GRAB A BYTE

KRUSKAL'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, Hashing, Dijkstras, Dynamic Programming and Union Find!

WHAT WE ARE COVERING TODAY

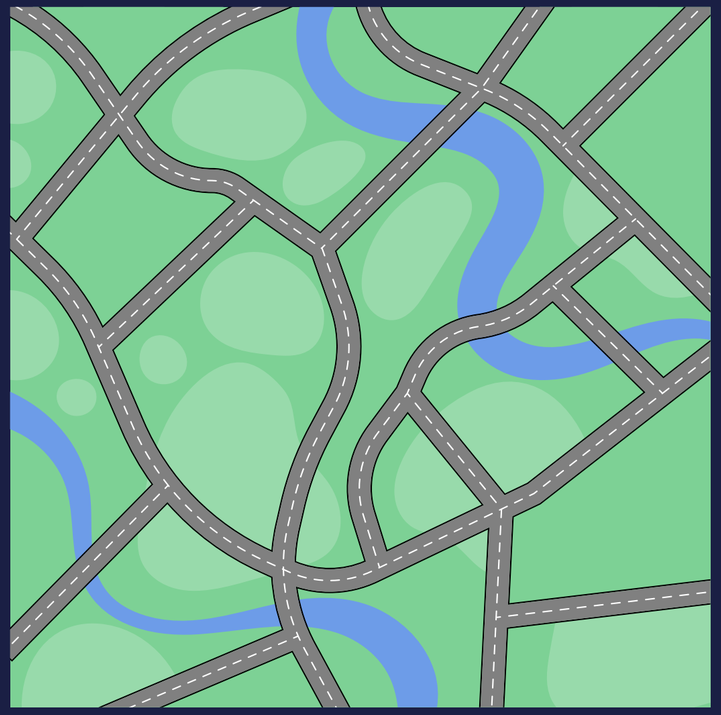
Today we are covering Kruskal’s Algorithm!

WHAT IS KRUSKALS?

Kruskal’s algorithm is a greedy algorithm used to find the minimum spanning tree (MST) for a connected, weighted graph.

It connects all the vertices together with the smallest total edge weight without creating any cycles.

WHAT?



Think of it like there is a bunch of cities and you want to build roads to connect them.

You want the cheapest overall cost without making loops.

KRUSKAL'S

- Always picks the cheapest available path first
- Keeps connecting points as cheaply as possible
- Don't create loops!

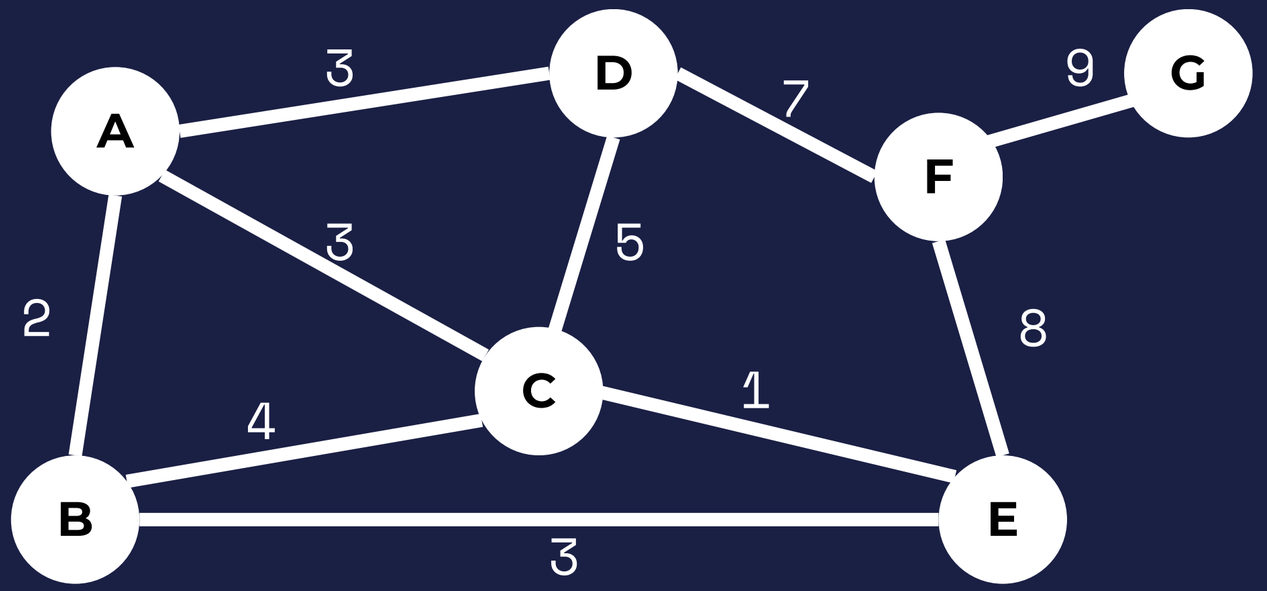
Lets consider a graph of values:

```
graph = {
  'A': {'B': 2, 'C': 3, 'D': 3},
  'B': {'A': 2, 'C': 4, 'E': 3},
  'C': {'A': 3, 'B': 4, 'D': 5, 'E': 1},
  'D': {'A': 2, 'C': 5, 'F': 7},
  'E': {'B': 3, 'C': 1, 'F': 8},
  'F': {'D': 7, 'E': 8, 'G': 9},
  'G': {'F': 9}
}
```

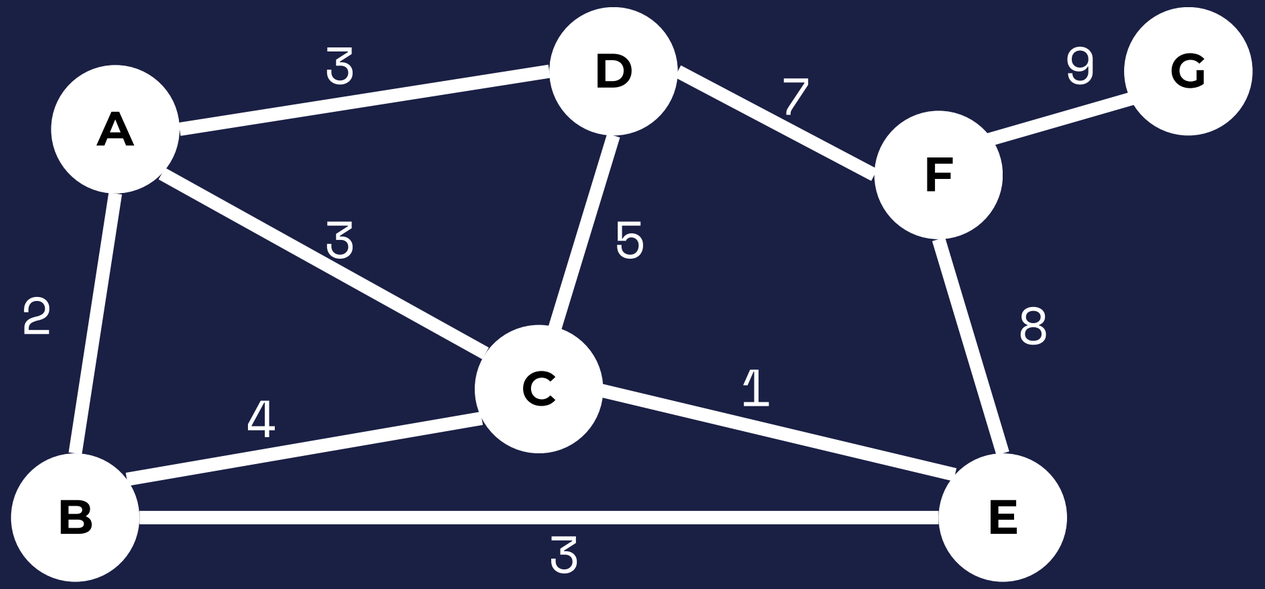
But lets make it look like a graph

```
graph = {
  'A': {'B': 2, 'C': 3, 'D': 3},
  'B': {'A': 2, 'C': 4, 'E': 3},
  'C': {'A': 3, 'B': 4, 'D': 5, 'E': 1},
  'D': {'A': 2, 'C': 5, 'F': 7},
  'E': {'B': 3, 'C': 1, 'F': 8},
  'F': {'D': 7, 'E': 8, 'G': 9},
  'G': {'F': 9}
}
```

