

Classical Algorithms

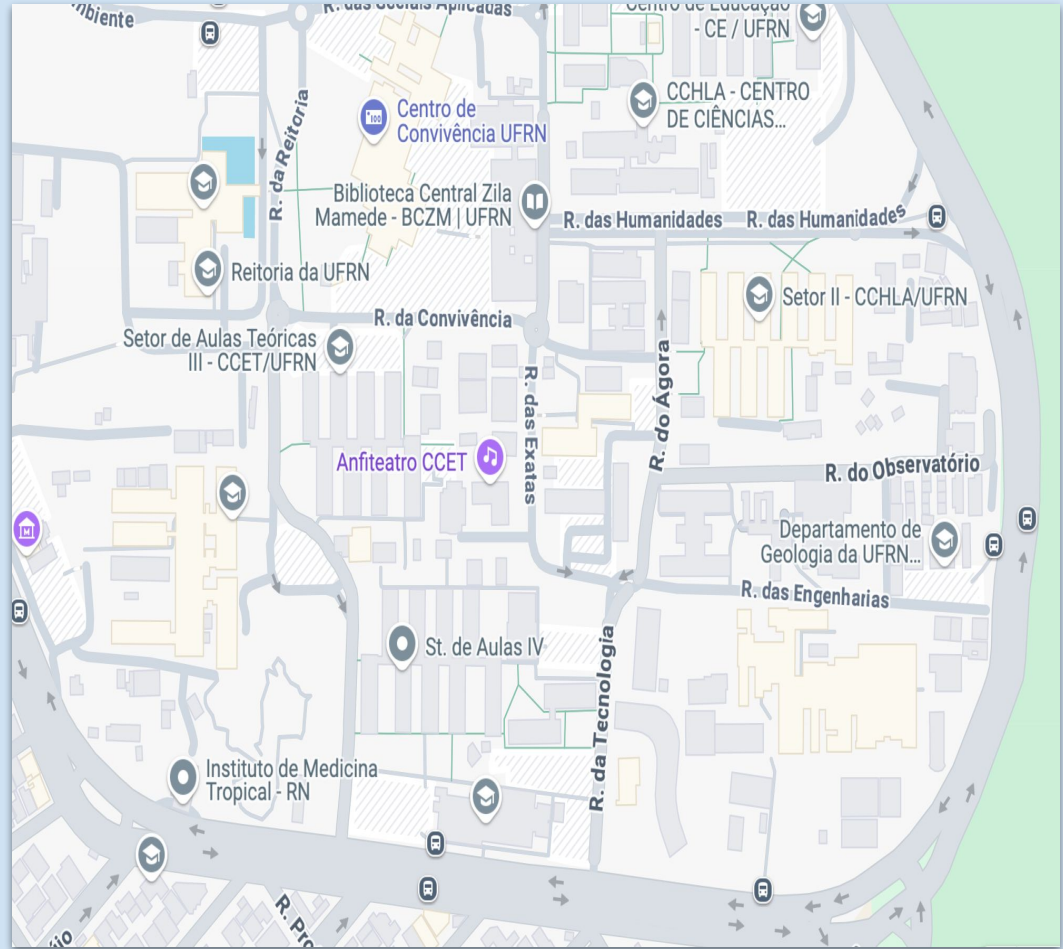
Kruskal
Minimum Spanning Tree (MST)

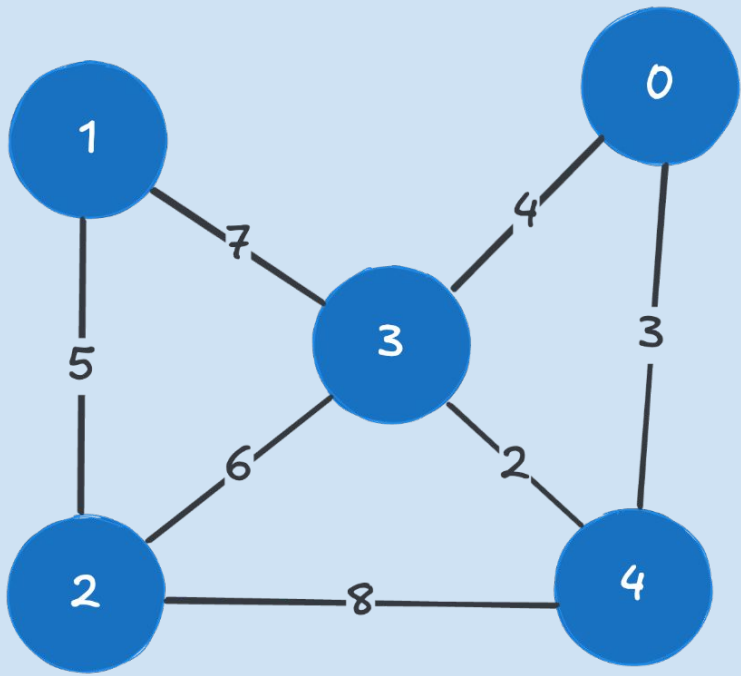
ivanovitch.silva@ufrn.br
@ivanovitchm



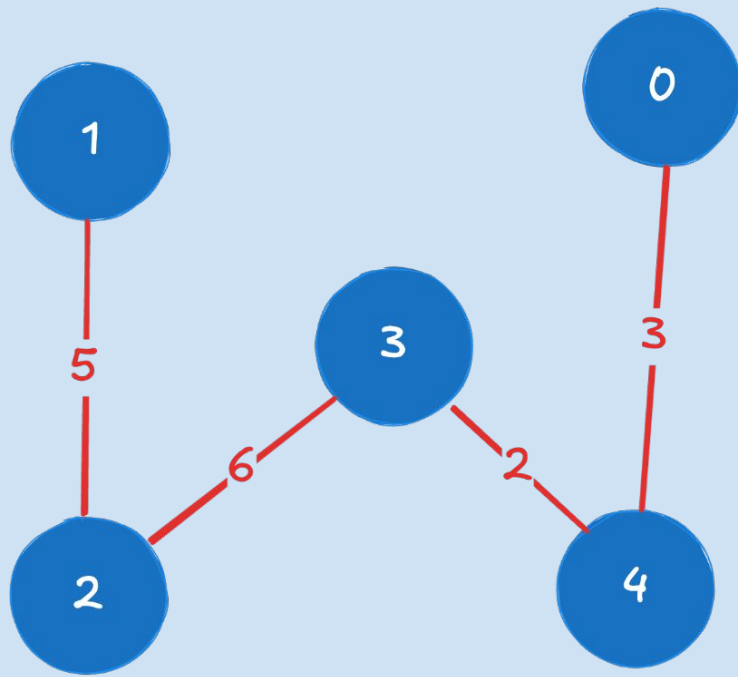
Imagine that you are installing internet cables at **UFRN**. You want to ensure that:

- All buildings are connected.
- You use the shortest total length of cables.
- There are no unnecessary cables going back and forth.

