Intro Math Problem Solving

Graph Theory Origin

Fleury's Algorithm

The Adjacency Matrix and Distances

Is There a Path From A to B?

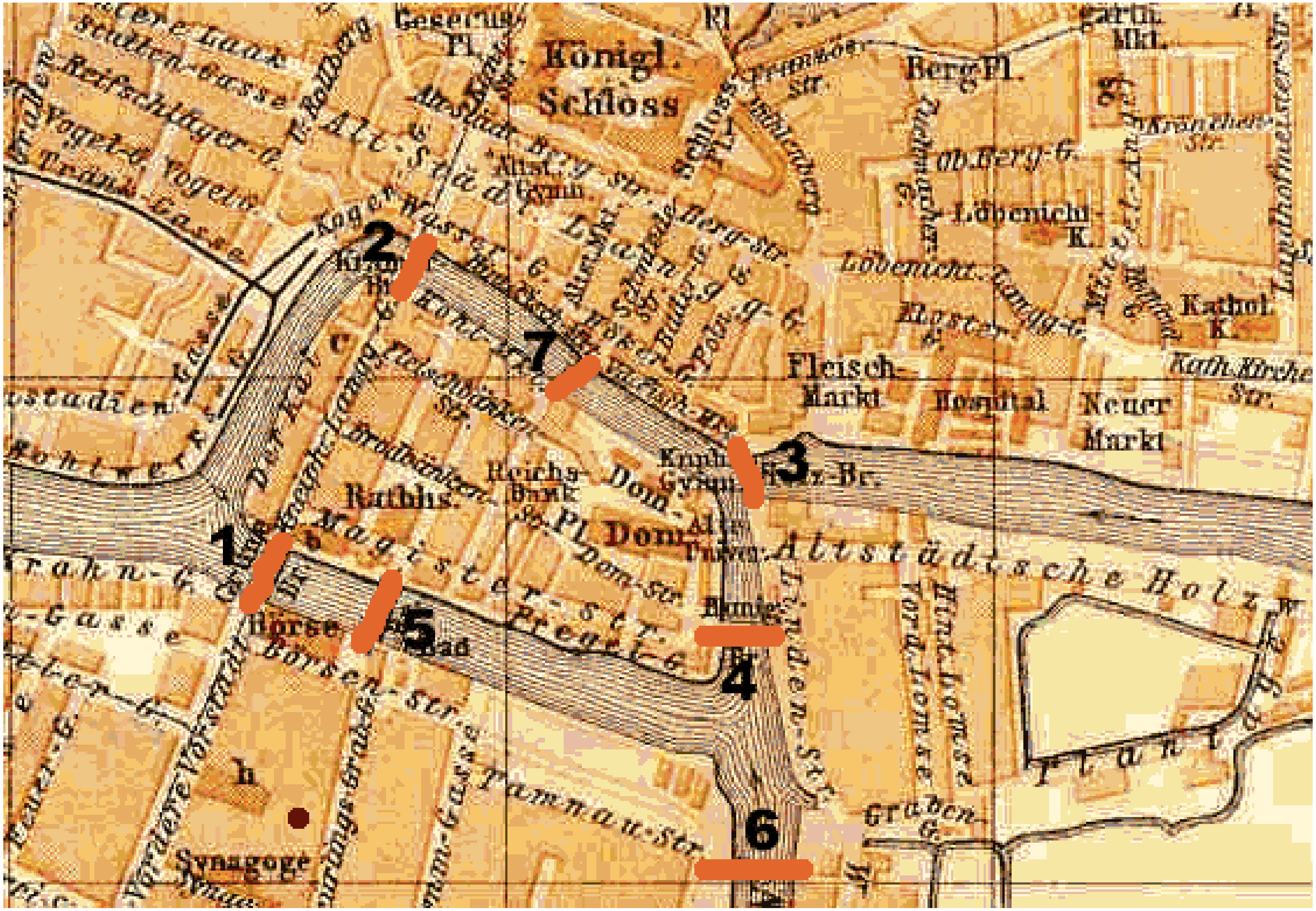
What is the Path from A to B?

Is There a Path From ANY A to ANY B?

Paths and the Adjacency Matrix

Homework #12

Graph Theory Origin



Euler's Bridge Problem

In the town of Koenigsberg, there were seven bridges across the river Pregel, some of which went to an island in the middle of the river.

The townspeople enjoyed trying to find a solution to the challenge of planning a trip that crossed every bridge exactly once, returning to the starting point.

No one could find a solution to this puzzle, even when the conditions were relaxed so that a round trip was not required.

Even though this puzzle did not seem to be about mathematics, it caught the attention of Euler.

Inventing Graph Theory

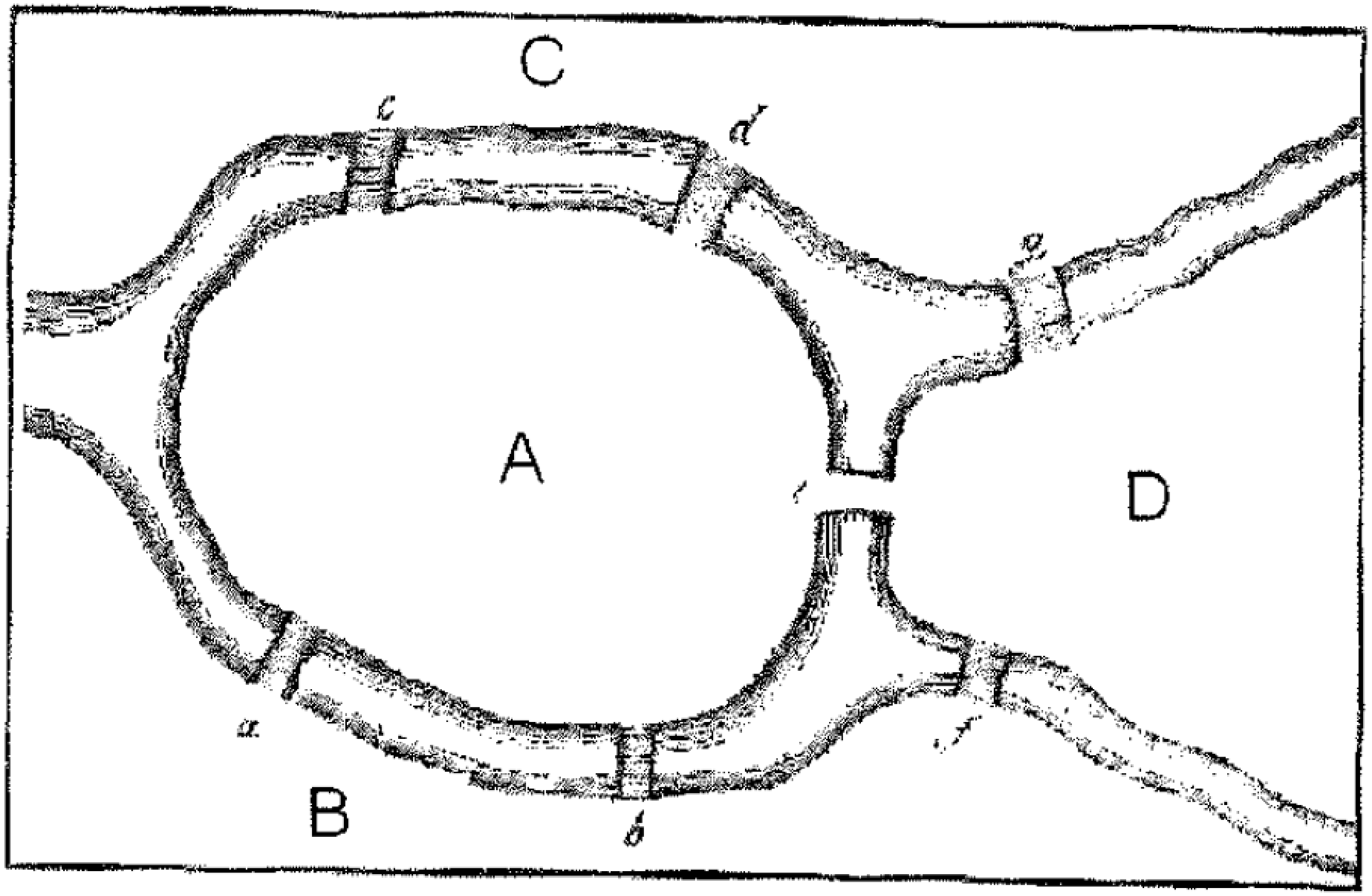
Now geometry counted as mathematics, but this puzzle did not seem to be geometric. There was nothing about the positions of the bridges that mattered.

But in a geometric diagram such as a map that displays regions, there are really two kinds of information: actual positions and shapes, but also connections.

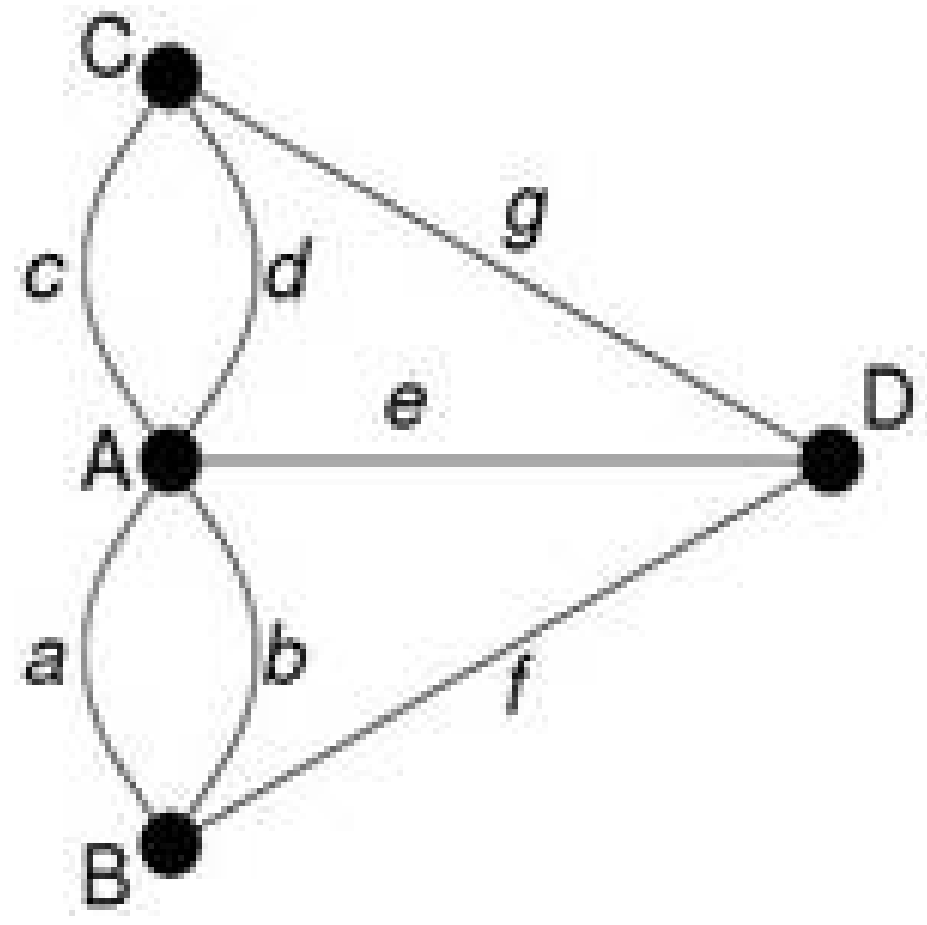
Euler looked at the city map and identified an abstract structure of objects (repesented by labeled points we'll call **nodes**) and connections (represented by lines connecting the points, which we will call **edges**) that contained all the information defining the problem, called the "graph" of the problem.

Euler had invented a kind of geometry without measurements, which opened up the new mathematical study of graph theory

Closeup of Bridges



Graph of Euler Bridge Problem



No Unnecessary Information

In Euler's graph, unnecessary information has been ruthlessly eliminated. The river is gone. The shape of the island is gone. The lengths of the bridges are not indicated.

We only have 4 points (which we will call nodes): A, B, C, D, which are connected by lines labeled a, b, c, d, e, f, g (which we will call edges).

The information in the graph is the bare minimum necessary to describe the problem.

We can use the graph to search for a solution. If we find a solution on the graph, we can transfer it to the map and plan our trip.

Adjacency of Euler Bridge Example

If we relabel the nodes A=1, B=2, C=3, and D=4, we can create an adjacency matrix that records how many bridges (edges) connect any pair of locations (nodes).

```
        1    2    3    4
     +----------------
  1  |  0    2    2    1
  2  |  2    0    0    1
  3  |  2    0    0    1
  4  |  1    1    1    0
```

Analyzing the Euler Bridge Problem

By reducing the bridge problem to a graph, Euler was able to focus on the features of the structure that would determine whether a solution was possible.

After some thought, Euler had an insight that allowed him to describe what a solution would look like, what made a solution possible or not, and how to determine whether any bridge graph had a solution.

Because Euler's analysis used an abstract model that could be applied to a wide class of cases, and his solution involved a logical argument, this was recognized as a new, legitimate branch of mathematics.

Solvable or Not?

After searching by hand for solutions, Euler changed his approach, and posed the following question:

Suppose I had a solution to the bridge problem. I could represent this as a list of the nodes I visit, in order, from first to last.

If I use every edge exactly once, and finish at the starting point, I call my solution an Euler circuit.

If I use every edge exactly once, but don't finish at the starting point, I call my solution an Euler path.

Now imagine that I have a solution to the Euler bridge problem, and I want to illustrate it with a pencil. I start by drawing the nodes A, B, C and D. Then I would place the pencil on my chosen starting node, and move from node to node, tracing out my route.

Is there something about such a drawing that indicates when the puzzle is solvable?

Is a Round Trip Possible?

Euler noticed that, when drawing a round trip, a pencil "arrives" at each node as many times as it leaves.

Put any set of nodes on a piece of paper, start at node A, move from one node to another in any way you want to, and come back to node A.

At any node B, the number of entering edges and exiting edges are equal. The total number of edges connected to any node B **must be even**.

The number of edges that touch B is called the degree of B.

For a graph to have an Euler circuit, every node must have even degree.

Is Any Solution Possible?

If we can't find a round trip solution, what about a solution that uses all the bridges, but doesn't end up at the starting position?

Think again about Euler's approach. If I have draw nodes A, B, C, ..., Z on paper, and start at A, and connect nodes in any way I like, and end at Z, then, except at A and Z, all the nodes must have even degree.

The starting node A will have one extra "exit" edge, and the ending node Z will have one extra "entrance" edge.

For a graph to have an Euler path, exactly two nodes must have odd degree.

Euler Bridge Problem Unsolvable

Now we can look at the graph for the Koenigsberg bridge problem, and determine the bad news.

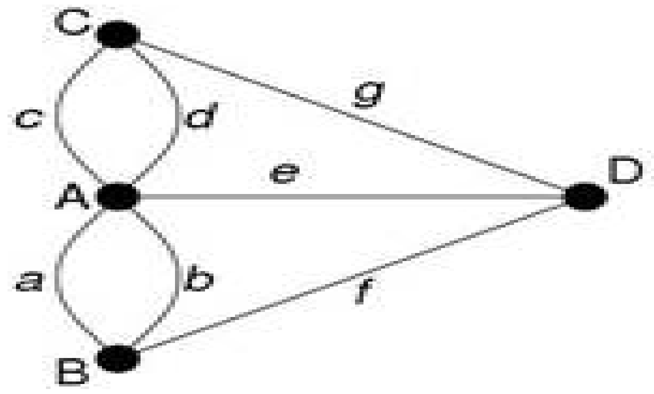
There are 4 nodes, and they have the following degrees (number of edges):

A: 5

B: 3

C: 3

D: 3



Therefore:

1) no Euler circuit (round trip) is possible, because there are nodes with odd degree.

2) no Euler path (use all the edges, but final node is not the same as starting node) is possible, because there are more than two nodes with odd degree.

Only a Partial Answer

Euler solved the Koenigsberg bridge problem by showing that there was no solution.

