GRAB A BYTE

DIJKSTRA'S ALGORITHM

WHAT WE'VE COVERED SO FAR

Prior to Spring Break we covered: Linear and Binary Search. Bubble, Selection, Insertion, Merge and Quick Sort. Since Spring Break we have covered Breadth-First Search, Depth-First Search and Hashing

WHAT WE ARE COVERING TODAY

Today we are covering Dijkstra’s Algorithm!

WHAT IS DIJKSTRA'S ALGORITHM?

Dijkstra's algorithm is specifically designed to find the shortest paths from a starting node (source) to all other nodes in a graph, where the edges have non-negative weights.

WHAT?

Think of it as a map with cities (nodes) and you are trying to get from Brownsville to Austin in the shortest possible path! You could go Brownsville → S.Padre → Corpus → San Antonio → Austin, but that would take 1 hour longer than if you went through Laredo.

Lets consider variables of a graph:

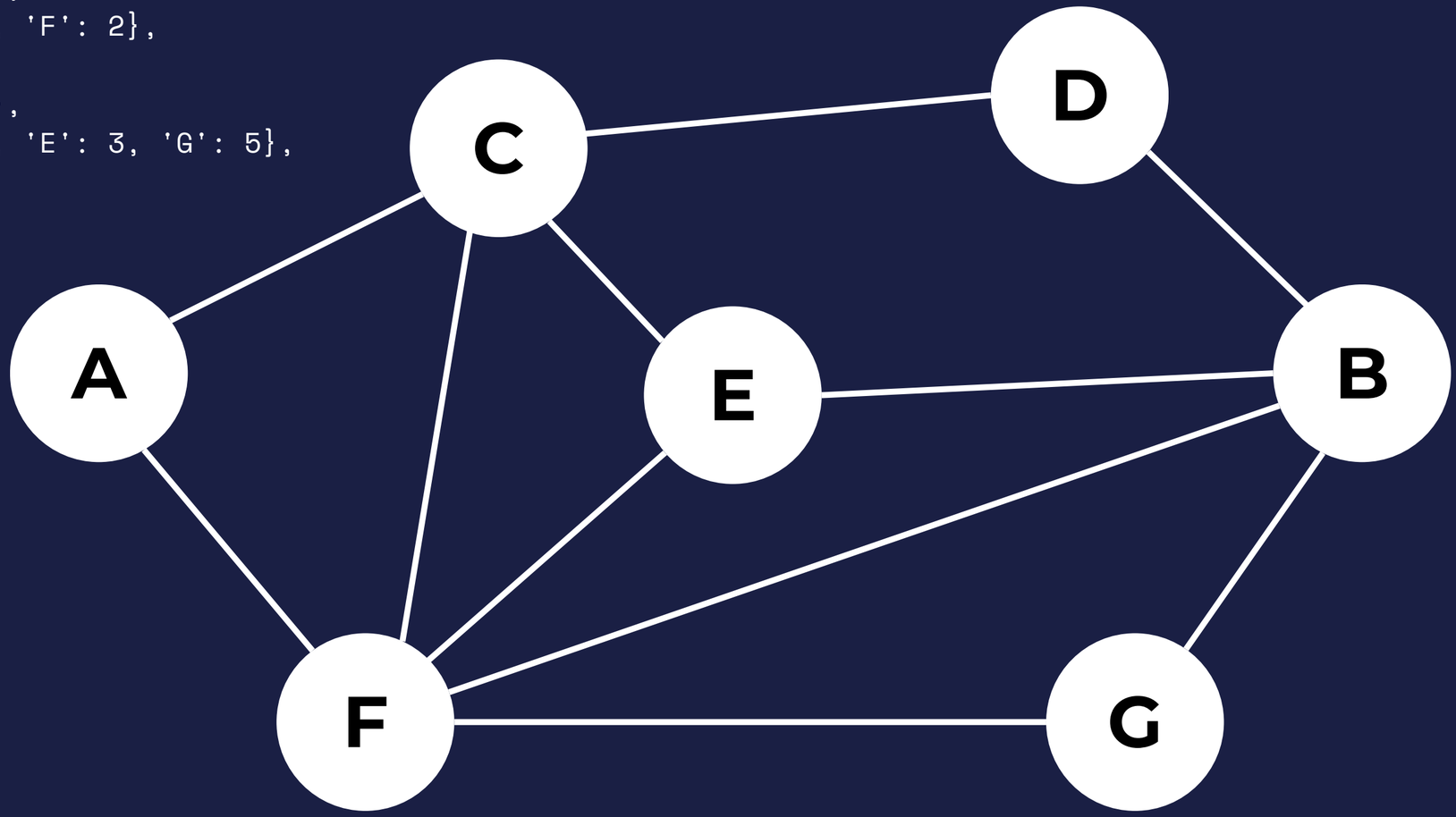
```
graph = {
 'A': {'C': 3, 'F': 2},
 'B': {'D': 1, 'E': 2, 'G': 2},
 'C': {'A': 3, 'D': 4, 'E': 1, 'F': 2},
 'D': {'B': 1, 'C': 4},
 ‘E’: {'B': 2, 'C': 1, 'F': 3},
 ‘F’: {'A': 2, 'B': 6, 'C': 2, 'E': 3, 'G': 5},
 ‘G’: {'B': 2, 'F': 5}
}
```

But lets make it look like a graph:

```
graph = {
 'A': {'C': 3, 'F': 2},
 'B': {'D': 1, 'E': 2, 'G': 2},
 'C': {'A': 3, 'D': 4, 'E': 1, 'F': 2},
 'D': {'B': 1, 'C': 4},
 ‘E’: {'B': 2, 'C': 1, 'F': 3},
 ‘F’: {'A': 2, 'B': 6, 'C': 2, 'E': 3, 'G': 5},
 ‘G’: {'B': 2, 'F': 5}
}
```

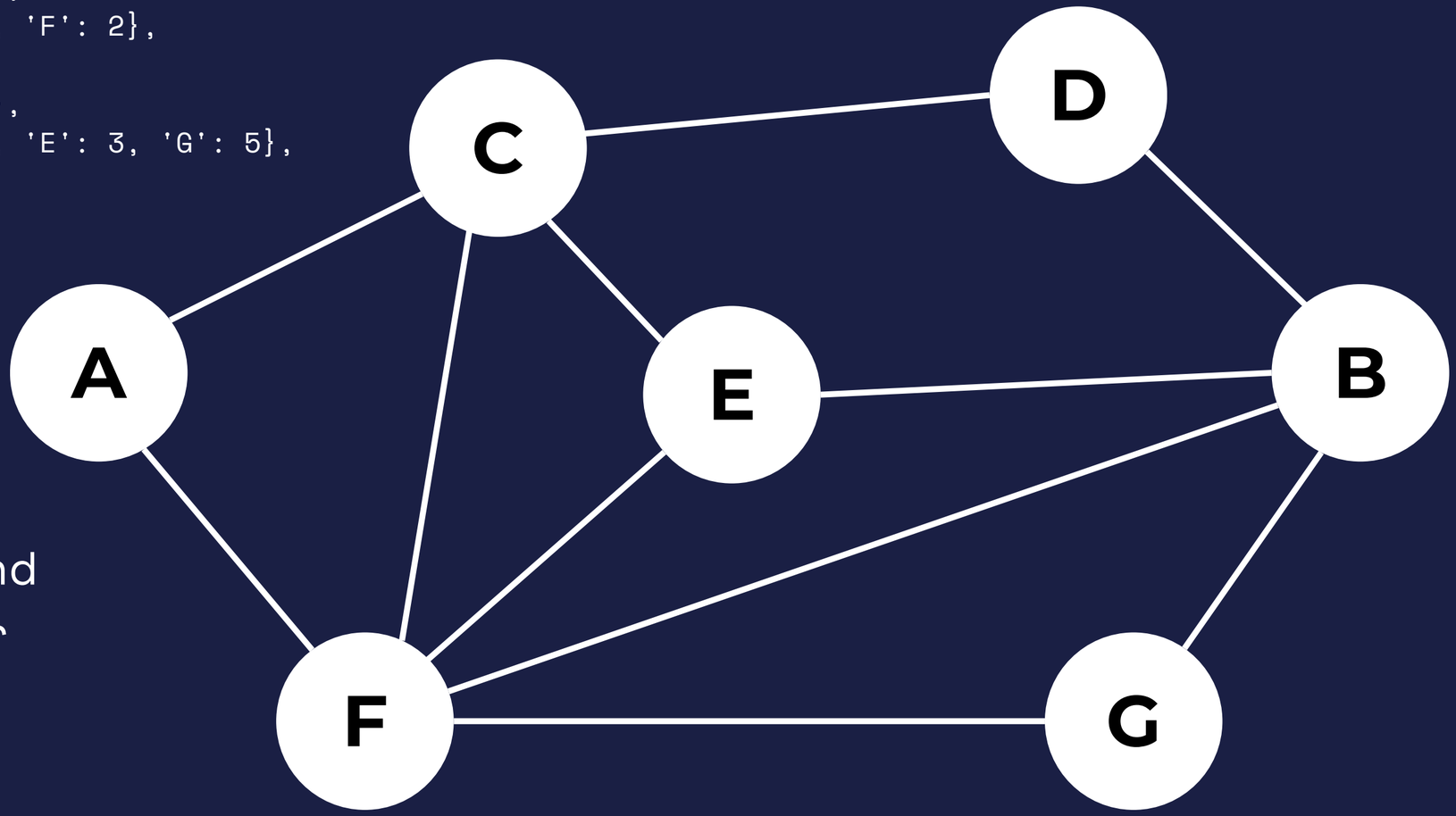
But lets make it look like a graph:

```
graph = {
  'A': {'C': 3, 'F': 2},
  'B': {'D': 1, 'E': 2, 'G': 2},
  'C': {'A': 3, 'D': 4, 'E': 1, 'F': 2},
  'D': {'B': 1, 'C': 4},
  'E': {'B': 2, 'C': 1, 'F': 3},
  'F': {'A': 2, 'B': 6, 'C': 2, 'E': 3, 'G': 5},
  'G': {'B': 2, 'F': 5}
}
```



In this graph, our vertices (the letter nodes) are connected by edges (the paths between)

```
graph = {
 'A': {'C': 3, 'F': 2},
 'B': {'D': 1, 'E': 2, 'G': 2},
 'C': {'A': 3, 'D': 4, 'E': 1, 'F': 2},
 'D': {'B': 1, 'C': 4},
 'E': {'B': 2, 'C': 1, 'F': 3},
 'F': {'A': 2, 'B': 6, 'C': 2, 'E': 3, 'G': 5},
 'G': {'B': 2, 'F': 5}
}
```

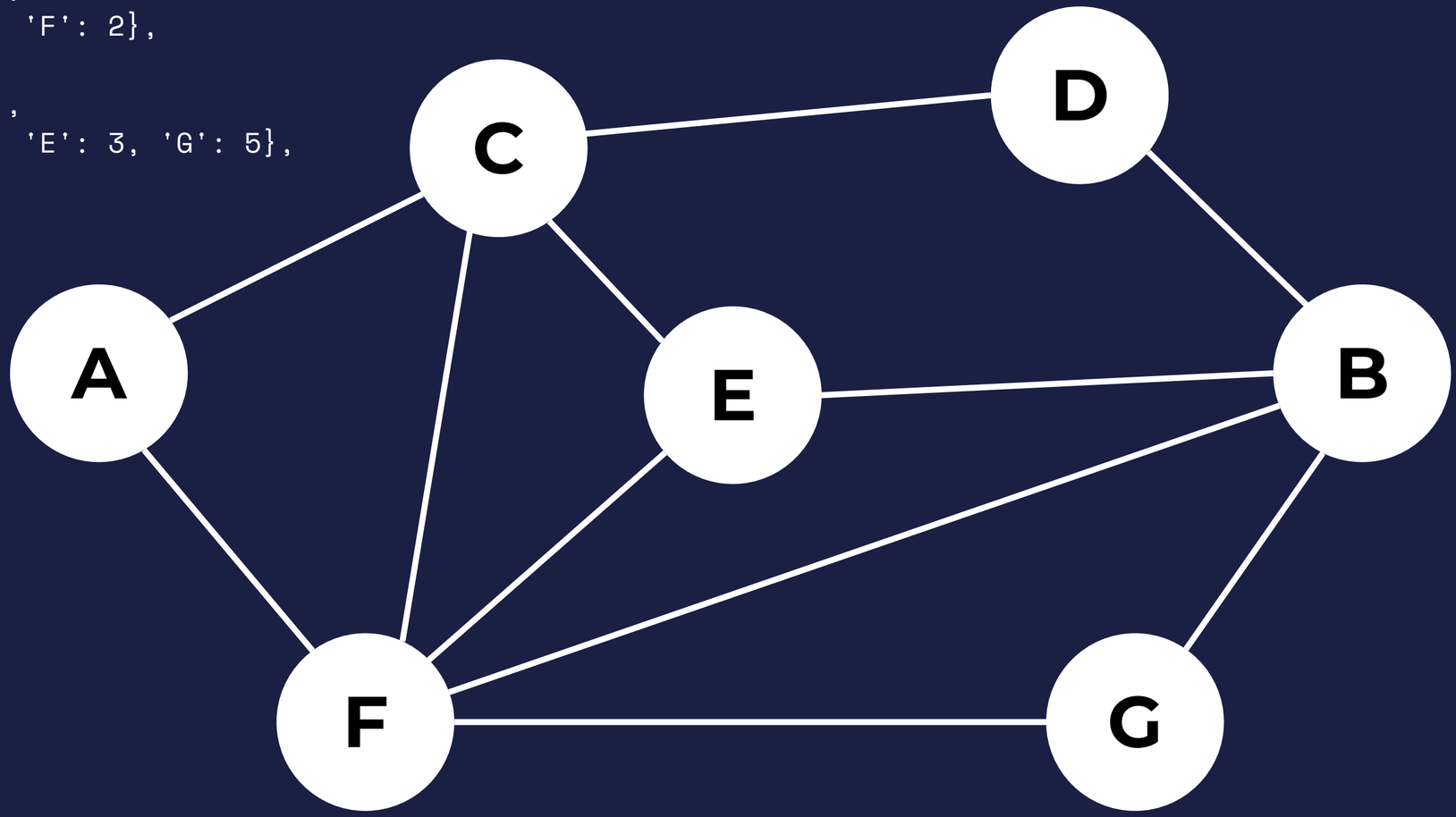


The vertices and edges form our graph

This graph is called a weighted graph.

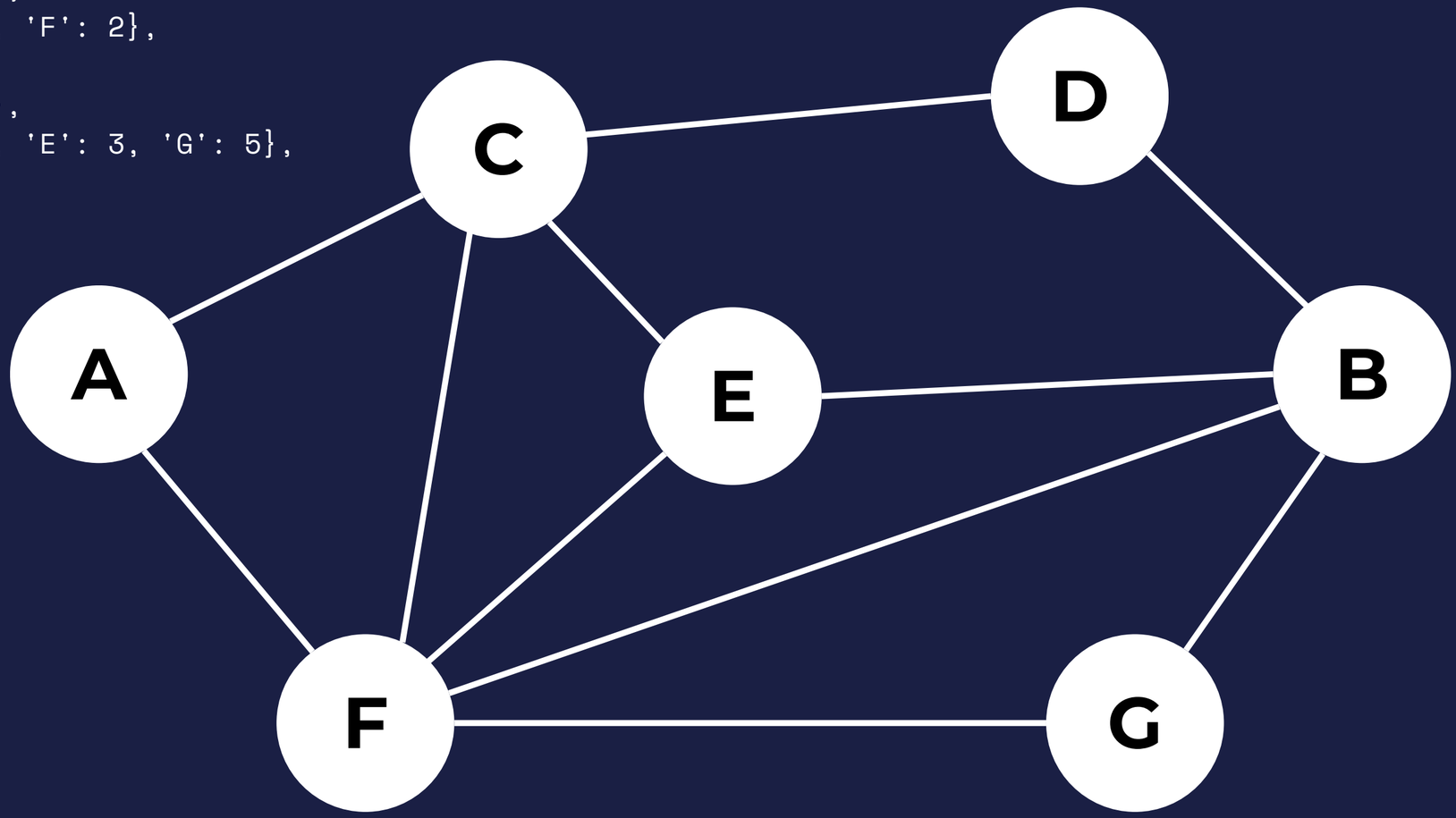
```
graph = {
 'A': {'C': 3, 'F': 2},
 'B': {'D': 1, 'E': 2, 'G': 2},
 'C': {'A': 3, 'D': 4, 'E': 1, 'F': 2},
 'D': {'B': 1, 'C': 4},
 'E': {'B': 2, 'C': 1, 'F': 3},
 'F': {'A': 2, 'B': 6, 'C': 2, 'E': 3, 'G': 5},
 'G': {'B': 2, 'F': 5}
}
```

This is because each edge has a value (or weight) that tells us how “costly” it is to travel that edge.