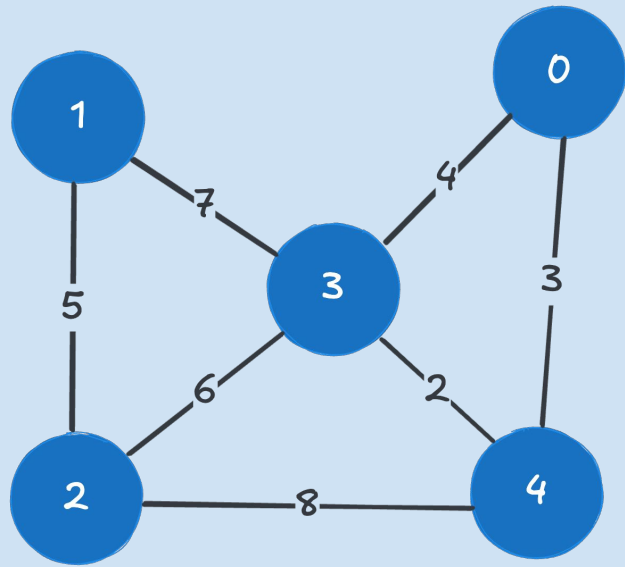




graph - connected, undirected,
weighted (non-negative)

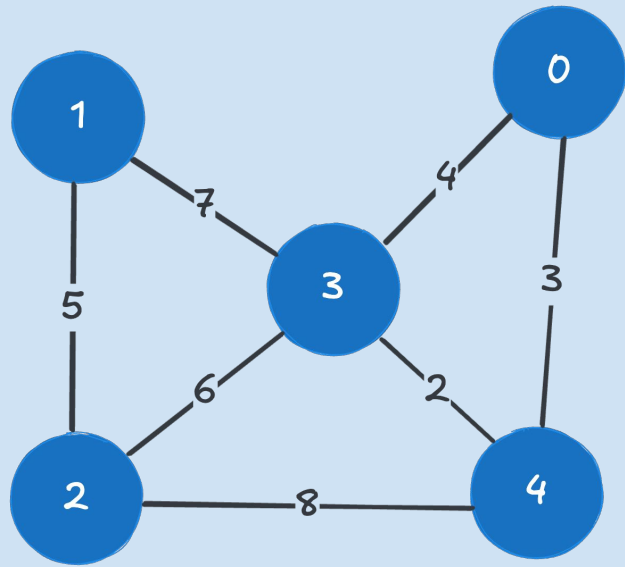


— Minimum Spanning Tree (MST)



Graph Representation

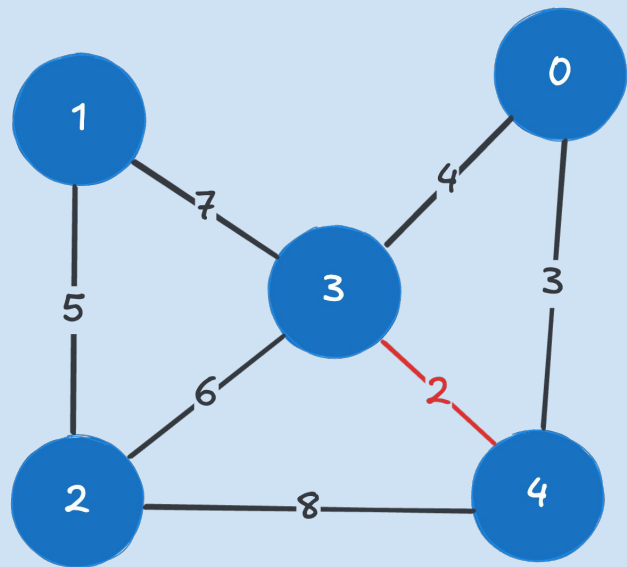
```
edges = [  
    [[4, 3], [3, 4]],      # Vertex 0  
    [[2, 5], [3, 7]],      # Vertex 1  
    [[1, 5], [3, 6], [4, 8]], # Vertex 2  
    [[0, 4], [1, 7], [2, 6], [4, 2]], # Vertex 3  
    [[3, 2], [0, 3], [2, 8]], # Vertex 4  
]
```

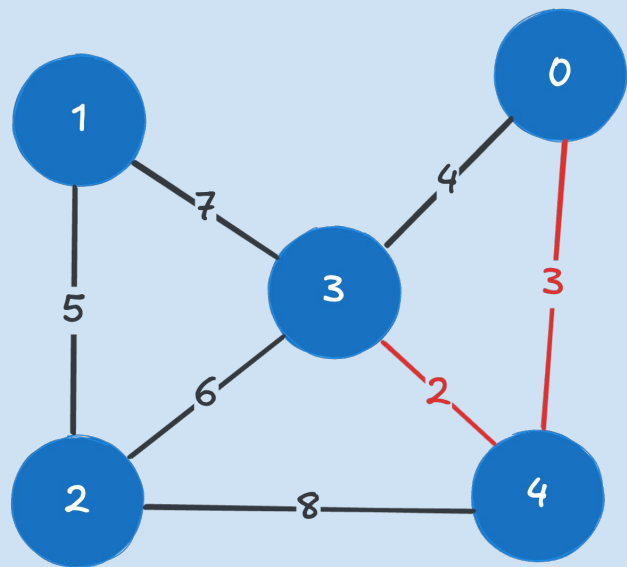
```
edgeList = []  
for sourceIndex, vertex in enumerate(edges):  
    for edge in vertex:  
        if edge[0] > sourceIndex: # Evita duplicatas  
            edgeList.append([sourceIndex, edge[0], edge[1]])  
sortedEdges = sorted(edgeList, key=lambda edge: edge[2])
```

```
[  
    [3, 4, 2], # Weight 2  
    [0, 4, 3], # Weight 3  
    [0, 3, 4], # Weight 4  
    [1, 2, 5], # Weight 5  
    [2, 3, 6], # Weight 6  
    [1, 3, 7], # Weight 7  
    [2, 4, 8], # Weight 8  
]
```

Sort the Edges by Weight



```
[  
  [3, 4, 2], # Weight 2  
  [0, 4, 3], # Weight 3  
  [0, 3, 4], # Weight 4  
  [1, 2, 5], # Weight 5  
  [2, 3, 6], # Weight 6  
  [1, 3, 7], # Weight 7  
  [2, 4, 8], # Weight 8  
]
```



[

[3, 4, 2], # Weight 2

[0, 4, 3], # Weight 3

[0, 3, 4], # Weight 4

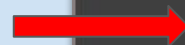
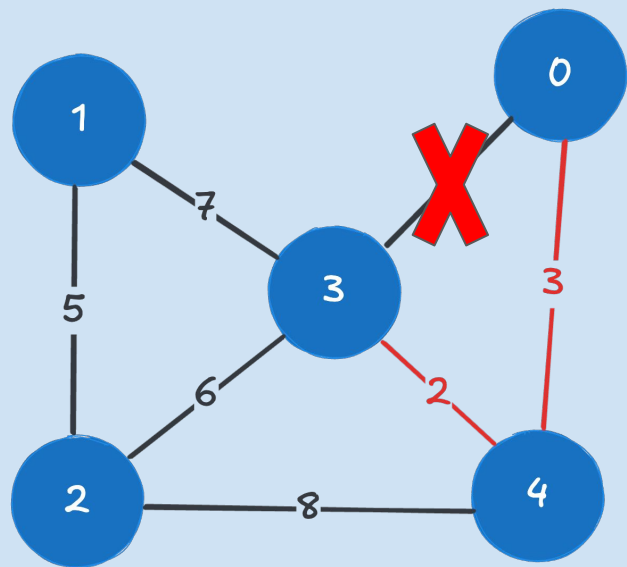
[1, 2, 5], # Weight 5