He did more than that: he showed how to look at **any graph** G and immediately say that:

1) an Euler circuit is possible;

2) an Euler path is possible;

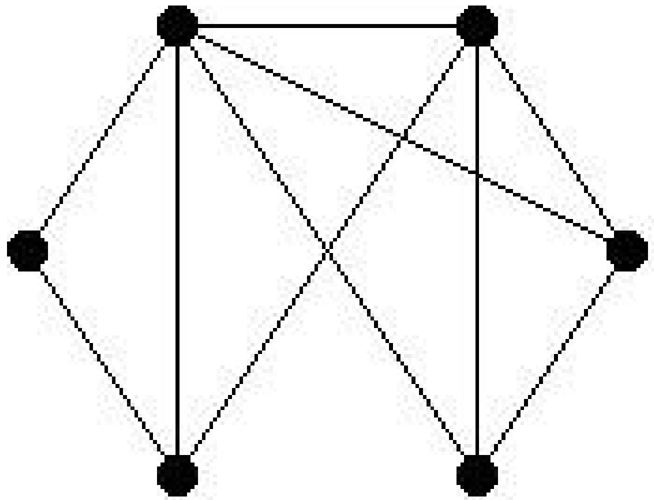
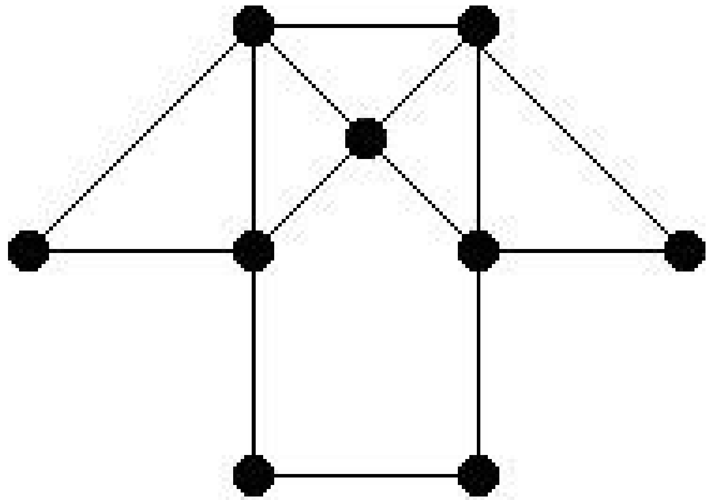
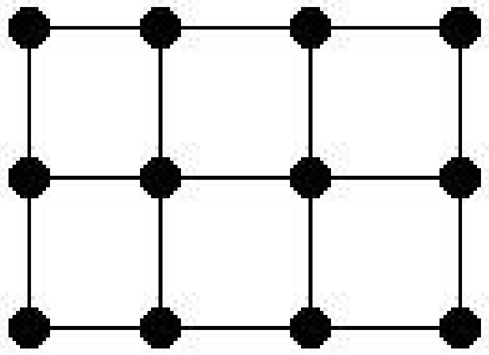
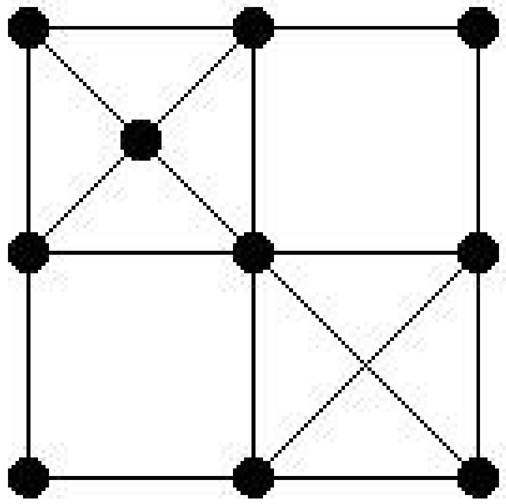
3) no Euler circuit or path is possible.

But Euler did not show:

1) that a circuit or path is not just possible, but *guaranteed*.

2) how to find a circuit or path.

Fleury’s Algorithm

A Guaranteed Answer

For a small graph, we may be able to find a suitable Euler circuit or path by trial and error, but to answer the problem confidently, we need an algorithm.

One such method is known as **Fleury's algorithm**. If a graph has exactly zero nodes of odd degree, the Fleury's algorithm will find an Euler circuit. If a graph has exactly two nodes of odd degree, it is guaranteed to find an Euler path.

In order to use Fleury's algorithm, we must define a new concept: when an edge of a graph is a bridge.

A Bridge

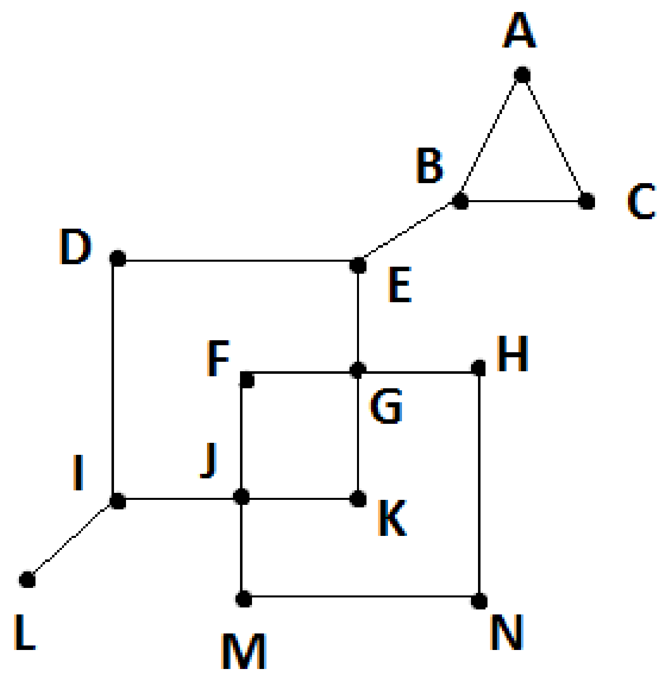
A graph is "connected" if there is a path between every pair of nodes A and B.

Any edge whose removal transforms a connected graph into a disconnected graph is called a bridge.

In this graph,

edge (B,E) is a bridge,

edge (I,L) is a bridge.



Fleury’s Algorithm

1) The graph G must be connected.
2) The graph G must have exactly 0 or 2 nodes of odd degree.
3) If G has 2 nodes of odd degree, start at one of them; otherwise start anywhere.
4) From the current node, choose an edge, using a non-bridge edge unless no other choice is available. Use the edge to move to a new node, and delete the edge.
5) Continue until you run out of edges.

The Graph Changes

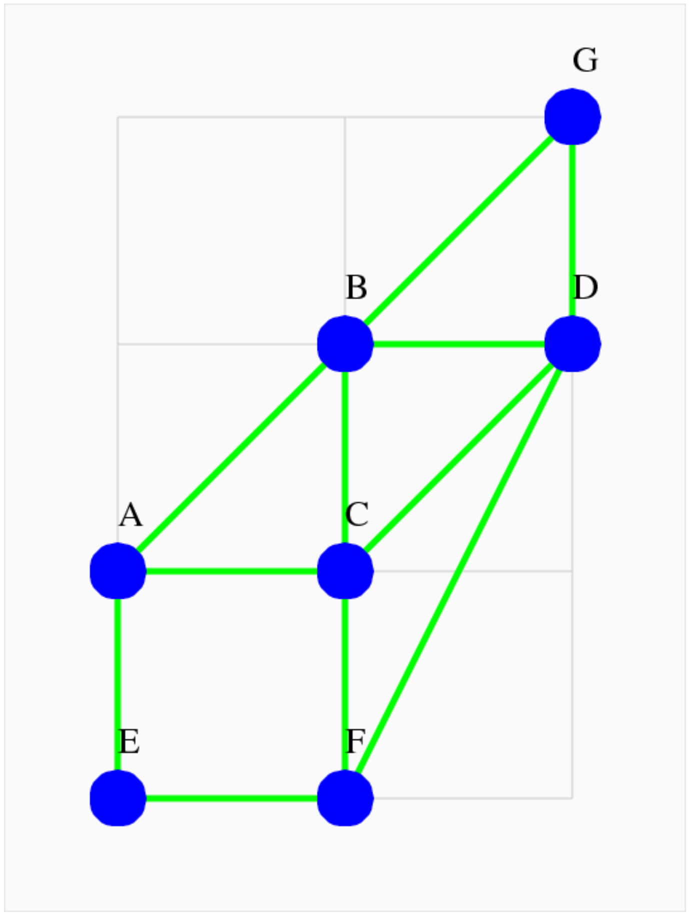
While using Fleury's algorithm, we are deleting edges.

If a node no longer has any active edges, we also delete that node.

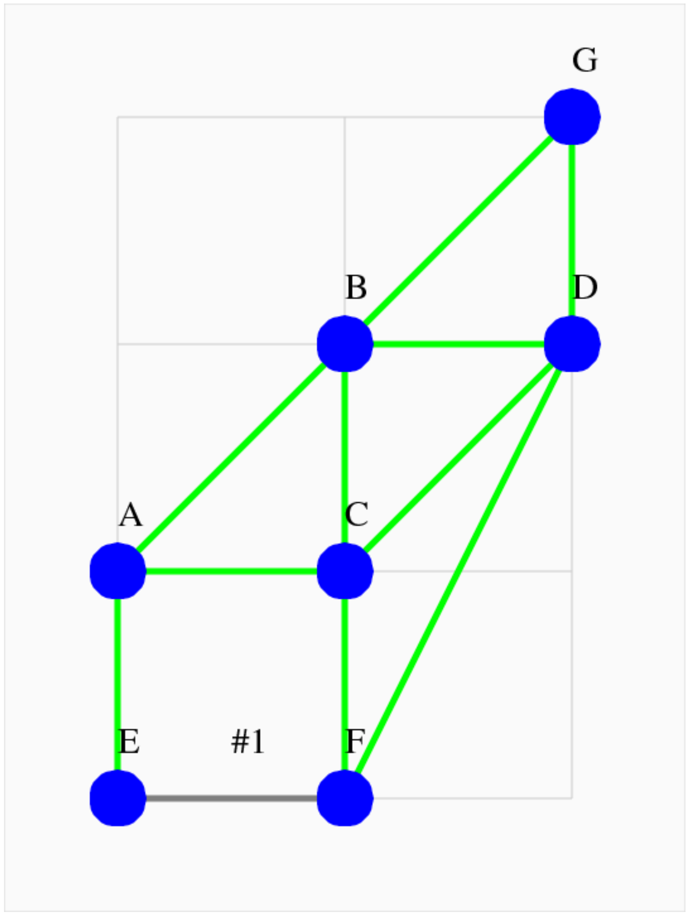
Because the graph changes on each step, we have to be careful to check whether an edge has suddenly become a bridge.

In the following demo, we begin with green edges and blue nodes. A deleted edge or node becomes gray. Bridges are highlighted in red.