But lets make it look like a graph

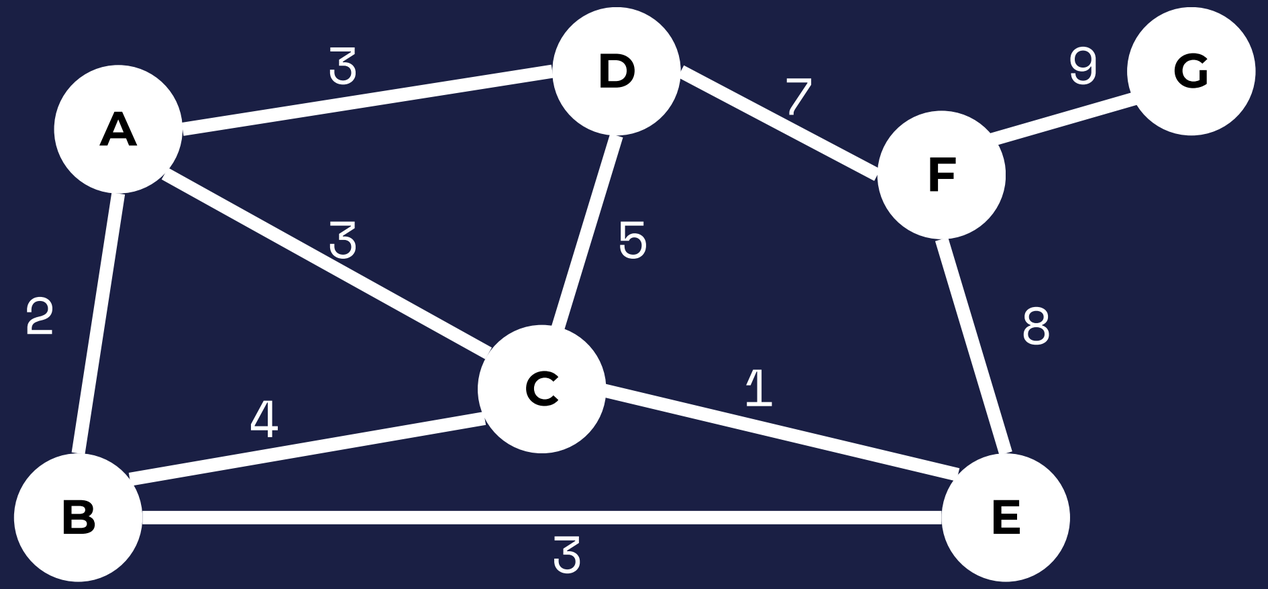


Kruskal’s algorithm always looks for the smallest weight not already apart of the spanning tree and adds it to the spanning tree.



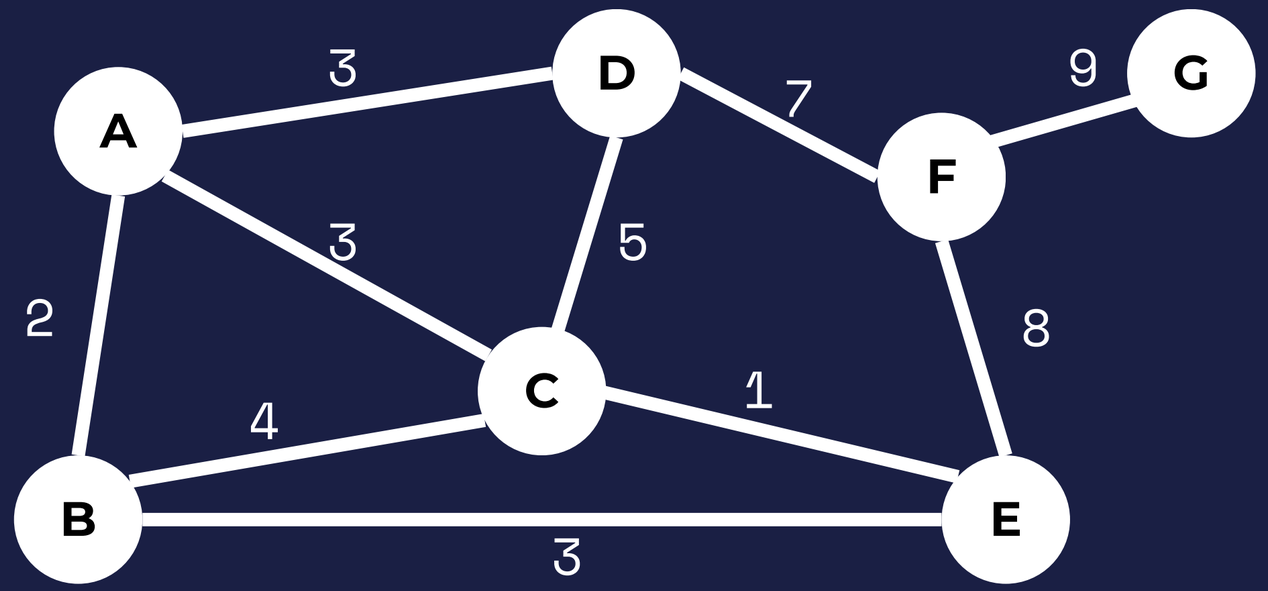
Kruskal’s algorithm always looks for the smallest weight not already apart of the spanning tree and adds it to the spanning tree.

Spanning Tree



The lowest number on this graph is C to E for 1.
Lets add C to E to the Spanning Tree

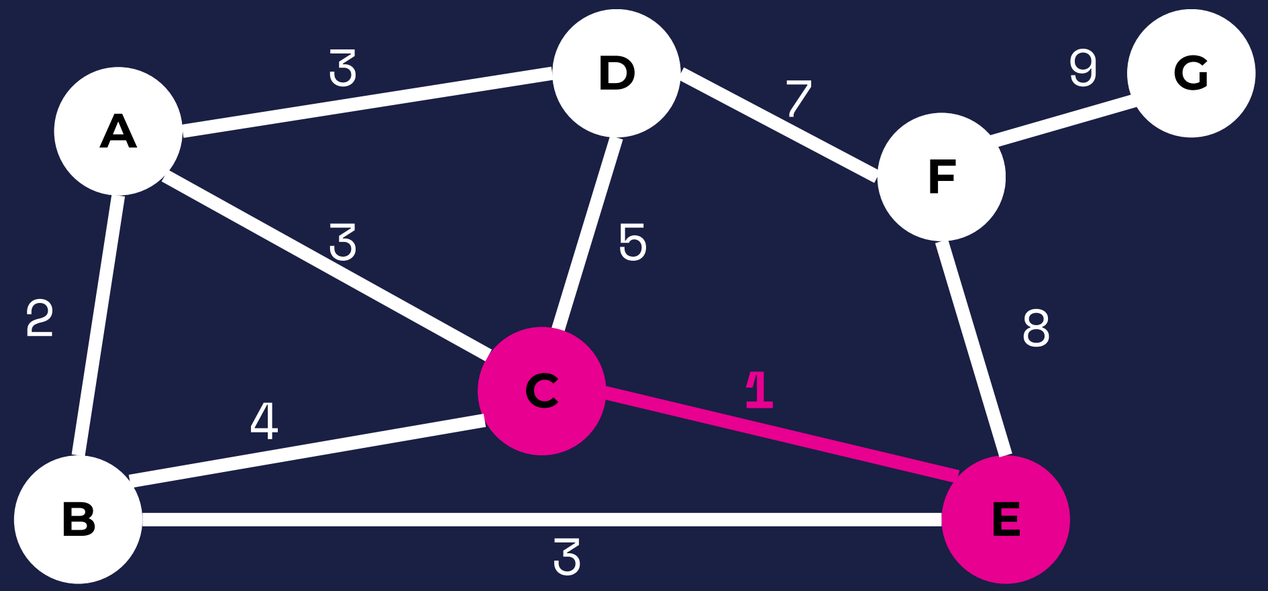
Spanning Tree



The lowest number on this graph is C to E for 1.
Lets add C to E to the Spanning Tree