[2, 3, 6], # Weight 6

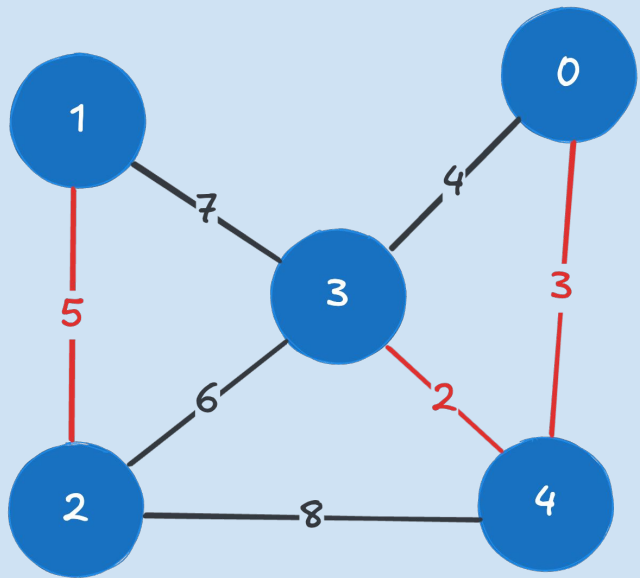
[1, 3, 7], # Weight 7

[2, 4, 8], # Weight 8

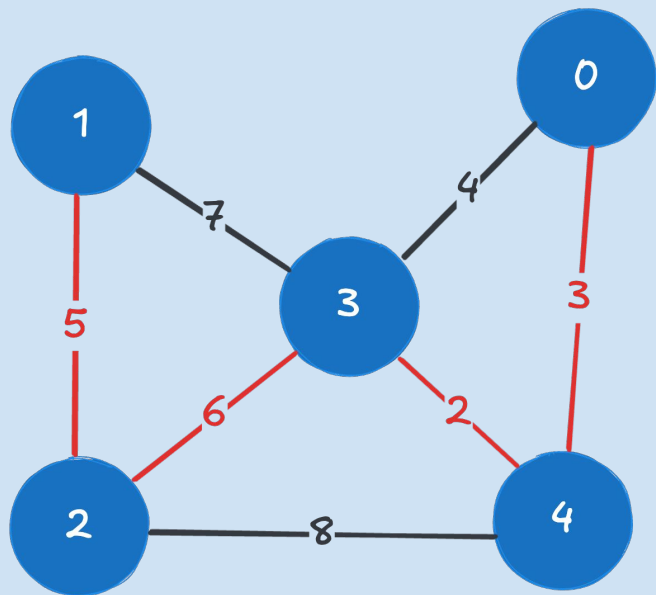
]



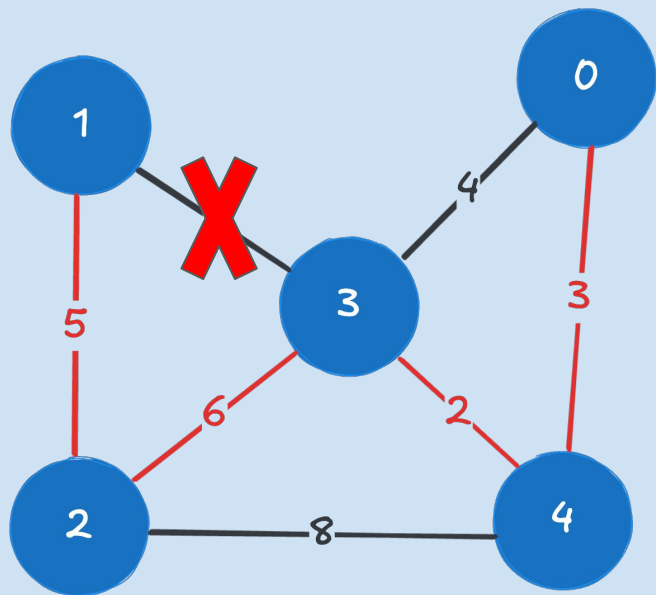
```
[  
    [3, 4, 2], # Weight 2  
    [0, 4, 3], # Weight 3  
    [0, 3, 4], # Weight 4  
    [1, 2, 5], # Weight 5  
    [2, 3, 6], # Weight 6  
    [1, 3, 7], # Weight 7  
    [2, 4, 8], # Weight 8  
]
```

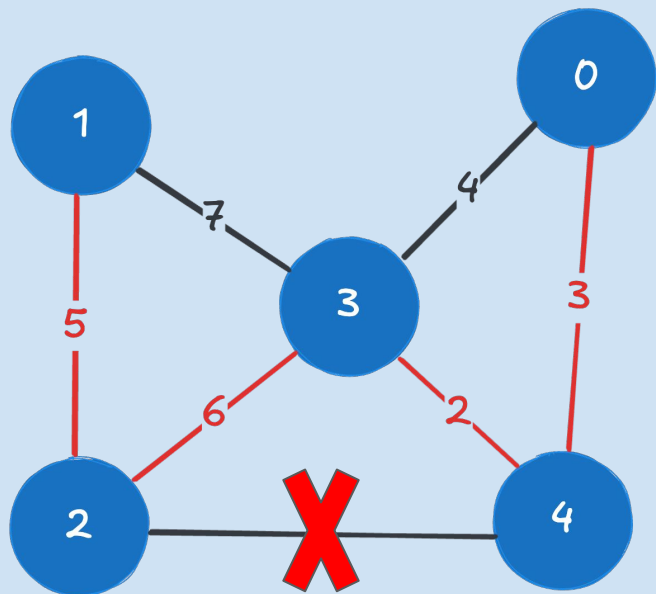
[
[3, 4, 2], # Weight 2
[0, 4, 3], # Weight 3
[0, 3, 4], # Weight 4
[1, 2, 5], # Weight 5
[2, 3, 6], # Weight 6
[1, 3, 7], # Weight 7
[2, 4, 8], # Weight 8
]



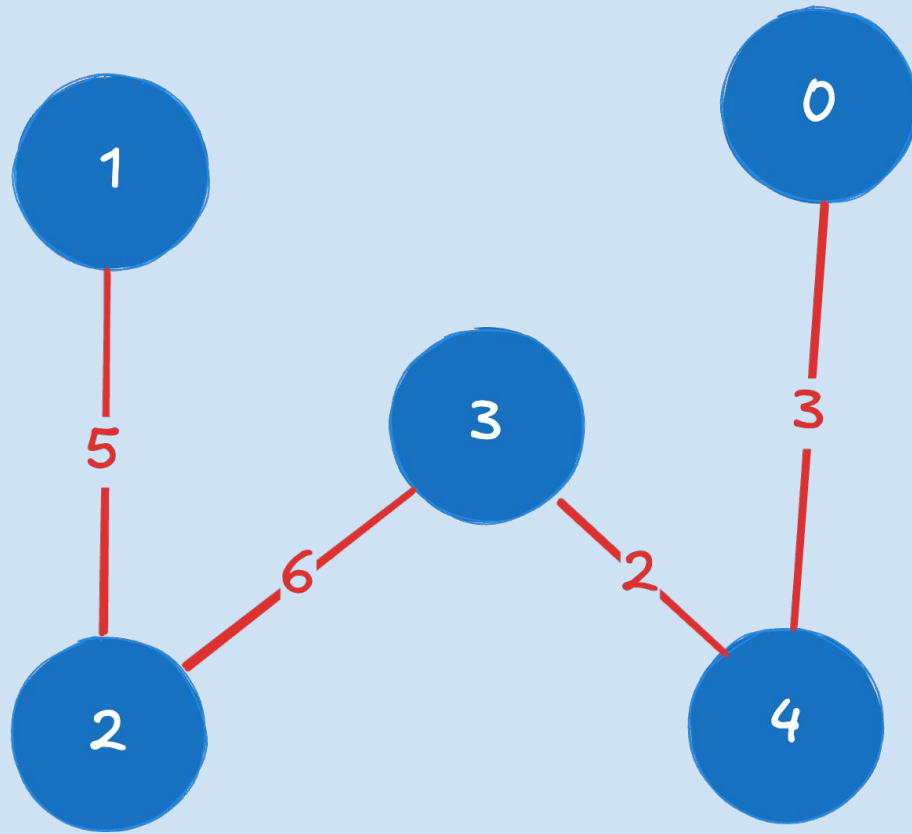
```
[  
    [3, 4, 2], # Weight 2  
    [0, 4, 3], # Weight 3  
    [0, 3, 4], # Weight 4  
    [1, 2, 5], # Weight 5  
    [2, 3, 6], # Weight 6  
    [1, 3, 7], # Weight 7  
    [2, 4, 8], # Weight 8  
]
```



```
[  
    [3, 4, 2], # Weight 2  
    [0, 4, 3], # Weight 3  
    [0, 3, 4], # Weight 4  
    [1, 2, 5], # Weight 5  
    [2, 3, 6], # Weight 6  
    [1, 3, 7], # Weight 7  
    [2, 4, 8], # Weight 8  
]
```



