method'	The algorithm used to find the shortest path. The possible inputs are: 'Bellman-Ford' - Assumes weights of the edges to be nonzero entries in Map. Time complexity is O(N*E). 'BFS' — Breadth-first search. Assumes all weights to be equal, and nonzero entries in Map to represent edges. Time complexity is O(N+E). 'Acyclic' — Assumes Map to be a directed acyclic graph and the weights of the edges to be non-zero entries in Map. Time complexity is O(N+E). 'Dijkstra' — Assumes weights of the edges to be positive values in Map. Time complexity is O(log(N)*E). Default. Note: N and E are the number of nodes and edges
'Weights'	A column matrix, which sets the customized weights of all the edges. The number of rows of this matrix must match that of the map. The order of edges will be used to determined which value in this matrix goes to which which edge; a better organized map will make the process of matching a lot easier.

Sample Problems

Problem 1 - Shortest Path:
This problem is found online, which is in the link below: (Problem 2)
https://courses.cs.washington.edu/courses/cse373/13au/more_examples_midterm2_solved.pdf

The problem asks us to find the shortest path from A to all the vertexes (node).



First we have to set up the network, or map. To do that, we have to find the starting node, ending node and the weight of every edge. This table contains all the edges and their information in an organized way(Alphabets are represented by numbers because of the syntax):

Edge	Starting Node	Ending Node	Weight
Edge 1	1	2	2
Edge 2	1	3	7
Edge 3	1	5	12
Edge 4	2	4	2
Edge 5	3	2	3
Edge 6	3	4	-1
Edge 7	3	5	2
Edge 8	4	6	2
Edge 9	5	1	-4

Edge 10	5	7	-7
Edge 11	6	7	2
Edge 12	7	4	1

Using this table, we can set up the map:

```
map = sparse( [1, 1, 1, 2, 3, 3, 3, 4, 5, 5, 6, 7], ...
              [2, 3, 5, 4, 2, 4, 5, 6, 1, 7, 7, 4], ...
              [2, 7, 12, 2, 3, -1, 2, 2, -4, -7, 2, 1], 7, 7)
```

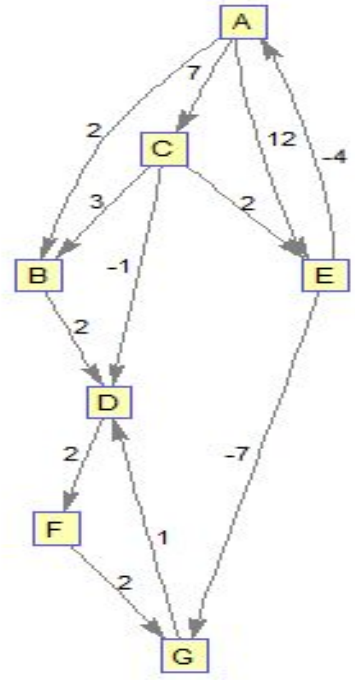
Note that this function does not work with the problem that has a 0 as one of its weights, because sparse matrix automatically takes out 0's Therefore that edge would be erased.

To check if the map is correct, you can graph it out using the following command:

```
view( biograph(map, [], 'ShowWeights','on'))
```

(Note:
Biograph is a command to create a biograph object.
View is a function that opens a window and draw the biograph object.
Map is the map that you want to draw.
[ ] is an array for nodes' IDs.
'ShowWeights' and 'on' are a name-value pair. It is used to set the visibility of flow values of each edge. The only inputs are 'on' and 'off'.
Visit http://www.mathworks.com/help/bioinfo/ref/biograph.html for more information)

Enter `view(biograph(map,['A', 'B', 'C', 'D', 'E', 'F', 'G'], 'ShowWeights','on'))`, you will get th following output:



This matches the graph that is provided.

Note that this map contains negative numbers, which does not match Dijkstra's algorithm requirements, that states that all weights must be positive numbers. So in this case, we will use the Bellman–Ford algorithm, which only needs numbers to be non-zero.

The starting node is A, use that information in the command, which will be

```
[dist, path, pred] = graphshortestpath(map, 1, 'Method', 'Bellman-Ford')
```

Then it will return the answers, which are

```
dist =
    0    2    7    3    9    5    2
path =
 Columns 1 through 6
  [1]   [1x2 double]   [1x2 double]   [1x5 double]   [1x3 double]   [1x6 double]
 Column 7
   [1x4 double]

pred =
   0    1    1    7    3    4    5
```

We can organize these numbers into a table.