Spanning Tree

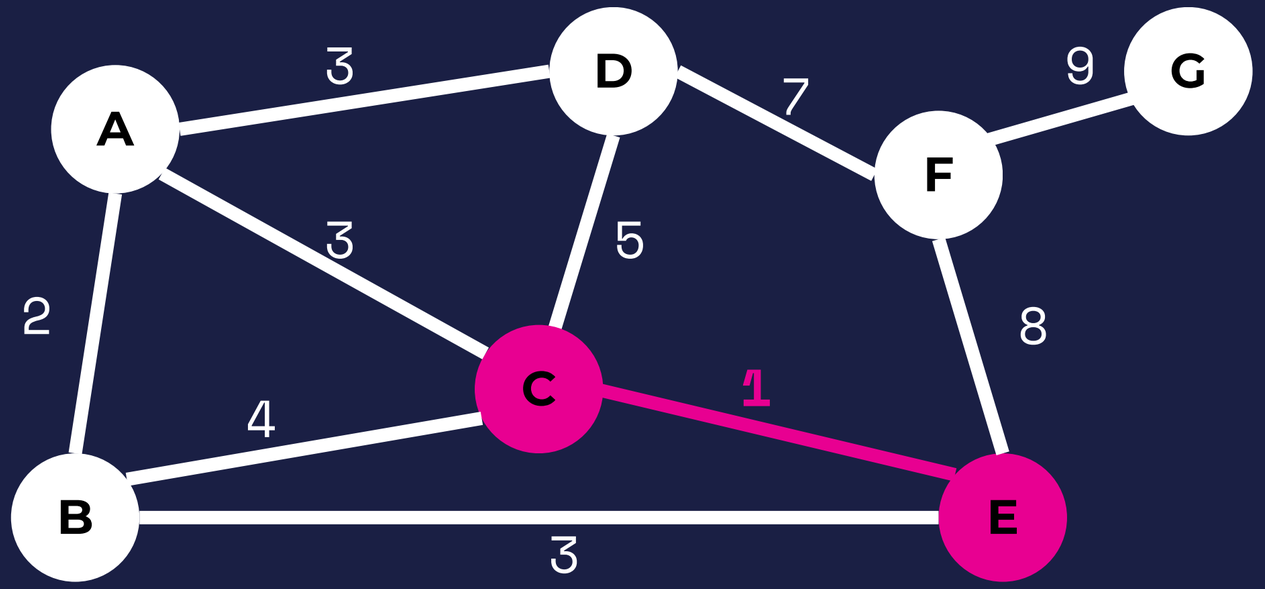
- (C, E, 1)



The next lowest number on this graph is A to B for 2. Lets add that one as well

Spanning Tree

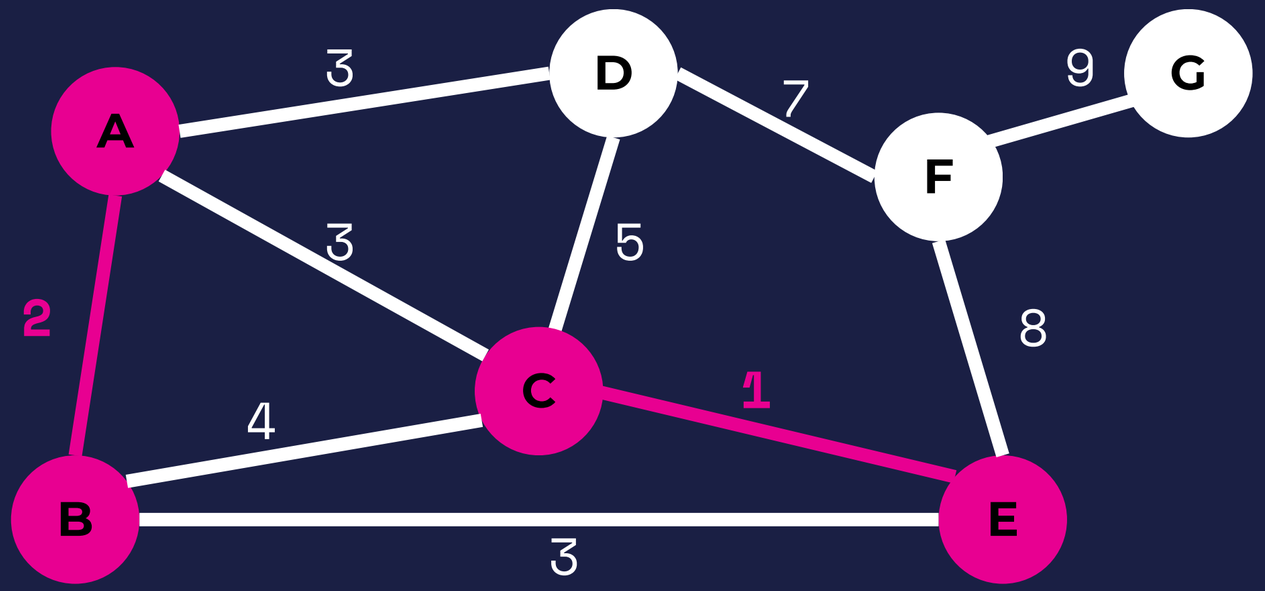
- (C, E, 1)



The next lowest number on this graph is A to B for 2. Lets add that one as well

Spanning Tree

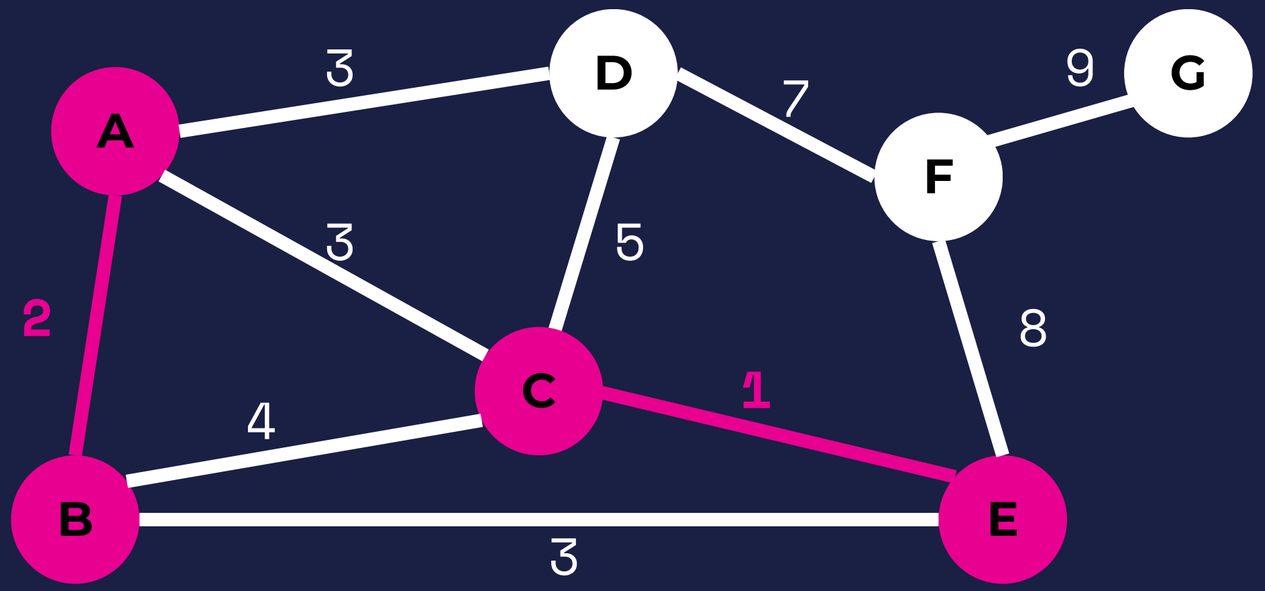
- (C, E, 1)
- (A, B, 2)