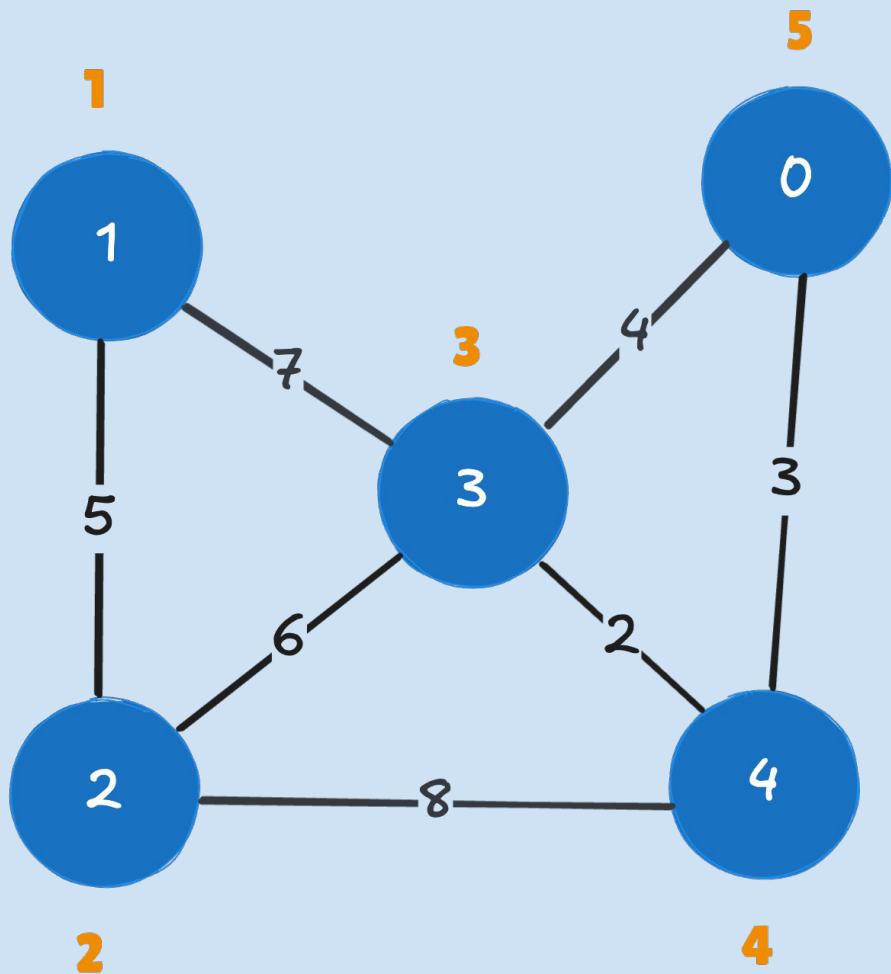
```
[  
    [3, 4, 2], # Weight 2  
    [0, 4, 3], # Weight 3  
    [0, 3, 4], # Weight 4  
    [1, 2, 5], # Weight 5  
    [2, 3, 6], # Weight 6  
    [1, 3, 7], # Weight 7  
    [2, 4, 8], # Weight 8  
]
```



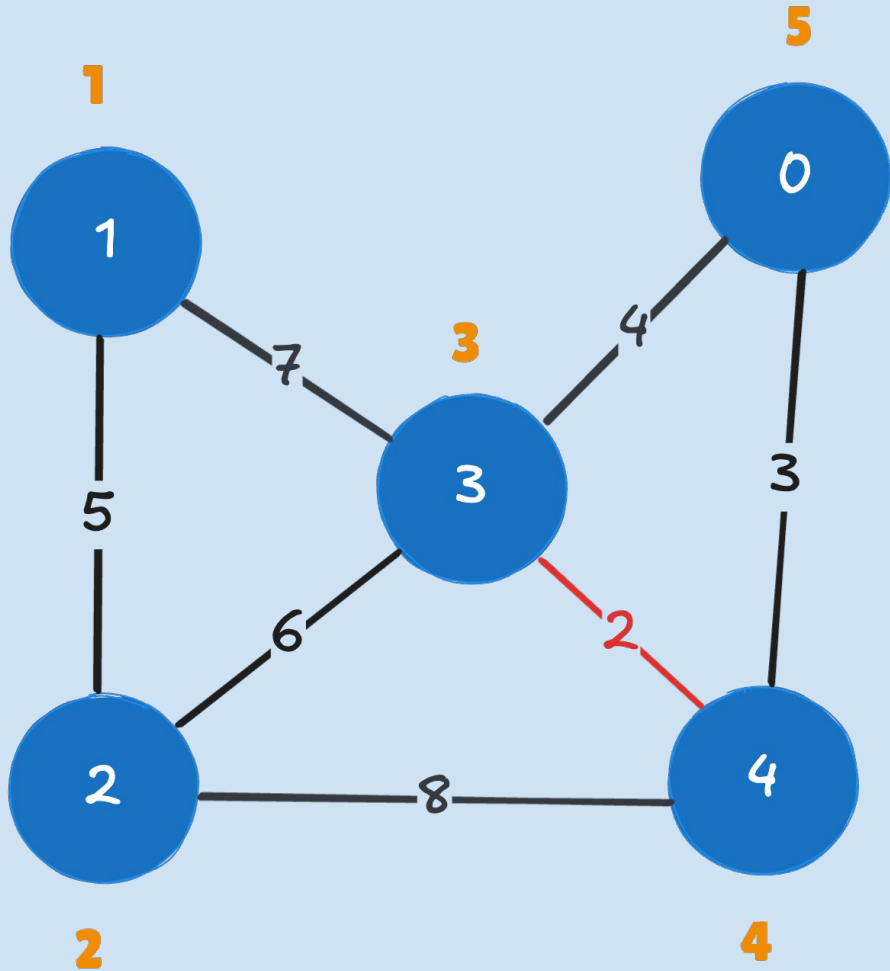
— Minimum Spanning Tree (MST)

How can we identify a cycle?

Ans: Union-Find (Disjoint Set Union - DSU)
data structure

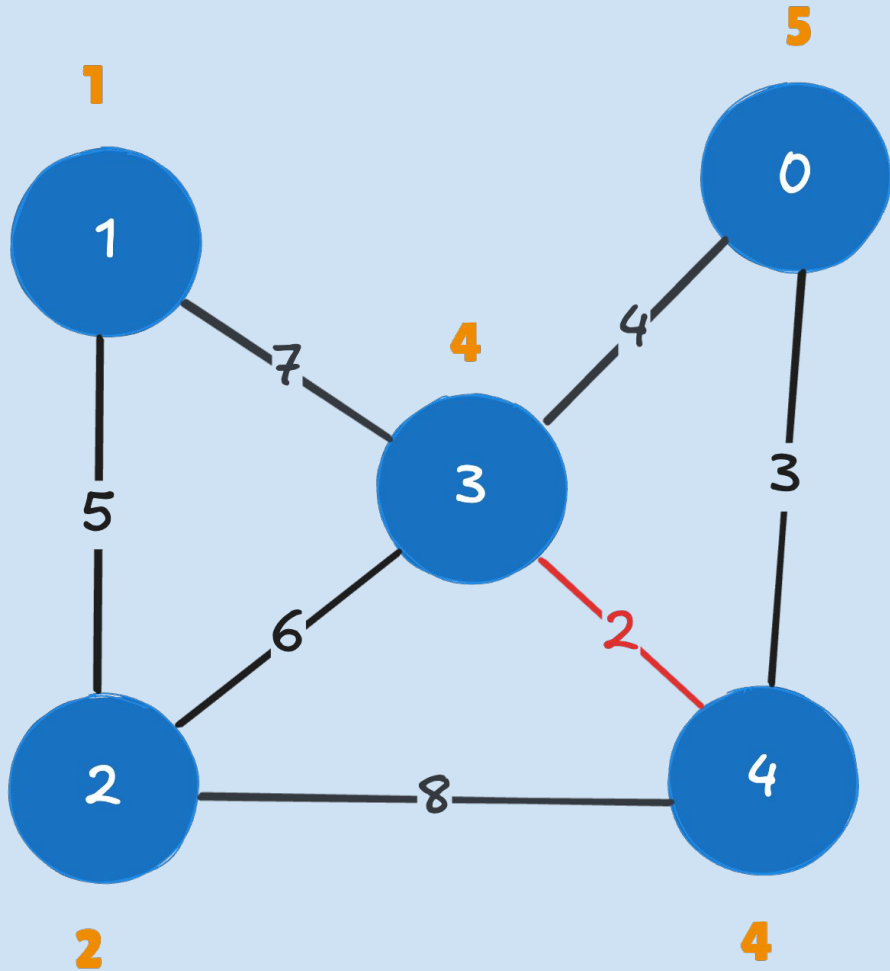


```
[  
    [3, 4, 2], # Weight 2  
    [0, 4, 3], # Weight 3  
    [0, 3, 4], # Weight 4  
    [1, 2, 5], # Weight 5  
    [2, 3, 6], # Weight 6  
    [1, 3, 7], # Weight 7  
    [2, 4, 8], # Weight 8  
]
```