Starting Node	Ending Node	Distance Traveled	Winning Path	Predecessor Nodes
A	A	0	A	N/A
A	B	2	A,B	A
A	C	7	A,C	A
A	D	3	A,C,E,G,D	G
A	E	9	A,C,E	C
A	F	5	A,C,E,G,D,F	D
A	G	2	A,C,E,G	E

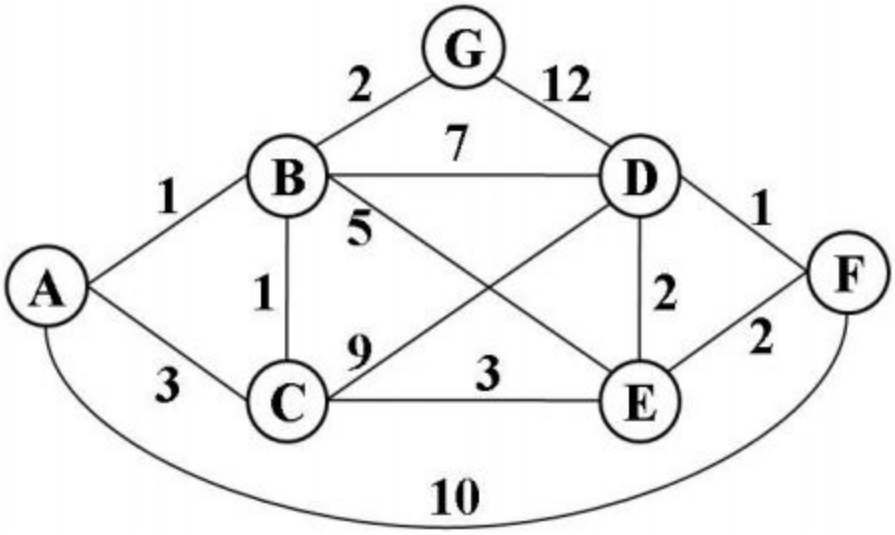
This is a good chance for those of you that did not completely understand the idea of predecessor node to understand it. Notice that the predecessor node is the last node before reaching the ending node in every corresponding winning path. That is what predecessor node mean, the second to the last node in a winning path.

We have the same result as the one that is given. (Their table is partially wrong, check question b in the solution section. Those are the real answers)

Problem 2 - Shortest Path:
This problem is found online, which is in the link below: (Problem 1)

https://courses.cs.washington.edu/courses/cse373/13au/more_examples_midterm2_solved.pdf

The problem asks to find the shortest path from A to F with the given map:



First we have to set up the network, or map. To do that, we have to find the starting node, ending node and the weight of every edge. This table contains all the edges and their information in an organized way(Alphabets are represented by numbers because of the syntax)

Important: for undirected map, organization is very important, it might affect the final answer.

Edge	Starting Node	Ending Node	Weight
Edge 1	1	2	1
Edge 2	1	3	3
Edge 3	1	6	10
Edge 4	2	3	1
Edge 5	2	4	7
Edge 6	2	5	5
Edge 7	2	7	2
Edge 8	3	4	9
Edge 9	3	5	3

Edge 10	4	5	2
Edge 11	4	6	1
Edge 12	4	7	12
Edge 13	5	6	2

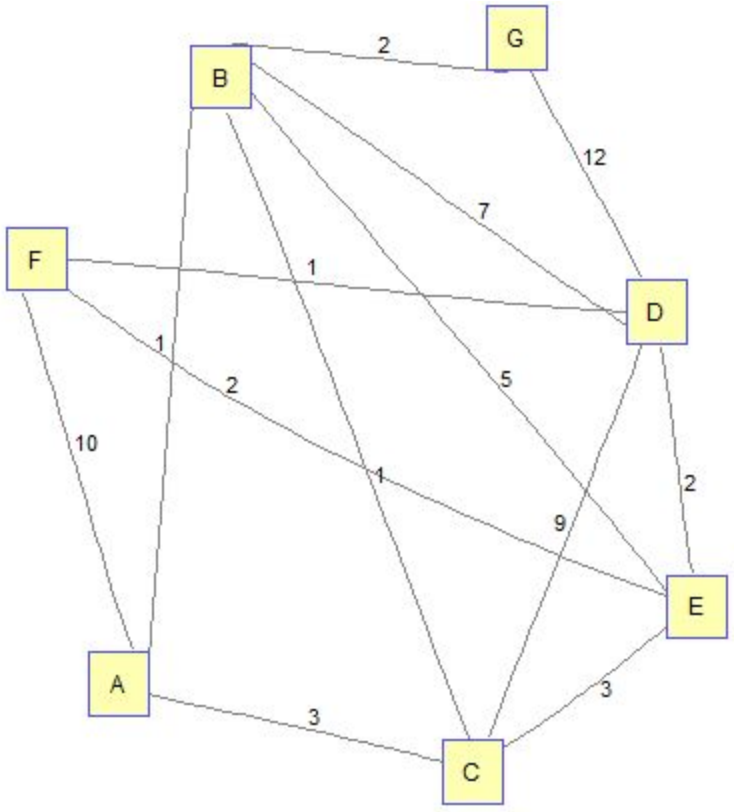
Using this table, we can set up the map:

```
map = sparse( [1, 1, 1, 2, 2, 2, 2, 3, 3, 4, 4, 4, 5], ...
              [2, 3, 6, 3, 4, 5, 7, 4, 5, 5, 6, 7, 6], ...
              [1, 3, 10, 1, 7, 5, 2, 9, 3, 2, 1, 12, 2], 7, 7)
```