Because they aren't touching, they are currently two different spanning trees

Spanning Tree

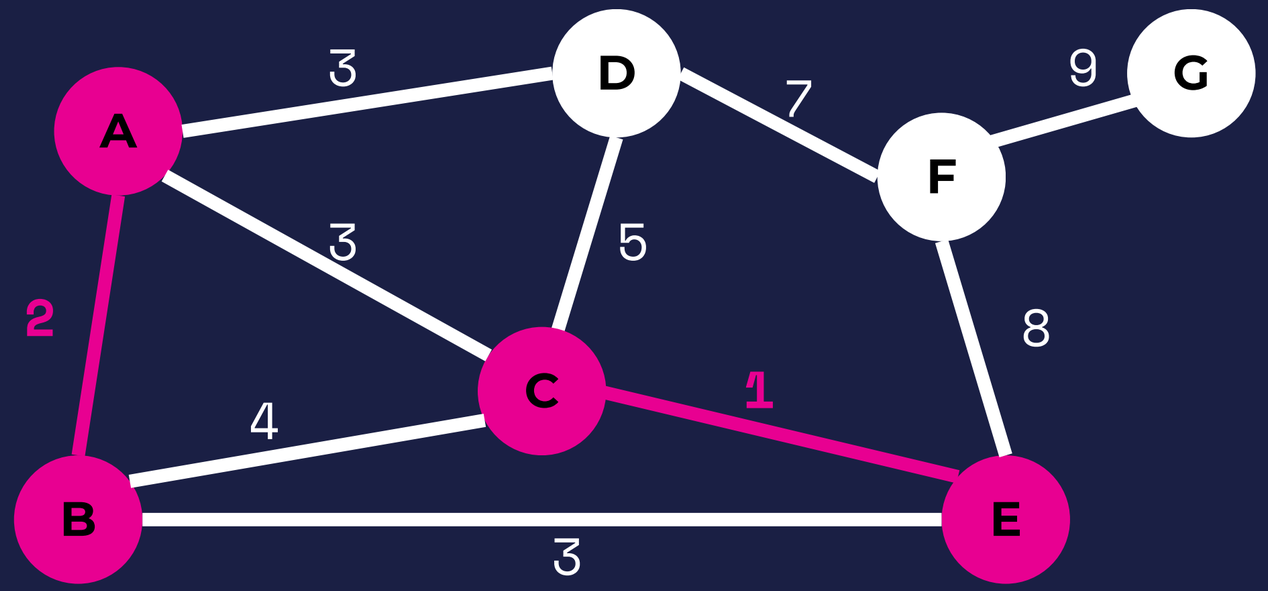
- (C, E, 1)
- (A, B, 2)



The next lowest number is 3. Lets start from the top, and add A to D to the Spanning Tree, connecting it to A to B because they connect via A

Spanning Tree

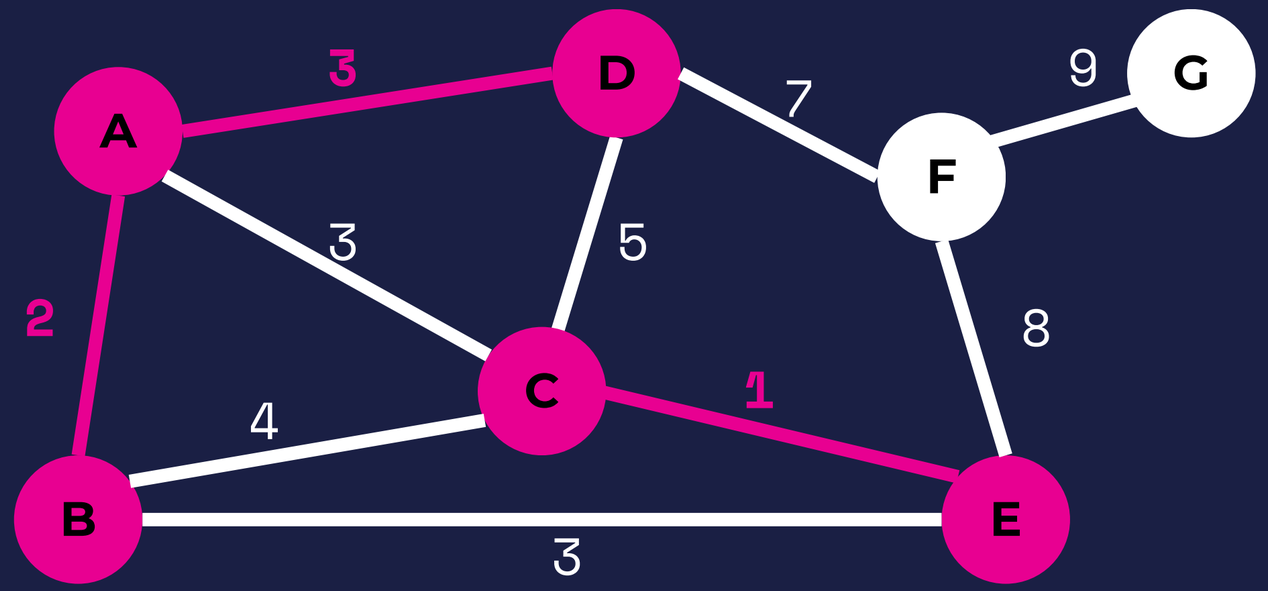
- (C, E, 1)
- (A, B, 2)



The next lowest number is 3. Lets start from the top, and add A to D to the Spanning Tree, connecting it to A to B because they connect via A

Spanning Tree

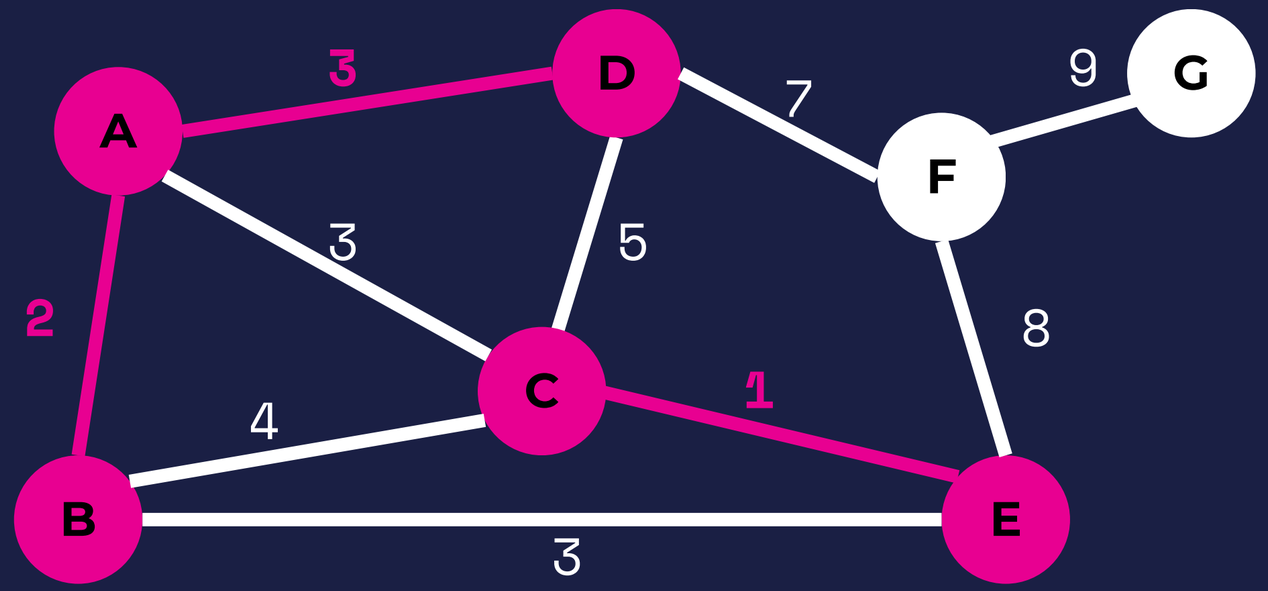
- (C, E, 1)
- (A, B, 2)
- (A, D, 3)



Next we will do A to C for three. Once we connect these two points, it connects the C to E spanning Tree to the A to B spanning tree making it one.