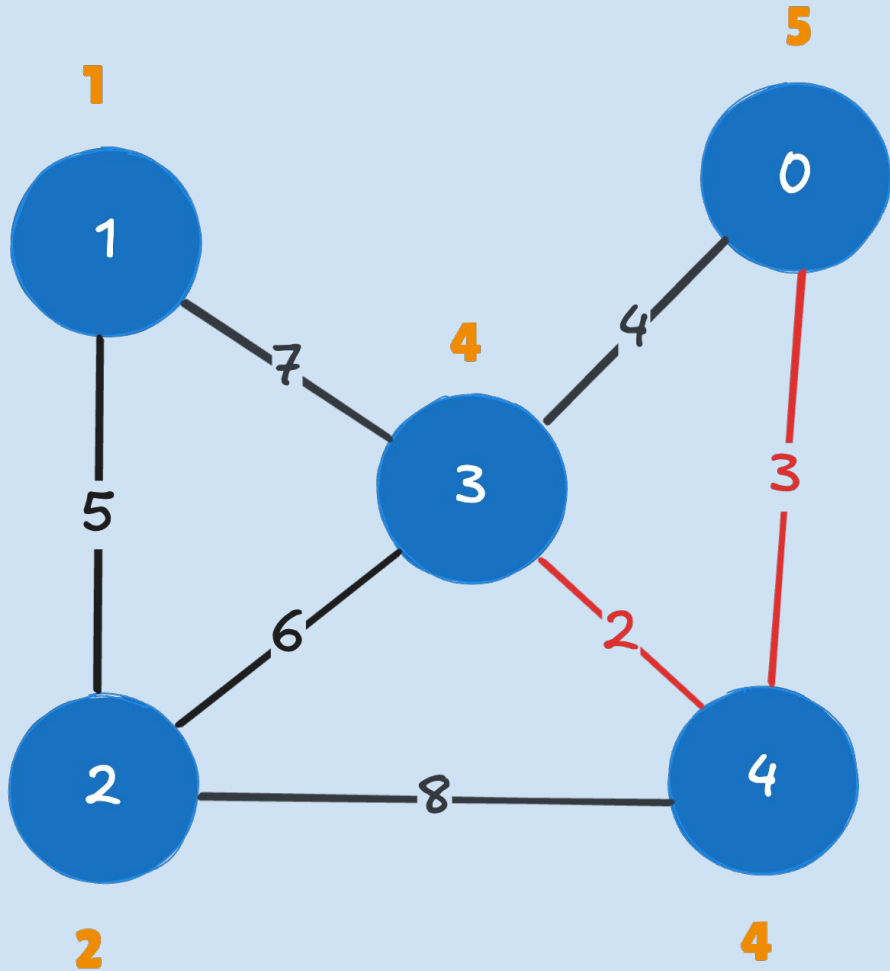
Edge [3,4,2] connects vertices 3 and 4, which belongs to different set.

```
[  
  [3, 4, 2], # Weight 2  
  [0, 4, 3], # Weight 3  
  [0, 3, 4], # Weight 4  
  [1, 2, 5], # Weight 5  
  [2, 3, 6], # Weight 6  
  [1, 3, 7], # Weight 7  
  [2, 4, 8], # Weight 8  
]
```



An union between the sets
3 and 4 is done.

```
[  
    [3, 4, 2], # Weight 2  
    [0, 4, 3], # Weight 3  
    [0, 3, 4], # Weight 4  
    [1, 2, 5], # Weight 5  
    [2, 3, 6], # Weight 6  
    [1, 3, 7], # Weight 7  
    [2, 4, 8], # Weight 8  
]
```

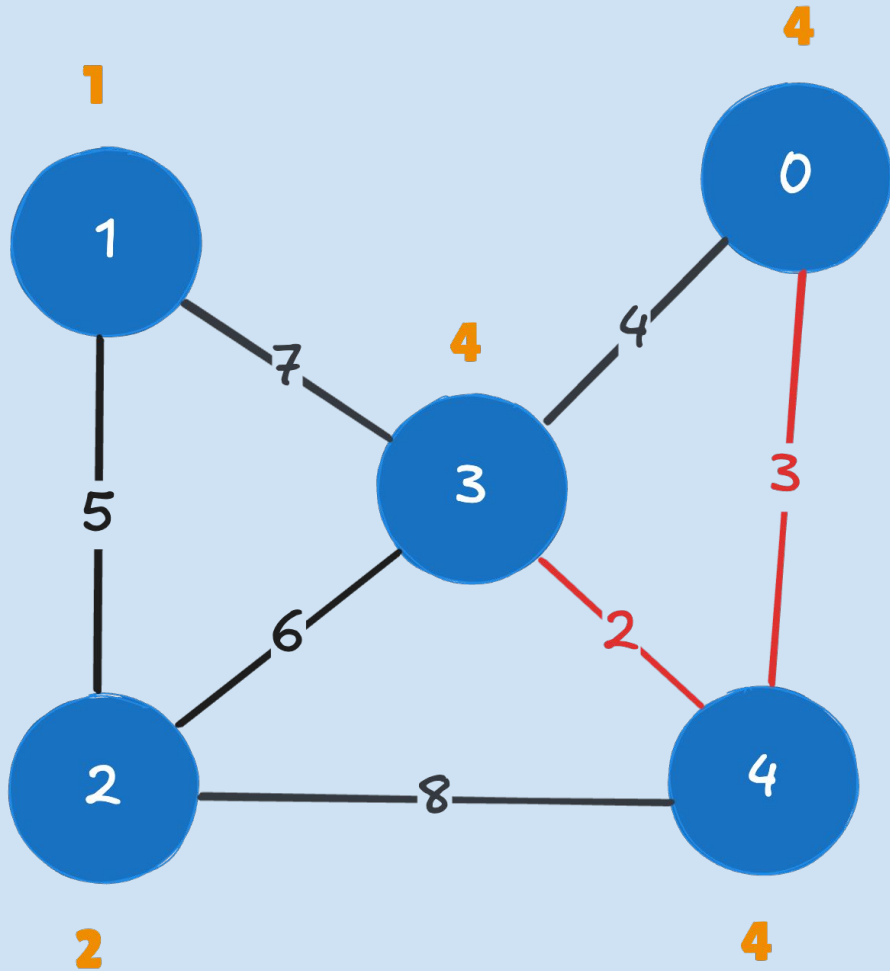


Edge $[0,4,3]$ connects vertices 0 and 4, which belong to different sets.