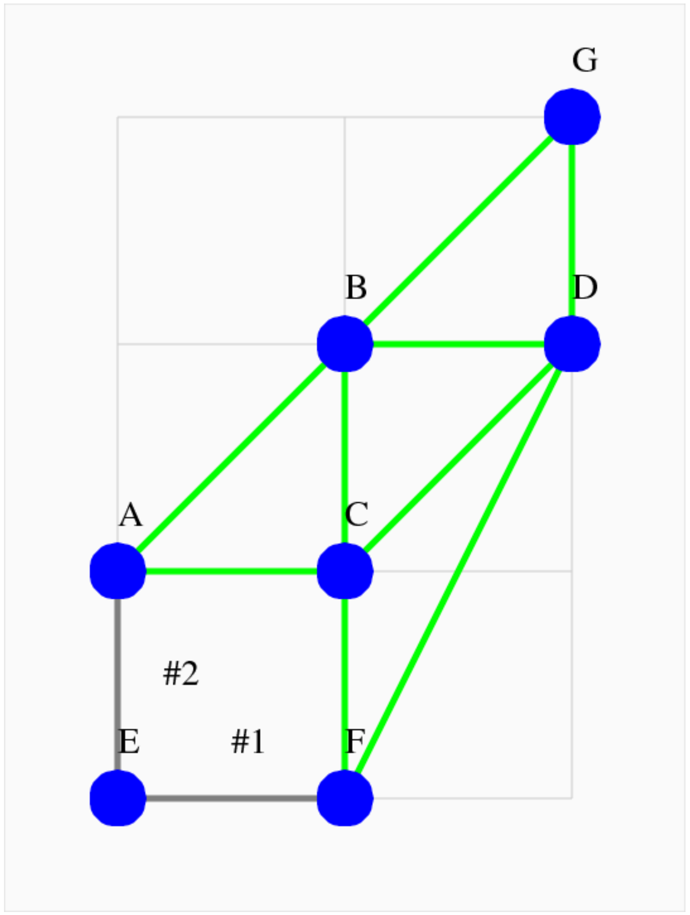
Fleury Demo



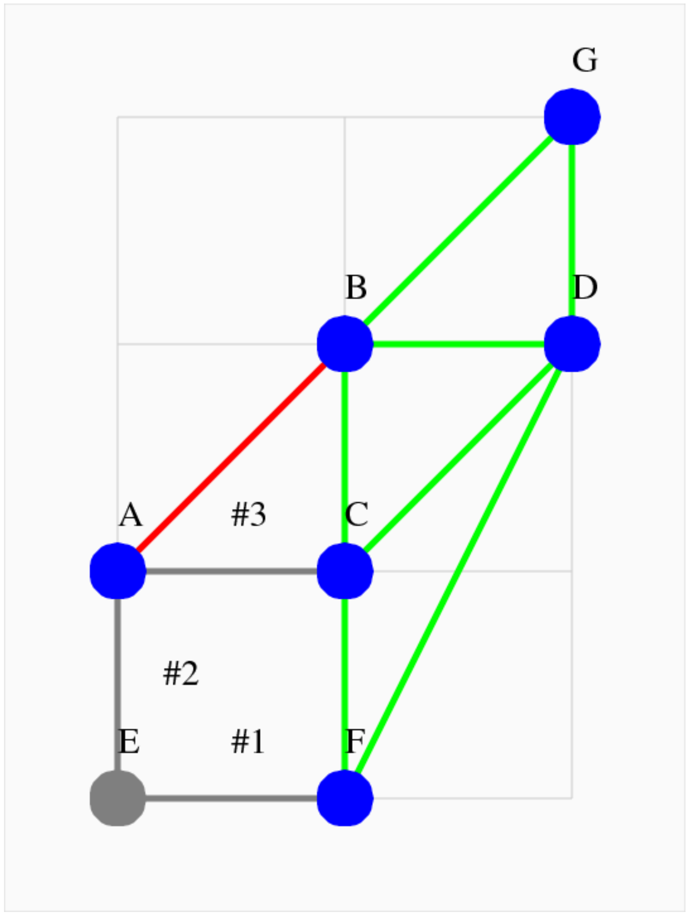
1: Start at F, move to E



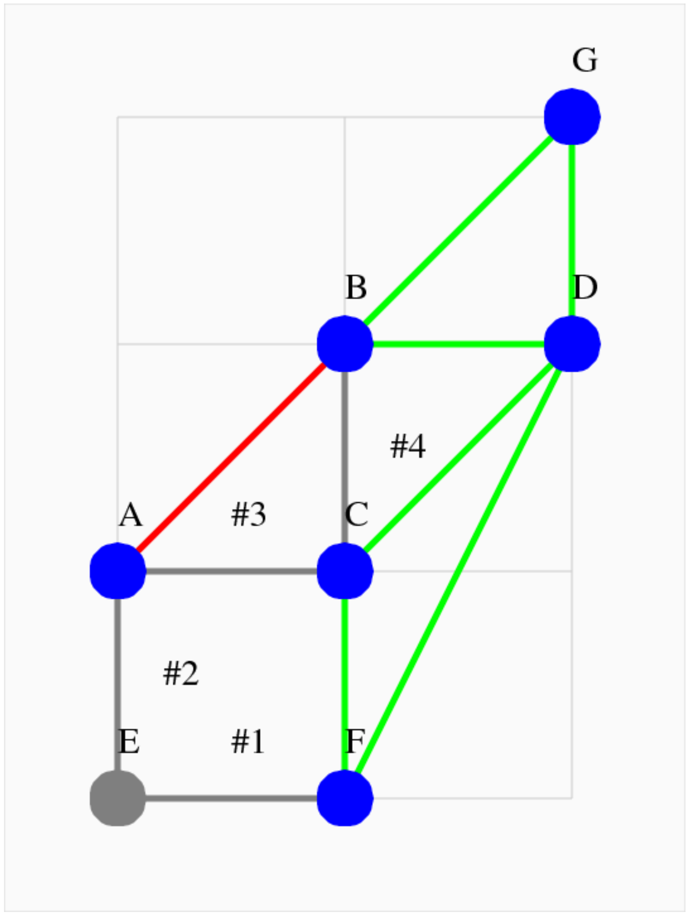
2: Move to A



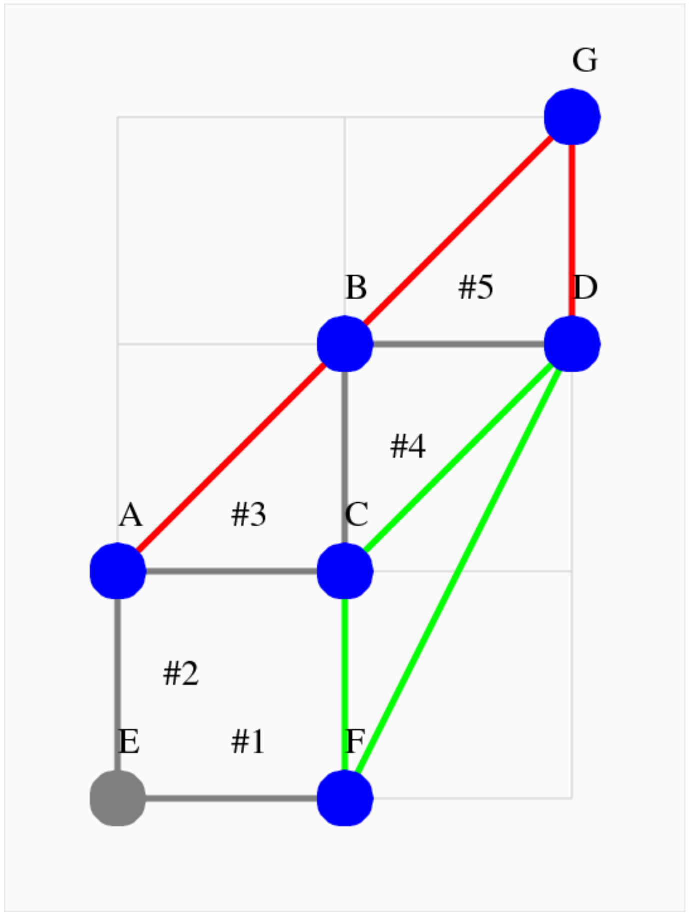
3: Move to C (A-B is a bridge now)



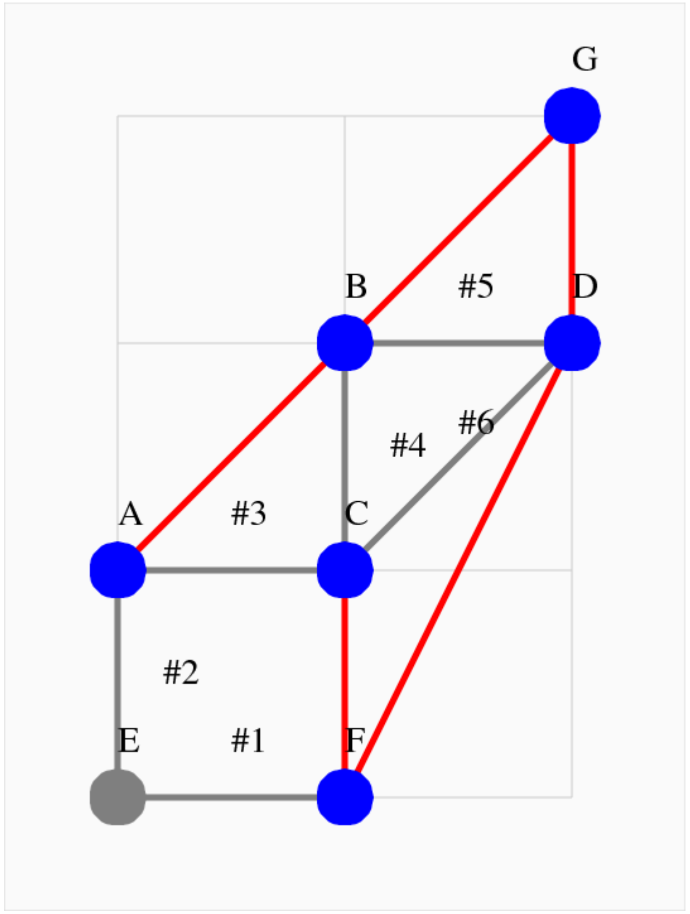
4: Move to B



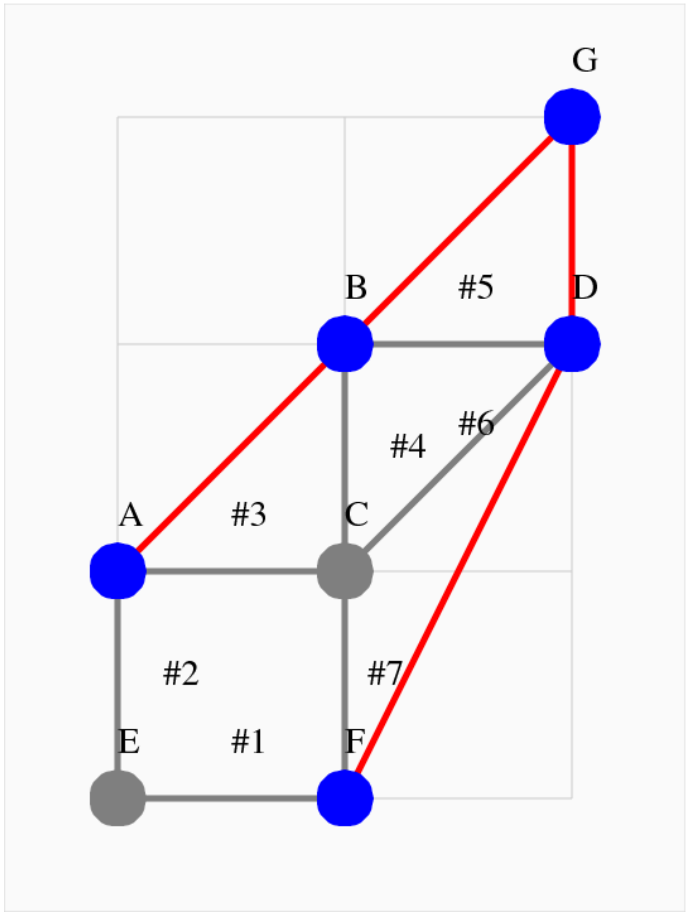
5: Move to D (now 3 bridges!)



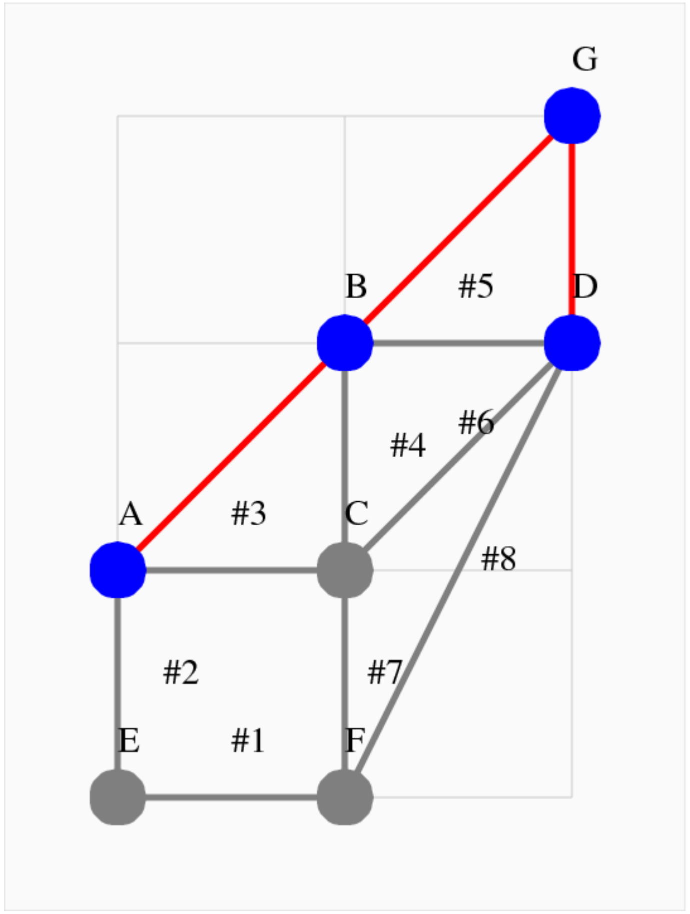
6: Move to C (Now 5 bridges)



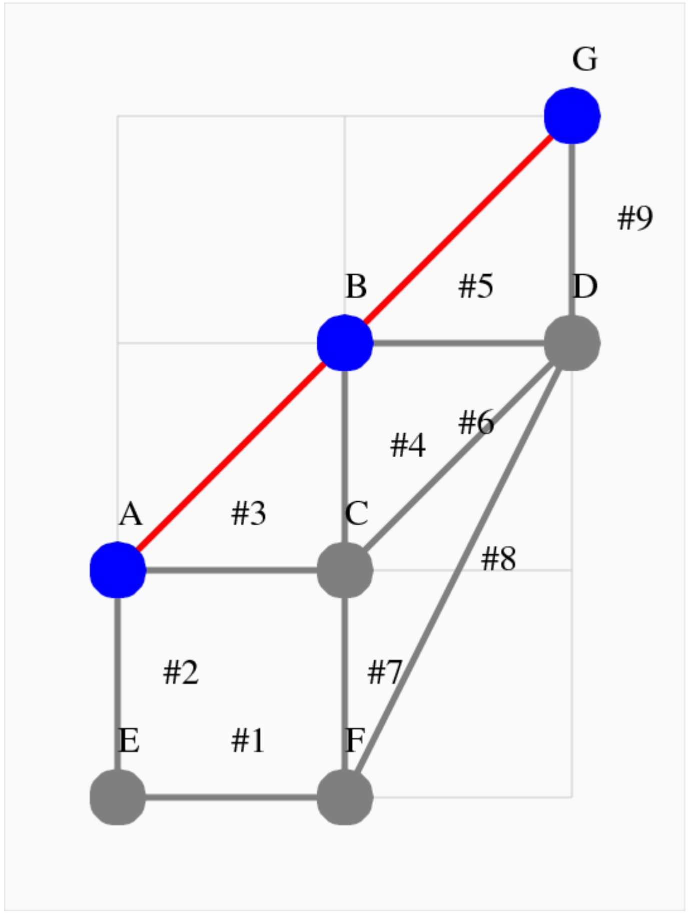
7: Move to F



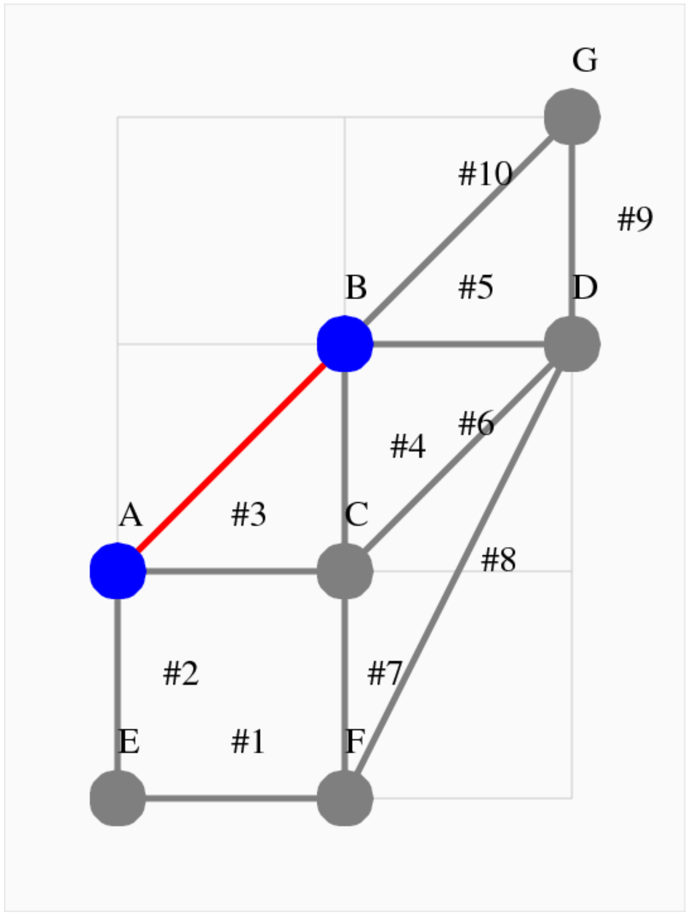
8: Move to D



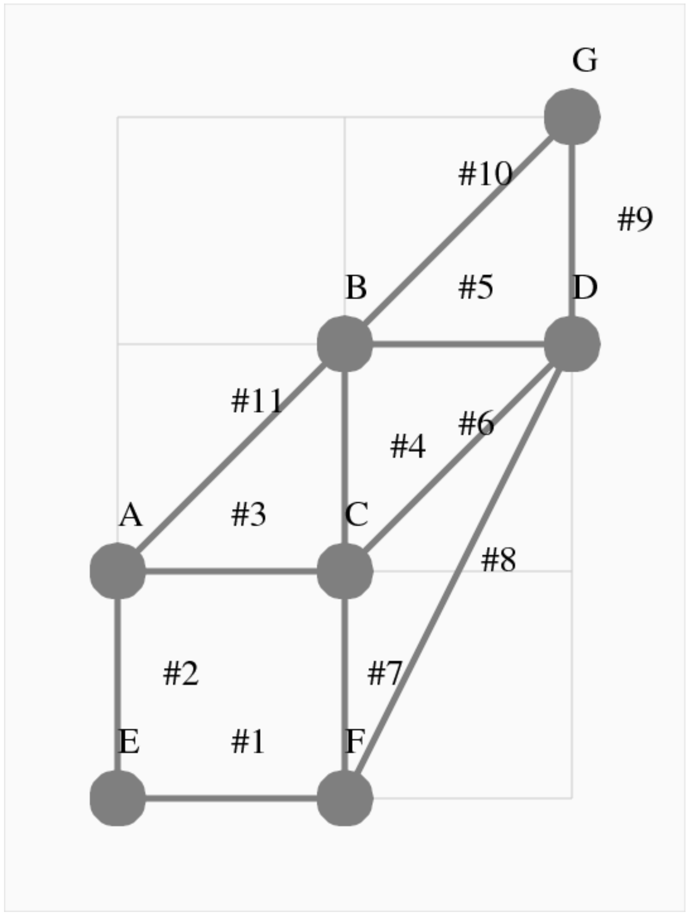
9: Move to G



10: Move to B

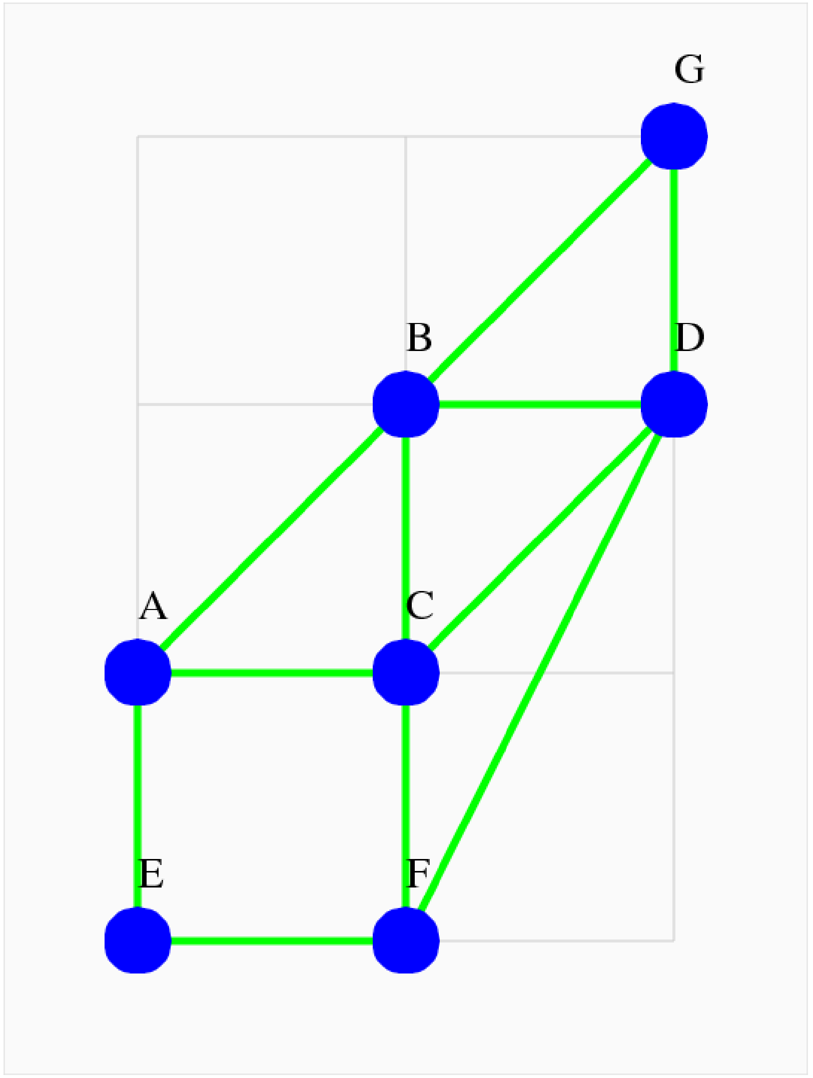


11: Move to A, done!