Spanning Tree

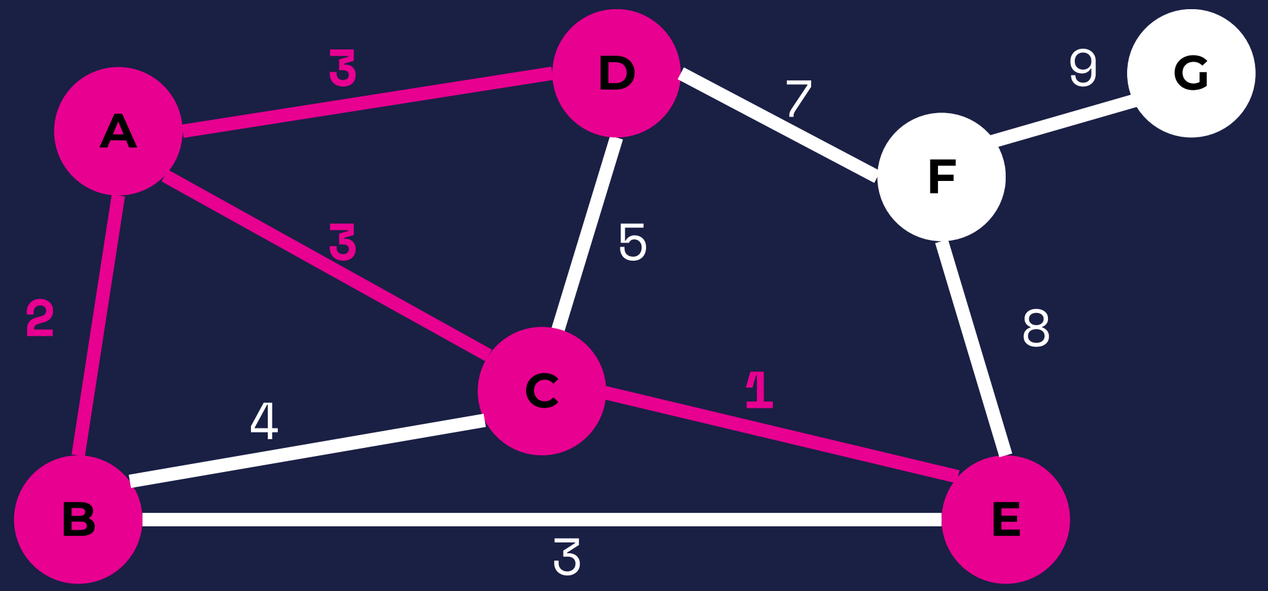
- (C, E, 1)
- (A, B, 2)
- (A, D, 3)



Next we will do A to C for three. Once we connect these two points, it connects the C to E spanning Tree to the A to B spanning tree making it one.

Spanning Tree

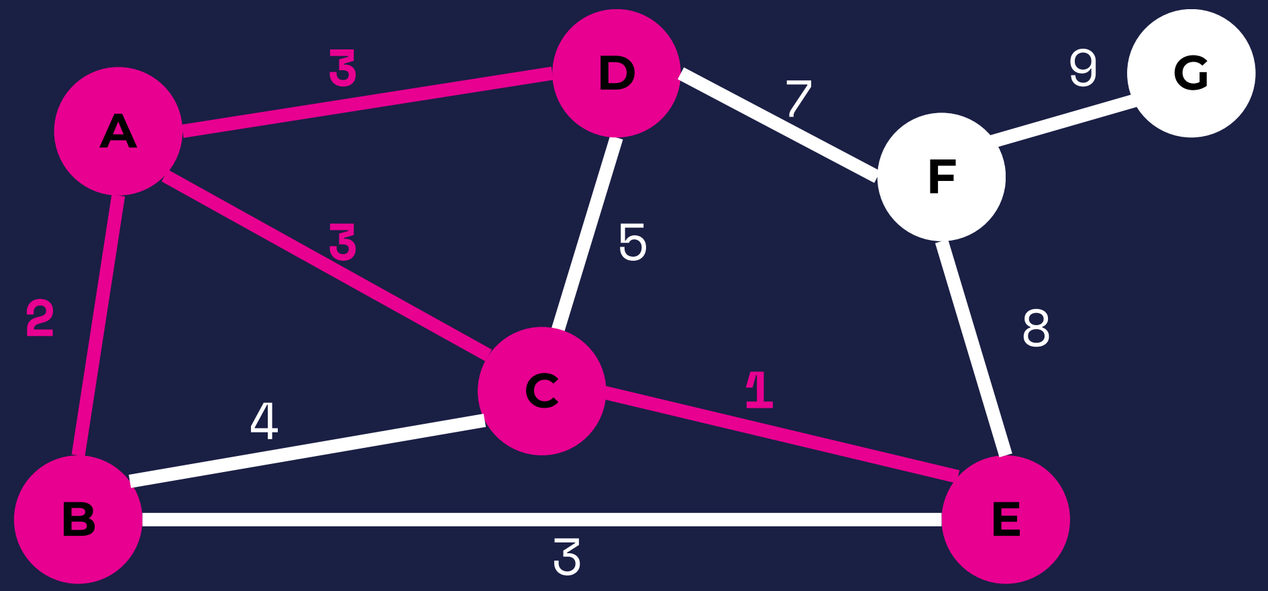
- (C, E, 1)
- (A, B, 2)
- (A, D, 3)
- (A, C, 3)



Next is B to E. However both points are a part of the same spanning tree so we skip this (otherwise it makes a loop and Kruskals says NO LOOPS!)

Spanning Tree

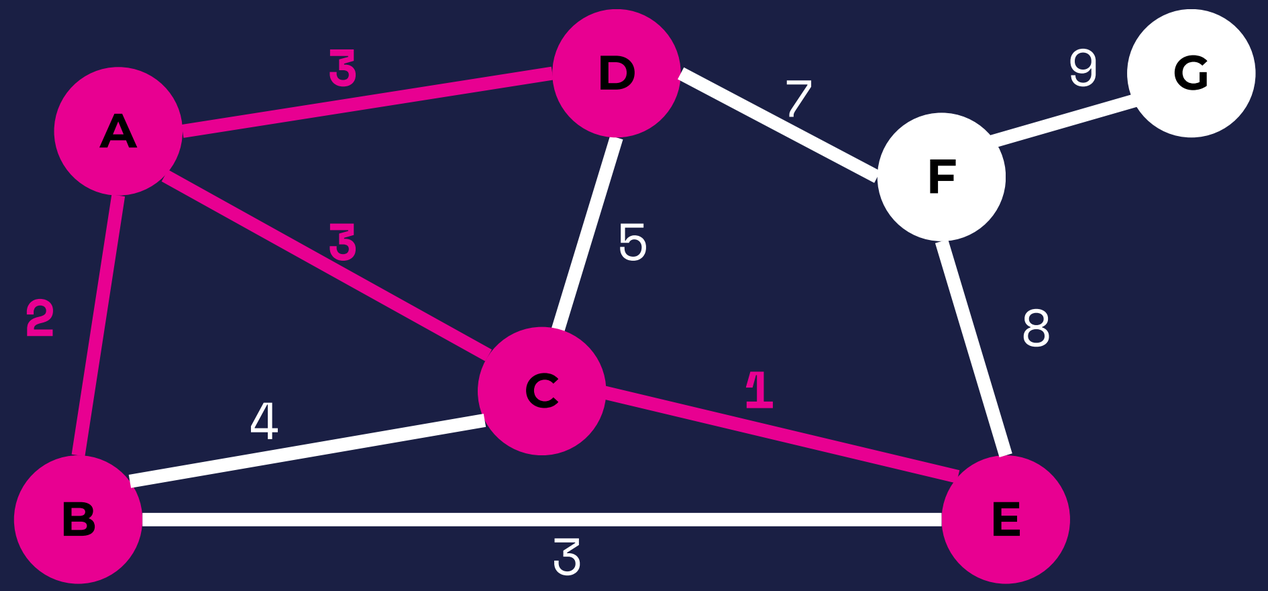
- (C, E, 1)
- (A, B, 2)
- (A, D, 3)
- (A, C, 3)



So next would be 4. B and C are apart of the small spanning tree so we will skip this, because NO LOOPS!