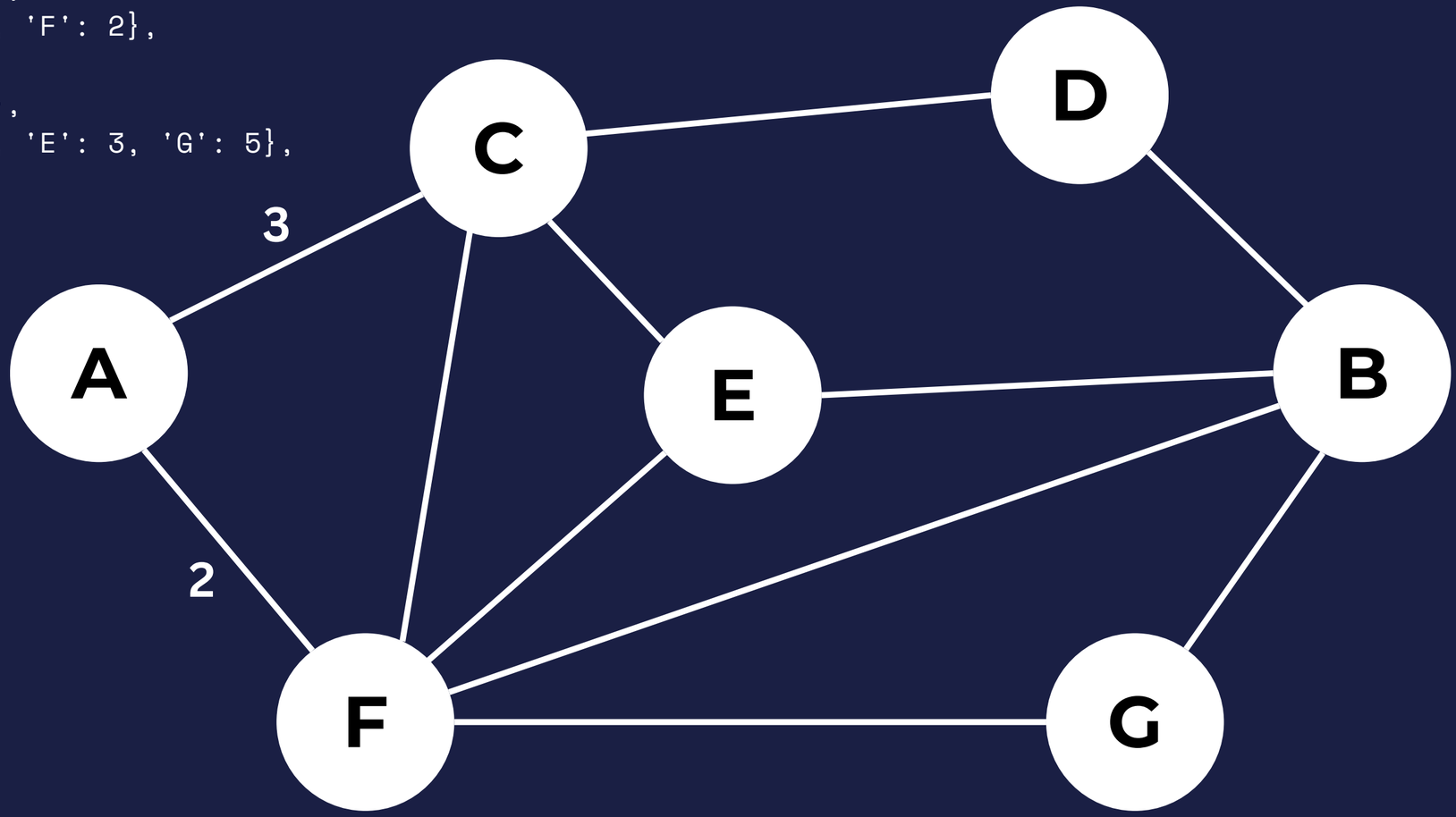
Lets fill out the weights

```
graph = {
 'A': {'C': 3, 'F': 2},
 'B': {'D': 1, 'E': 2, 'G': 2},
 'C': {'A': 3, 'D': 4, 'E': 1, 'F': 2},
 'D': {'B': 1, 'C': 4},
 'E': {'B': 2, 'C': 1, 'F': 3},
 'F': {'A': 2, 'B': 6, 'C': 2, 'E': 3, 'G': 5},
 'G': {'B': 2, 'F': 5}
}
```



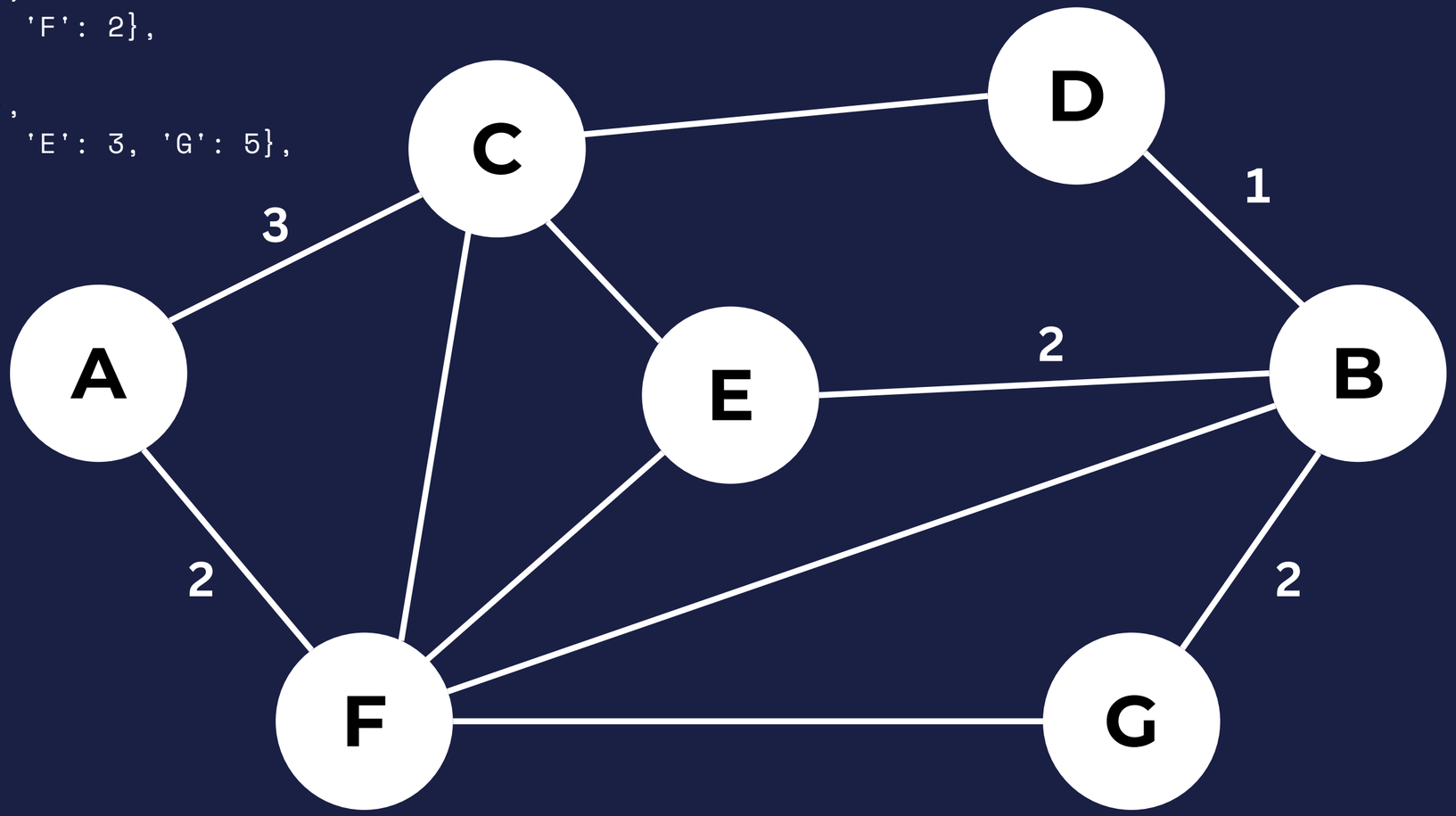
A connects to C by 3 and F by 2.

```
graph = {
  'A': {'C': 3, 'F': 2},
  'B': {'D': 1, 'E': 2, 'G': 2},
  'C': {'A': 3, 'D': 4, 'E': 1, 'F': 2},
  'D': {'B': 1, 'C': 4},
  'E': {'B': 2, 'C': 1, 'F': 3},
  'F': {'A': 2, 'B': 6, 'C': 2, 'E': 3, 'G': 5},
  'G': {'B': 2, 'F': 5}
}
```



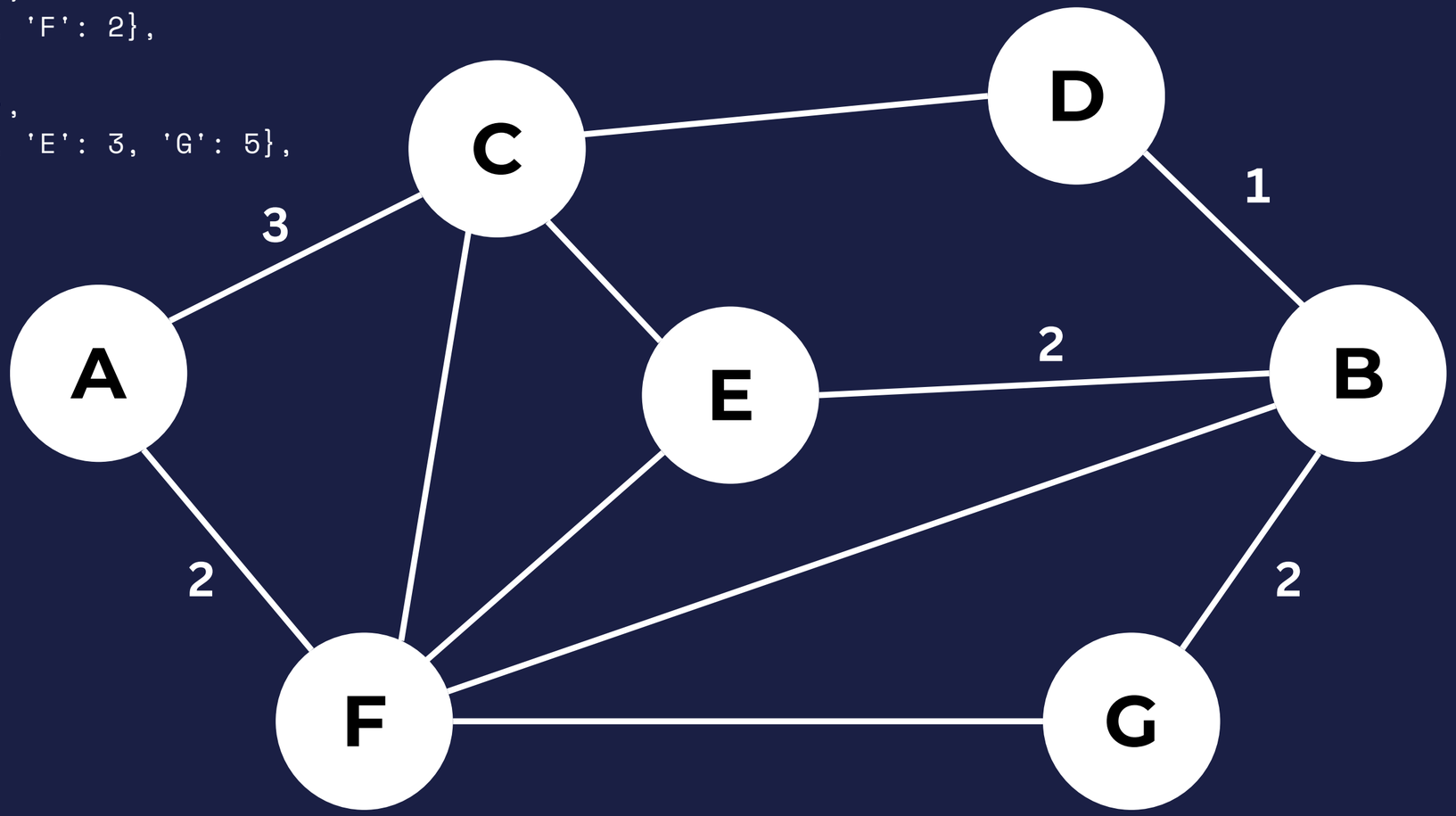
B connects to D by 1, E by 2, and G by 2

```
graph = {
  'A': {'C': 3, 'F': 2},
  'B': {'D': 1, 'E': 2, 'G': 2},
  'C': {'A': 3, 'D': 4, 'E': 1, 'F': 2},
  'D': {'B': 1, 'C': 4},
  'E': {'B': 2, 'C': 1, 'F': 3},
  'F': {'A': 2, 'B': 6, 'C': 2, 'E': 3, 'G': 5},
  'G': {'B': 2, 'F': 5}
}
```



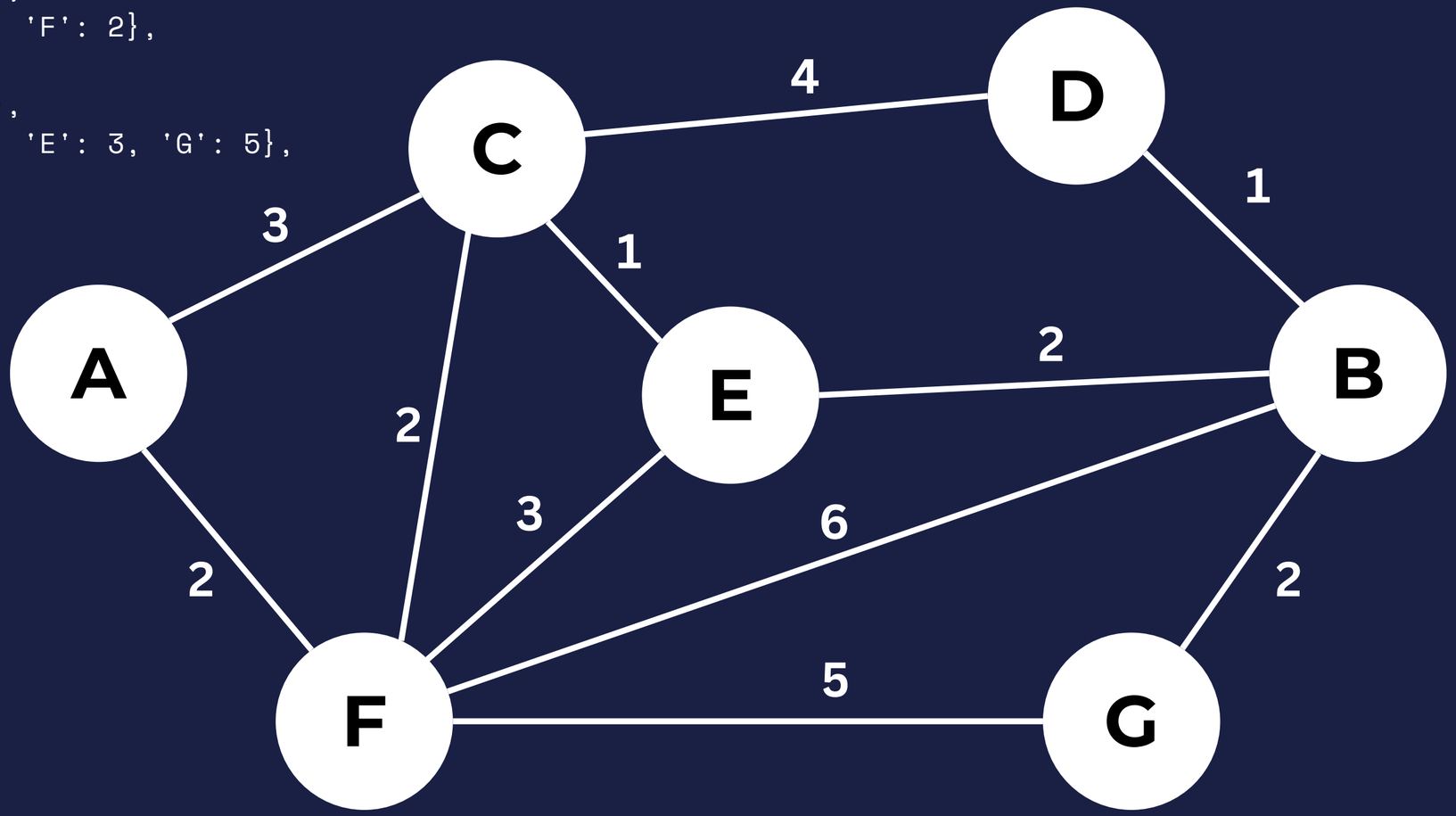
And so forth...

```
graph = {
  'A': {'C': 3, 'F': 2},
  'B': {'D': 1, 'E': 2, 'G': 2},
  'C': {'A': 3, 'D': 4, 'E': 1, 'F': 2},
  'D': {'B': 1, 'C': 4},
  'E': {'B': 2, 'C': 1, 'F': 3},
  'F': {'A': 2, 'B': 6, 'C': 2, 'E': 3, 'G': 5},
  'G': {'B': 2, 'F': 5}
}
```



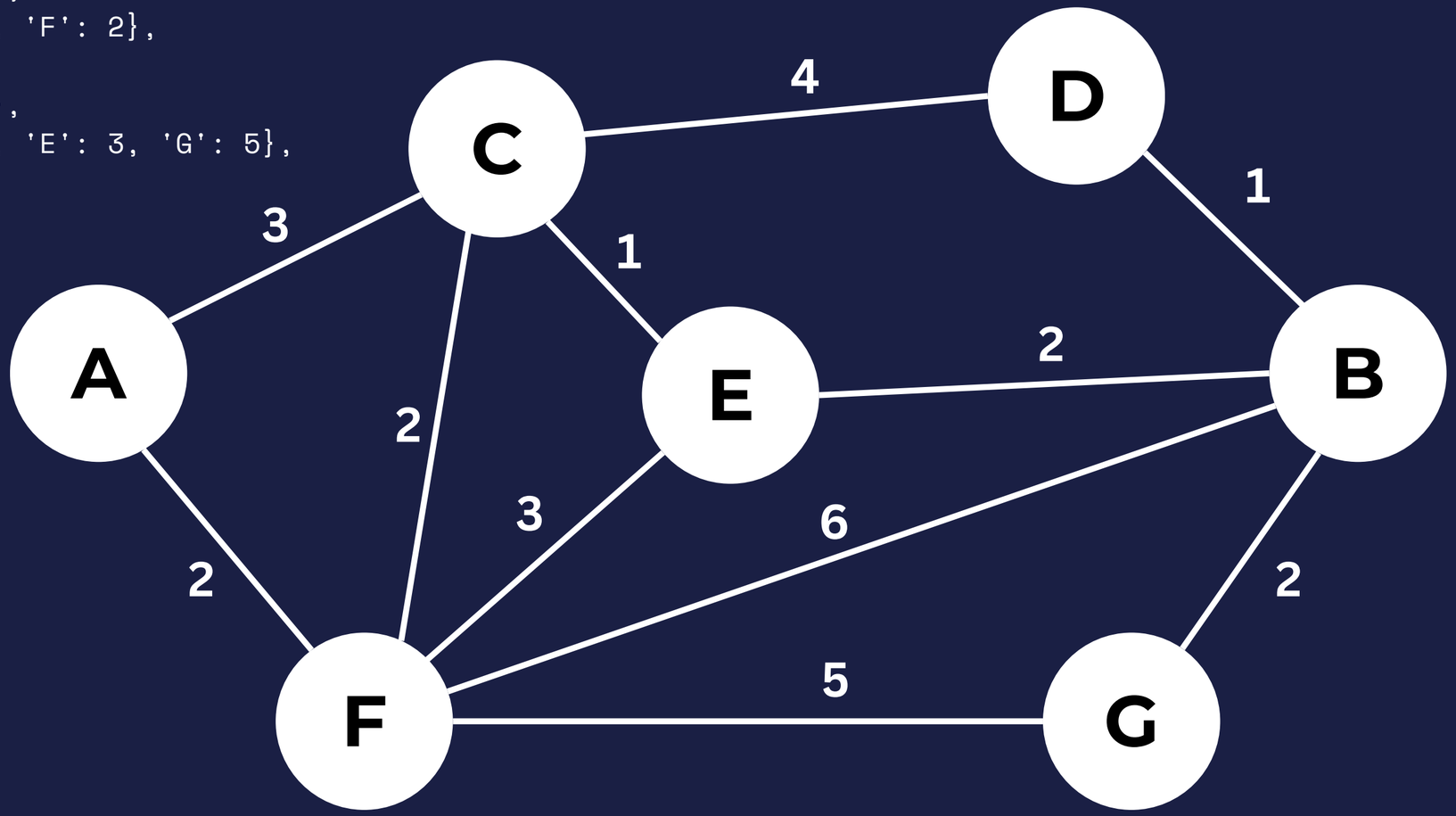
And so forth...

```
graph = {
  'A': {'C': 3, 'F': 2},
  'B': {'D': 1, 'E': 2, 'G': 2},
  'C': {'A': 3, 'D': 4, 'E': 1, 'F': 2},
  'D': {'B': 1, 'C': 4},
  'E': {'B': 2, 'C': 1, 'F': 3},
  'F': {'A': 2, 'B': 6, 'C': 2, 'E': 3, 'G': 5},
  'G': {'B': 2, 'F': 5}
}
```



With our weighted graph we want to find the shortest path from one vertex to another.

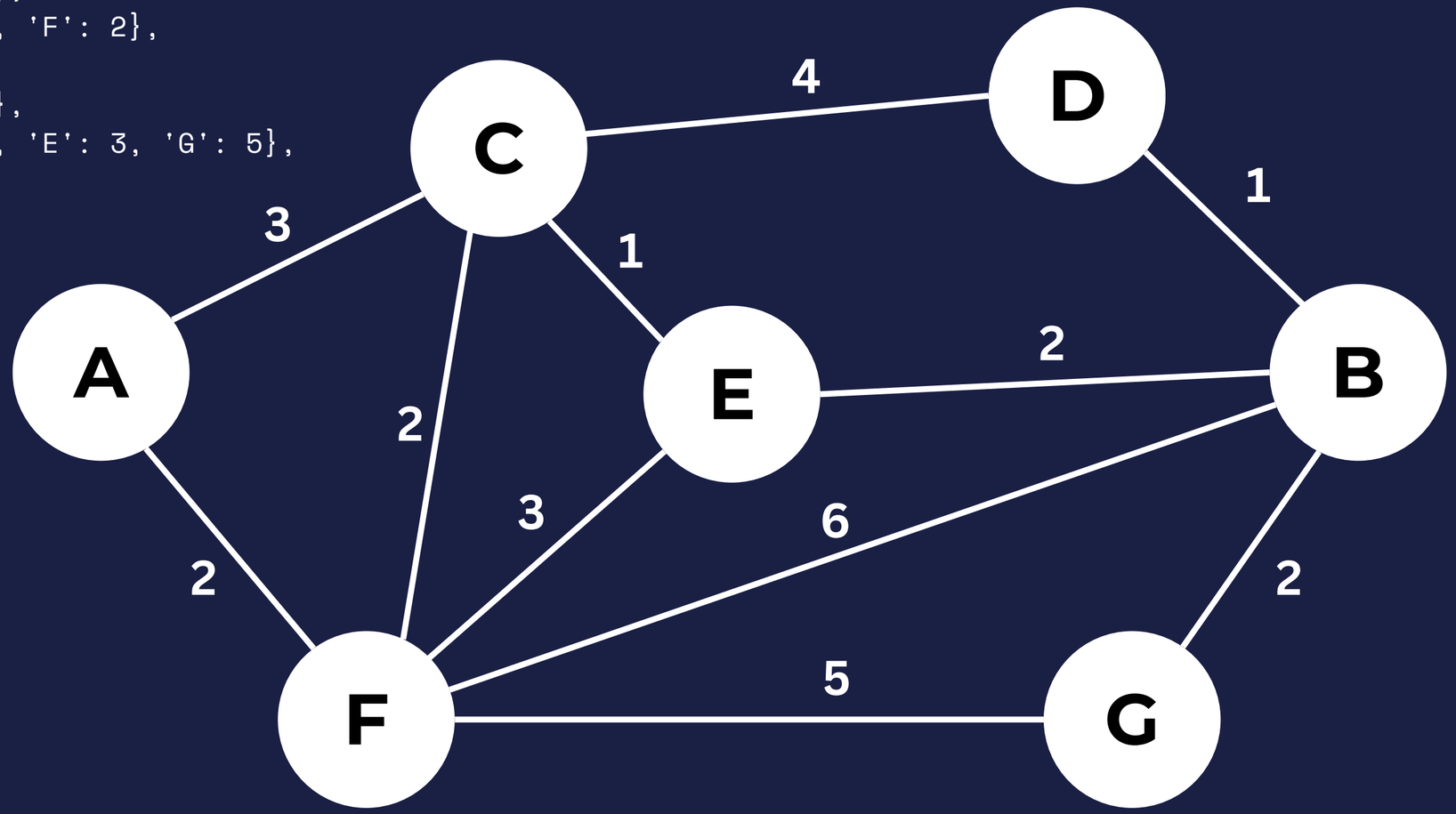
```
graph = {
 'A': {'C': 3, 'F': 2},
 'B': {'D': 1, 'E': 2, 'G': 2},
 'C': {'A': 3, 'D': 4, 'E': 1, 'F': 2},
 'D': {'B': 1, 'C': 4},
 'E': {'B': 2, 'C': 1, 'F': 3},
 'F': {'A': 2, 'B': 6, 'C': 2, 'E': 3, 'G': 5},
 'G': {'B': 2, 'F': 5}
}
```



Like from A to B.