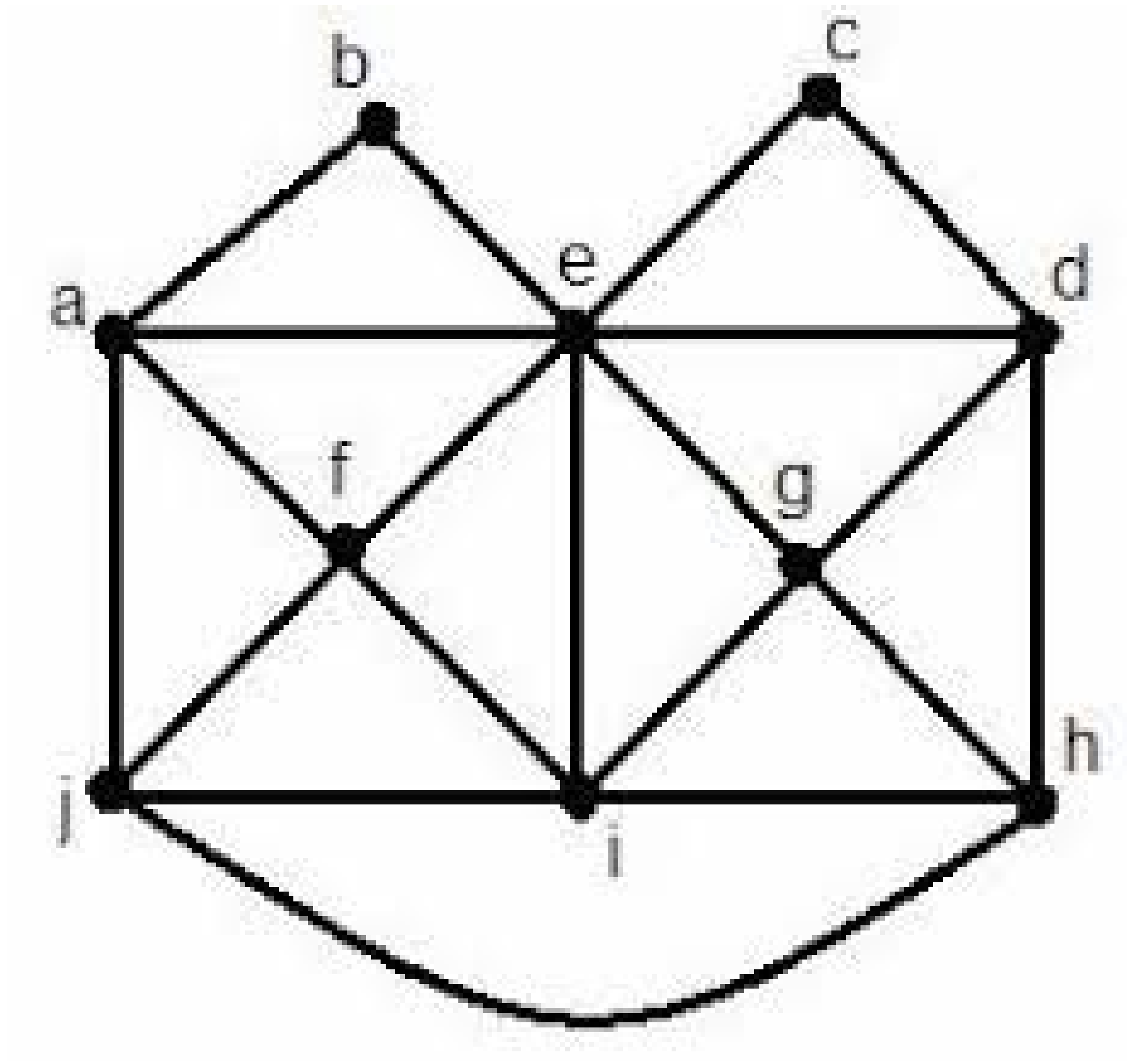


We Have an Euler Path

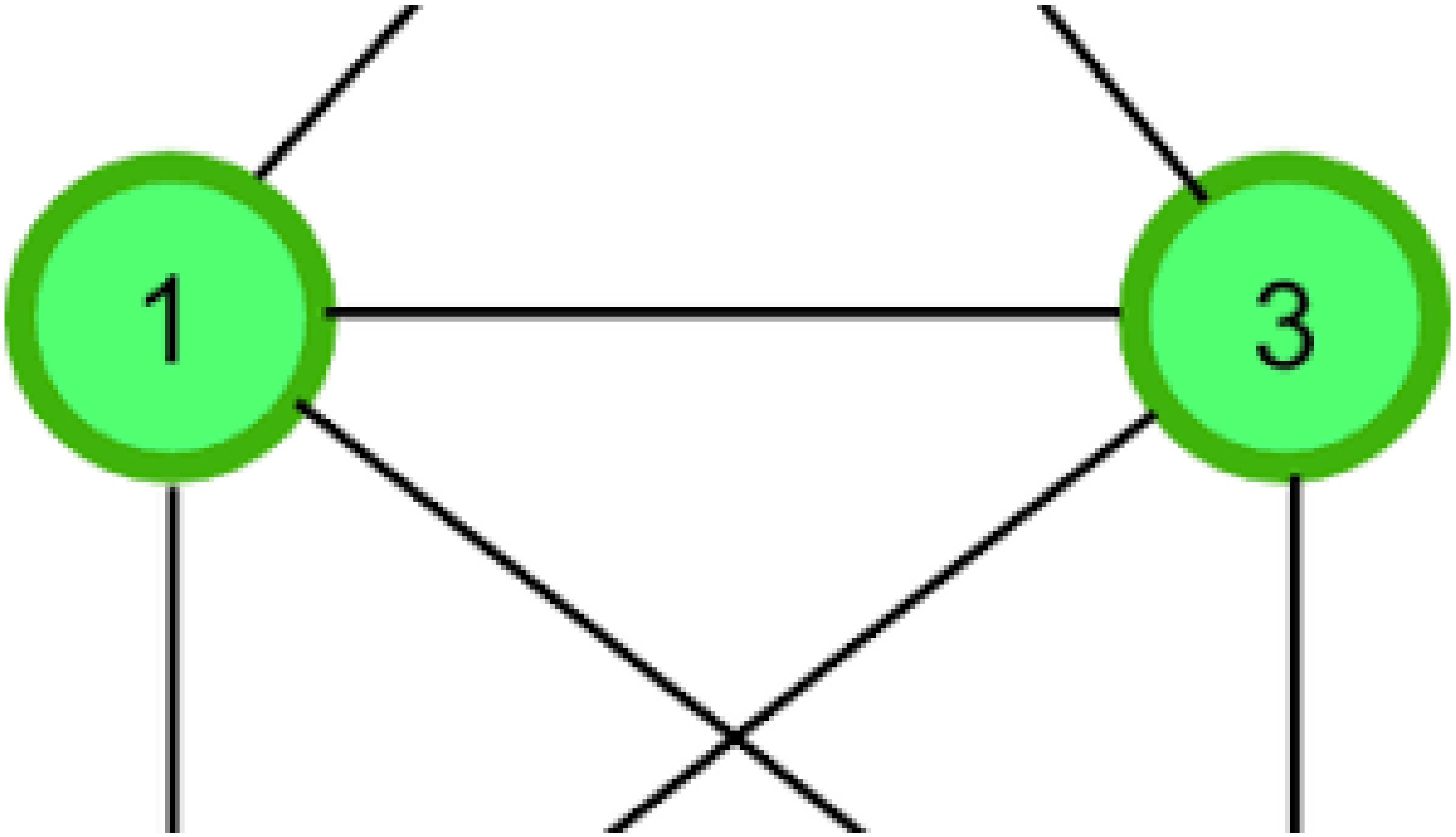
Fleury result: Euler path is FEACBDCFDGBA



A Second Example



The Adjacency Matrix



Graphs and Multigraphs

If we want to set up and solve graph problems on a computer, we need to decide on a way to represent a graph using the tools that MATLAB provides.

We have seen that a graph can be described as a set of N nodes, and a number of edges which connecting pairs of nodes.

Notice that in the Euler Bridge example, in some cases there were two edges connecting the same pair of nodes.

To simplify our work, from now on we will assume that, in a graph, there can only be at most one connection between nodes. We will reclassify the Euler bridge example as a multigraph, and leave such cases alone for now.

The Adjacency Matrix

Suppose that we have a graph G, consisting of N nodes, labeled 1 through N.

We suppose that there may be a single edge connecting a pair of distinct nodes I and J.

The simplest way to store such information in a computer defines an NxN adjacency matrix Adj:

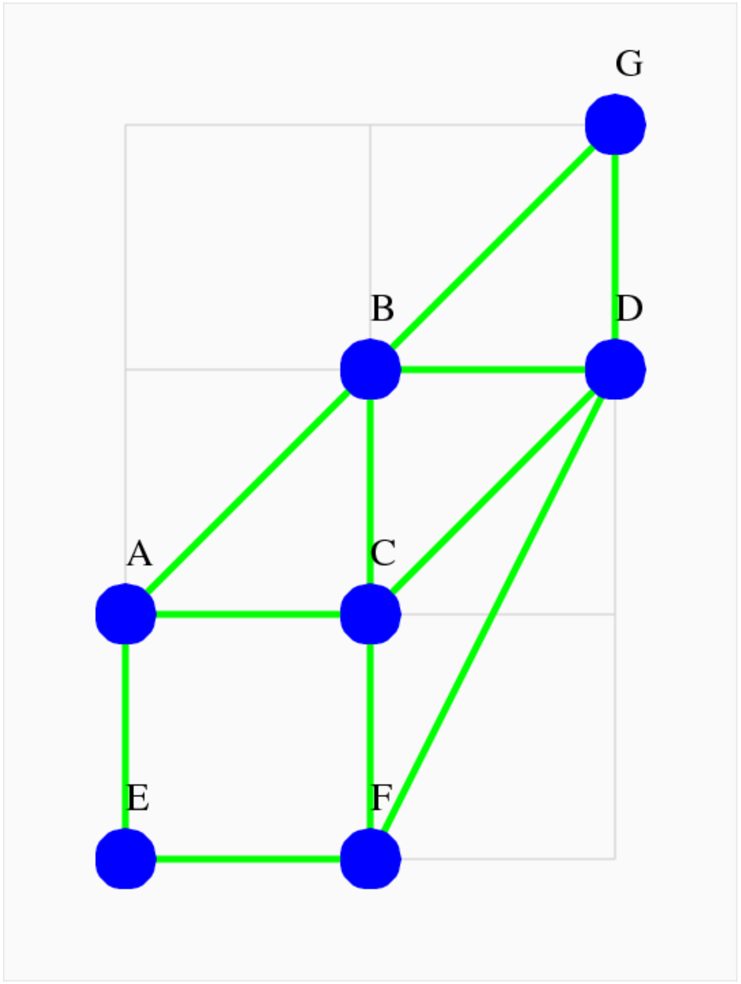
Adj(I,J) = 1, if an edge connects nodes I and J

0, otherwise

We can also think of Adj(I,J) as storing TRUE or FALSE.

Fleury Example Adjacency Matrix

```
     A  B  C  D  E  F  G
   +--------------------
A  | .  1  1  .  1  .  .
B  | 1  .  1  1  .  .  1
C  | 1  1  .  1  .  1  .
D  | .  1  1  .  .  1  1
E  | 1  .  .  .  .  1  .
F  | .  .  1  1  1  .  .
G  | .  1  .  1  .  .  .
```



Adjacency Matrix Properties

For a graph G, the adjacency matrix Adj:

* is a square matrix, NxN;

* is a 0/1 or logical (true/false) matrix;

* has a zero diagonal: A(I,I) = 0;

* is symmetric: Adj(I,J) = Adj(J,I).

Now that we have a way of putting a graph into the computer's memory, we have to figure out how to teach the computer to solve various problems associated with a graph.

Graph Theory Questions

If the computer has the adjacency matrix of a graph, it "sees" the same things we do. Can the computer:

* Find a path from node 1 to node 6?

* Tell if the graph is "connected" (all one piece)?

* Find a path that visits every node exactly once?

* Find a path that uses every edge exactly once?

* Determine how many steps it takes to reach a particular node?

How Far from node E?

Suppose we are standing at node E in the Fleury example graph, and wonder how far away the other nodes are, that is, how many steps it would take to reach each one.

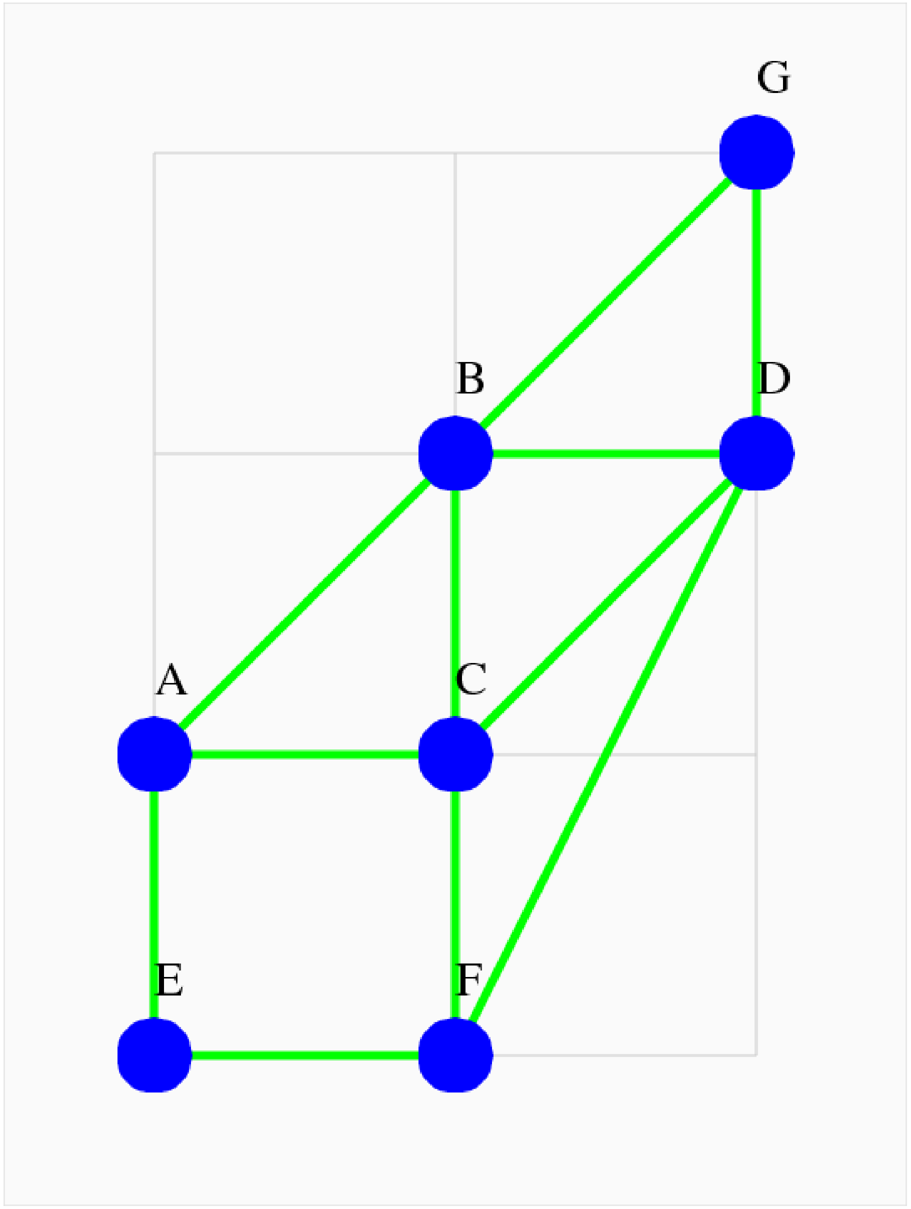
The graph’s adjacency matrix Adj can tell us.