Spanning Tree

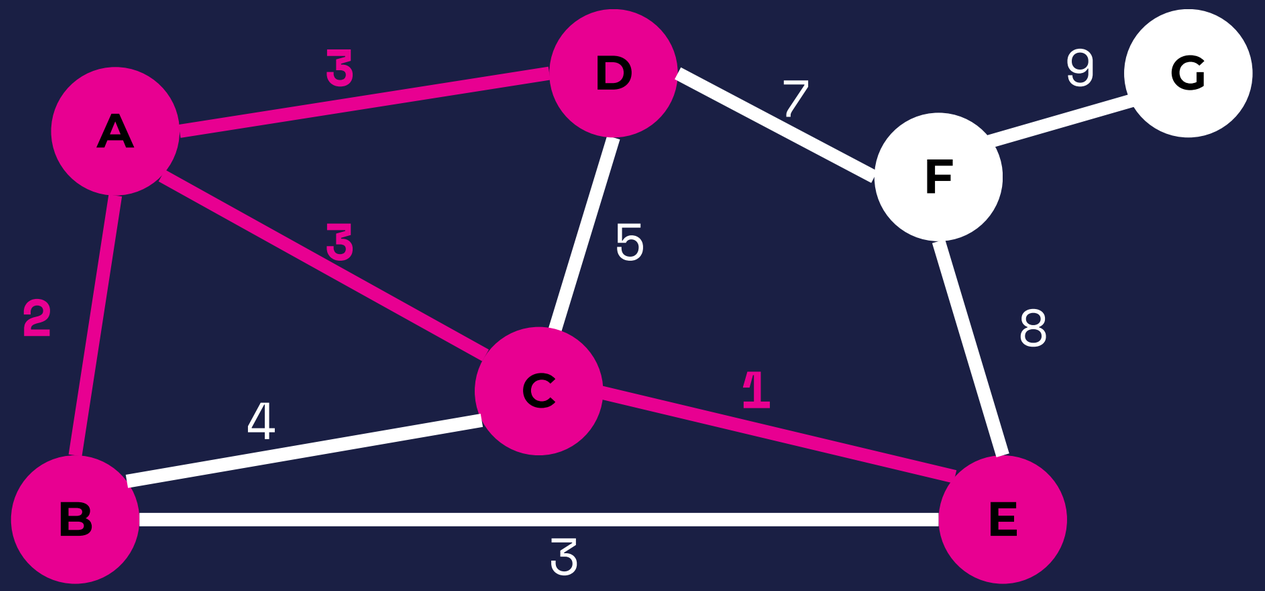
- (C, E, 1)
- (A, B, 2)
- (A, D, 3)
- (A, C, 3)



Next is 5, and again: NO LOOPS!

Spanning Tree

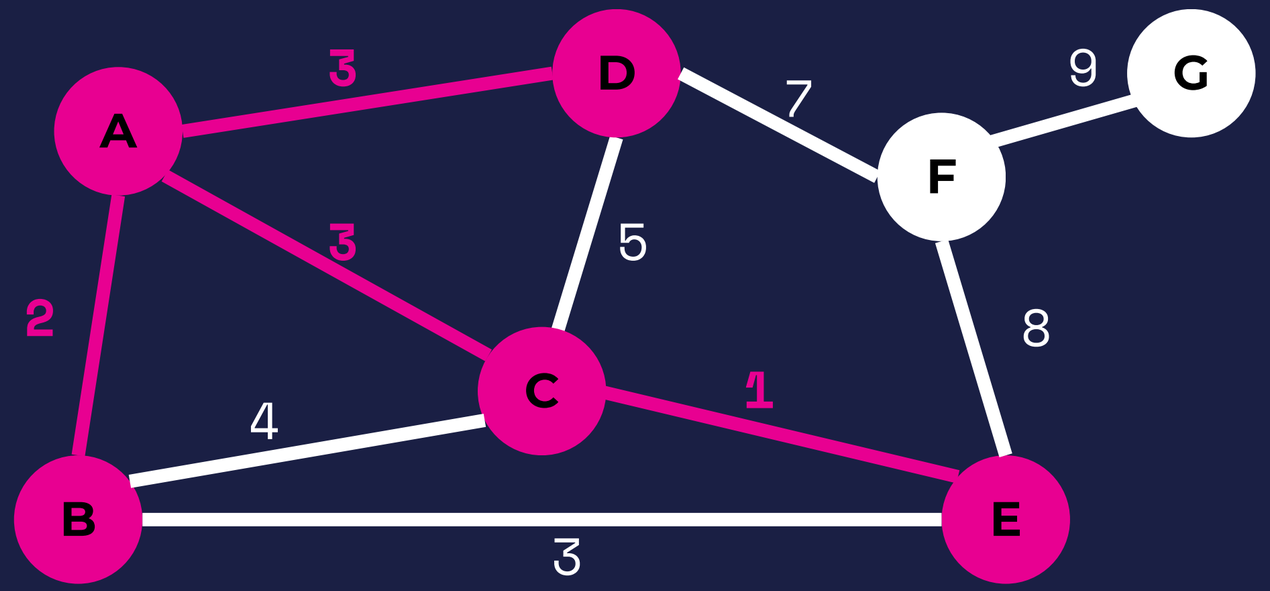
- (C, E, 1)
- (A, B, 2)
- (A, D, 3)
- (A, C, 3)



There is no 6, so we move on to 7. D to F is 7, we'll add those to the spanning tree

Spanning Tree

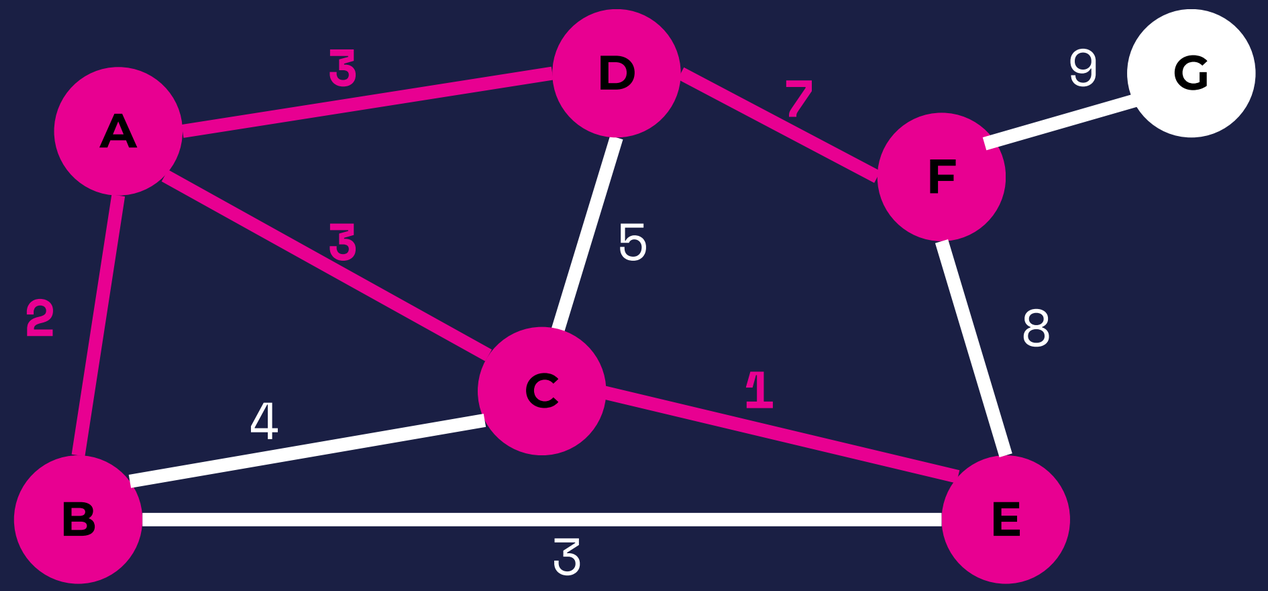
- (C, E, 1)
- (A, B, 2)
- (A, D, 3)
- (A, C, 3)