[

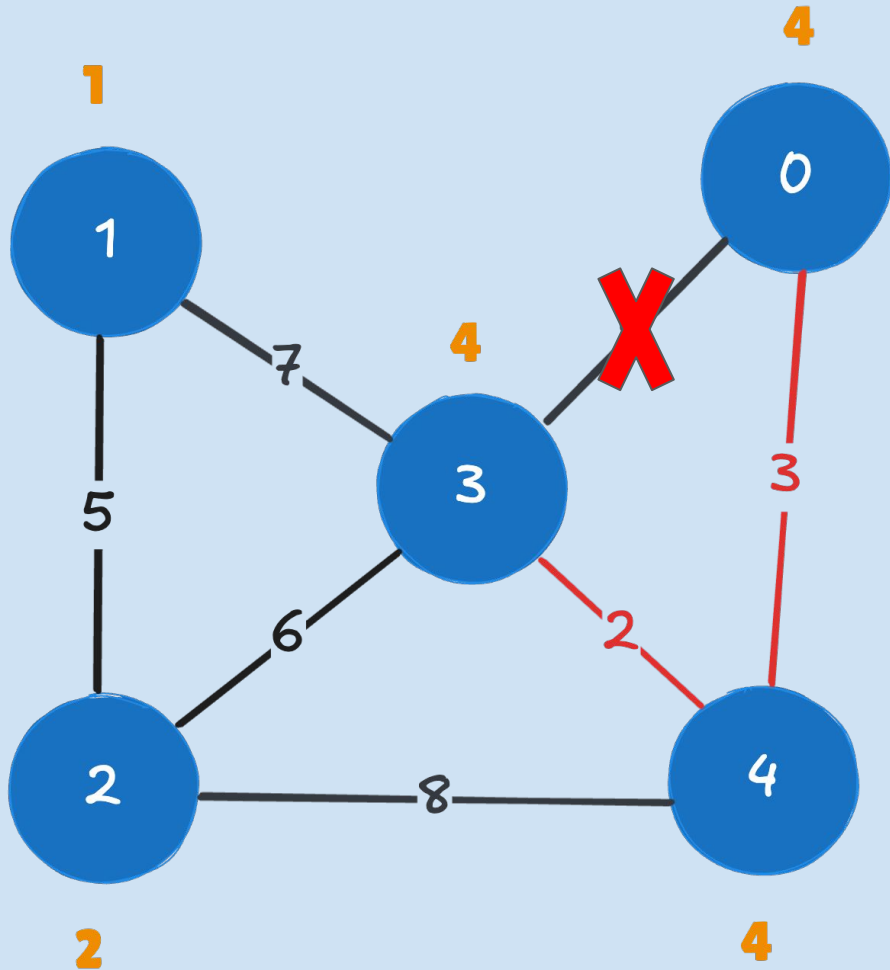
[3, 4, 2],	# Weight 2
[0, 4, 3],	# Weight 3
[0, 3, 4],	# Weight 4
[1, 2, 5],	# Weight 5
[2, 3, 6],	# Weight 6
[1, 3, 7],	# Weight 7
[2, 4, 8],	# Weight 8

]

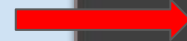


An union between the sets 4 and 5 is done.

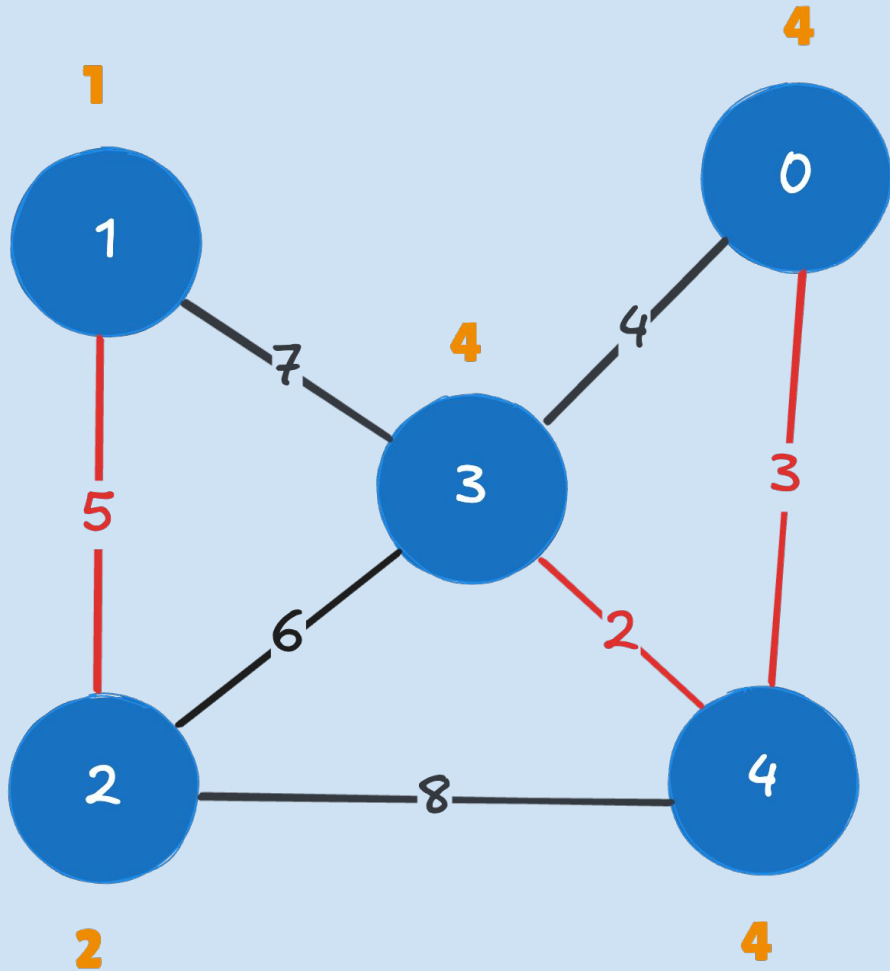
```
[  
    [3, 4, 2], # Weight 2  
    [0, 4, 3], # Weight 3  
    [0, 3, 4], # Weight 4  
    [1, 2, 5], # Weight 5  
    [2, 3, 6], # Weight 6  
    [1, 3, 7], # Weight 7  
    [2, 4, 8], # Weight 8  
]
```



Edge [0,3,4] connects vertices 0 and 3, which belong to same sets.

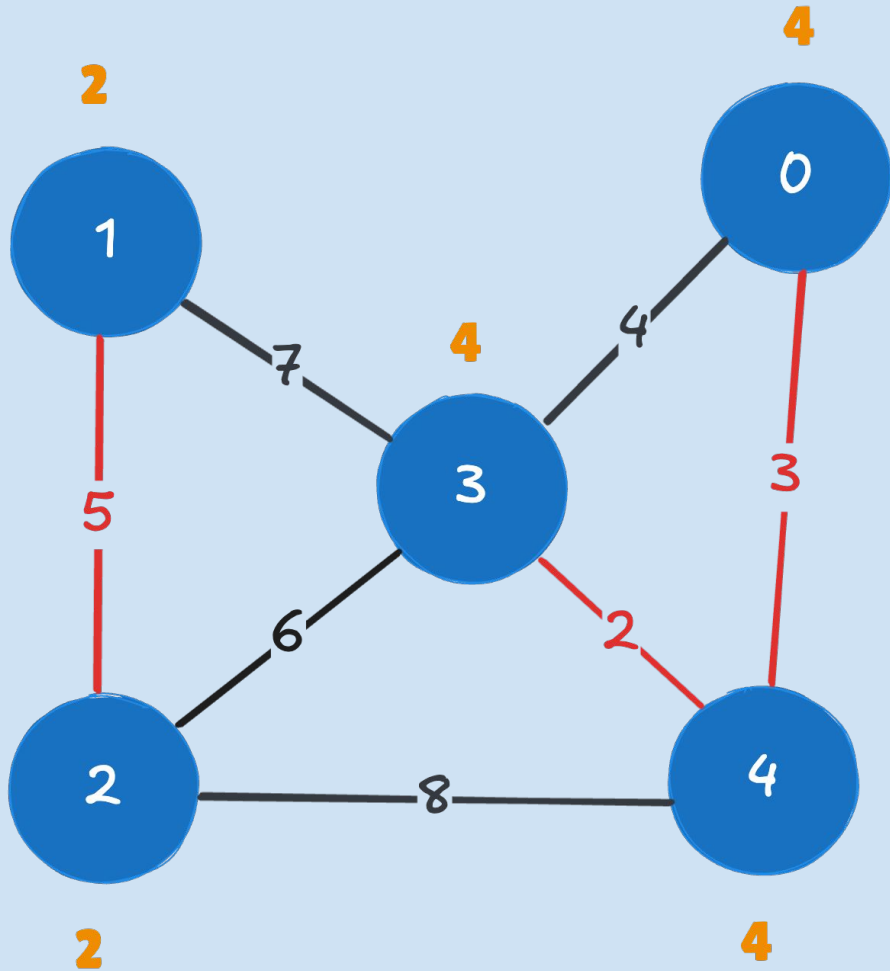


```
[  
  [3, 4, 2], # Weight 2  
  [0, 4, 3], # Weight 3  
  [0, 3, 4], # Weight 4  
  [1, 2, 5], # Weight 5  
  [2, 3, 6], # Weight 6  
  [1, 3, 7], # Weight 7  
  [2, 4, 8], # Weight 8  
]
```