Start with a (column) vector **v0** containing a 1 in position 5, corresponding to us standing at node E.

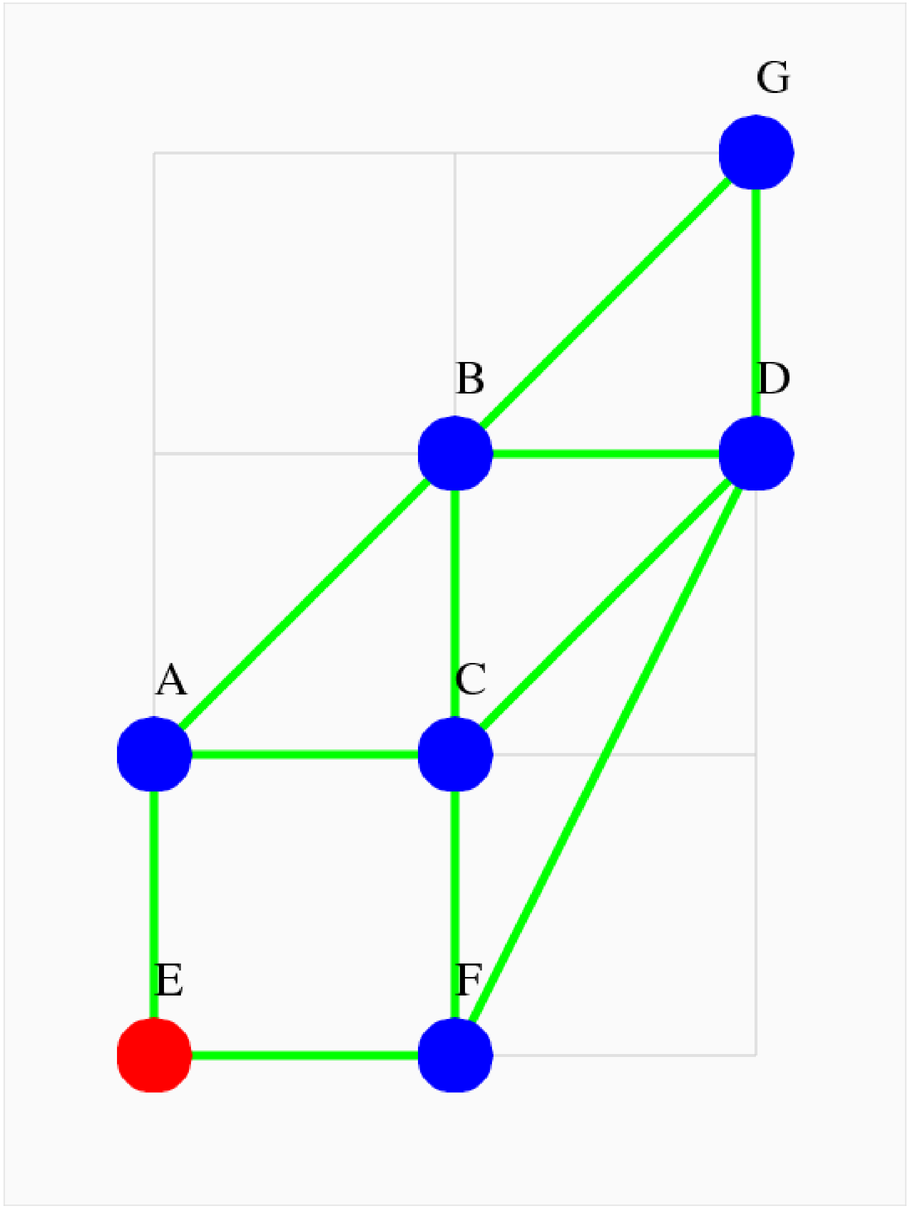
Compute v1 = Adj * v0. This shows us all the places we could be after one step.

Computing v2 = Adj * v1 shows us how many places we can be after two steps (**and** how many different ways we could get there!)

Our Example Graph



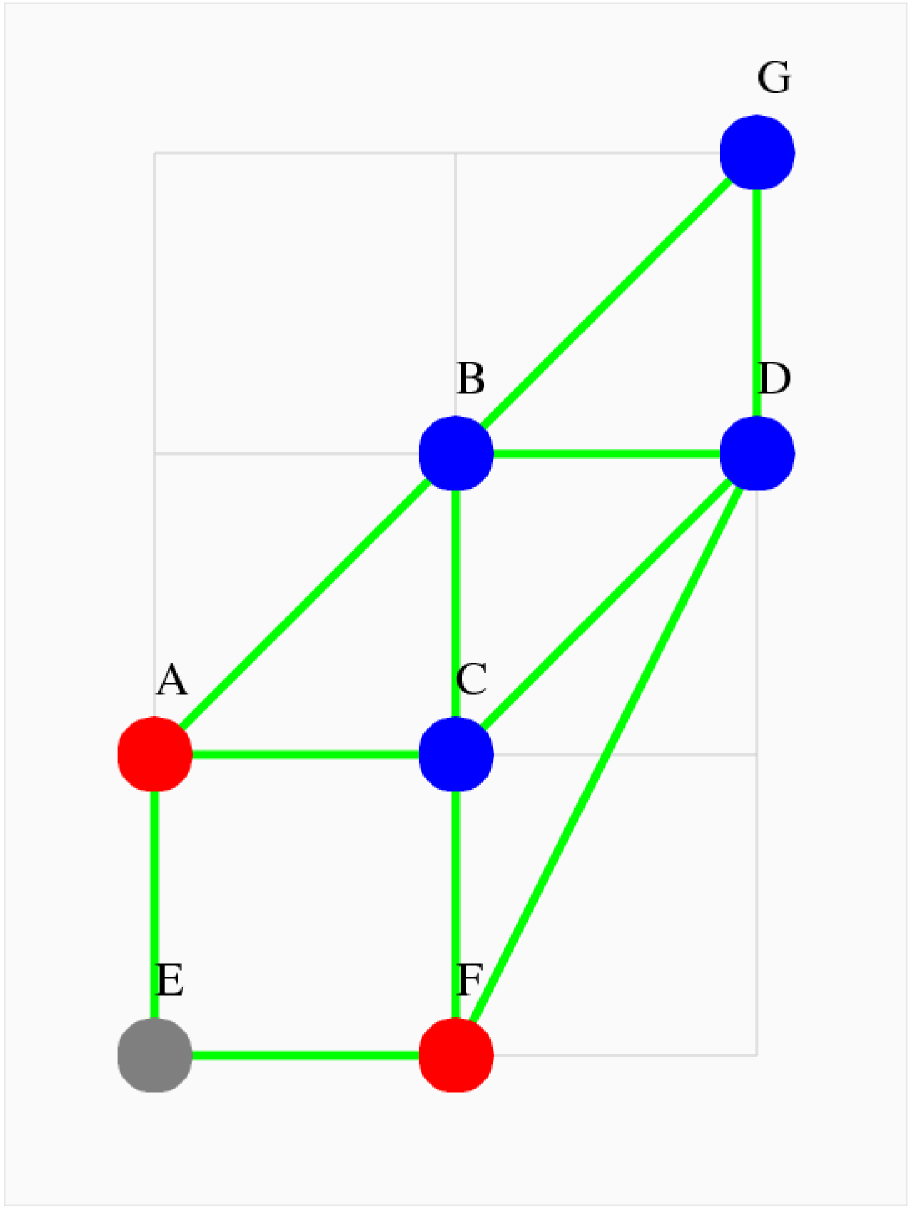
Start at Node E



Where Can I Be After 1 Step?

```
                   Adj  *  v0  =  v1
-------------------     --     --
0  1  1  0  1  0  0      0      1
1  0  1  1  0  0  1      0      0
1  1  0  1  0  1  0      0      0
0  1  1  0  0  1  1  *   0  =   0
1  0  0  0  0  1  0      1      0
0  0  1  1  1  0  0      0      1
0  1  0  1  0  0  0      0      0
```

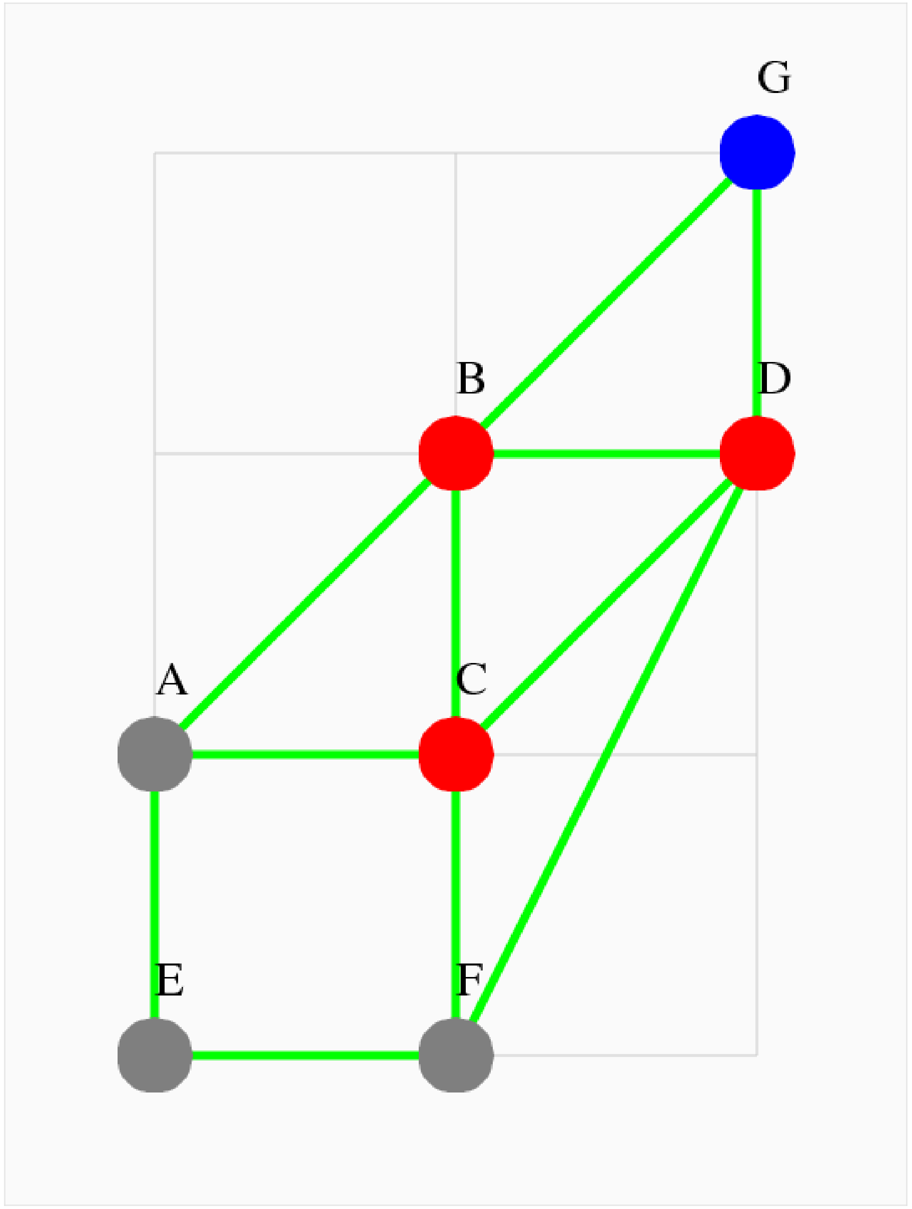
Step 1: A & F



Where Can I Be After 2 Steps?

```
                   Adj  *  v1  =  v2
-------------------        --     --
0  1  1  0  1  0  0         1      0
1  0  1  1  0  0  1         0      1
1  1  0  1  0  1  0         0      2
0  1  1  0  0  1  1  *      0  =   1
1  0  0  0  0  1  0         0      2
0  0  1  1  1  0  0         1      0
0  1  0  1  0  0  0         0      0
```