There is no 6, so we move on to 7. D to F is 7, we'll add those to the spanning tree

Spanning Tree

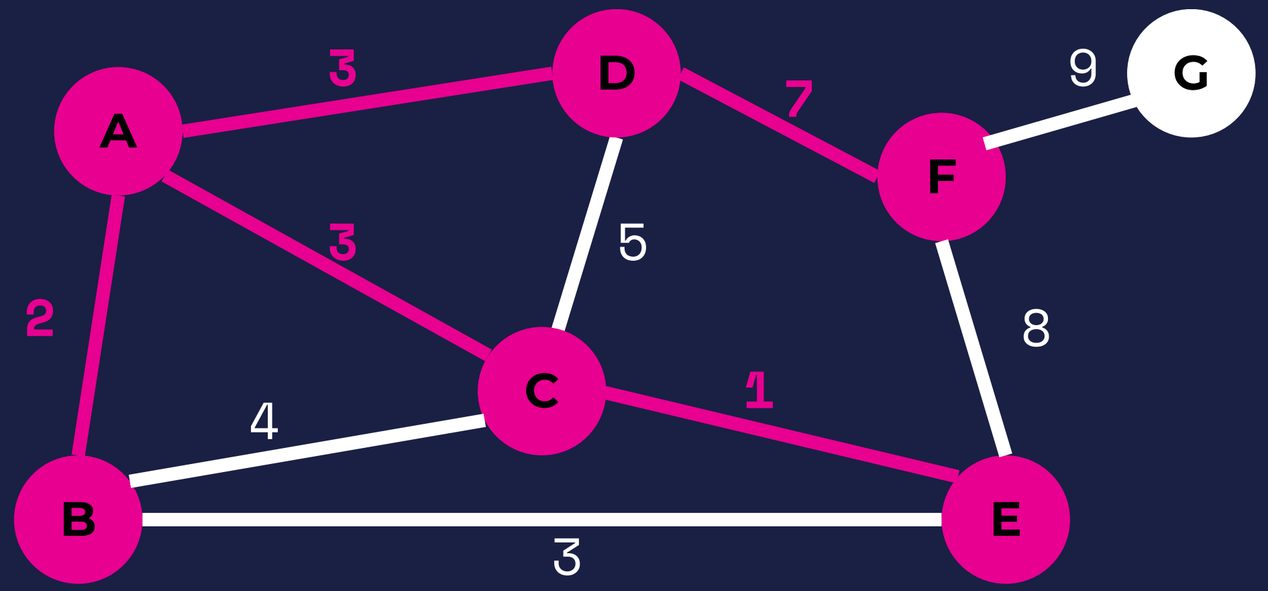
- (C, E, 1)
- (A, B, 2)
- (A, D, 3)
- (A, C, 3)
- (D, F, 7)



Next is 8. F and E are apart of the same spanning tree so we skip because NO LOOPS!

Spanning Tree

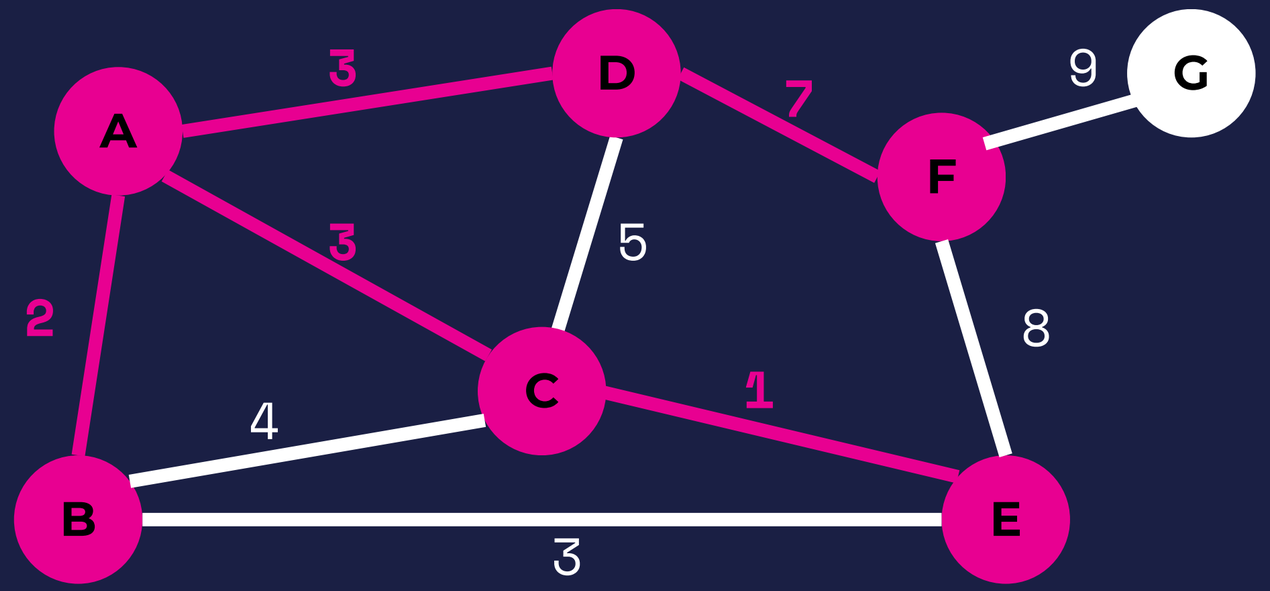
- (C, E, 1)
- (A, B, 2)
- (A, D, 3)
- (A, C, 3)
- (D, F, 7)



Last is 9. We connect F to G for 9.

Spanning Tree

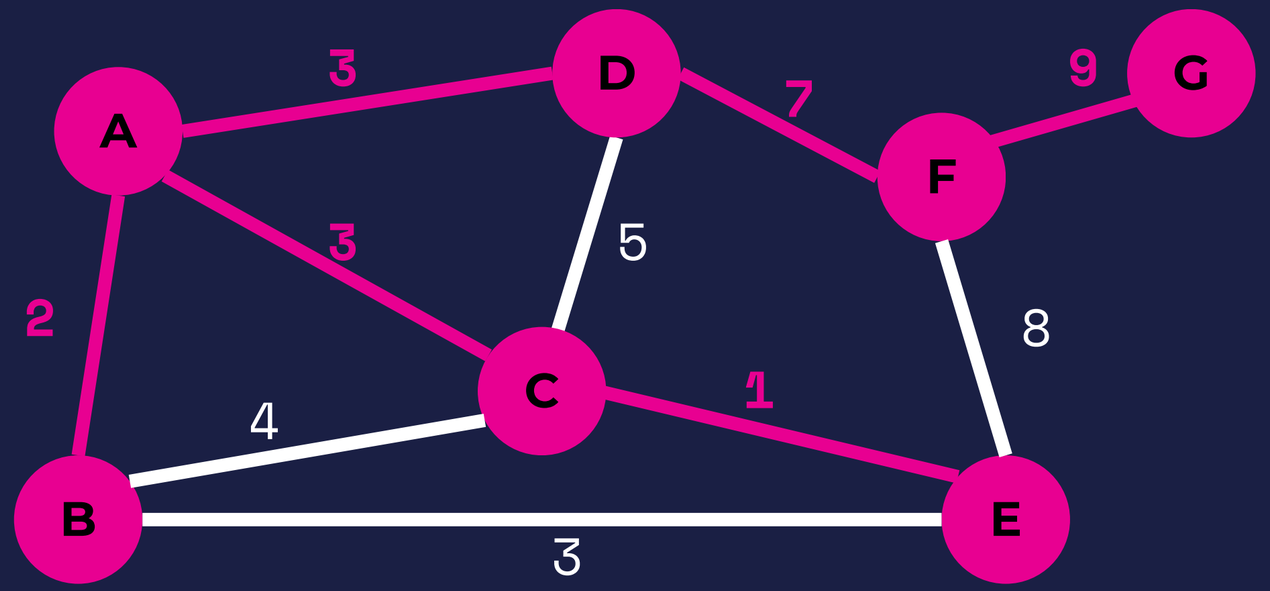
- (C, E, 1)
- (A, B, 2)
- (A, D, 3)
- (A, C, 3)
- (D, F, 7)