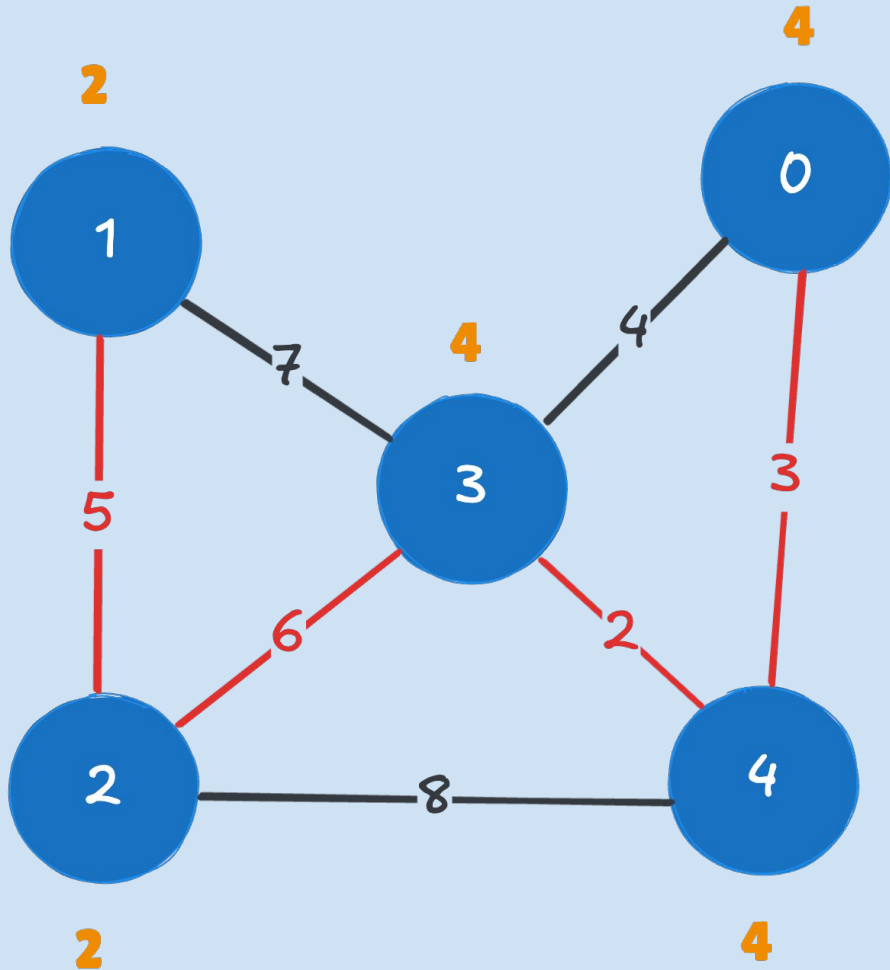
Edge [1,2,5] connects vertices 1 and 2, which belong to different sets.

```
[  
  [3, 4, 2], # Weight 2  
  [0, 4, 3], # Weight 3  
  [0, 3, 4], # Weight 4  
  [1, 2, 5], # Weight 5  
  [2, 3, 6], # Weight 6  
  [1, 3, 7], # Weight 7  
  [2, 4, 8], # Weight 8  
]
```



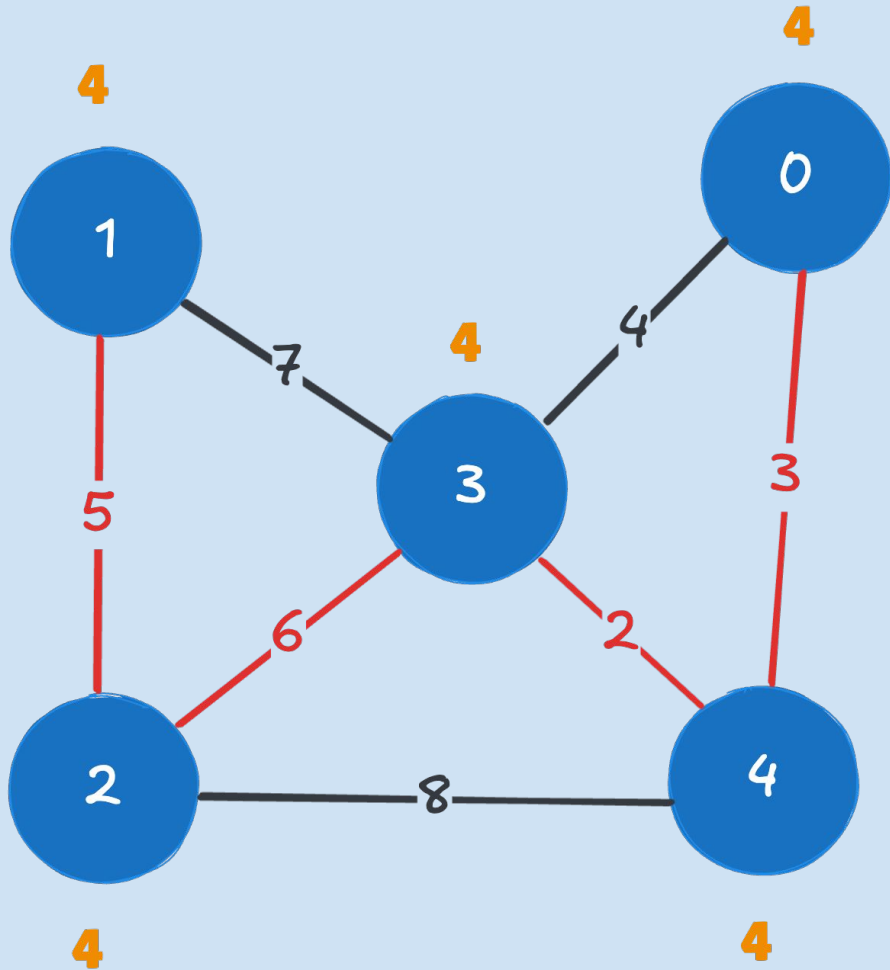
An union between the sets
1 and 2 is done.

```
[  
    [3, 4, 2], # Weight 2  
    [0, 4, 3], # Weight 3  
    [0, 3, 4], # Weight 4  
    [1, 2, 5], # Weight 5  
    [2, 3, 6], # Weight 6  
    [1, 3, 7], # Weight 7  
    [2, 4, 8], # Weight 8  
]
```