With our weighted graph we want to find the shortest path from one vertex to another.

```
graph = {
 'A': {'C': 3, 'F': 2},
 'B': {'D': 1, 'E': 2, 'G': 2},
 'C': {'A': 3, 'D': 4, 'E': 1, 'F': 2},
 'D': {'B': 1, 'C': 4},
 'E': {'B': 2, 'C': 1, 'F': 3},
 'F': {'A': 2, 'B': 6, 'C': 2, 'E': 3, 'G': 5},
 'G': {'B': 2, 'F': 5}
}
```

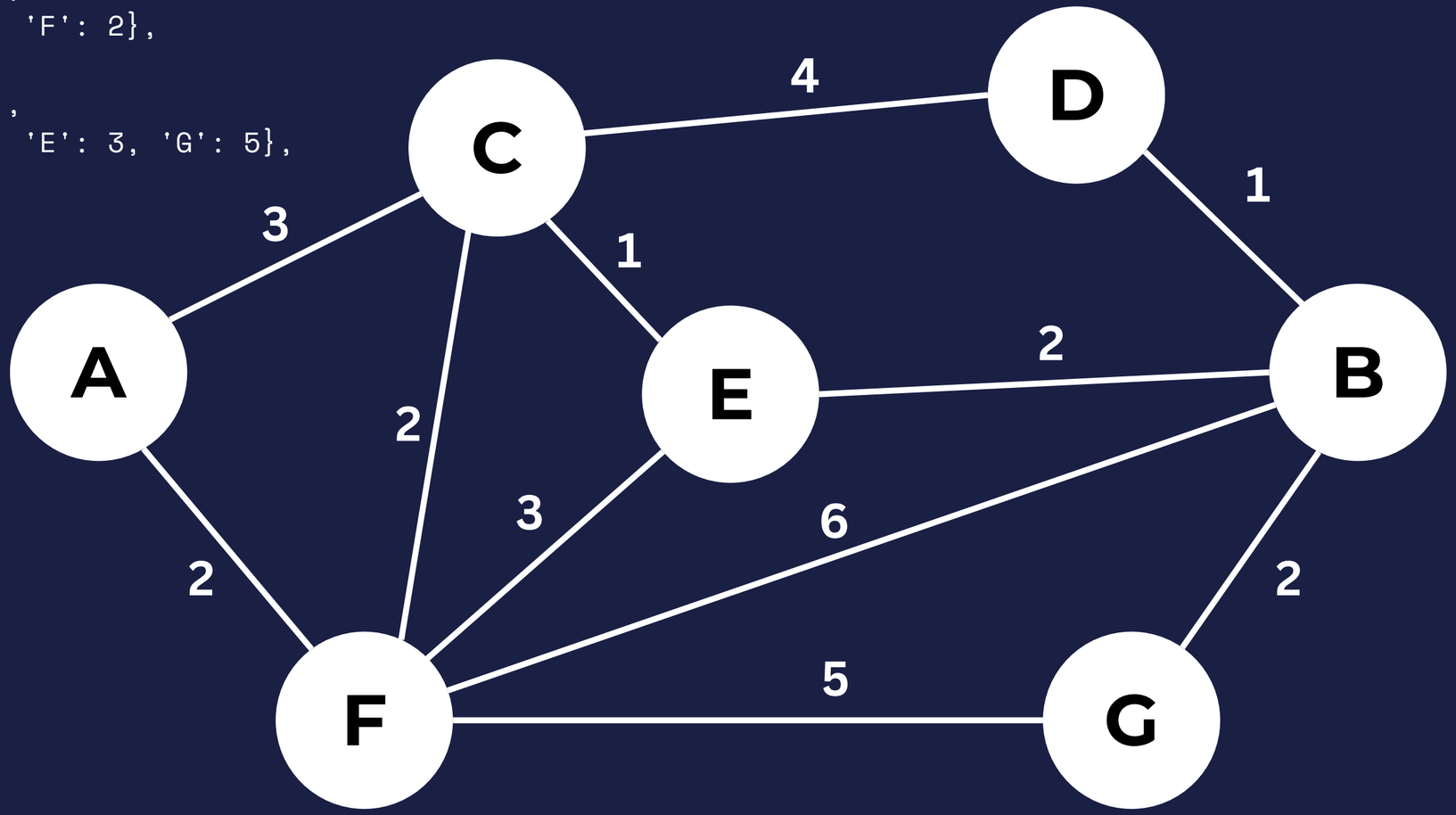


Like from A to B

So this algorithm will start checking the shortest distance between A and the rest of the vertices.

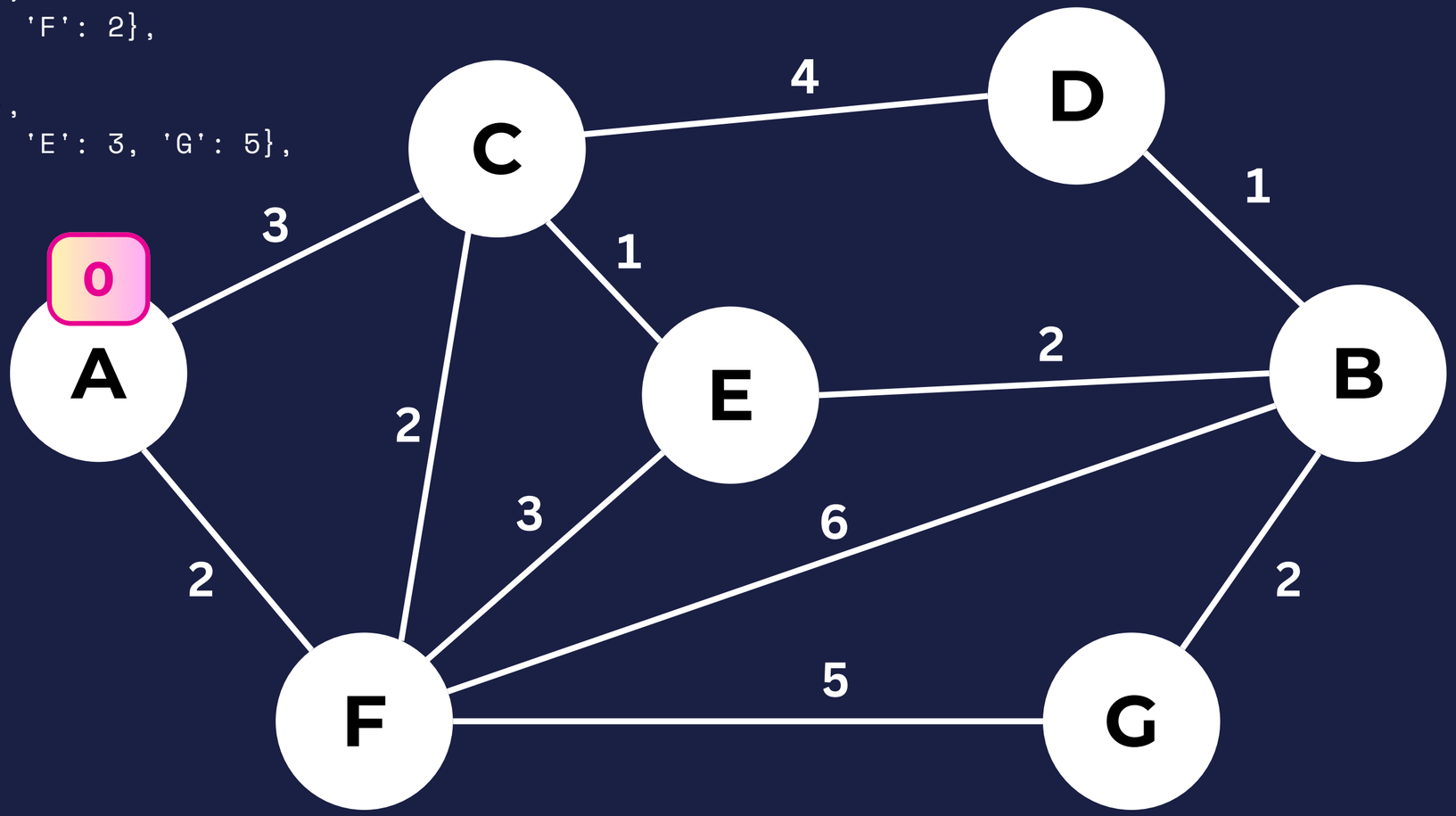
```
graph = {
 'A': {'C': 3, 'F': 2},
 'B': {'D': 1, 'E': 2, 'G': 2},
 'C': {'A': 3, 'D': 4, 'E': 1, 'F': 2},
 'D': {'B': 1, 'C': 4},
 'E': {'B': 2, 'C': 1, 'F': 3},
 'F': {'A': 2, 'B': 6, 'C': 2, 'E': 3, 'G': 5},
 'G': {'B': 2, 'F': 5}
}
```

First we need to label each vertex with a number representing the shortest distance.



Since the distance from A to A is 0, we can label A 0.

```
graph = {
  'A': {'C': 3, 'F': 2},
  'B': {'D': 1, 'E': 2, 'G': 2},
  'C': {'A': 3, 'D': 4, 'E': 1, 'F': 2},
  'D': {'B': 1, 'C': 4},
  'E': {'B': 2, 'C': 1, 'F': 3},
  'F': {'A': 2, 'B': 6, 'C': 2, 'E': 3, 'G': 5},
  'G': {'B': 2, 'F': 5}
}
```

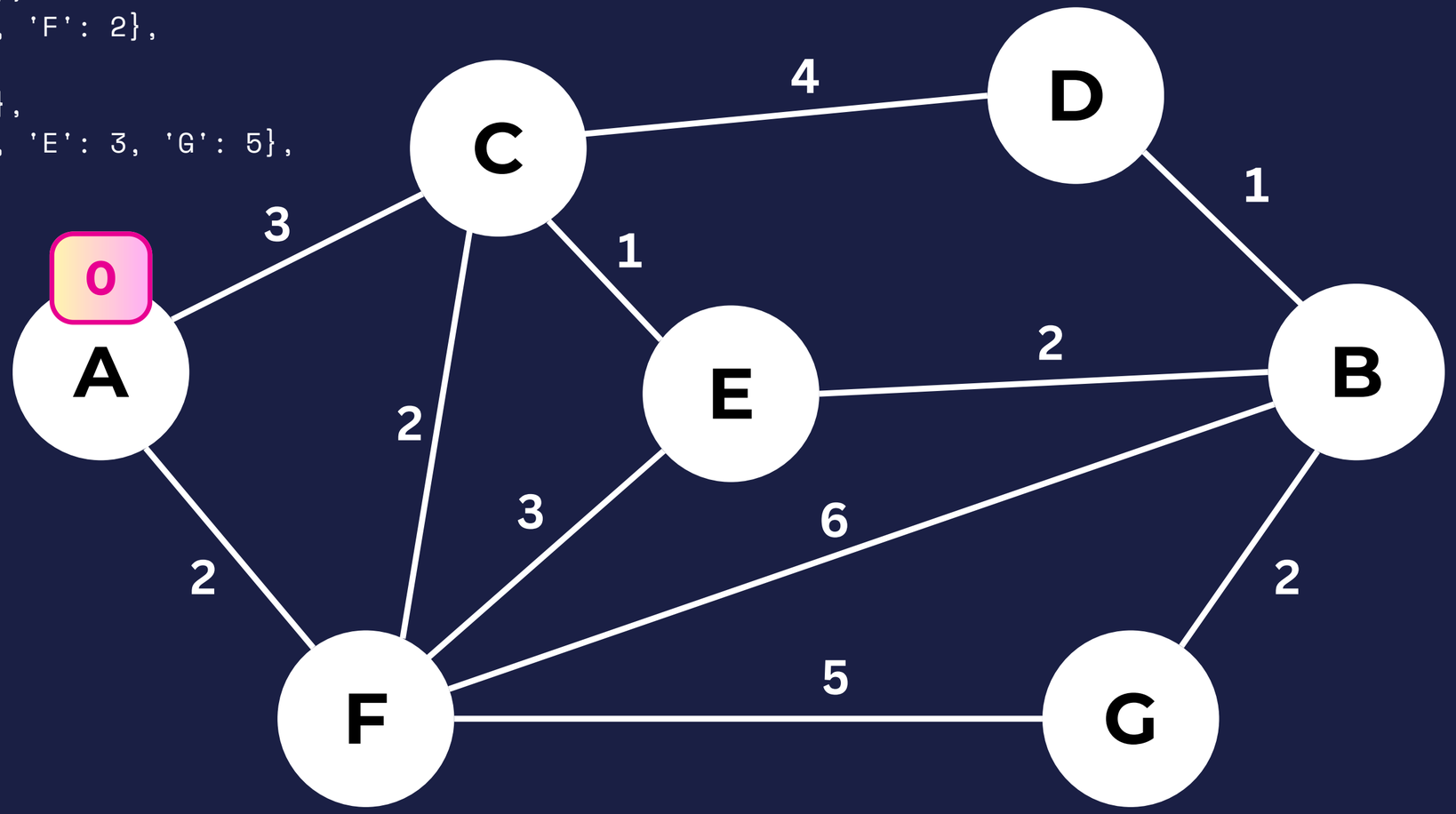


But since we don't know the distance for the other vertices yet, we need a value as a placeholder.

```
graph = {
  'A': {'C': 3, 'F': 2},
  'B': {'D': 1, 'E': 2, 'G': 2},
  'C': {'A': 3, 'D': 4, 'E': 1, 'F': 2},
  'D': {'B': 1, 'C': 4},
  'E': {'B': 2, 'C': 1, 'F': 3},
  'F': {'A': 2, 'B': 6, 'C': 2, 'E': 3, 'G': 5},
  'G': {'B': 2, 'F': 5}
}
```

So, we will choose infinity.

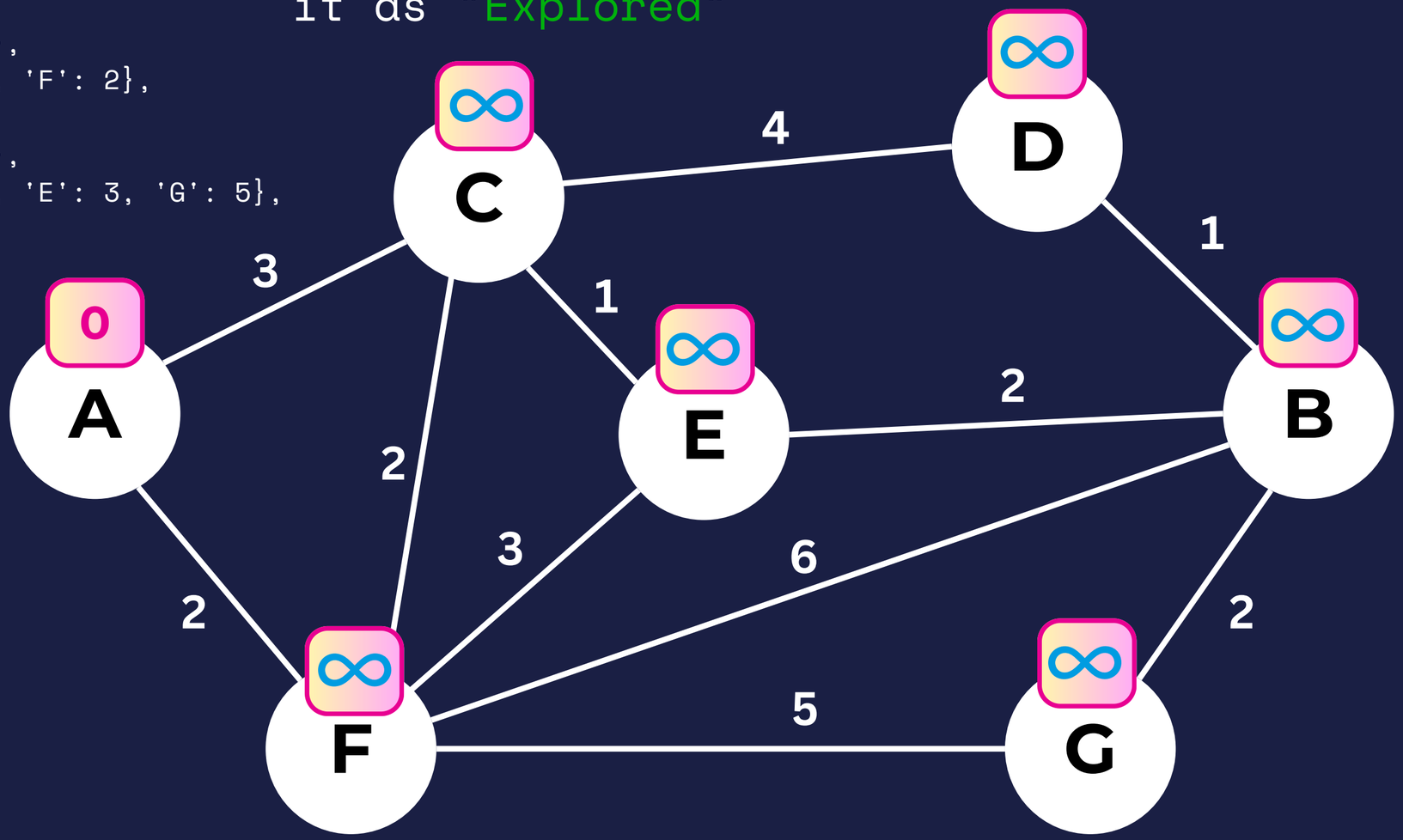
∞



Once we've found the shortest distance to a vertex, we'll mark it as "Explored"

```
graph = {
 'A': {'C': 3, 'F': 2},
 'B': {'D': 1, 'E': 2, 'G': 2},
 'C': {'A': 3, 'D': 4, 'E': 1, 'F': 2},
 'D': {'B': 1, 'C': 4},
 'E': {'B': 2, 'C': 1, 'F': 3},
 'F': {'A': 2, 'B': 6, 'C': 2, 'E': 3, 'G': 5},
 'G': {'B': 2, 'F': 5}
}
```