Last is 9. We connect F to G for 9.

Spanning Tree

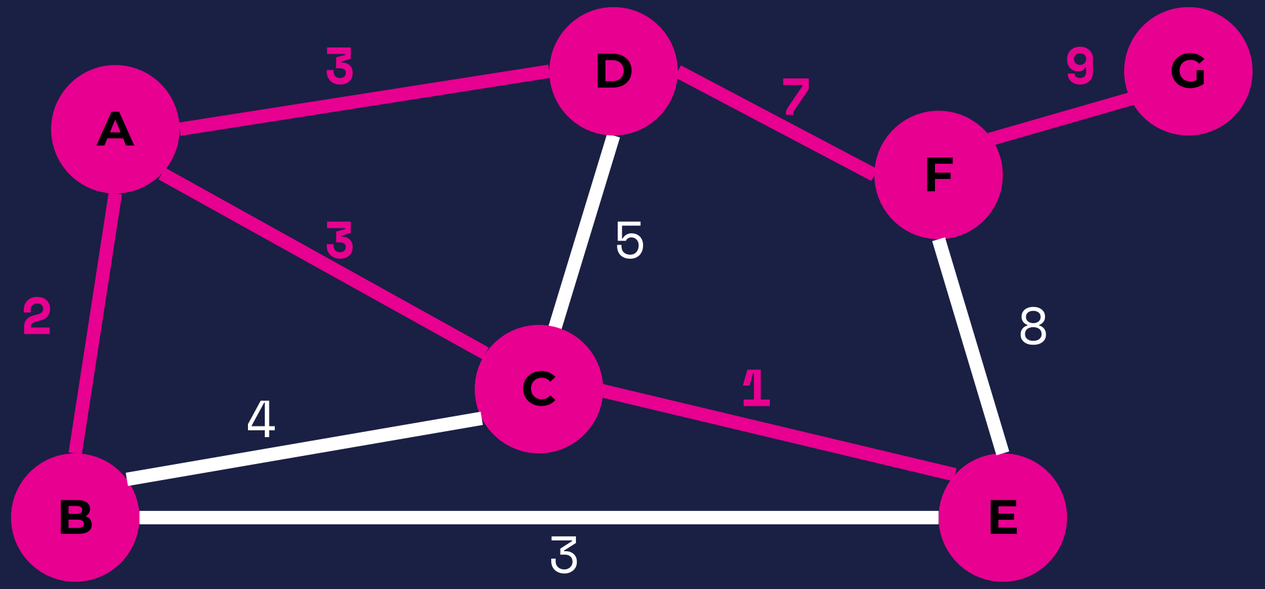
- (C, E, 1)
- (A, B, 2)
- (A, D, 3)
- (A, C, 3)
- (D, F, 7)
- (F, G, 9)



So there is the Minimum Spanning Tree. We can remove all the unused paths and it will show all points connected with no loops.

Spanning Tree

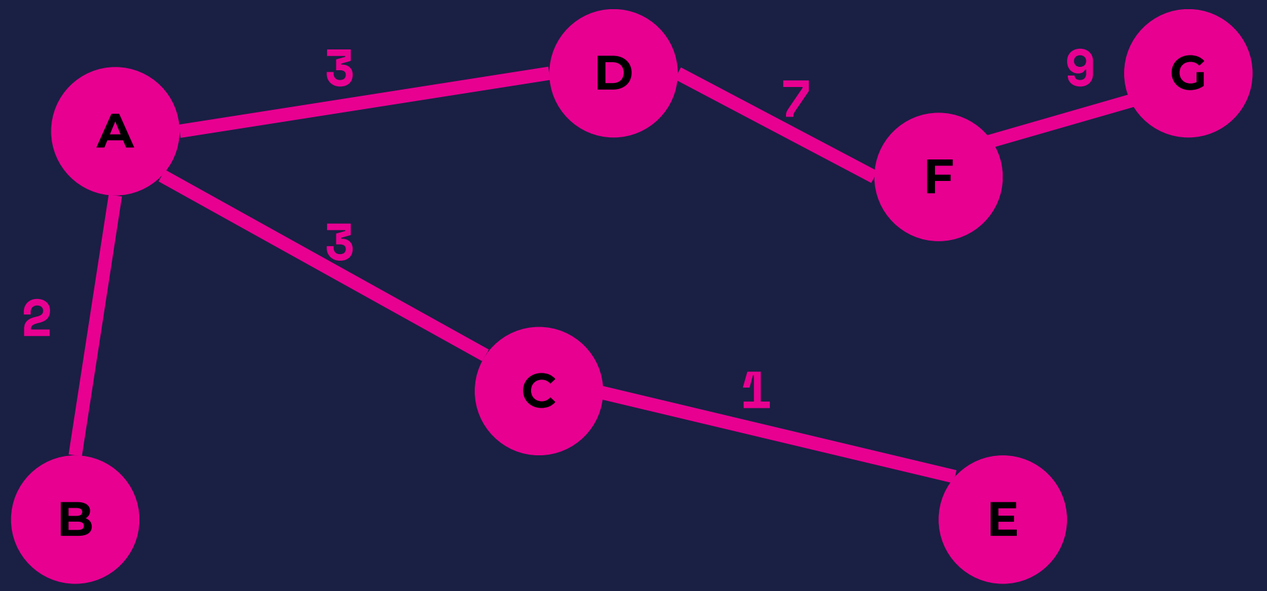
- (C, E, 1)
- (A, B, 2)
- (A, D, 3)
- (A, C, 3)
- (D, F, 7)
- (F, G, 9)



Tada!

Spanning Tree

- (C, E, 1)
- (A, B, 2)
- (A, D, 3)
- (A, C, 3)
- (D, F, 7)
- (F, G, 9)



THE PSEUDOCODE

```
function KRUSKALS(graph):
    edges = SORT graph.edges
    parent = x for x in graph.vertices
    rank = 0 for x in graph.vertices

    function FIND(x):
        if parent[x] != x:
            parent[x] = find(parent[x])
        return parent[x]

    function UNION(x, y):
        rootX = FIND(x)
        rootY = FIND(y)
        if rootX != rootY:
            if rank[rootX] > rank[rootY]: parent[rootY] = rootX
            elif rank[rootX] < rank[rootY]: parent[rootX] = rootY
            else:
                parent[rootY] = rootX
                rank[rootX] += 1
```

THE PSEUDOCODE

```
for u, v, weight in edges:
    if FIND(u) != FIND(v):
        UNION(u, v)
        spanningTree.add(u, v, weight)

    return spanningTree
```

EXAMPLES REPLIT AND GITHUB! PLEASE GO TO:

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

HTTPS://GITHUB.COM/RIKKITOMIKOEHRHART/GRABABYTE

UP NEXT

May 14 - Prim's Algorithm

And then we are done!!

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

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