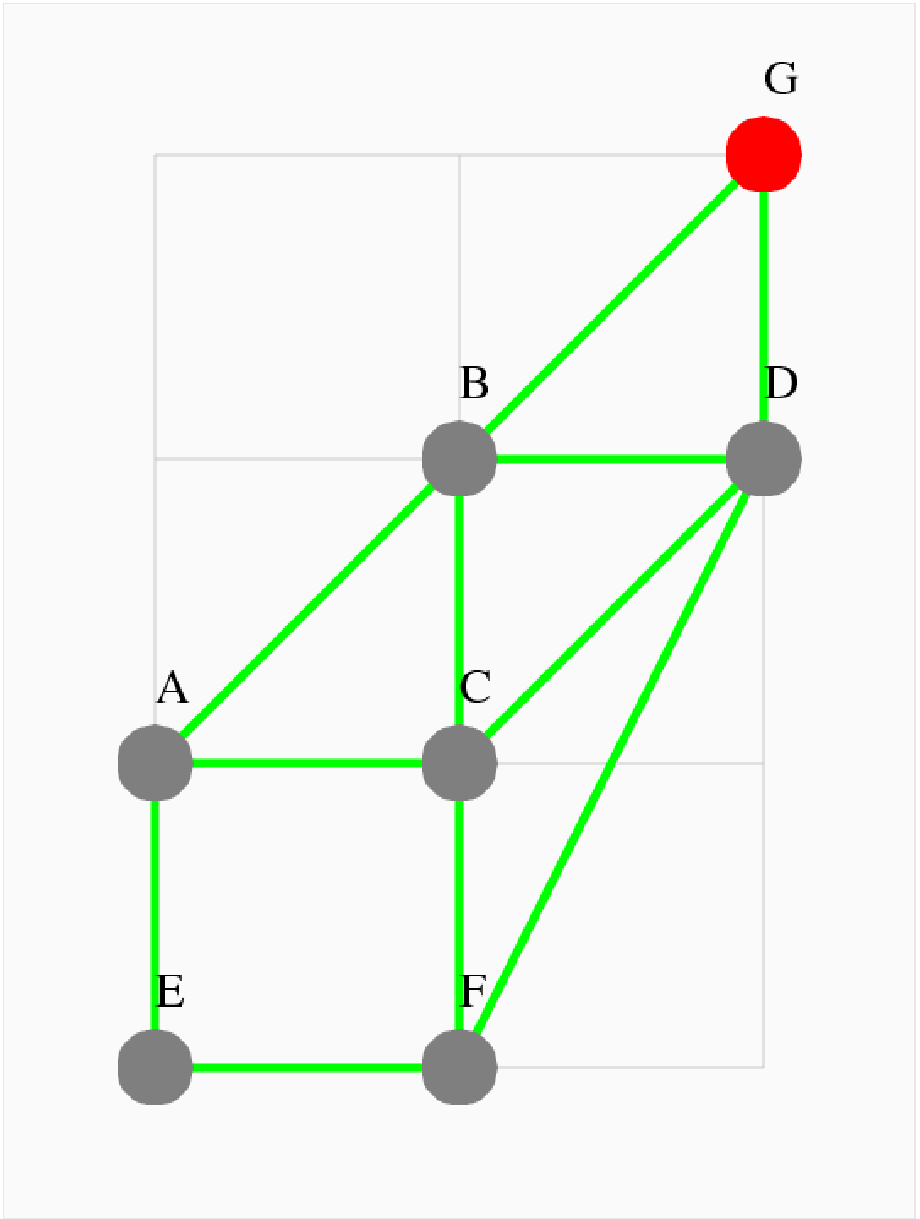
Step 2: B, C, & D



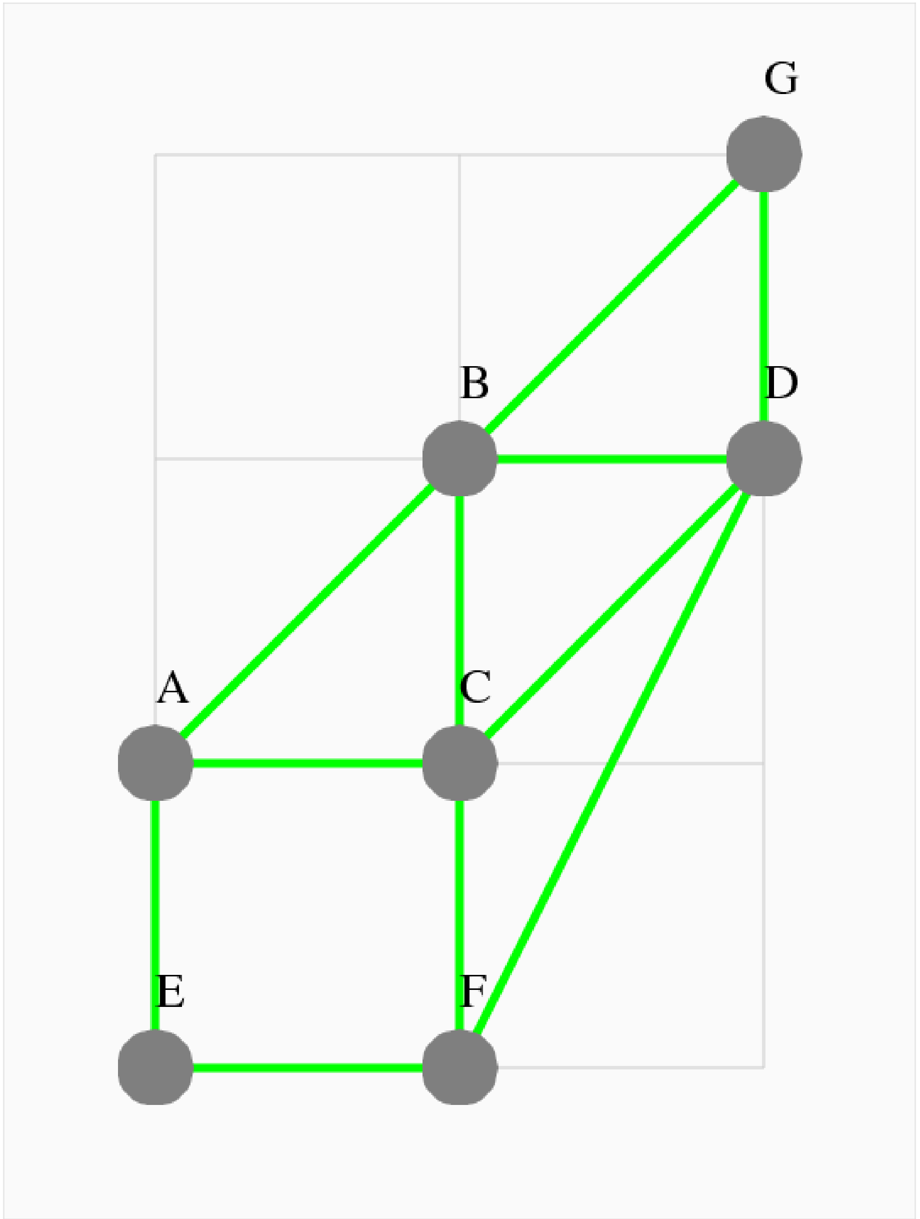
Where Can I Be After 3 Steps?

```
                   Adj  * v2 = v3
-------------------     --   --
0  1  1  0  1  0  0      0    5
1  0  1  1  0  0  1      1    3
1  1  0  1  0  1  0      2    2
0  1  1  0  0  1  1  *   1 =  3
1  0  0  0  0  1  0      2    0
0  0  1  1  1  0  0      0    5
0  1  0  1  0  0  0      0    2
```

Step 3: G



All Nodes Reached



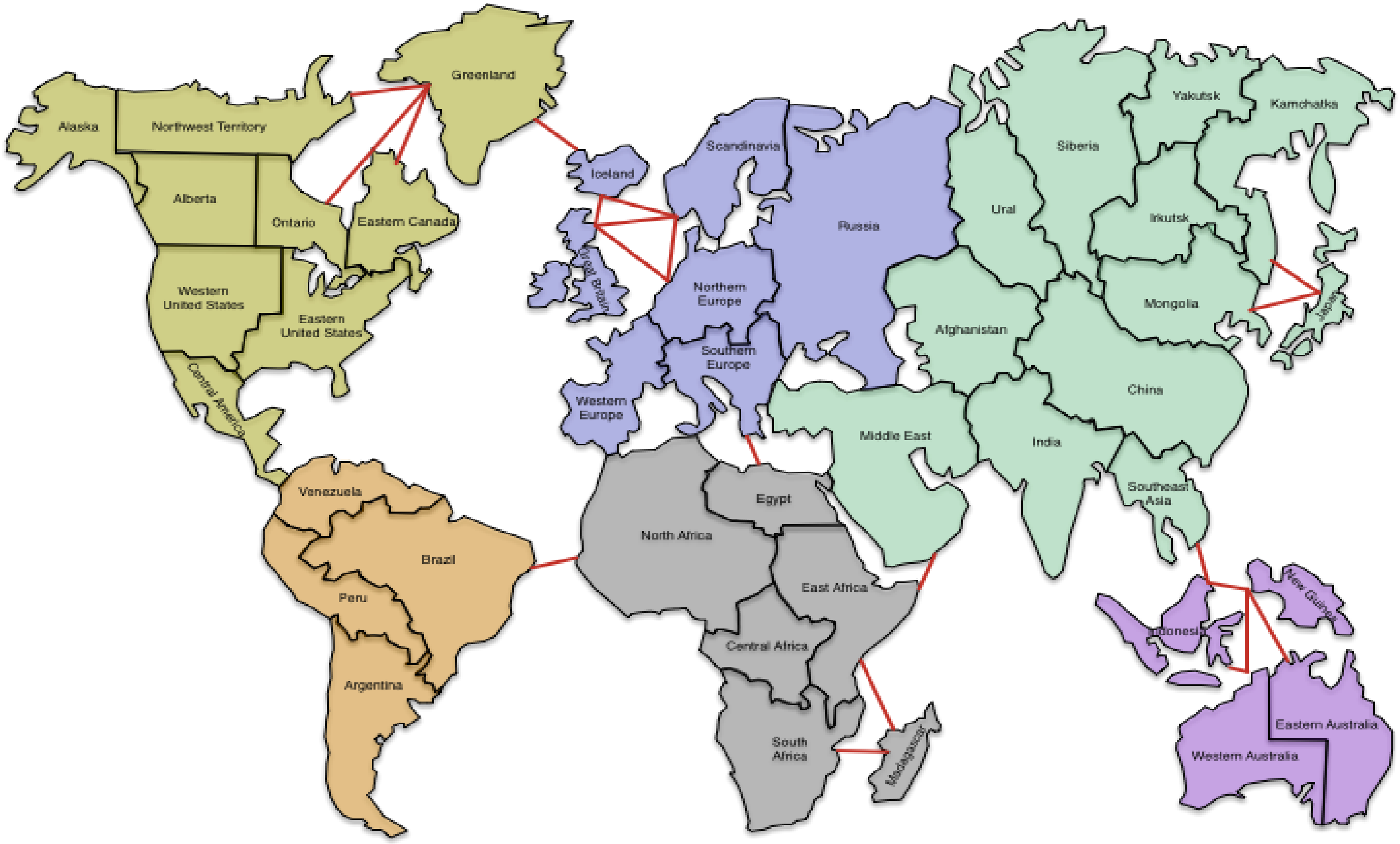
Distance from Node E

So, using the adjacency matrix Adj, and a column vector **v0** that gives our initial position at E, we can use repeated matrix multiplication to “walk” from E to the other nodes.

By keeping track of the first time we reach each node, we have a distance map:

```
From E to:  A  B  C  D  E  F  G
Steps:      1  2  2  2  0  1  3
```

The RISK Board



The RISK Example

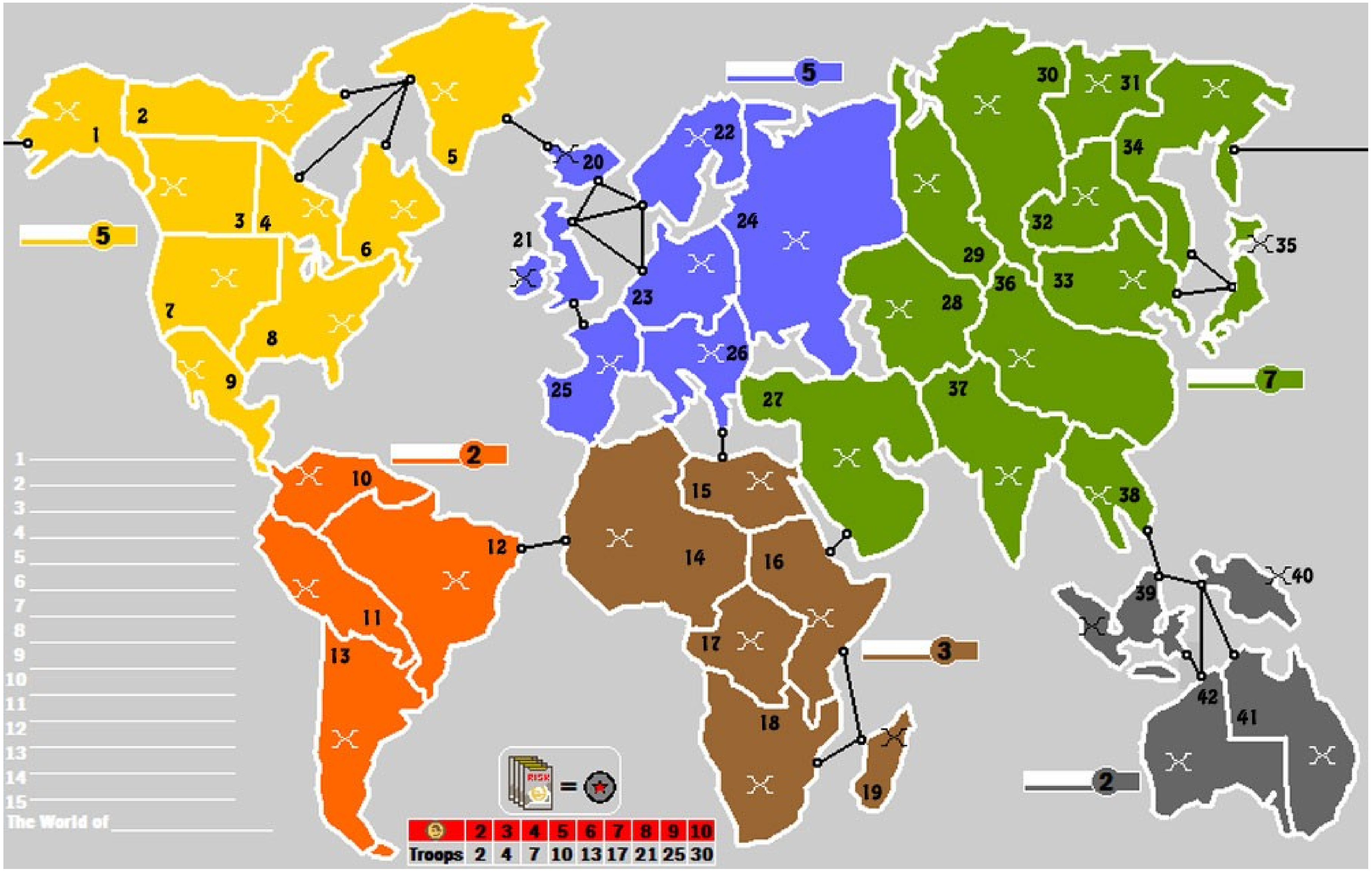
RISK is a multiplayer game of strategy played on a global map divided into N=52 regions.

A player who has armies in one region can use them to attack any neighboring region.

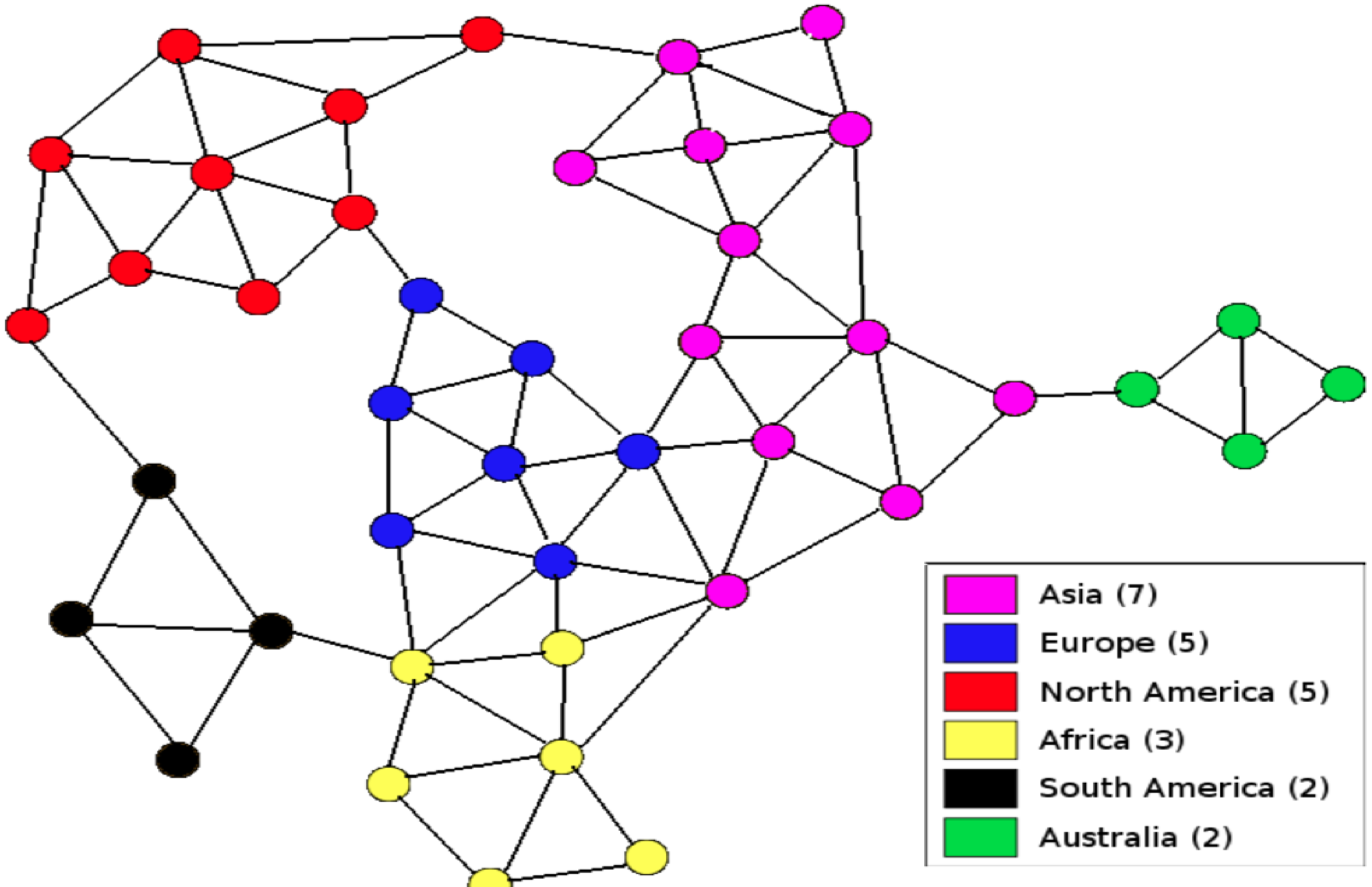
Thus, the essential information about the game board can be described by an adjacency matrix.