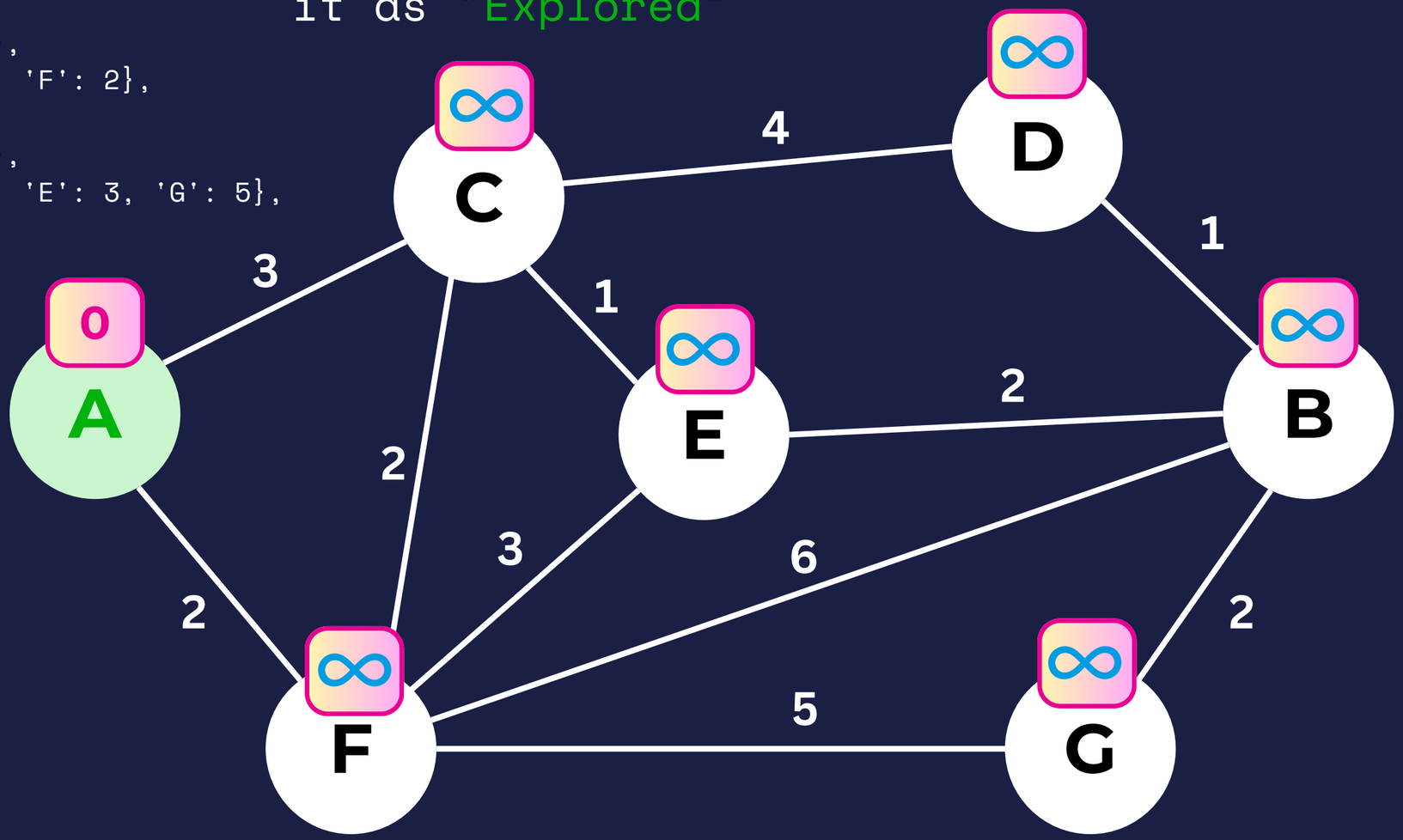
We know the shortest distance from A to A, so we will mark A as explored



Once we've found the shortest distance to a vertex, we'll mark it as "Explored"

```
graph = {
 'A': {'C': 3, 'F': 2},
 'B': {'D': 1, 'E': 2, 'G': 2},
 'C': {'A': 3, 'D': 4, 'E': 1, 'F': 2},
 'D': {'B': 1, 'C': 4},
 'E': {'B': 2, 'C': 1, 'F': 3},
 'F': {'A': 2, 'B': 6, 'C': 2, 'E': 3, 'G': 5},
 'G': {'B': 2, 'F': 5}
}
```

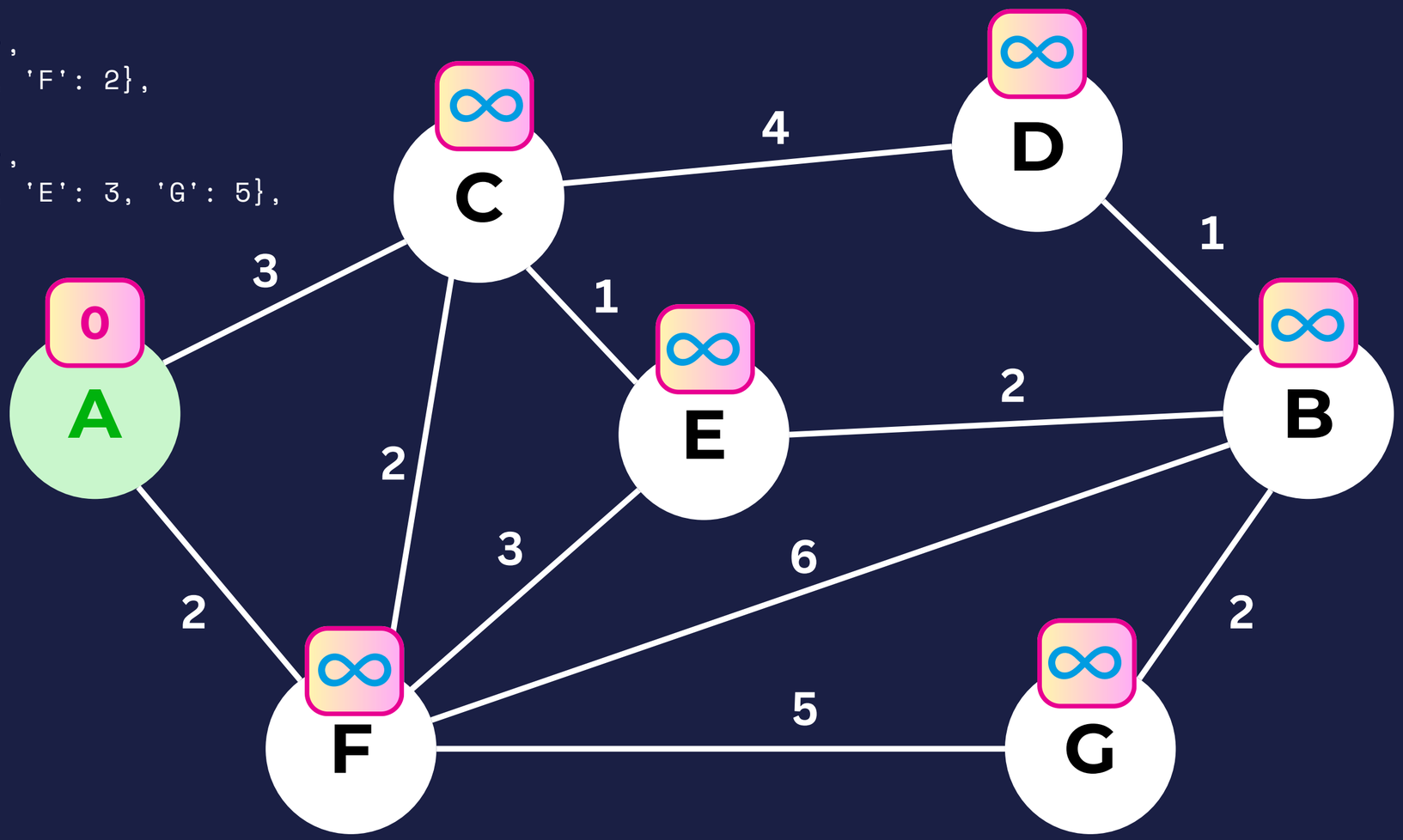
We know the shortest distance from A to A, so we will mark A as explored



Dijkstra's is a 2 step process. Step 1: Update Estimates

```
graph = {
 'A': {'C': 3, 'F': 2},
 'B': {'D': 1, 'E': 2, 'G': 2},
 'C': {'A': 3, 'D': 4, 'E': 1, 'F': 2},
 'D': {'B': 1, 'C': 4},
 'E': {'B': 2, 'C': 1, 'F': 3},
 'F': {'A': 2, 'B': 6, 'C': 2, 'E': 3, 'G': 5},
 'G': {'B': 2, 'F': 5}
}
```

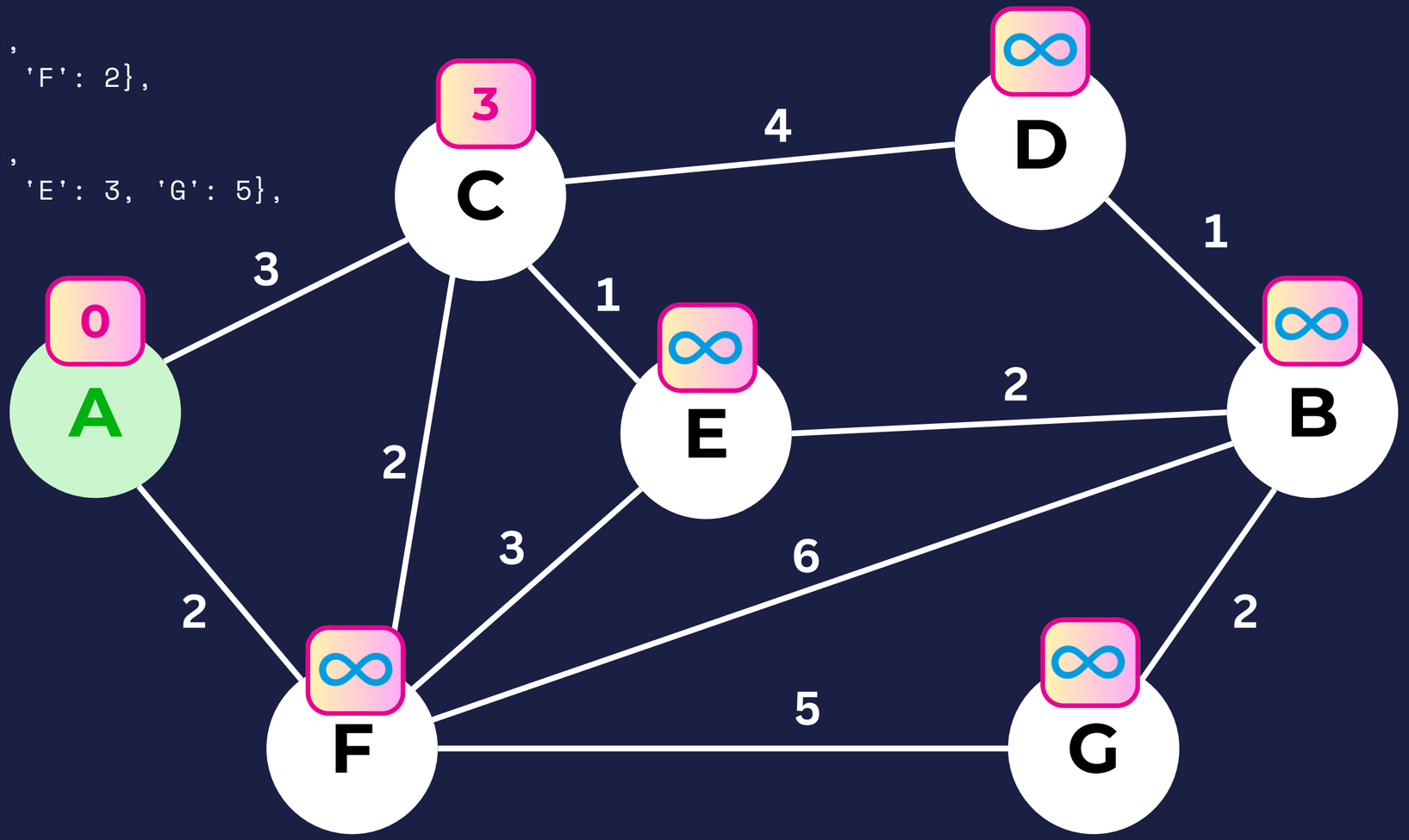
We will check the path from A to C. We know the cost is 3 and we know that 3 is less than infinity.



Dijkstra's is a 2 step process. Step 1: Update Estimates

```
graph = {
 'A': {'C': 3, 'F': 2},
 'B': {'D': 1, 'E': 2, 'G': 2},
 'C': {'A': 3, 'D': 4, 'E': 1, 'F': 2},
 'D': {'B': 1, 'C': 4},
 'E': {'B': 2, 'C': 1, 'F': 3},
 'F': {'A': 2, 'B': 6, 'C': 2, 'E': 3, 'G': 5},
 'G': {'B': 2, 'F': 5}
}
```

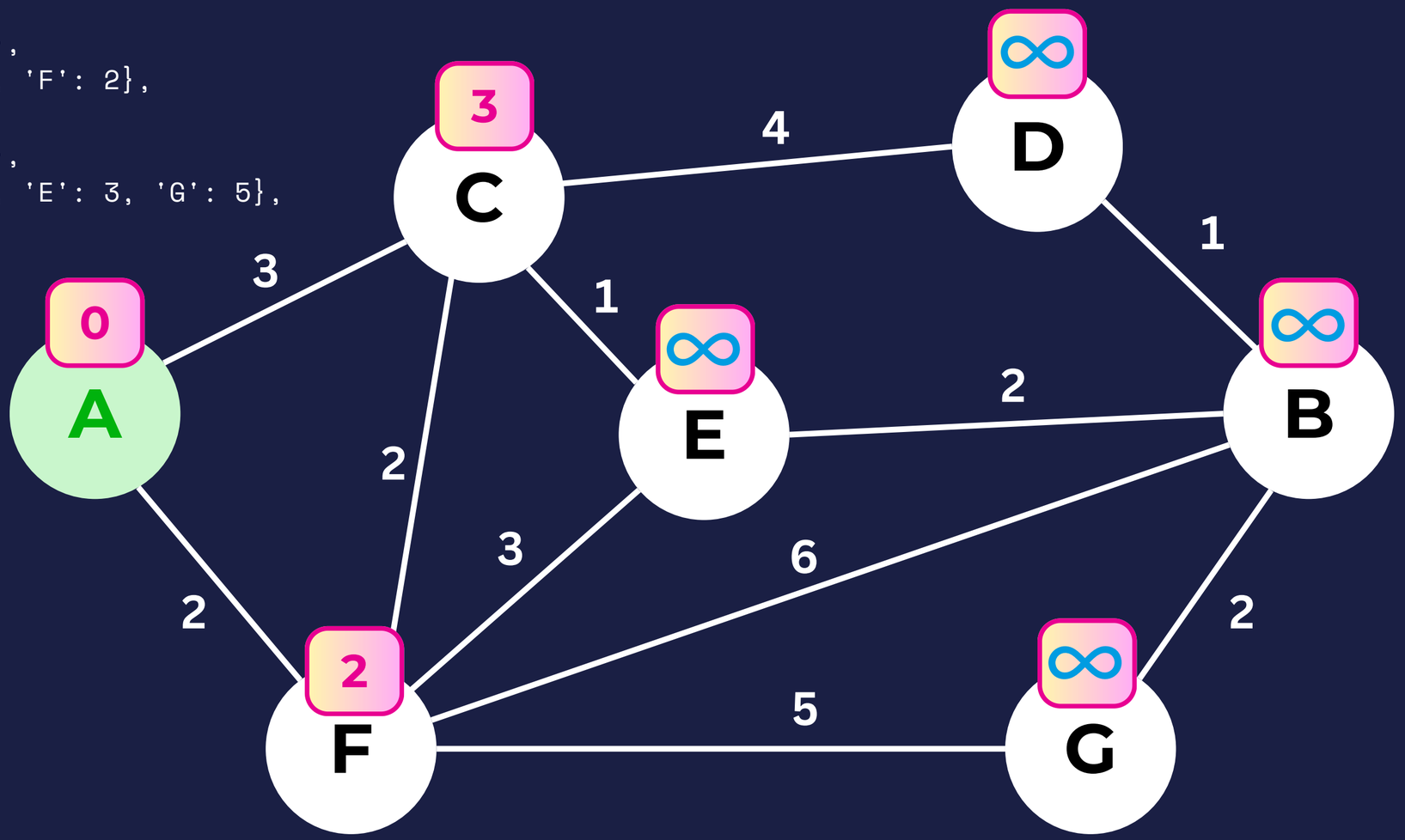
So we update infinity on C to 3. That estimate is done, we'll move on to F



Dijkstra's is a 2 step process. Step 1: Update Estimates

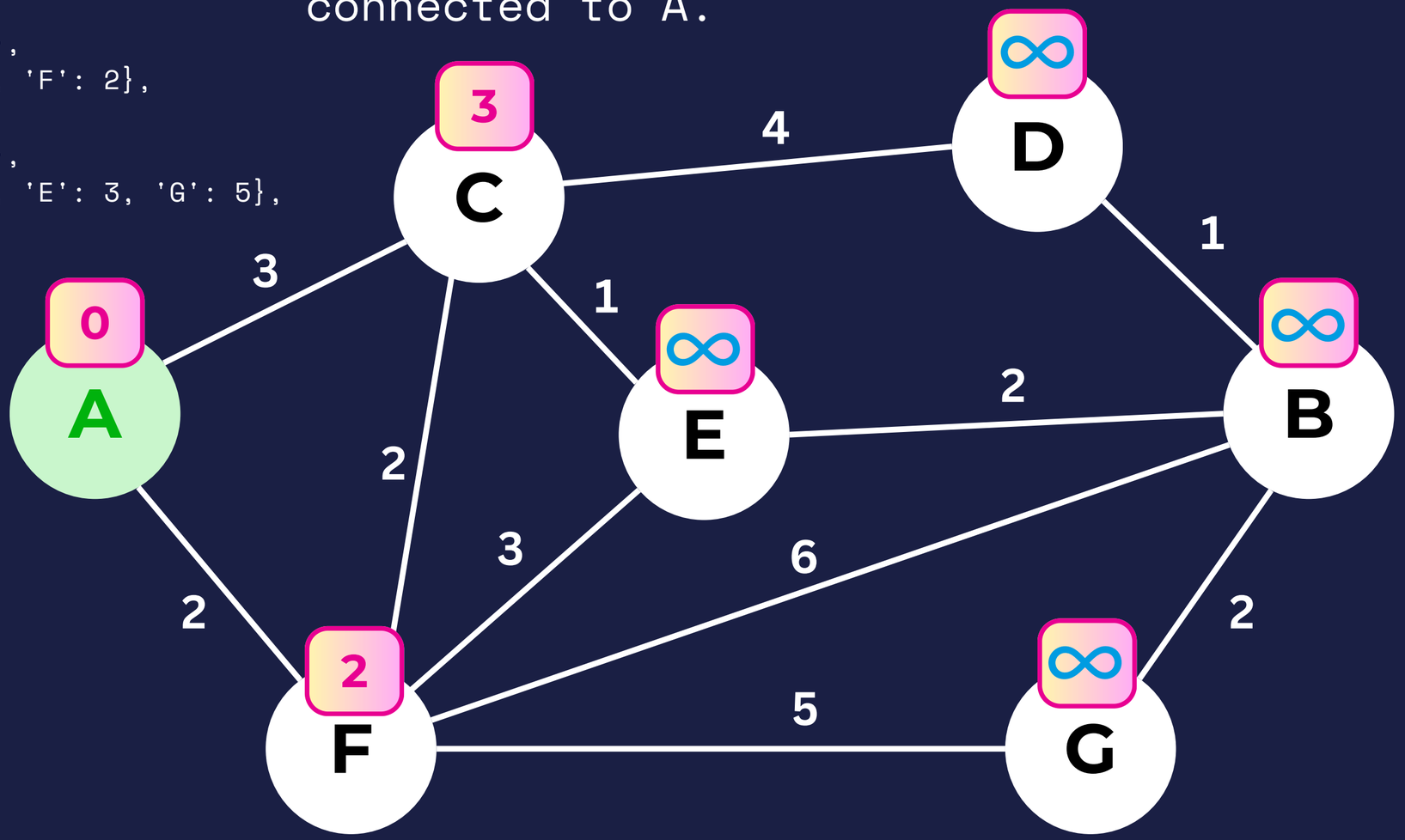
```
graph = {
 'A': {'C': 3, 'F': 2},
 'B': {'D': 1, 'E': 2, 'G': 2},
 'C': {'A': 3, 'D': 4, 'E': 1, 'F': 2},
 'D': {'B': 1, 'C': 4},
 'E': {'B': 2, 'C': 1, 'F': 3},
 'F': {'A': 2, 'B': 6, 'C': 2, 'E': 3, 'G': 5},
 'G': {'B': 2, 'F': 5}
}
```

So we update infinity on C to 3. That estimate is done, we'll move on to F



That is Step 1 done for A. We've checked all available edges connected to A.

```
graph = {
 'A': {'C': 3, 'F': 2},
 'B': {'D': 1, 'E': 2, 'G': 2},
 'C': {'A': 3, 'D': 4, 'E': 1, 'F': 2},
 'D': {'B': 1, 'C': 4},
 'E': {'B': 2, 'C': 1, 'F': 3},
 'F': {'A': 2, 'B': 6, 'C': 2, 'E': 3, 'G': 5},
 'G': {'B': 2, 'F': 5}
}
```

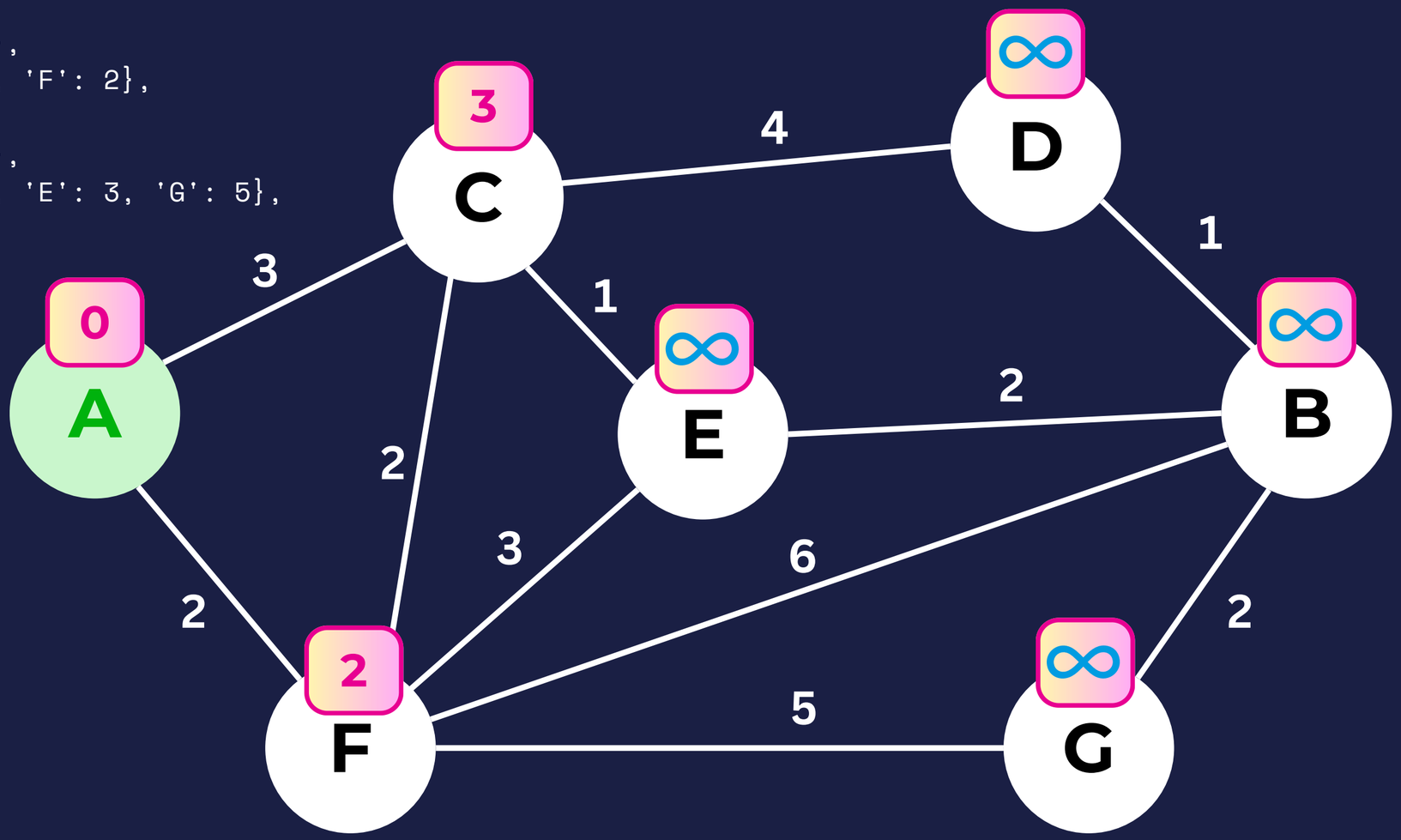


Now we can move on to the next step...