Enter

```
view(biograph(map,['A', 'B', 'C', 'D', 'E', 'F', 'G'], 'ShowArrows', 'off', 'LayoutType', 'equilibrium' , 'ShowWeights','on'))
```

You might not get the same graph, because the original graph was messy, so some nodes have been dragged around to make it clear. This is the picture after dragging:



For undirected map, you must use the following command before solving the problem:

```
map = tril( map + map' )
```

(Note: tril function returns the lower triangular part of the input. Visit http://www.mathworks.com/help/matlab/ref/tril.html for more information)

The question asks us for a path from A to F. Using the numbers 1 to 6, plug each of those into the command. Remember, this map is undirected so we must set the name-value pair 'directed' to false.

```
[dist, path, pred] = graphshortestpath(map, 1, 6, 'directed', false)
```

This is the output:

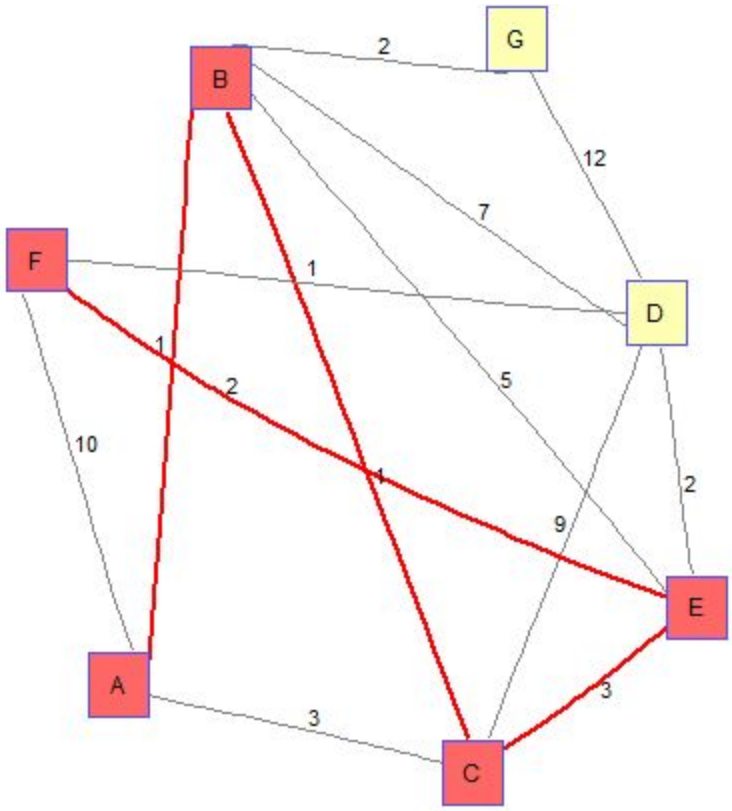
```
dist =
    7
path =
    1    2    3    5    6
pred =
    0    1    2    5    3    5    2
```

Compared to the answer that is given, which is A->B->C->E->F (1->2->3->5->6) and 7 for total distance, we get the exact same result.

To show the solution in picture form, which thickens the edges and change the color of nodes and edges in the winning path, do the following, where h is the biograph object:

```
set(h.Nodes(path),'Color',[1 0.4 0.4]) %find the nodes and change color to red
fowEdges = getedgesbynodeid(h,get(h.Nodes(path),'ID')); %find the edges
revEdges = getedgesbynodeid(h,get(h.Nodes(fliplr(path)),'ID'));
edges = [fowEdges;revEdges];
set(edges,'LineColor',[1 0 0]) %change their color to red
set(edges,'LineWidth',1.5) %set the width to 1.5
view(h) %view the biograph
```

Then you will see


G
B
2
12
7
F
1
D
1
2
5
10
1
2
9
E
A
3
3
C