We have to assign a number 1 through N to each region, set up an NxN adjacency matrix A, and store a 1 in Adj(I,J) and Adj(J,I) if regions I and J are adjacent.

RISK Game Board Numbered



The RISK Graph



The RISK Adjacency Matrix

There are 42 regions on the RISK game board, so the adjacency matrix is 42x42. It's hard to print out the numerical values and see the pattern.

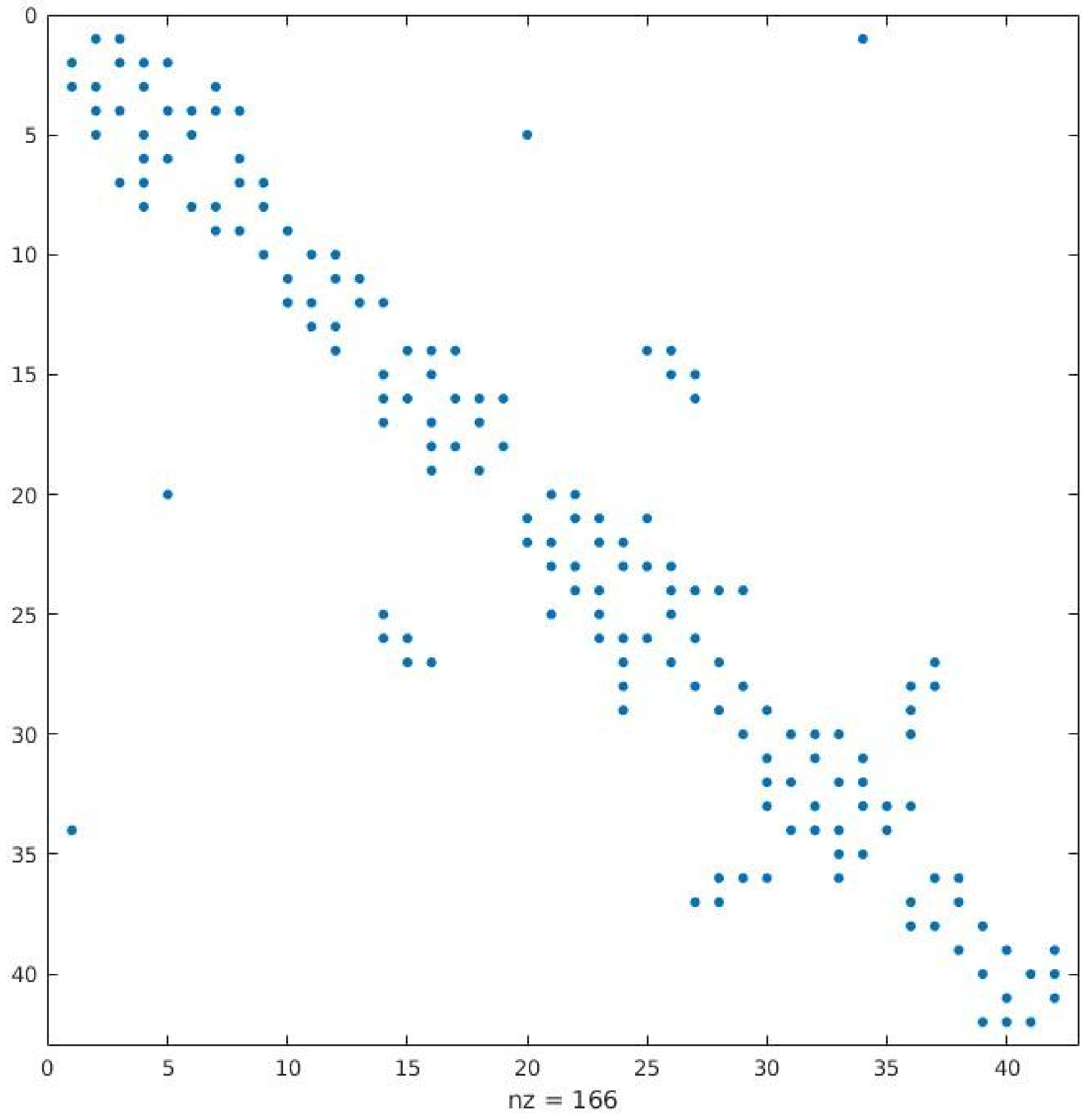
However, the MATLAB command

```
spy ( A )
```

will read a matrix A and print out a picture showing the location of nonzero entries. This is enough information to check the RISK matrix.

We want to make sure it is symmetric; also, the numbering of the regions should keep most of the nonzeros near the diagonal.

spy(ADJ) for RISK



Using the RISK Adjacency Matrix

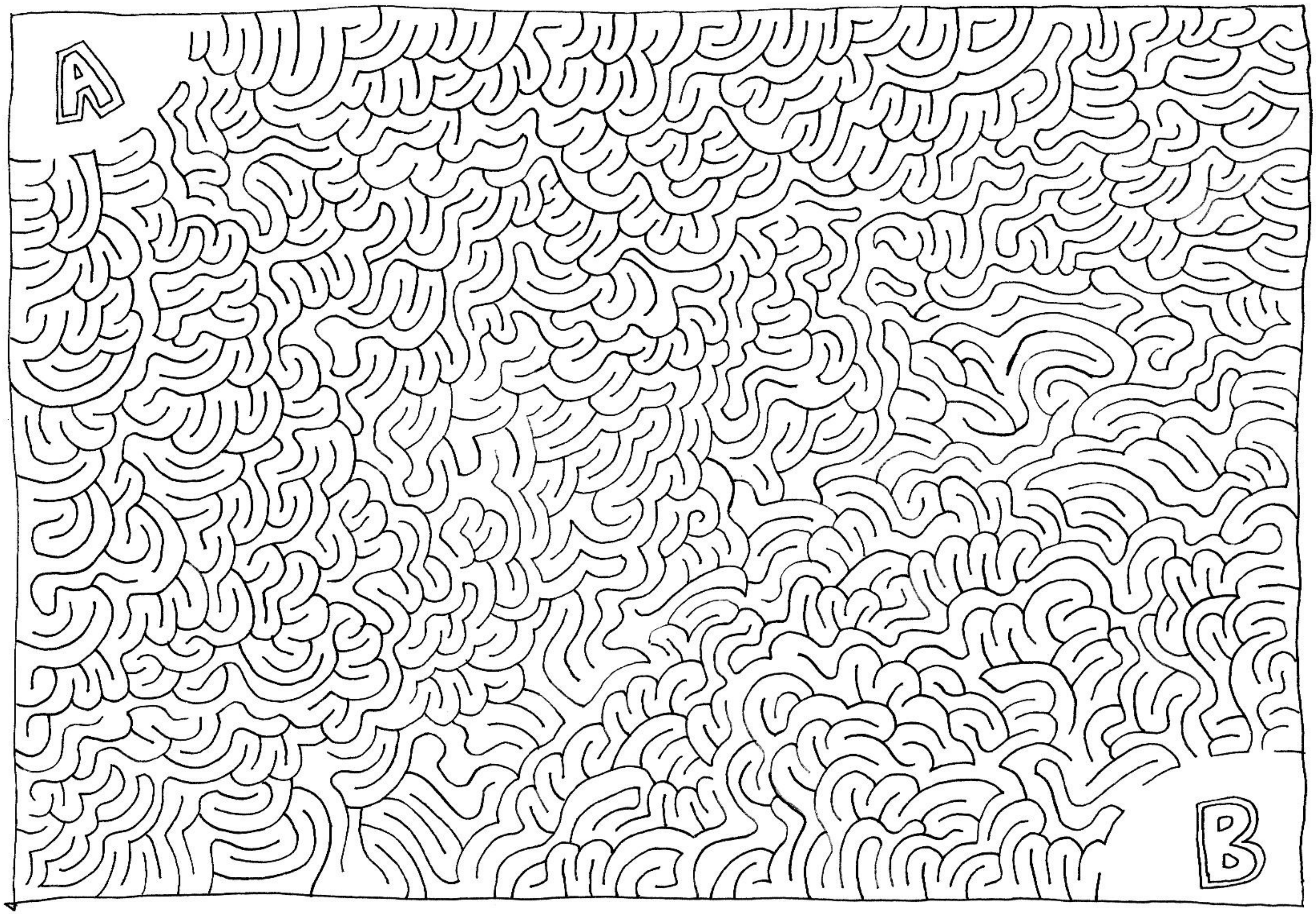
If we have created the adjacency matrix A for the game of RISK, assigning numbers to each region, then we have set up a substantial amount of the information needed to play the game.

In particular, if player "John" wants to attack player "Karen's" region 17 from his region 32, the value of Adj(17,32) will be 1 if region 32 is adjacent to 17, meaning an attack is possible.

Of course, we would need more information to determine that region 32 belongs to John, 17 to Karen, and how many armies each player has on their regions. But the adjacency matrix is able to store the connection information that human players refer to by looking at the game board.

As we have seen, the RISK adjacency matrix could also be used to determine the distance between one region and any other region on the map.

Can You Get From A to B?



Long Distance Connections

A graph can be thought of as a collection of local connections between pairs of nodes.

But it's natural to think of a graph as a sort of city-to-city road map that we can use to plan a trip.

Looking at a road map, we know we can drive from Blacksburg to Denver, using a sequence of city-to-city connections.

We also know that there is no way to drive to Paris.

The Connecting Path Problem

We are given nodes A and B, and an adjacency matrix Adj. Our task is to determine whether or not there is a path connecting A to B.

A path connecting A to B is a sequence of pairs of nodes such that:

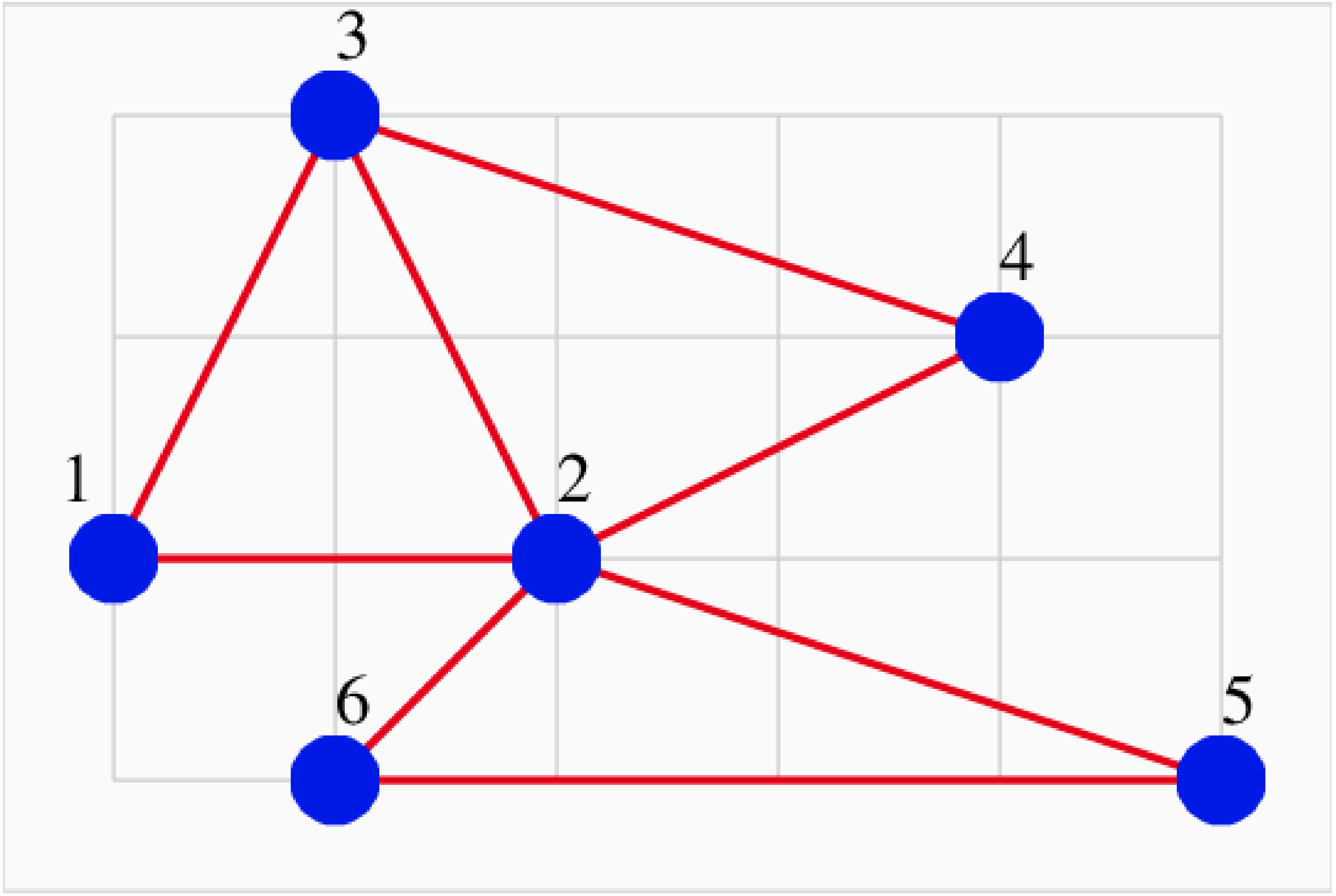
* The first pair begins with A: (A,*).

* The last pair ends with B: (*,B).

* Each pair (C,D) in the path is connected, that is, Adj(C,D) = 1.

The length of the path is the number of pairs in the sequence.

Graph Example #2



Graph #2: Paths from 3 to 5

In graph #2, here are some examples of paths from node 3 to node 5:

```
(3,1), (1,2), (2,5)                     length 3
(3,1), (1,2), (2,6), (6,5)              length 4
(3,4), (4,2), (2,6), (6,5)              length 4
(3,1), (1,3), (3,1), (1,2), (2,5) length 5
```

Is There a Path from A to B?

Since the adjacency matrix describes the graph, it really must be possible to determine a path from A to B.

The problem is that, for each node, the adjacency matrix only tells us where we can get to in one step.

Supposing we start at A, our list of "reachable nodes" begins with just A.

The adjacency matrix will tell us all the places we can reach from A. Perhaps this is nodes F and R.

Now our list of reachable nodes is A, F and R.

Checking our newest reachable nodes, F, and R, the adjacency matrix might tell us F connects to C and W, while R connects to S.

The reachable list is now A, C, F, R and S.

Perhaps we can keep doing this until either:

* we are able to add node B to our reachable list, or

* we have not reached B, and there are no more reachable nodes.

A-to-B Algorithm Test

Start at node 3 in Graph #2, try to reach node 5:

1) 3 is reachable

2) Edges (3,1) and (3,4), so 1,3,4 are reachable

3) Edge (1,2), so 1,2,3,4 are reachable

4) Edge (2,5) and (2,6) so 1,2,3,4,5,6 reachable.

5) Yes, there is a path from 3 to 5!