Step 2: Choosing the next vertex.

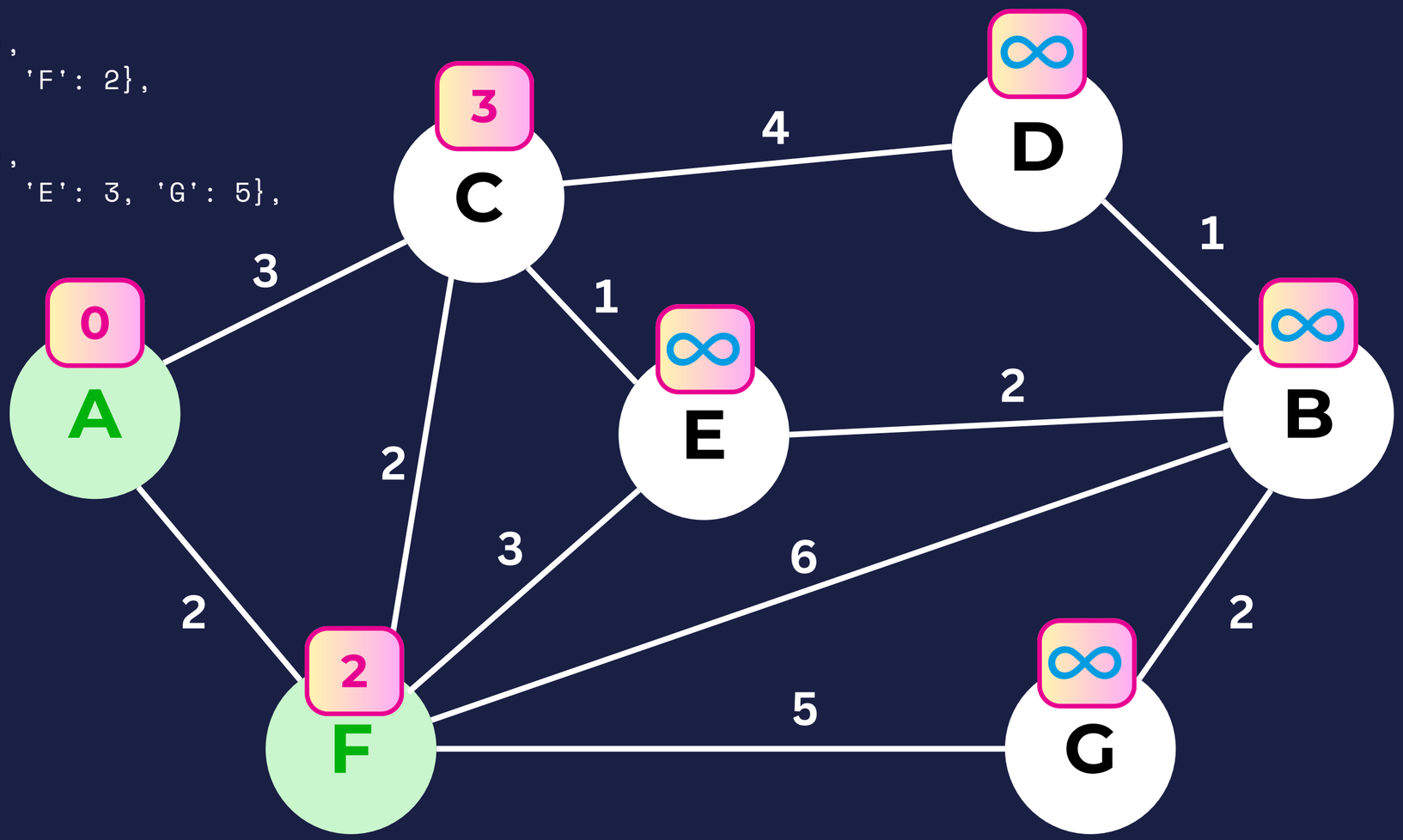
```
graph = {
 'A': {'C': 3, 'F': 2},
 'B': {'D': 1, 'E': 2, 'G': 2},
 'C': {'A': 3, 'D': 4, 'E': 1, 'F': 2},
 'D': {'B': 1, 'C': 4},
 'E': {'B': 2, 'C': 1, 'F': 3},
 'F': {'A': 2, 'B': 6, 'C': 2, 'E': 3, 'G': 5},
 'G': {'B': 2, 'F': 5}
}
```

The next vertex to explore will always be the next unexplored node with the smallest estimate. In this case: F



And we start over by marking F as explored

```
graph = {
 'A': {'C': 3, 'F': 2},
 'B': {'D': 1, 'E': 2, 'G': 2},
 'C': {'A': 3, 'D': 4, 'E': 1, 'F': 2},
 'D': {'B': 1, 'C': 4},
 'E': {'B': 2, 'C': 1, 'F': 3},
 'F': {'A': 2, 'B': 6, 'C': 2, 'E': 3, 'G': 5},
 'G': {'B': 2, 'F': 5}
}
```

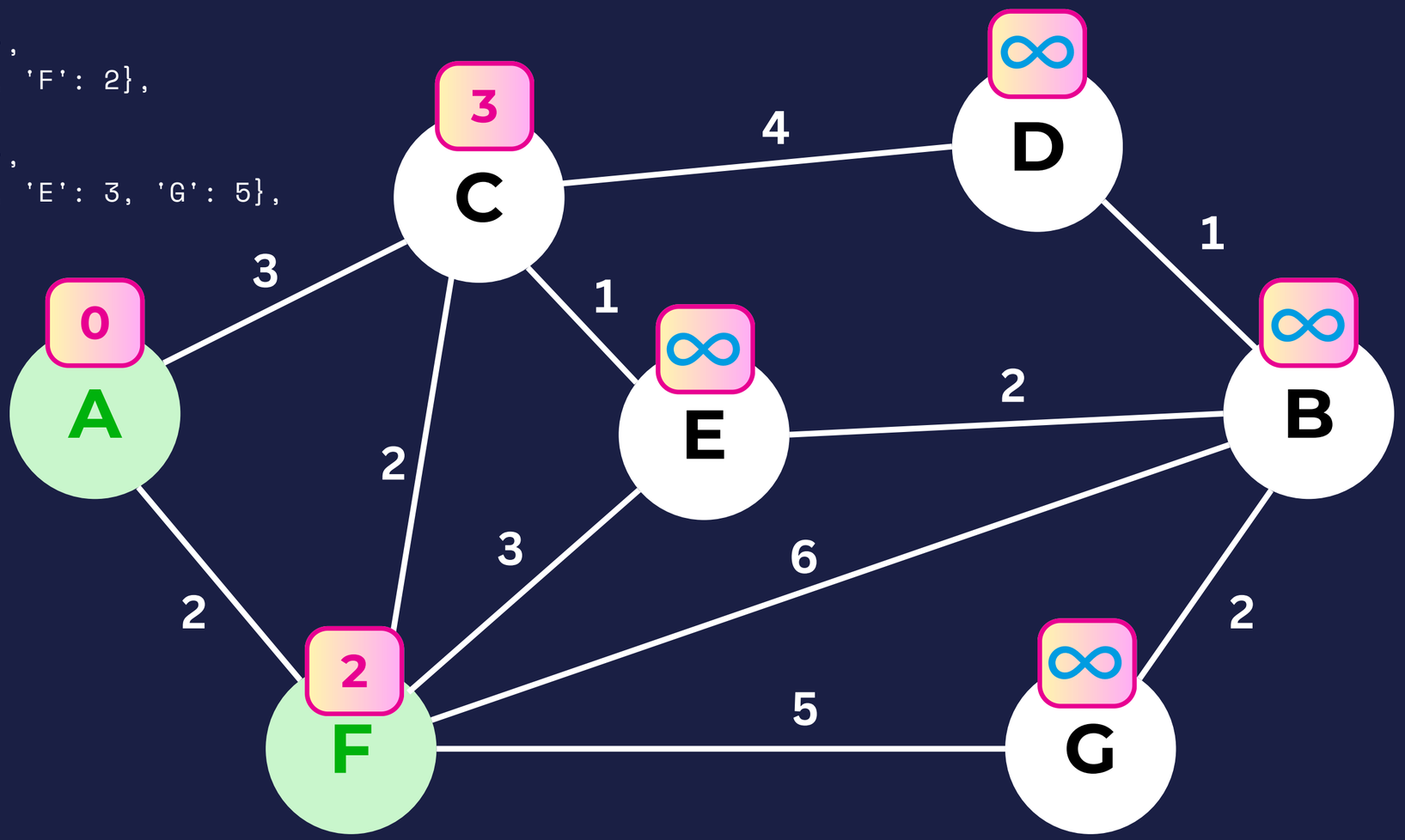


And then restarting the 2 step process with F

Step 1: Updating Estimates

```
graph = {
  'A': {'C': 3, 'F': 2},
  'B': {'D': 1, 'E': 2, 'G': 2},
  'C': {'A': 3, 'D': 4, 'E': 1, 'F': 2},
  'D': {'B': 1, 'C': 4},
  'E': {'B': 2, 'C': 1, 'F': 3},
  'F': {'A': 2, 'B': 6, 'C': 2, 'E': 3, 'G': 5},
  'G': {'B': 2, 'F': 5}
}
```

F to G

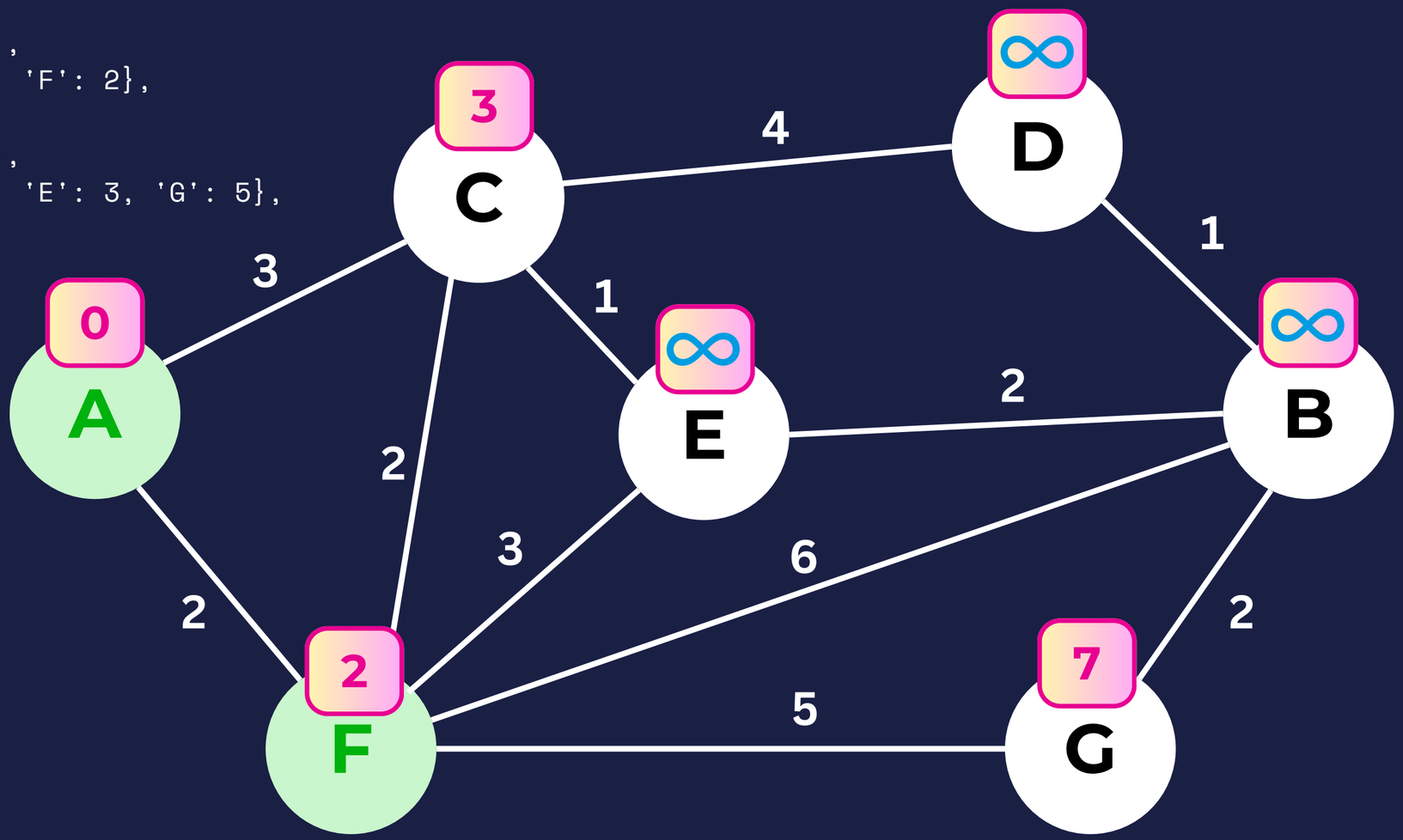


Step 1: Updating Estimates

```
graph = {
 'A': {'C': 3, 'F': 2},
 'B': {'D': 1, 'E': 2, 'G': 2},
 'C': {'A': 3, 'D': 4, 'E': 1, 'F': 2},
 'D': {'B': 1, 'C': 4},
 'E': {'B': 2, 'C': 1, 'F': 3},
 'F': {'A': 2, 'B': 6, 'C': 2, 'E': 3, 'G': 5},
 'G': {'B': 2, 'F': 5}
}
```

F to G:

if we can get from A to F in 2 and F to G in 5, then G's estimate is 7

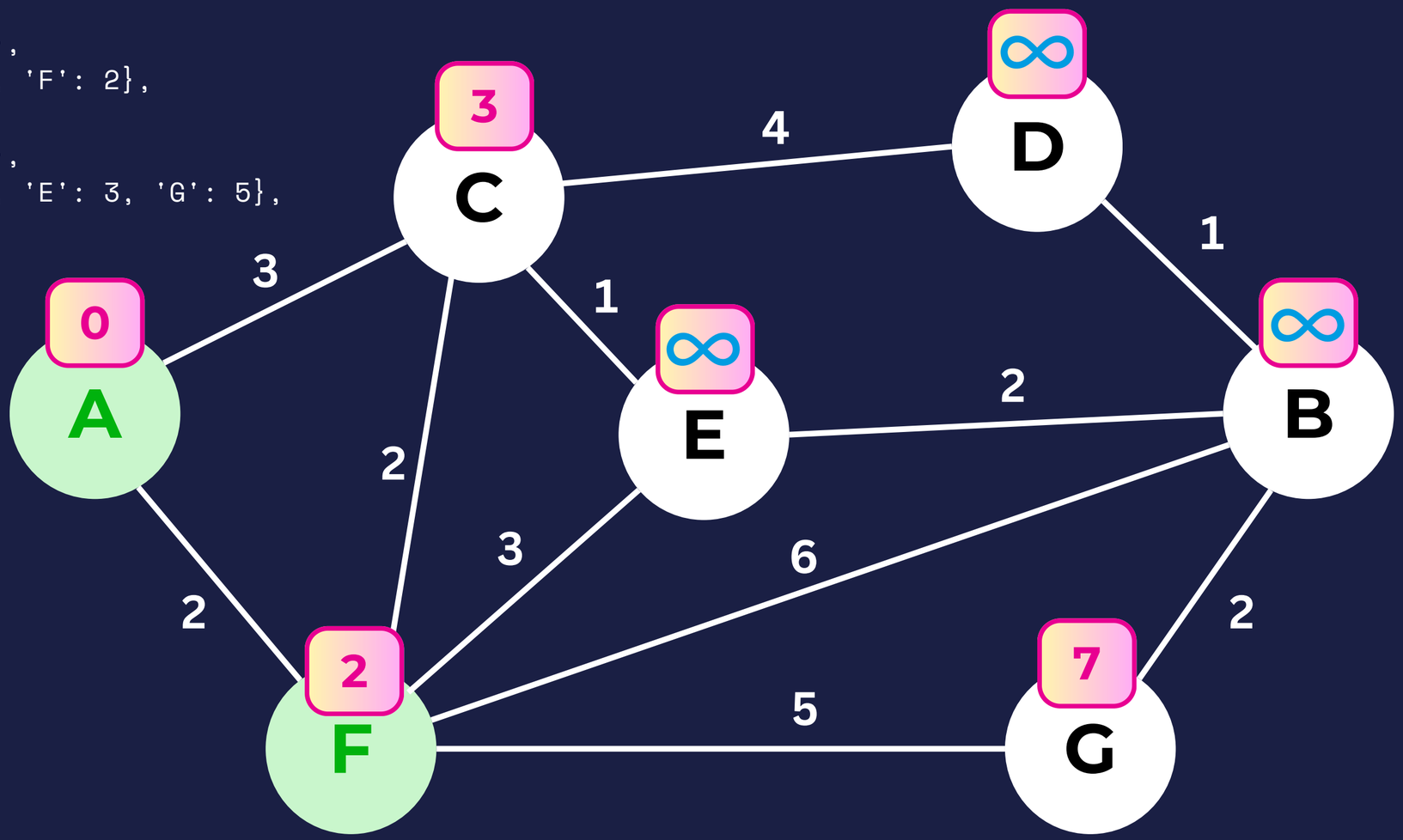


```
graph = {
  'A': {'C': 3, 'F': 2},
  'B': {'D': 1, 'E': 2, 'G': 2},
  'C': {'A': 3, 'D': 4, 'E': 1, 'F': 2},
  'D': {'B': 1, 'C': 4},
  'E': {'B': 2, 'C': 1, 'F': 3},
  'F': {'A': 2, 'B': 6, 'C': 2, 'E': 3, 'G': 5},
  'G': {'B': 2, 'F': 5}
}
```