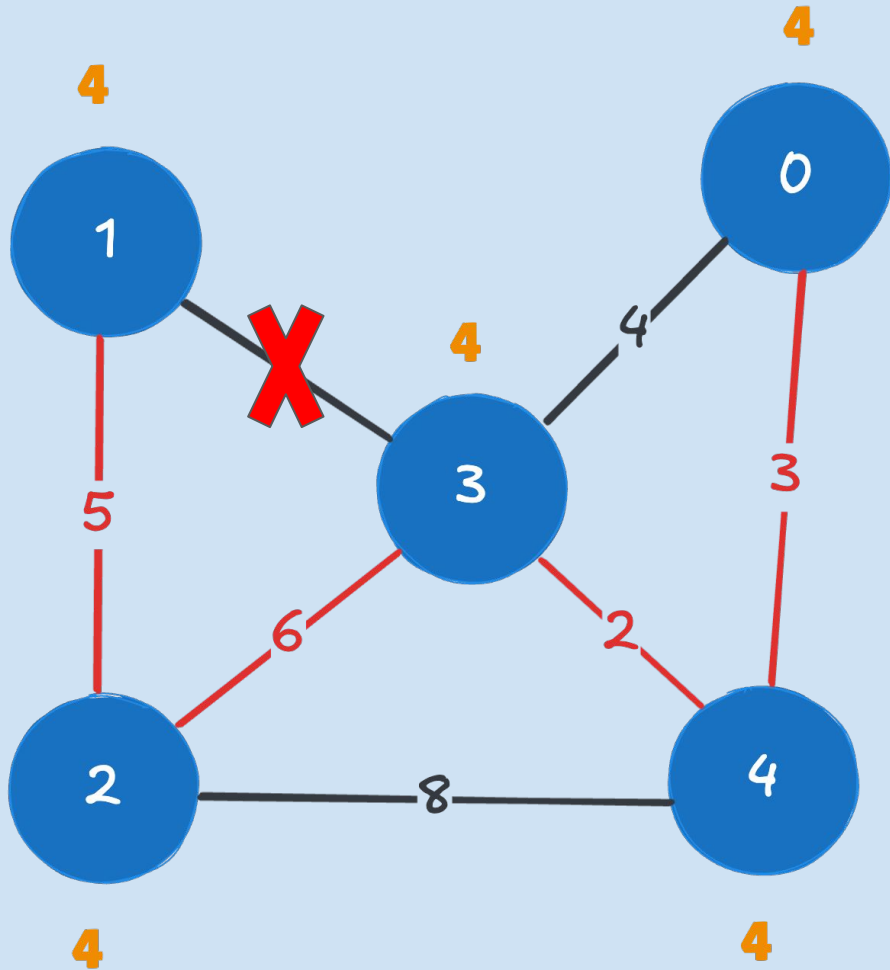
Edge [2,3,6] connects vertices 2 and 3, which belong to different sets.

```
[  
  [3, 4, 2], # Weight 2  
  [0, 4, 3], # Weight 3  
  [0, 3, 4], # Weight 4  
  [1, 2, 5], # Weight 5  
  [2, 3, 6], # Weight 6  
  [1, 3, 7], # Weight 7  
  [2, 4, 8], # Weight 8  
]
```



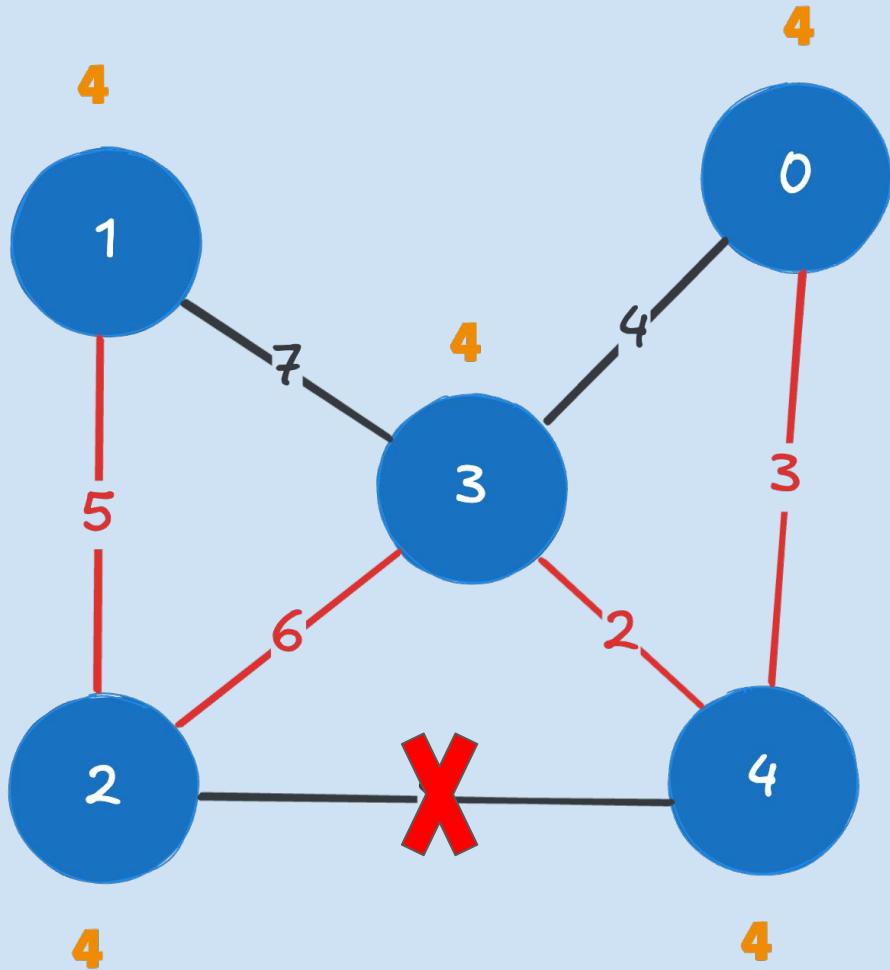
An union between the sets 2 and 4 is done.

```
[  
    [3, 4, 2], # Weight 2  
    [0, 4, 3], # Weight 3  
    [0, 3, 4], # Weight 4  
    [1, 2, 5], # Weight 5  
    [2, 3, 6], # Weight 6  
    [1, 3, 7], # Weight 7  
    [2, 4, 8], # Weight 8  
]
```



Edge [1,3,7] connects vertices 1 and 3, which belong to same sets.

```
[  
  [3, 4, 2], # Weight 2  
  [0, 4, 3], # Weight 3  
  [0, 3, 4], # Weight 4  
  [1, 2, 5], # Weight 5  
  [2, 3, 6], # Weight 6  
  [1, 3, 7], # Weight 7  
  [2, 4, 8], # Weight 8  
]
```



Edge $[2,4,8]$ connects vertices 2 and 4, which belong to same sets.

```
[  
  [3, 4, 2], # Weight 2  
  [0, 4, 3], # Weight 3  
  [0, 3, 4], # Weight 4  
  [1, 2, 5], # Weight 5  
  [2, 3, 6], # Weight 6  
  [1, 3, 7], # Weight 7  
  [2, 4, 8], # Weight 8  
]
```


How about the complexity?

$O(E \log(E))$ time
 $O(E+V)$ space



Tirol neighborhoods, Natal-RN

Infrastructure Optimization:

Use Kruskal's Algorithm to design the most cost-effective layout for infrastructure like fiber optics, water pipelines, or power grids by minimizing the total length of required connections.

Transportation Planning:

Connect major transportation hubs (e.g., bus terminals, train stations, and airports) using a Minimum Spanning Tree (MST) to identify the shortest routes and reduce travel distances.

Tourism Route Optimization:

Generate an MST connecting key tourist attractions (e.g., museums, landmarks, beaches) to design efficient sightseeing routes that minimize travel distance.

Urban Expansion Planning:

Use an MST to connect developing neighborhoods or areas under construction to the existing city network with minimal infrastructure costs.

Critical Infrastructure Analysis:

Identify critical roads or intersections by comparing the MST with the original road network, highlighting streets essential for maintaining connectivity.

MST entre POIs (hospitais) em Natal