A_TO_B in words

```
function value = a_to_b ( adj, a, b )
%
%% A_TO_B: Is there a path from node A to node B?
%
%  ADJ(N,N) is the adjacency matrix for the graph G.
%  A and B are nodes in the graph.  So we expect 1 <= A, B <= N.
%  VALUE is output by the function, and is TRUE if there is a path from A to B.
%
%  How many nodes are there?
%
%  REACH(I) will be TRUE if we can reach node I from node A.
%
%  NEW holds the nodes we have just reached on this step.
%
%  As long as NEW isn't empty, we reached some new nodes.
%  Use them as stepping stones, and try to reach more nodes.
%
%  For each node J that we found on the previous step...
%
%  ...look for adjacent neighbors I...
%
%  ...and if node I hasn't been reached yet...
%
%  ...then add node I to the reachable set, note that we just found it, and...
%
%  ...if the node is B, then B is reachable from A and we are done.
%
%  If we got here, we never reached B.
```

a_to_b.m

```
function value = a_to_b ( adj, a, b )
  value = false;
  [ n, n ] = size ( adj );
  reach = zeros(1,n);
  reach(a) = true;
  new = [ a ];
  while ( 0 < length ( new ) )
    old = new;
    new = [];
    for k = 1 : length ( old )
      j = old(k);
      for i = 1 : n
        if ( adj(i,j) == 1 )
          if ( ~ reach(i) )
            reach(i) = true;
            new = [ new, i ];
            if ( i == b )
              value = true;
              return
            end
          end
        end
      end
    end
  end
  return
end
```

What is the Path From A to B?

When we ask whether there is a path from A to B, our algorithm correctly returns TRUE or FALSE.

But if the answer is TRUE, the algorithm doesn't actually tell us the path to use.

Can we create a method that leaves a trail from node A, so that if we find B, we can work our way back?

Right now, REACH(J) turns TRUE when node J is "reached" from another node, say node I. What if REACH(J) stored that value, "I", the connecting node?

If we reach node B, then REACH(B) will contain the node that connected to B. So from B, we could use REACH(B) to move to a node that is one step closer to node A. But that node also has a REACH value, and so we can keep stepping back until we find A.

And that means we have a path (listed in reverse order, B to A...)

A_TO_B_PATH in Words

```
function path = a_to_b_path ( adj, a, b )
%
%% A_TO_B_PATH: Return a path from node A to node B.
%
%  ADJ(N,N) is the adjacency matrix for the graph G.
%  A and B are nodes in the graph.  So we expect 1 <= A, B <= N.
%  PATH is output by the function:
%
%  How many nodes are there?
%
%  REACH(I) is the node that first "reached" node I.
%  REACH(A) is -1, to indicate that this is the start of the path.
%
%  NEW holds the nodes we have just reached on this step.
%
%  As long as NEW isn't empty, we reached some new nodes.
%  Use them as stepping stones, and try to reach more nodes.
%
%  For each node J that we found on the previous step...
%
%  ...look for adjacent neighbors I...
%
%  ...and if node I has not been reached already,
%  point back to node J, and add node I to the "new" list.
%
%  If node I is actually our goal, then use REACH to figure out
%  how we got to node B from node A, and RETURN.
%
%  If we got here, no path was found.
```

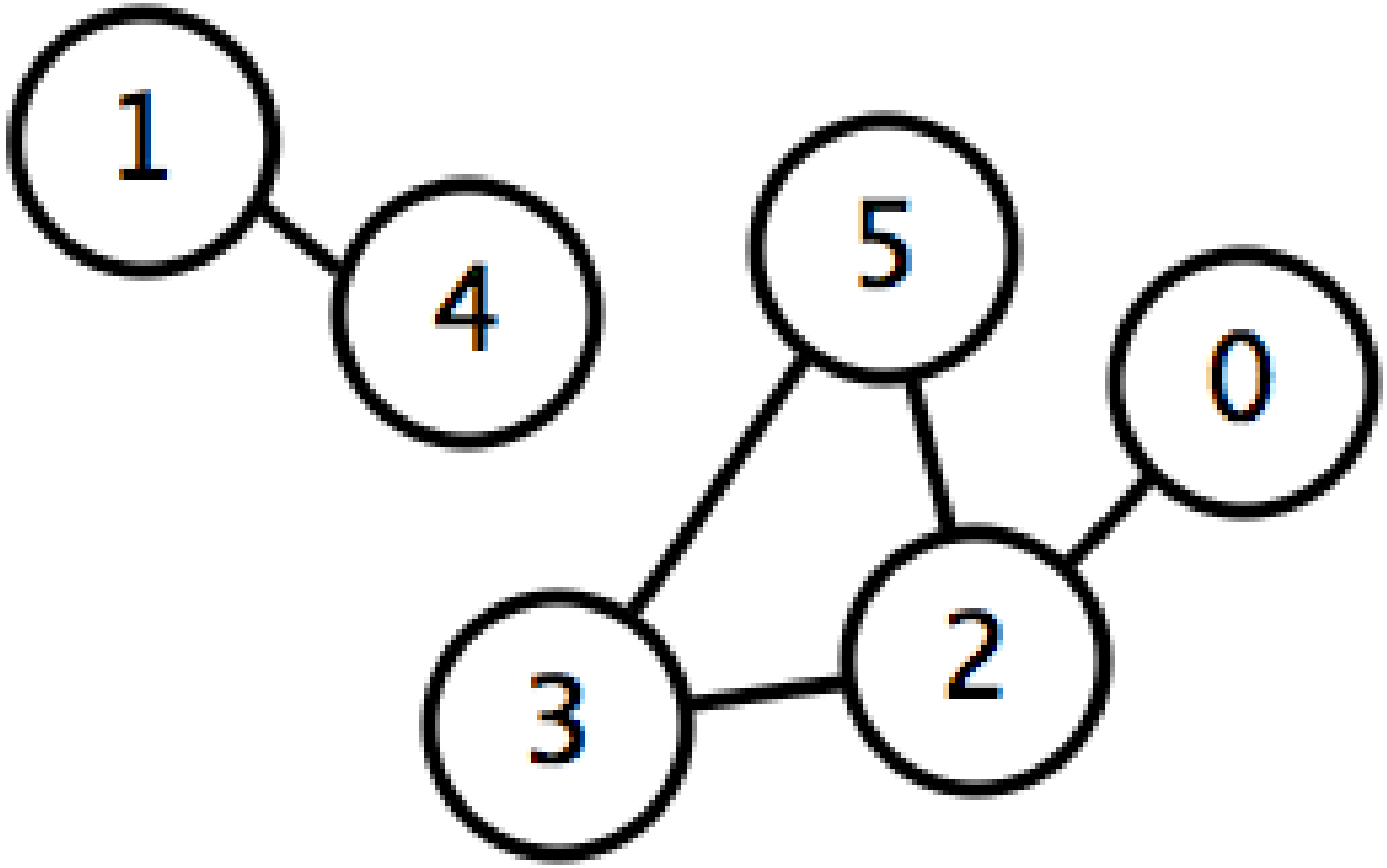
a_to_b_path.m

```
function path = a_to_b_path ( adj, a, b )

 path = [];
 [ n, n ] = size ( adj );
 reach = zeros(1,n);
 reach(a) = -1;
 new = [ a ];
 while ( 0 < length ( new ) )
   old = new;
   new = [];
   for k = 1 : length ( old )
     j = old(k);
     for i = 1 : n
       if ( adj(i,j) == 1 )
         if ( reach(i) == 0 )
           reach(i) = j;
           new = [ new, i ];
           if ( i == b )
             node = b;
             while ( true )
               path = [ node, path ];
               node = reach(node);
               if ( node == -1 )
                 return

               ... 8 END statements, RETURN and END.
```

Path From ANY A to ANY B?



Connection Detection

A graph is connected if it is possible to find a path (a sequence of edges) that lead from any node A to any other node B.

Our eye can easily decide whether a graph is connected, at least for small graphs.

If we think about this problem computationally, the input is the graph G, and the output is the value, "true" (the graph is connected) or "false", (the graph is not connected):

Try Connecting Every Pair

From the definition, we know that a graph G is connected if we can get from any node A to any node B. This might suggest the following algorithm.:

```
value = true;
for a = 1 : n
  for b = 1 : n
    value = a_to_b ( adj, a, b );
    if ( value == false )
      return
    end
  end
end
```

A Quicker Solution

Although we get the correct answer by trying to connect every pair, we do many calculations over and over.

What if we modify our "A-TO-B" algorithm so that we start at A, but instead of aiming for B, we simply continue adding as many nodes as we can.

When we have to stop, we have reached all the nodes (the graph is connected) or there is at least one node left out (the graph is not connected.)

This approach will also get the correct answer, but with a much shorter number of computations.

GRAPH_CONNECTED in Words

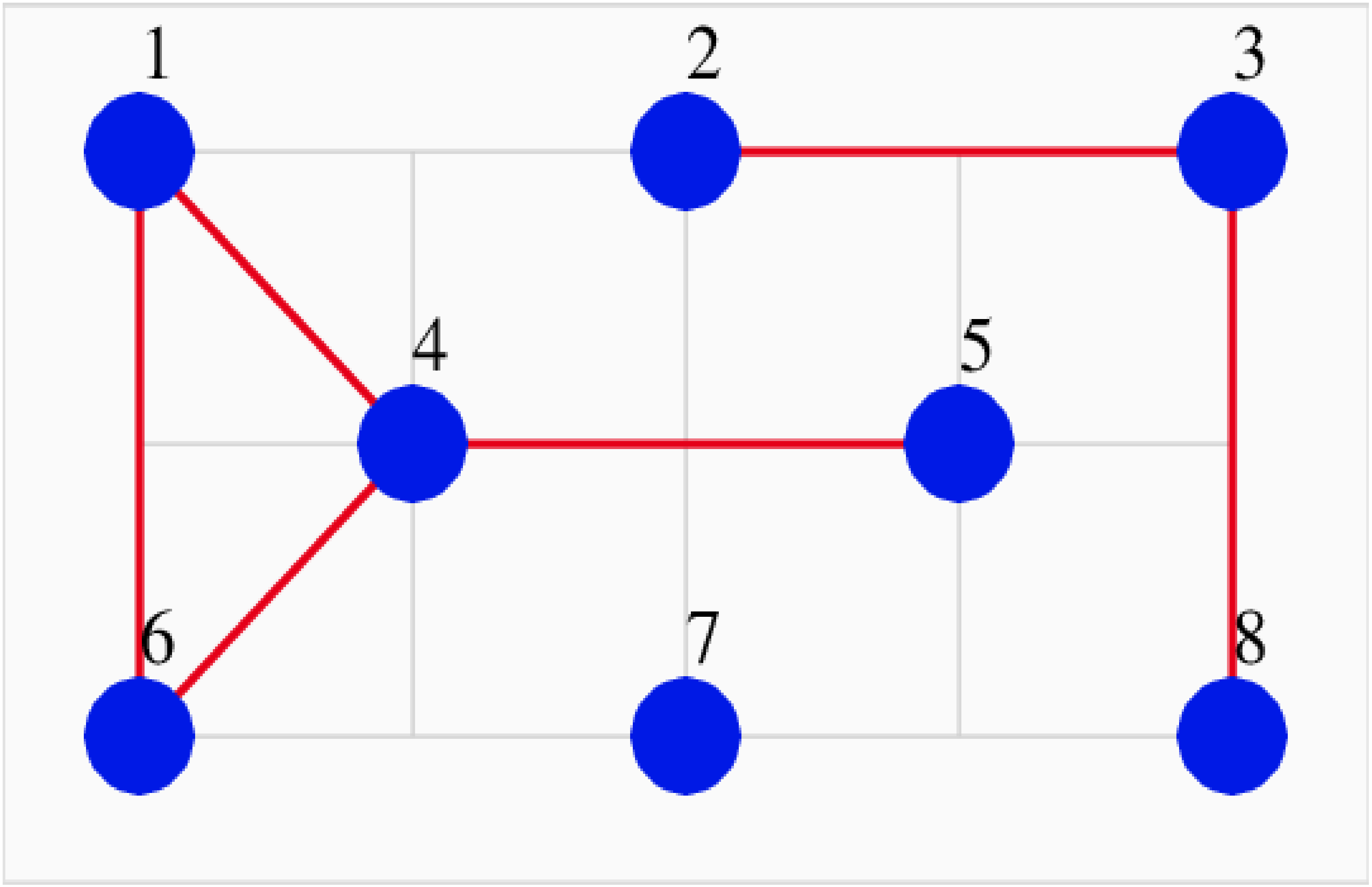
```
function value = graph_connected ( adj )
%
%% GRAPH_CONNECTED is true if the graph G is connected.
%
%  ADJ(N,N) is the adjacency matrix for the graph G.
%  VALUE isTRUE if the graph is connected.
%
%  How many nodes are there?
%
%  REACH(I) will be TRUE if we can reach node I from node 1.
%
%  NEW holds the nodes we have just reached on this step.
%
%  REACHED counts how many nodes we have reached.
%
%  As long as NEW isn't empty, we reached some new nodes.
%  Use them as stepping stones, and try to reach more nodes.
%
%  For each node J that we found on the previous step...
%
%  ...look for adjacent neighbors I...
%
%  ...and if node I hasn't been reached yet:
%  mark that is has been reached,
%  add it to list,
%  and increase the count of nodes we have reached.
%
%  When we get here, we have reached all the nodes we can from node 1.
```

graph_connnected.m

```
function value = graph_connected ( adj )

  [ n, n ] = size ( adj );
  reach = zeros(1,n);
  reach(1) = true;
  new = [1];
  reached = 1;
  while ( 0 < length ( new ) )
    old = new;
    new = [];
    for k = 1 : length ( old )
      j = old(k);
      for i = 1 : n
        if ( adj(i,j) == 1 )
          if ( ~ reach(i) )
            reach(i) = true;
            new = [ new, i ];
            reached = reached + 1;
          end
        end
      end
    end
  end
  value = ( reached == n );
  return
end
```

Graph Example #4 (Disconnected)



Disconnected Graph Adjacency

$$
\begin{array}{c|cccccccc}
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
\hline
1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\
2 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
3 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\
4 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\
5 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
6 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
8 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0
\end{array}
$$

The row of zeros makes it easy to see that this graph is not connected.

Usually, it is not so easy!

Paths and the Adjacency Matrix

We have seen that the adjacency matrix ADJ of a graph G contains all the information about the graph.

But to answer questions about the graph (for instance, is graph G connected?) we usually have to think about ADJ in complicated ways.

However, it turns out that ADJ can answer some questions about paths in a very efficient and simple manner.

Let's look at the adjacency matrix for our disconnected graph.

A1 = ADJ Matrix

```
     1  2  3  4  5  6  7  8
   +-----------------------
1  |  .  .  .  1  .  1  .  .
2  |  .  .  1  .  .  .  .  .
3  |  .  1  .  .  .  .  .  1
4  |  1  .  .  .  1  1  .  .
5  |  .  .  .  1  .  .  .  .
6  |  1  .  .  1  .  .  .  .
7  |  .  .  .  .  .  .  .  .
8  |  .  .  1  .  .  .  .  .
```

A2 = A^2 = A * A

```
 2    .    .    1    1    1    .    .
 .    1    .    .    .    .    .    1
 .    .    2    .    .    .    .    .
 1    .    .    3    .    1    .    .
 1    .    .    .    1    1    .    .
 1    .    .    1    1    2    .    .
 .    .    .    .    .    .    .    .
 .    1    .    .    .    .    .    1
```

```
A^2 counts paths of length 2 from J to I.
```

Powers of ADJ

Abbreviate the ADJ matrix as "A".

A(I,J) counts the paths of length 1 from node J to node I.

Surprisingly, A^2(I,J) counts the number of paths of length exactly 2, from J to I.

A^K counts paths of length K.

Therefore, A+A^2+A^3+...+A^7 counts the paths of lengths 1 through 7 from any node J to any node I.

If the graph is connected, then no entry of this matrix can be zero!

A1+A2+A3+A4+A5+A6+A7

```
   113     0     0   136    60   114     0     0
     0     7    15     0     0     0     0     7
     0    15    14     0     0     0     0    15
   136     0     0   151    76   136     0     0
    60     0     0    76    31    60     0     0
   114     0     0   136    60   113     0     0
     0     0     0     0     0     0     0     0
     0     7    15     0     0     0     0     7
```

```
From node 1, cannot reach nodes 2, 3, 7 or 8;
from node 2, cannot reach nodes 1, 4, 5, 6, 7, ...
and so on.

Because there are zeros in this matrix, the graph is not connected.
```

Homework #12

HW056: Convert a list of edges into an adjacency matrix.

HW057: Using the RISK adjacency matrix, find the maximum distance from Western Australia to any other region on the RISK game board.

HW058: Complete the SPOT survey and provide evidence (screen shot, or a written statement).

Homework #12 is due Friday, December 8.

Homework #11 is due tomorrow night!