Step 1: Updating Estimates

F to E:

if we can get from A to F in 2 and F to E in 3 then E's estimate is 5

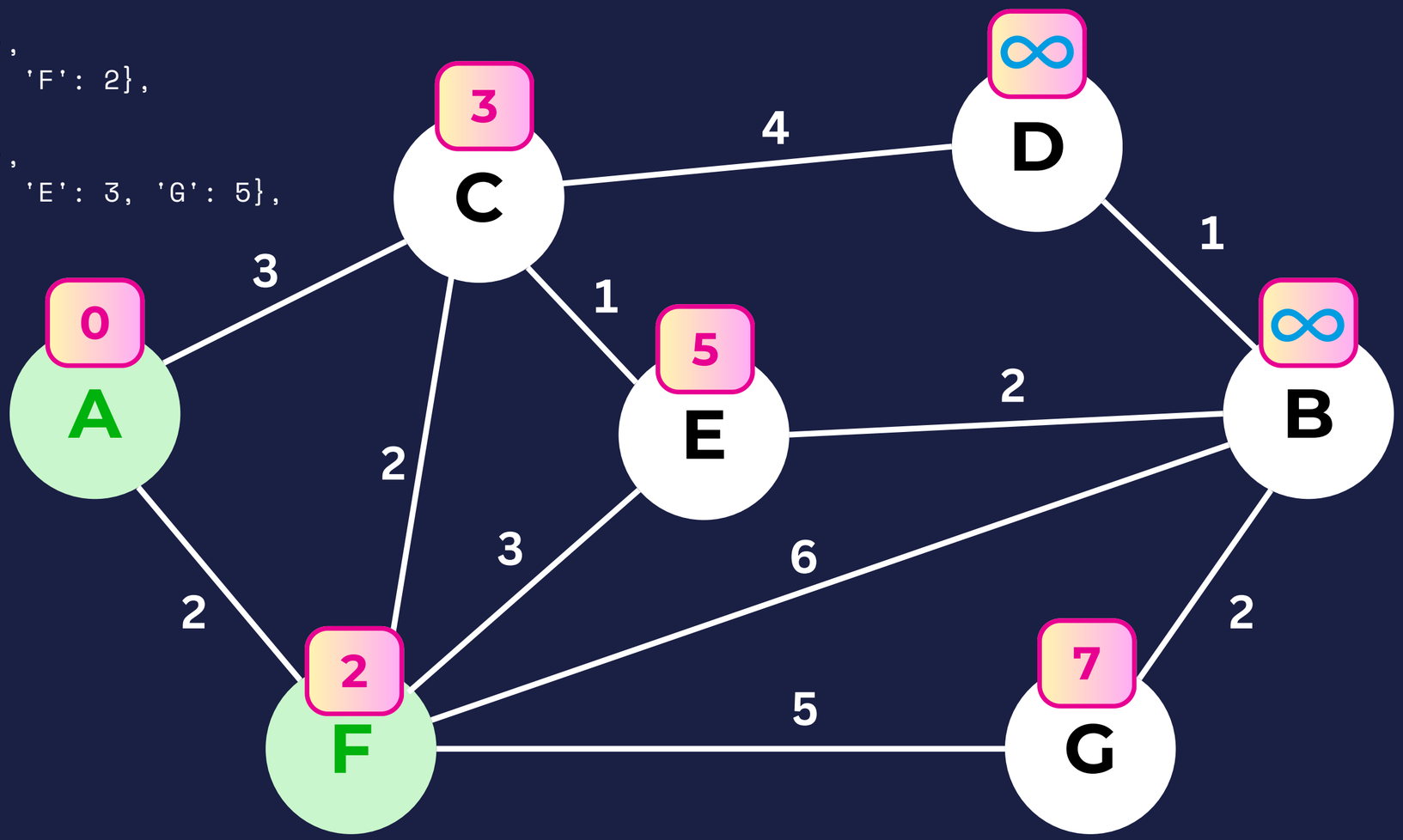


Step 1: Updating Estimates

```
graph = {
 'A': {'C': 3, 'F': 2},
 'B': {'D': 1, 'E': 2, 'G': 2},
 'C': {'A': 3, 'D': 4, 'E': 1, 'F': 2},
 'D': {'B': 1, 'C': 4},
 'E': {'B': 2, 'C': 1, 'F': 3},
 'F': {'A': 2, 'B': 6, 'C': 2, 'E': 3, 'G': 5},
 'G': {'B': 2, 'F': 5}
}
```

F to E:

if we can get from A to F in 2 and F to E in 3 then E's estimate is 5

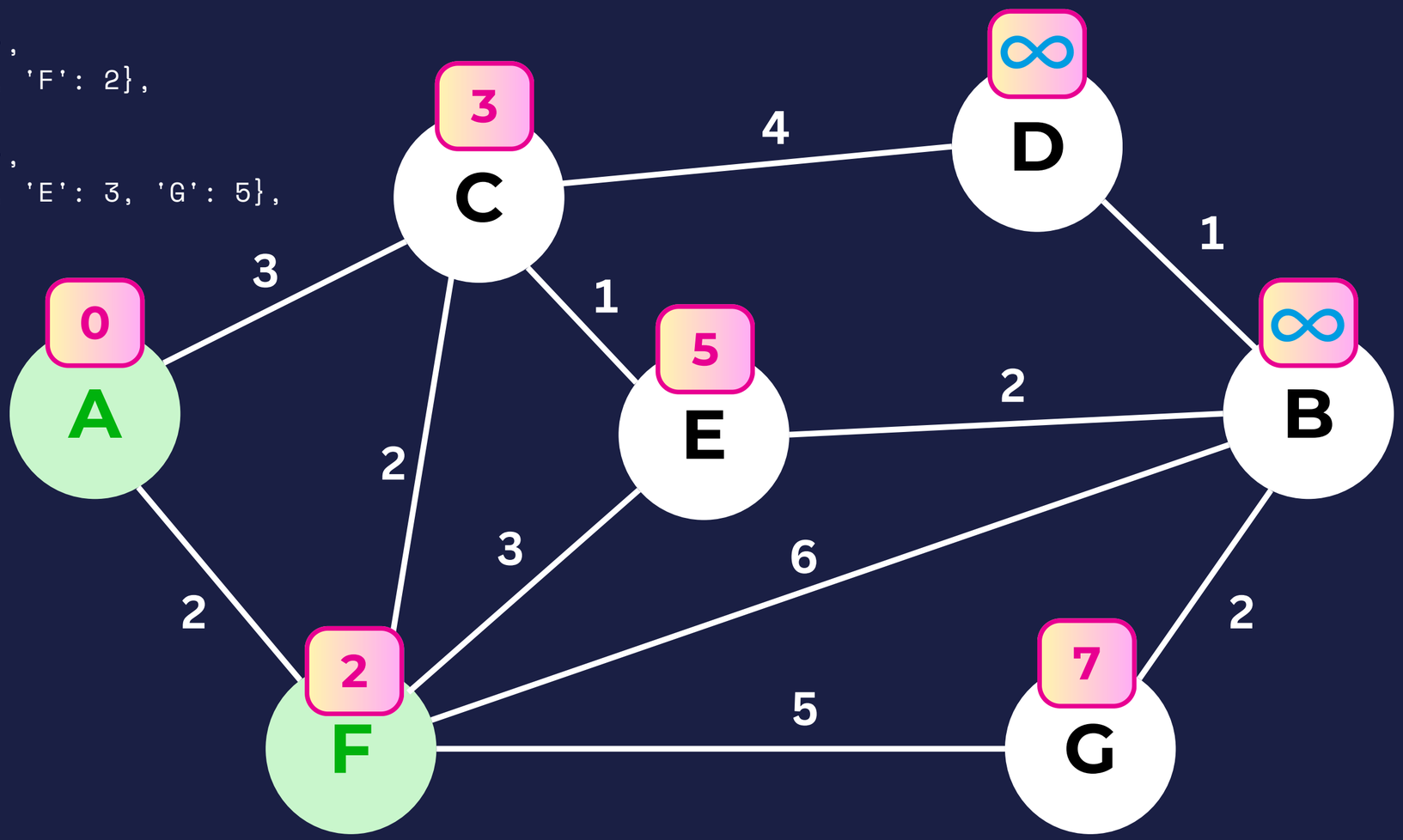


Step 1: Updating Estimates

```
graph = {
 'A': {'C': 3, 'F': 2},
 'B': {'D': 1, 'E': 2, 'G': 2},
 'C': {'A': 3, 'D': 4, 'E': 1, 'F': 2},
 'D': {'B': 1, 'C': 4},
 'E': {'B': 2, 'C': 1, 'F': 3},
 'F': {'A': 2, 'B': 6, 'C': 2, 'E': 3, 'G': 5},
 'G': {'B': 2, 'F': 5}
}
```

F to C:

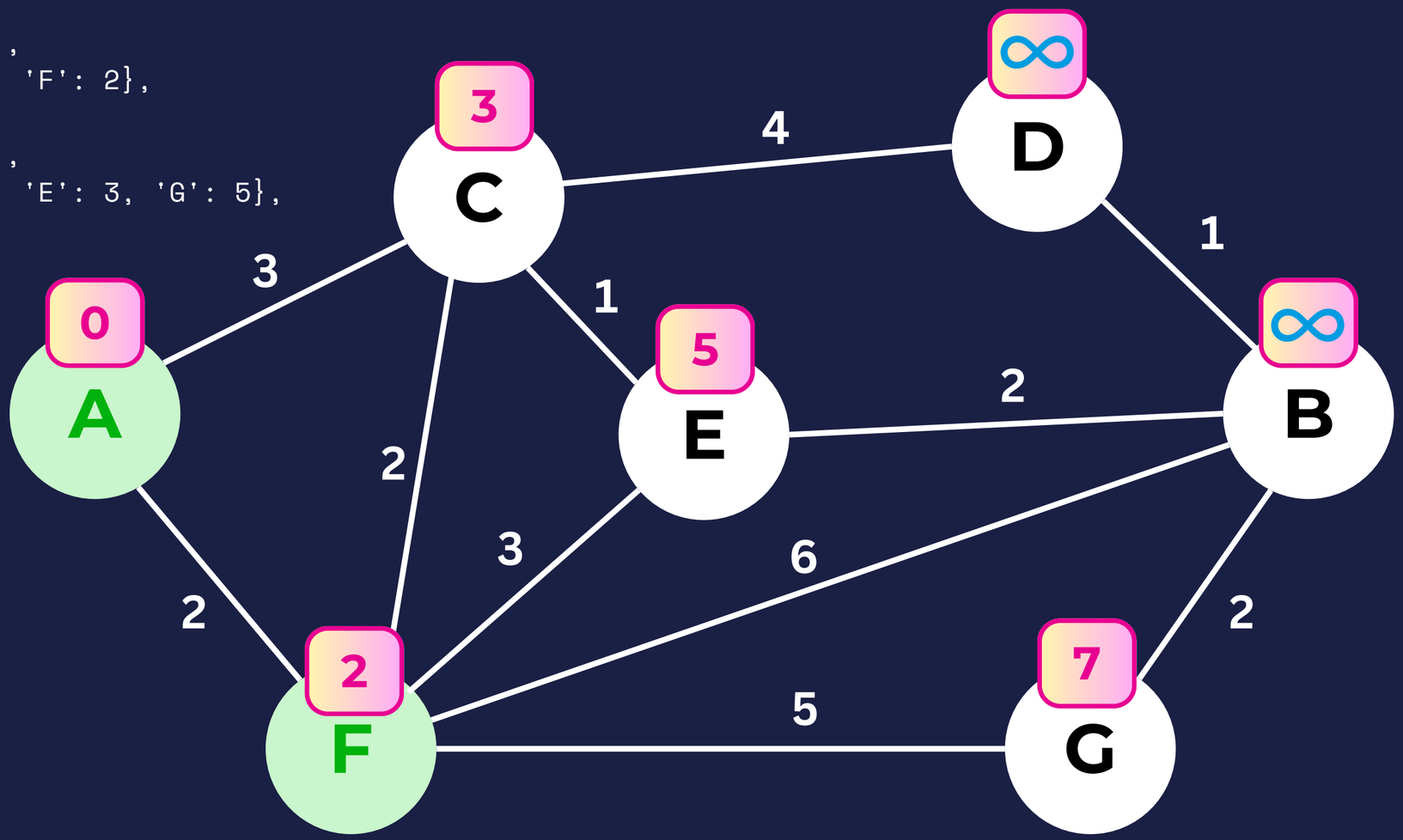
is currently 3, and A to F to C would be 4. Since 3 is less than 4, it stays the same



Step 2: Choosing the next Vertex

```
graph = {
  'A': {'C': 3, 'F': 2},
  'B': {'D': 1, 'E': 2, 'G': 2},
  'C': {'A': 3, 'D': 4, 'E': 1, 'F': 2},
  'D': {'B': 1, 'C': 4},
  'E': {'B': 2, 'C': 1, 'F': 3},
  'F': {'A': 2, 'B': 6, 'C': 2, 'E': 3, 'G': 5},
  'G': {'B': 2, 'F': 5}
}
```

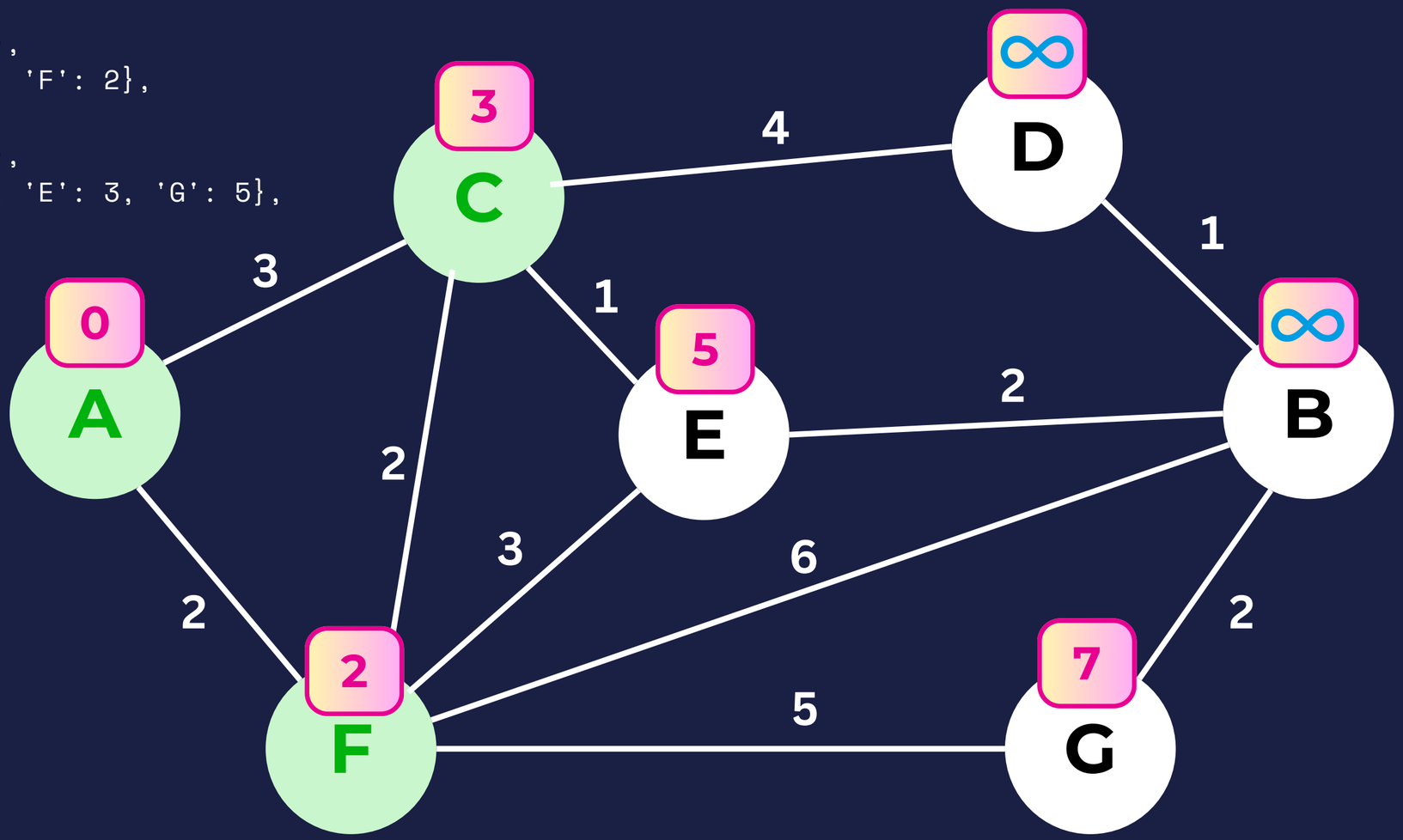
The next smallest unexplored node is: C



```
graph = {
  'A': {'C': 3, 'F': 2},
  'B': {'D': 1, 'E': 2, 'G': 2},
  'C': {'A': 3, 'D': 4, 'E': 1, 'F': 2},
  'D': {'B': 1, 'C': 4},
  'E': {'B': 2, 'C': 1, 'F': 3},
  'F': {'A': 2, 'B': 6, 'C': 2, 'E': 3, 'G': 5},
  'G': {'B': 2, 'F': 5}
}
```

Mark C as Explored

And then we'll move on to step 1

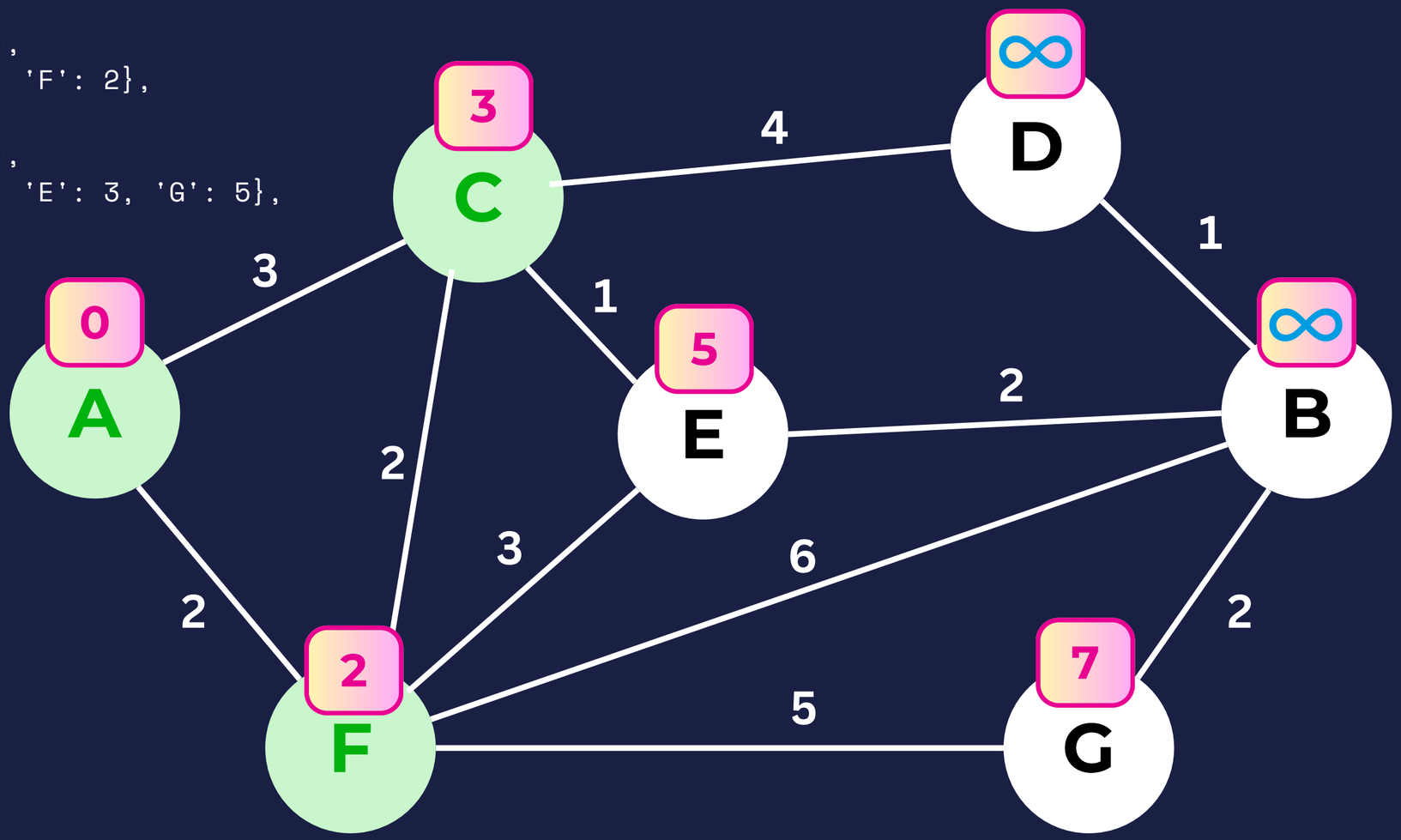


Step 1: Updating Estimates

```
graph = {
  'A': {'C': 3, 'F': 2},
  'B': {'D': 1, 'E': 2, 'G': 2},
  'C': {'A': 3, 'D': 4, 'E': 1, 'F': 2},
  'D': {'B': 1, 'C': 4},
  'E': {'B': 2, 'C': 1, 'F': 3},
  'F': {'A': 2, 'B': 6, 'C': 2, 'E': 3, 'G': 5},
  'G': {'B': 2, 'F': 5}
}
```

C to D:

A to C is 3 and
C to D is 4 so
D gets updated
to 7

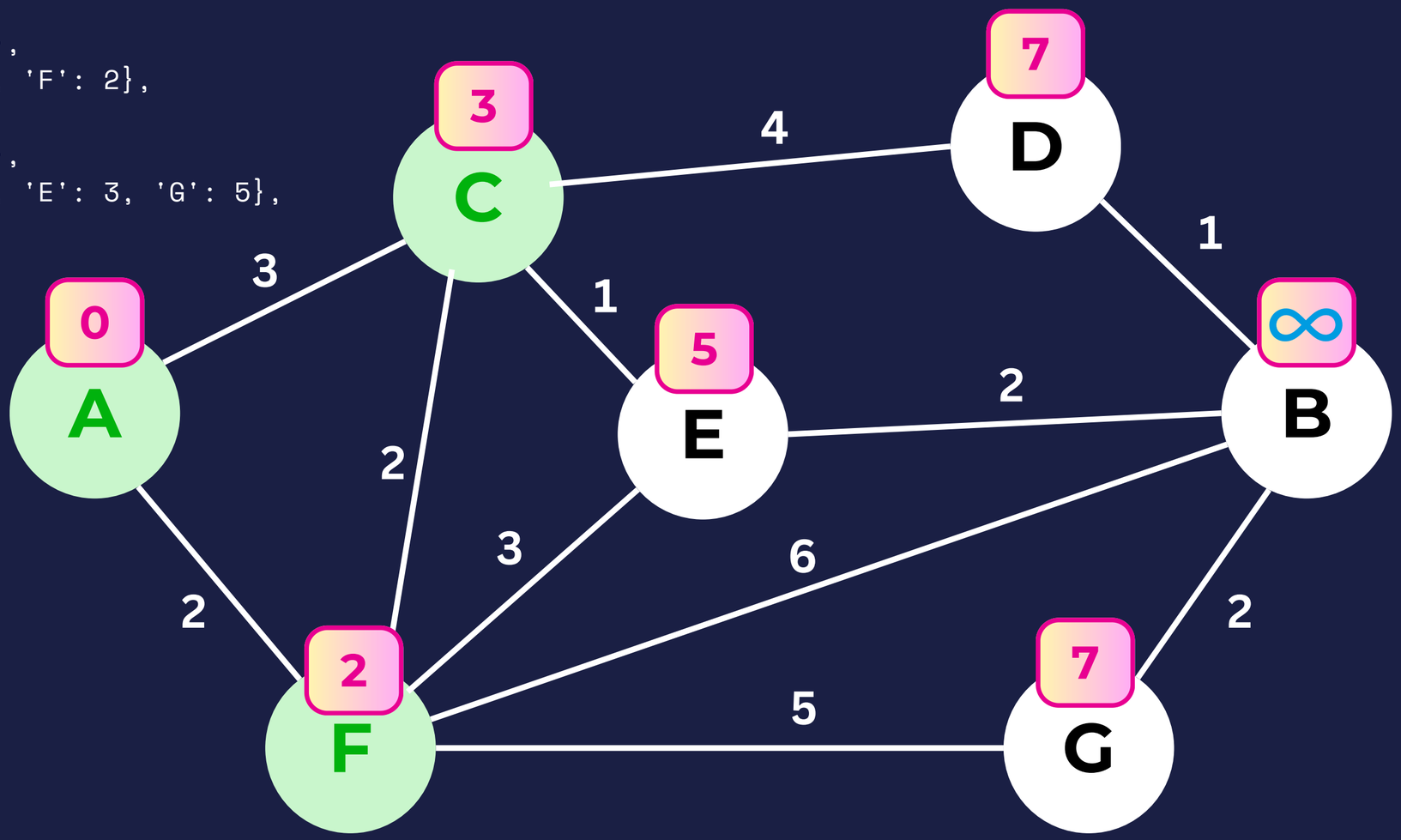


Step 1: Updating Estimates

```
graph = {
  'A': {'C': 3, 'F': 2},
  'B': {'D': 1, 'E': 2, 'G': 2},
  'C': {'A': 3, 'D': 4, 'E': 1, 'F': 2},
  'D': {'B': 1, 'C': 4},
  'E': {'B': 2, 'C': 1, 'F': 3},
  'F': {'A': 2, 'B': 6, 'C': 2, 'E': 3, 'G': 5},
  'G': {'B': 2, 'F': 5}
}
```

C to D:

A to C is 3 and
C to D is 4 so
D gets updated
to 7

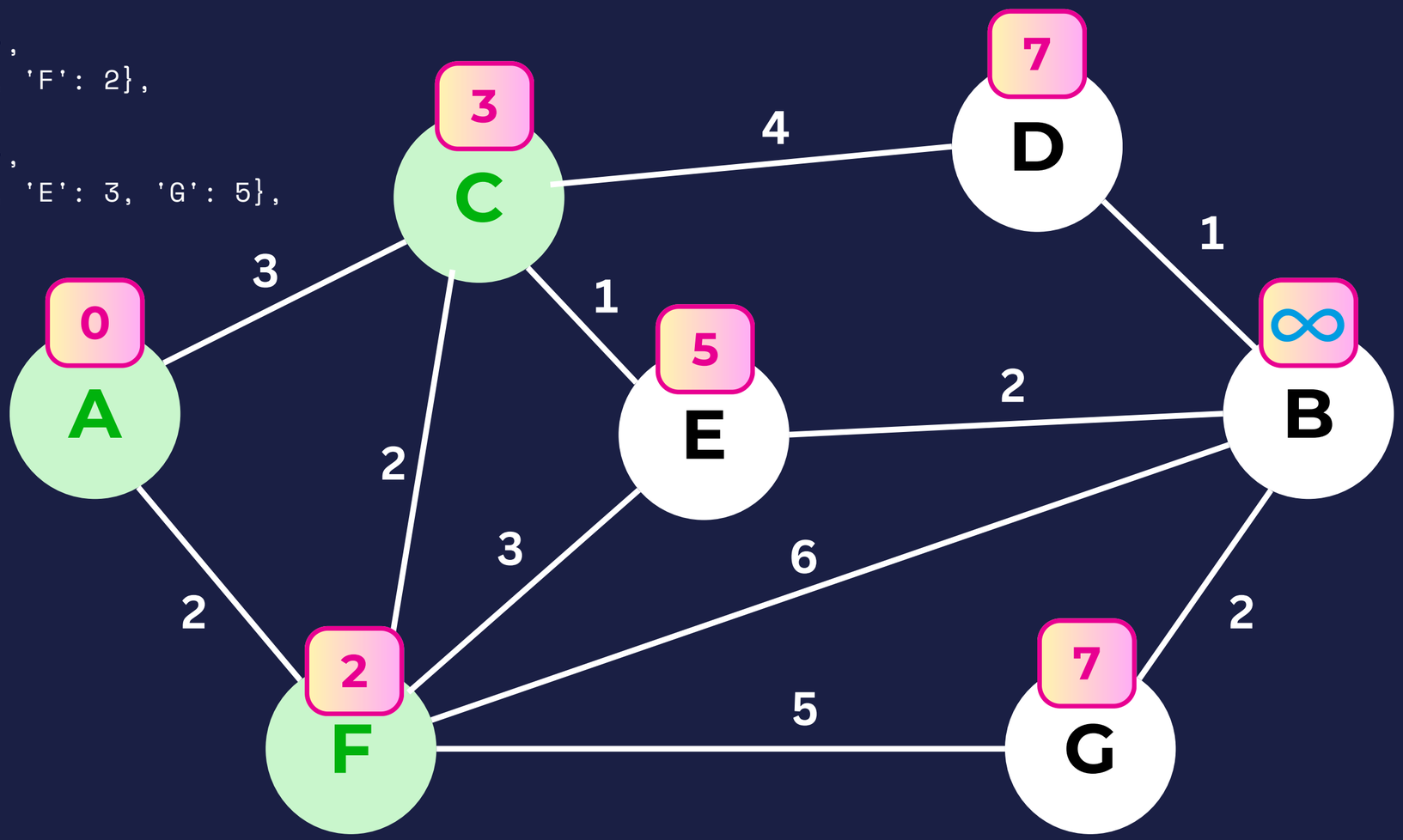


Step 1: Updating Estimates

```
graph = {
 'A': {'C': 3, 'F': 2},
 'B': {'D': 1, 'E': 2, 'G': 2},
 'C': {'A': 3, 'D': 4, 'E': 1, 'F': 2},
 'D': {'B': 1, 'C': 4},
 'E': {'B': 2, 'C': 1, 'F': 3},
 'F': {'A': 2, 'B': 6, 'C': 2, 'E': 3, 'G': 5},
 'G': {'B': 2, 'F': 5}
}
```

C to E:

A to C is 3 and C to E is 1 which makes 4. 4 is less than 5, so we update E to 4.

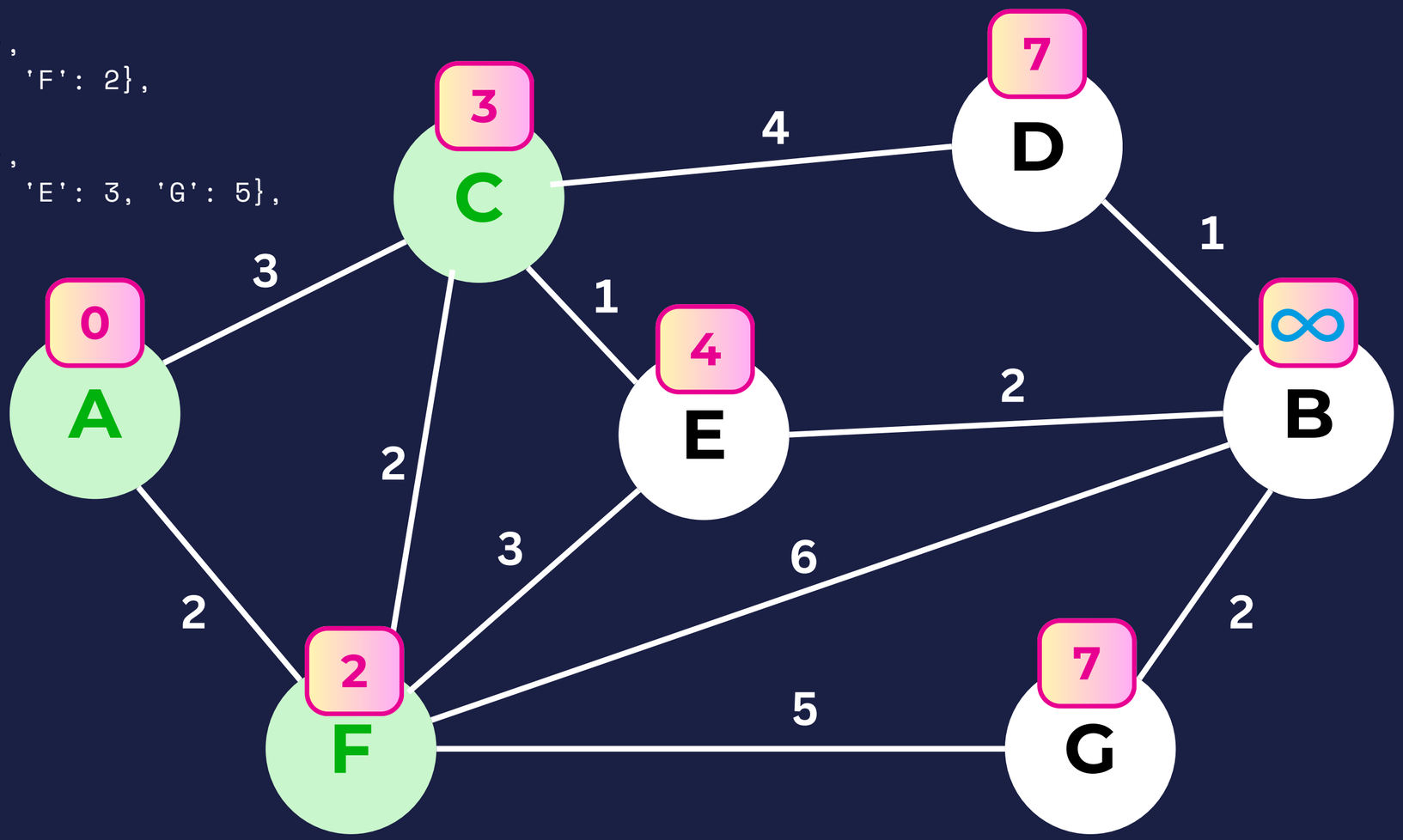


Step 1: Updating Estimates

```
graph = {
 'A': {'C': 3, 'F': 2},
 'B': {'D': 1, 'E': 2, 'G': 2},
 'C': {'A': 3, 'D': 4, 'E': 1, 'F': 2},
 'D': {'B': 1, 'C': 4},
 'E': {'B': 2, 'C': 1, 'F': 3},
 'F': {'A': 2, 'B': 6, 'C': 2, 'E': 3, 'G': 5},
 'G': {'B': 2, 'F': 5}
}
```

C to E:

A to C is 3 and C to E is 1 which makes 4. 4 is less than 5, so we update E to 4.

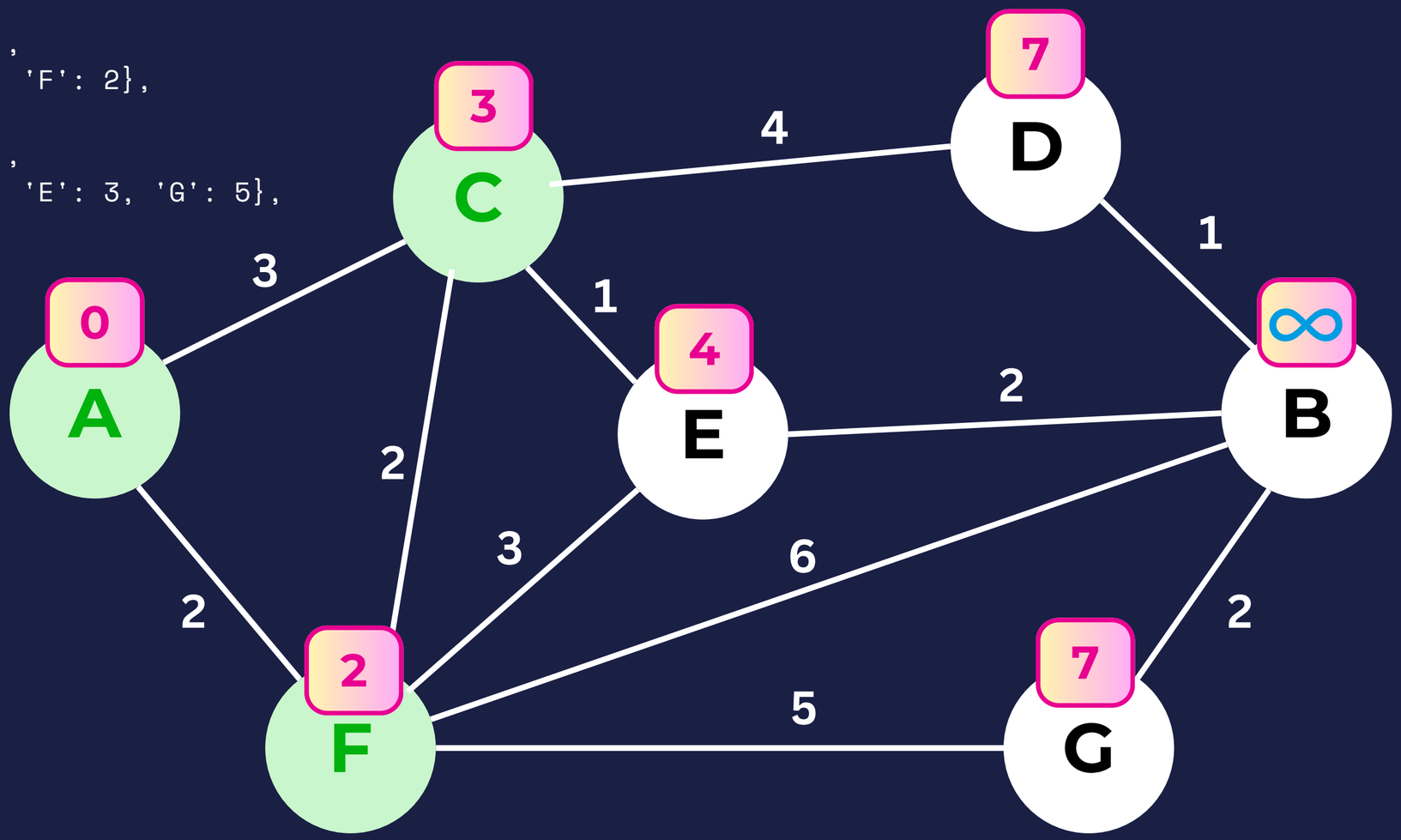


Step 1: Updating Estimates

```
graph = {
 'A': {'C': 3, 'F': 2},
 'B': {'D': 1, 'E': 2, 'G': 2},
 'C': {'A': 3, 'D': 4, 'E': 1, 'F': 2},
 'D': {'B': 1, 'C': 4},
 'E': {'B': 2, 'C': 1, 'F': 3},
 'F': {'A': 2, 'B': 6, 'C': 2, 'E': 3, 'G': 5},
 'G': {'B': 2, 'F': 5}
}
```

C to F:

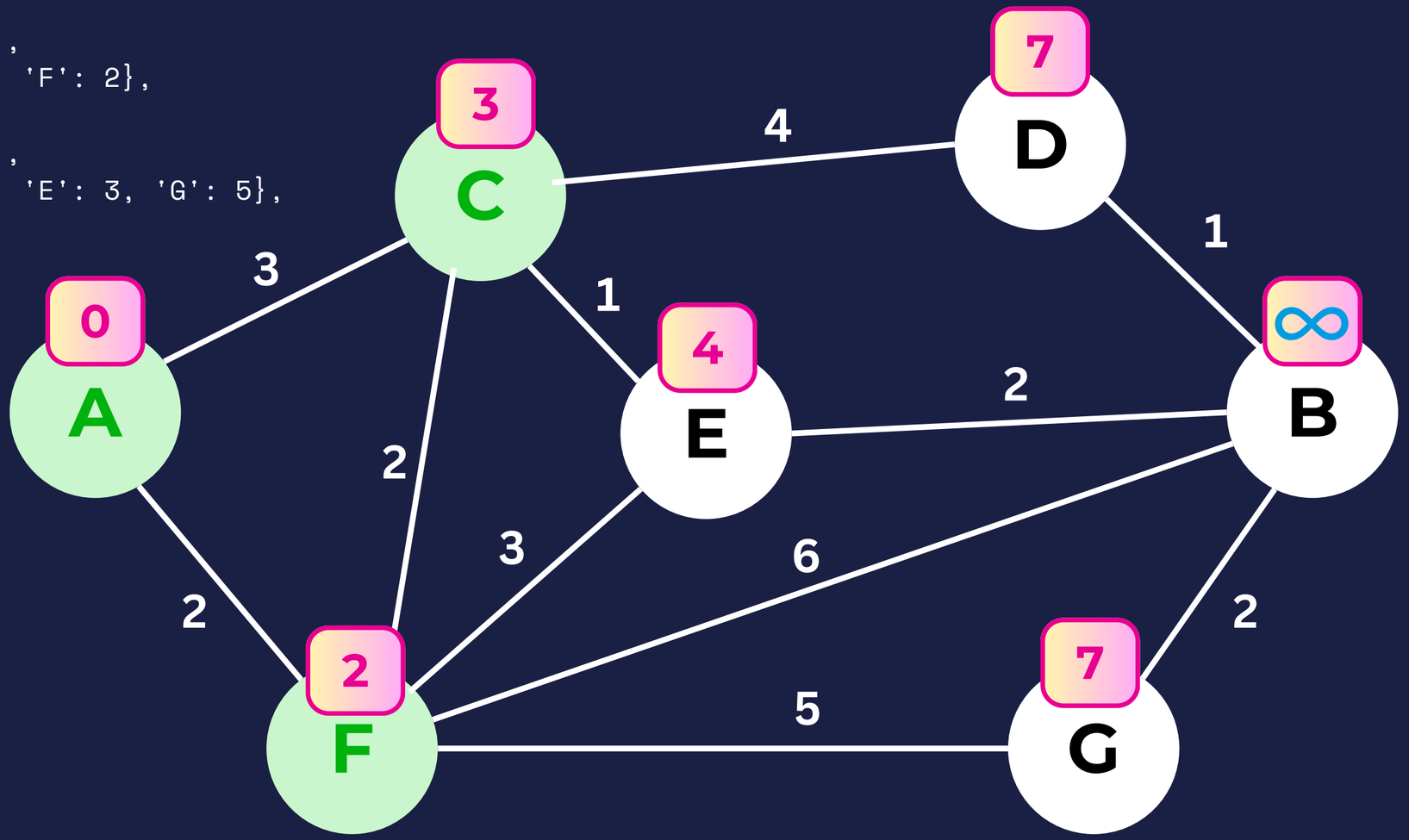
A to C is 3 and C to F is 2 which is 5. F is currently at 2 which is less, so that stays.



Step 2: Choosing the next Vertex

```
graph = {
 'A': {'C': 3, 'F': 2},
 'B': {'D': 1, 'E': 2, 'G': 2},
 'C': {'A': 3, 'D': 4, 'E': 1, 'F': 2},
 'D': {'B': 1, 'C': 4},
 'E': {'B': 2, 'C': 1, 'F': 3},
 'F': {'A': 2, 'B': 6, 'C': 2, 'E': 3, 'G': 5},
 'G': {'B': 2, 'F': 5}
}
```

The next smallest unexplored node is: E

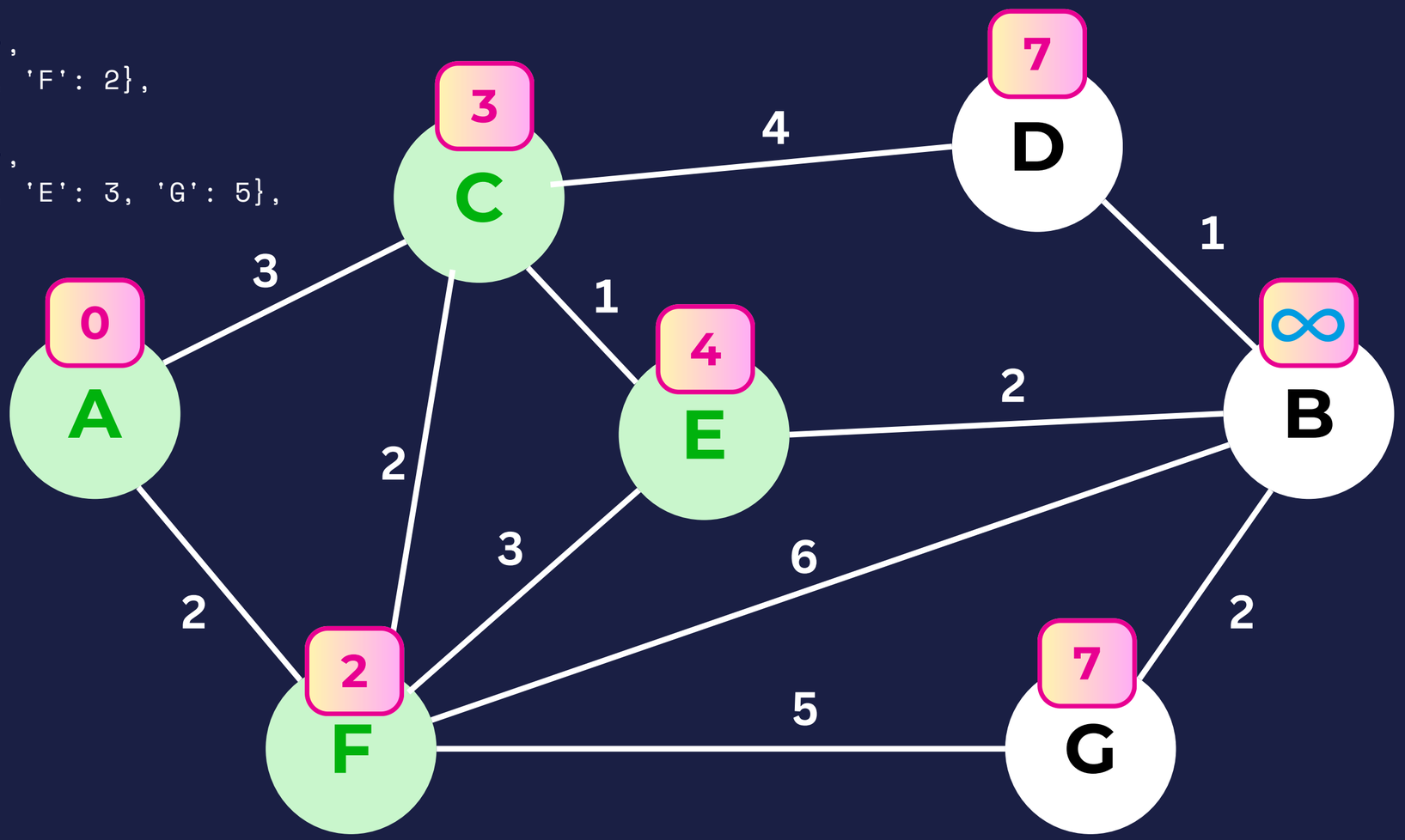


Mark E as Explored and Step 1: Updating Estimates

```
graph = {
 'A': {'C': 3, 'F': 2},
 'B': {'D': 1, 'E': 2, 'G': 2},
 'C': {'A': 3, 'D': 4, 'E': 1, 'F': 2},
 'D': {'B': 1, 'C': 4},
 'E': {'B': 2, 'C': 1, 'F': 3},
 'F': {'A': 2, 'B': 6, 'C': 2, 'E': 3, 'G': 5},
 'G': {'B': 2, 'F': 5}
}
```

E to B:

A to E (shortest path) is 4, E to B is 2. So B gets updated to 6

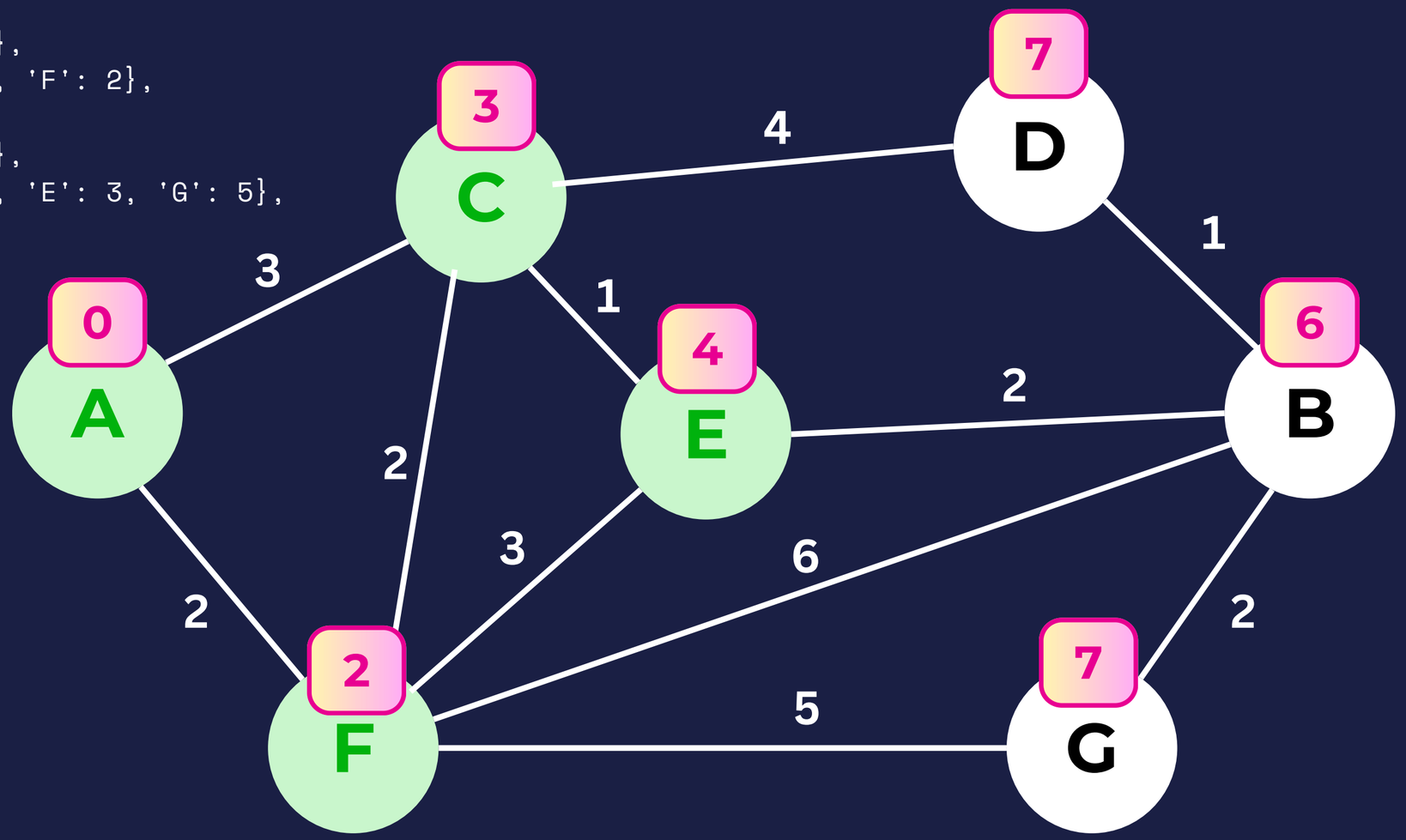


Mark E as Explored and Step 1: Updating Estimates

```
graph = {
  'A': {'C': 3, 'F': 2},
  'B': {'D': 1, 'E': 2, 'G': 2},
  'C': {'A': 3, 'D': 4, 'E': 1, 'F': 2},
  'D': {'B': 1, 'C': 4},
  'E': {'B': 2, 'C': 1, 'F': 3},
  'F': {'A': 2, 'B': 6, 'C': 2, 'E': 3, 'G': 5},
  'G': {'B': 2, 'F': 5}
}
```

E to B:

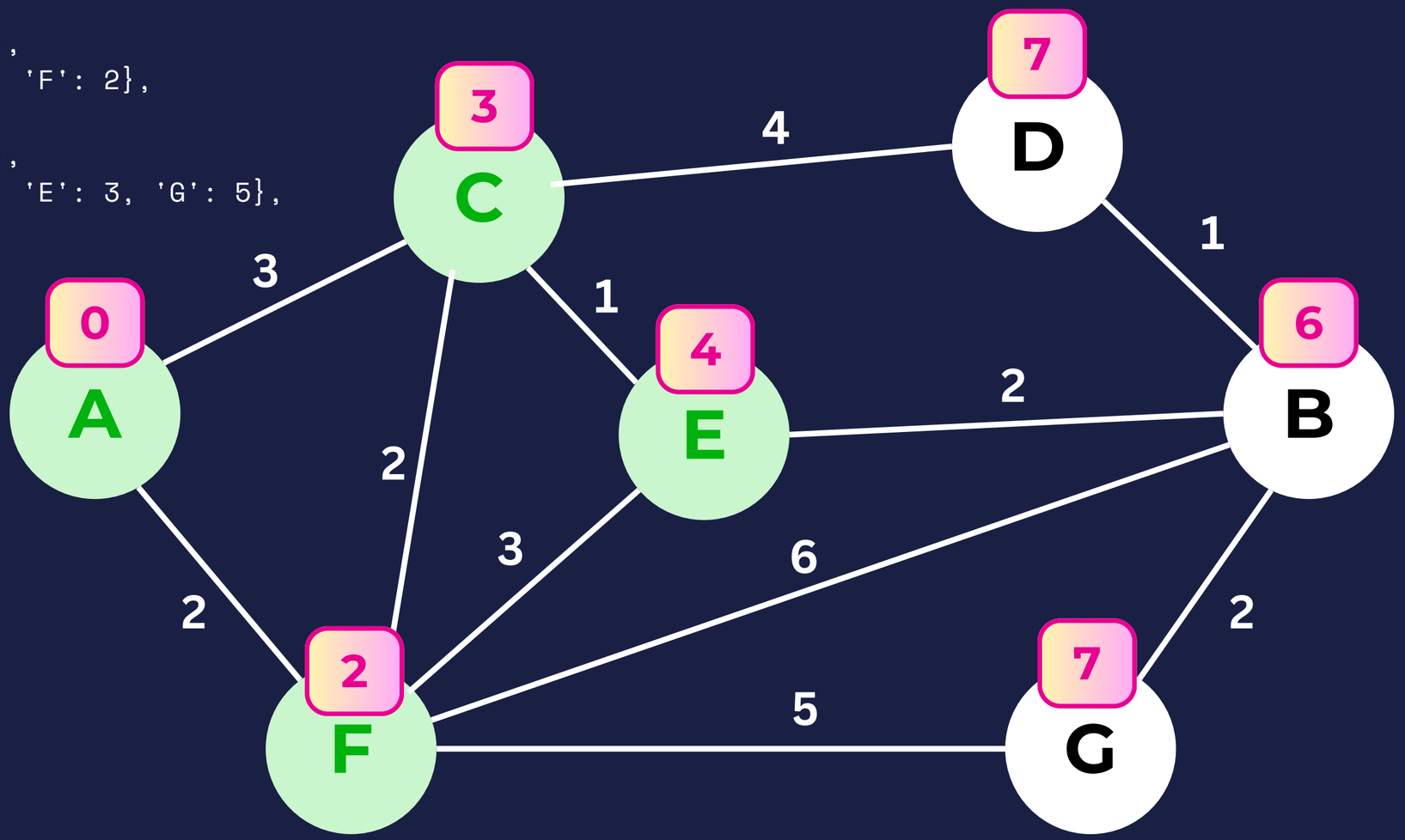
A to E (shortest path) is 4, E to B is 2. So B gets updated to 6



Step 2: Choosing the next Vertex

```
graph = {
 'A': {'C': 3, 'F': 2},
 'B': {'D': 1, 'E': 2, 'G': 2},
 'C': {'A': 3, 'D': 4, 'E': 1, 'F': 2},
 'D': {'B': 1, 'C': 4},
 'E': {'B': 2, 'C': 1, 'F': 3},
 'F': {'A': 2, 'B': 6, 'C': 2, 'E': 3, 'G': 5},
 'G': {'B': 2, 'F': 5}
}
```

The next smallest unexplored node is: B

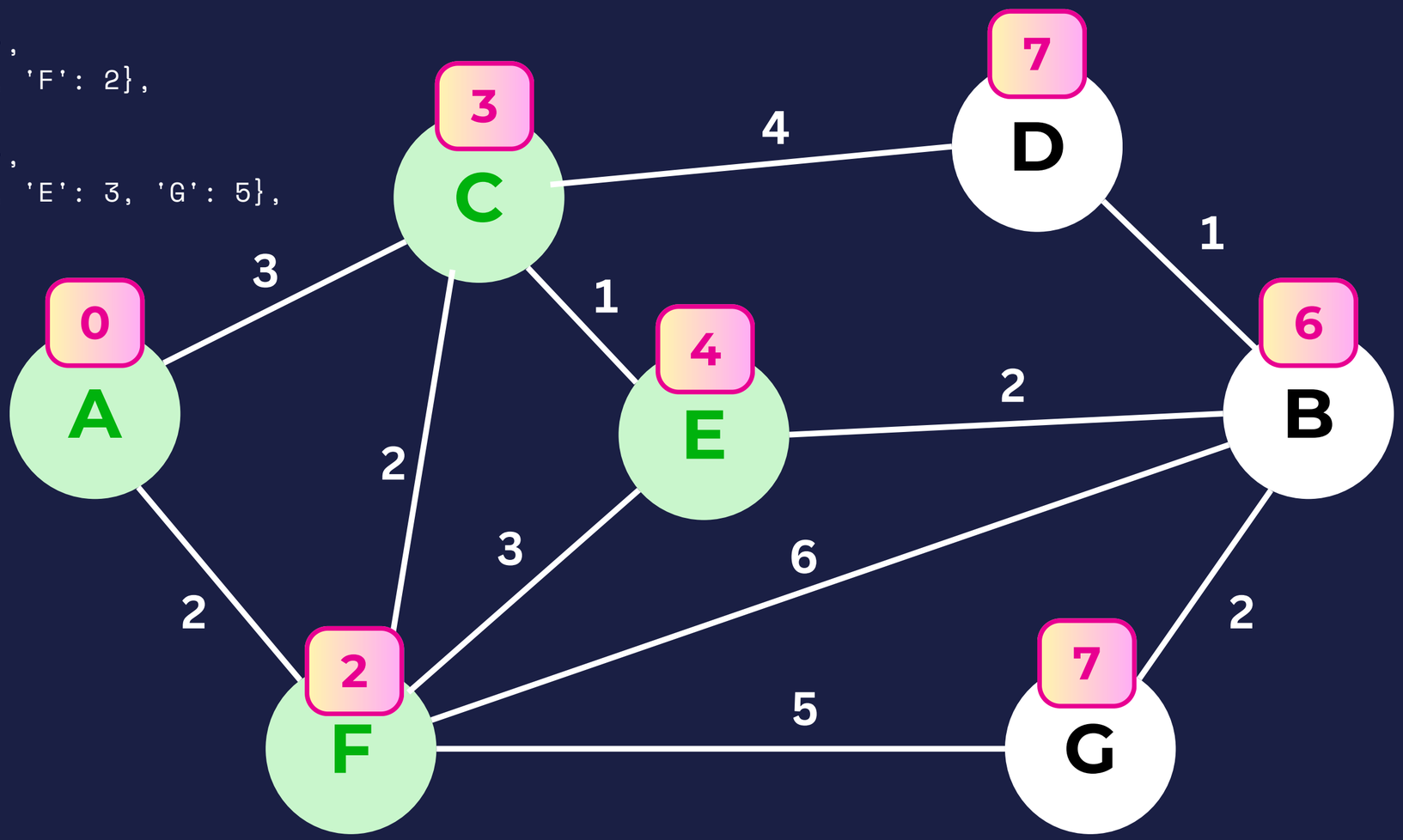


Mark B as Explored and we're done!

```
graph = {
 'A': {'C': 3, 'F': 2},
 'B': {'D': 1, 'E': 2, 'G': 2},
 'C': {'A': 3, 'D': 4, 'E': 1, 'F': 2},
 'D': {'B': 1, 'C': 4},
 'E': {'B': 2, 'C': 1, 'F': 3},
 'F': {'A': 2, 'B': 6, 'C': 2, 'E': 3, 'G': 5},
 'G': {'B': 2, 'F': 5}
}
```

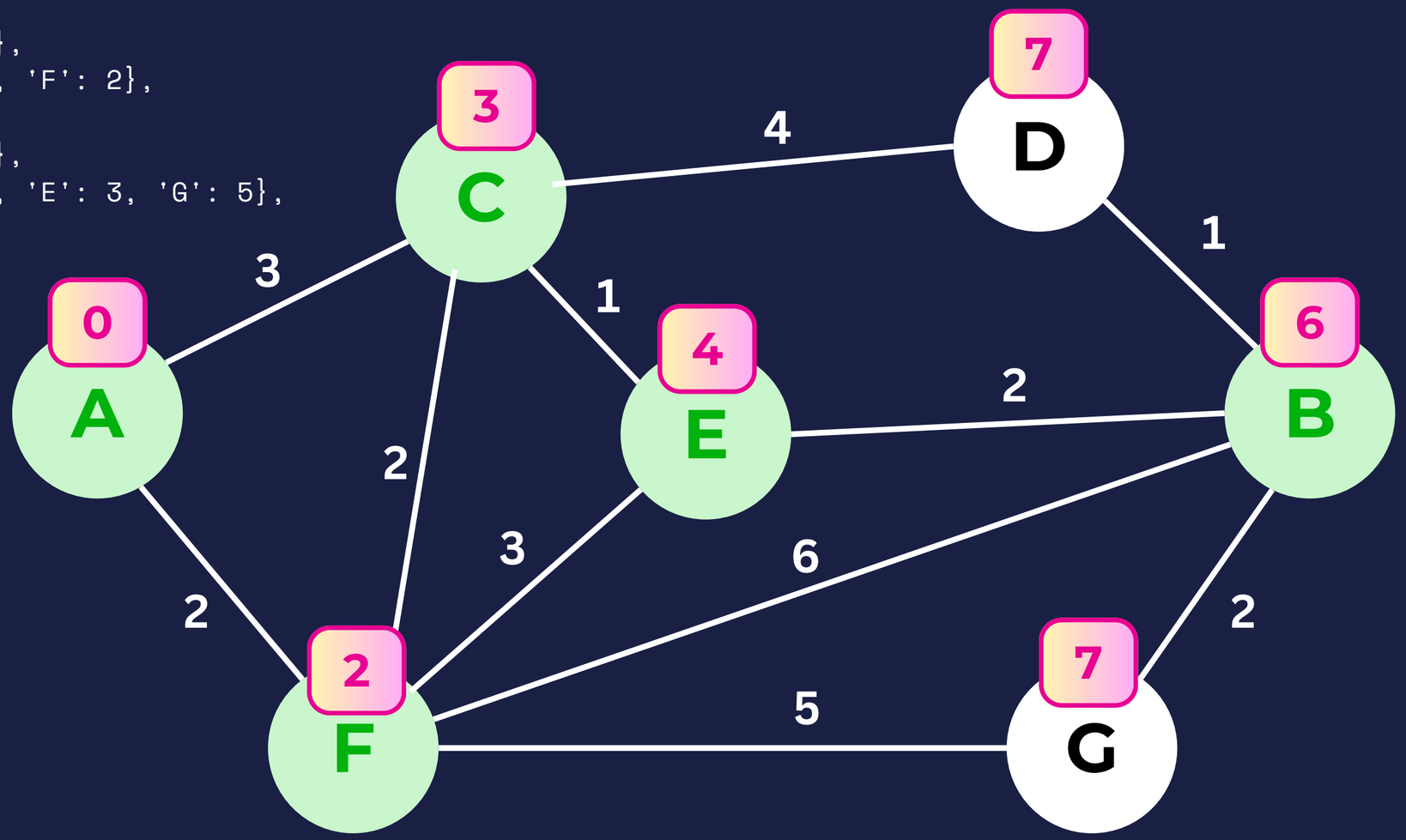
Since our goal was to find the shortest possible path from A to B!

Once B is explored, then we've found it!



```
graph = {
 'A': {'C': 3, 'F': 2},
 'B': {'D': 1, 'E': 2, 'G': 2},
 'C': {'A': 3, 'D': 4, 'E': 1, 'F': 2},
 'D': {'B': 1, 'C': 4},
 'E': {'B': 2, 'C': 1, 'F': 3},
 'F': {'A': 2, 'B': 6, 'C': 2, 'E': 3, 'G': 5},
 'G': {'B': 2, 'F': 5}
}
```

Mark B as Explored and we're done!



Since our goal was to find the shortest possible path from A to B!

Once B is explored, then we've found it!

THE PSEUDOCODE

```
function dijkstra(graph, start):
    distances = CREATE dictionary
    SET all nodes distances to inifity
    SET start node distance to 0

    queue = CREATE priority
    queue.ADD (0, start)

    WHILE (queue is NOT empty):
        queue.REMOVE(smallest distance node)
        current_distance, current_node = smallest distance node value

        IF current_distance > distances[current_node]:
            CONTINUE to next node

        FOR neighbors in graph[current_node]:
            CALCULATE distance = current_distance + weight

            IF distance < neighbor distance:
                UPDATE distances[neighbor] = distance
                queue.ADD(distance, neighbor)
    return distances
```

EXAMPLES REPLIT! PLEASE GO TO:

HTTPS://REPLIT.COM/@RIKKIEHRHART/GRABABYTE

UP NEXT

Apr 23 - Dynamic Programming (Knapsack Problem)

Apr 30 - Union-Find

May 7 - Kruskal's Algorithm

May 14 - Prim's Algorithm

Questions? - rikki.ehrhart@g.ausitncc.edu

If you'd like the opportunity to run a Grab a Byte algorithm workshop, please let me know!