
C
CodeIT

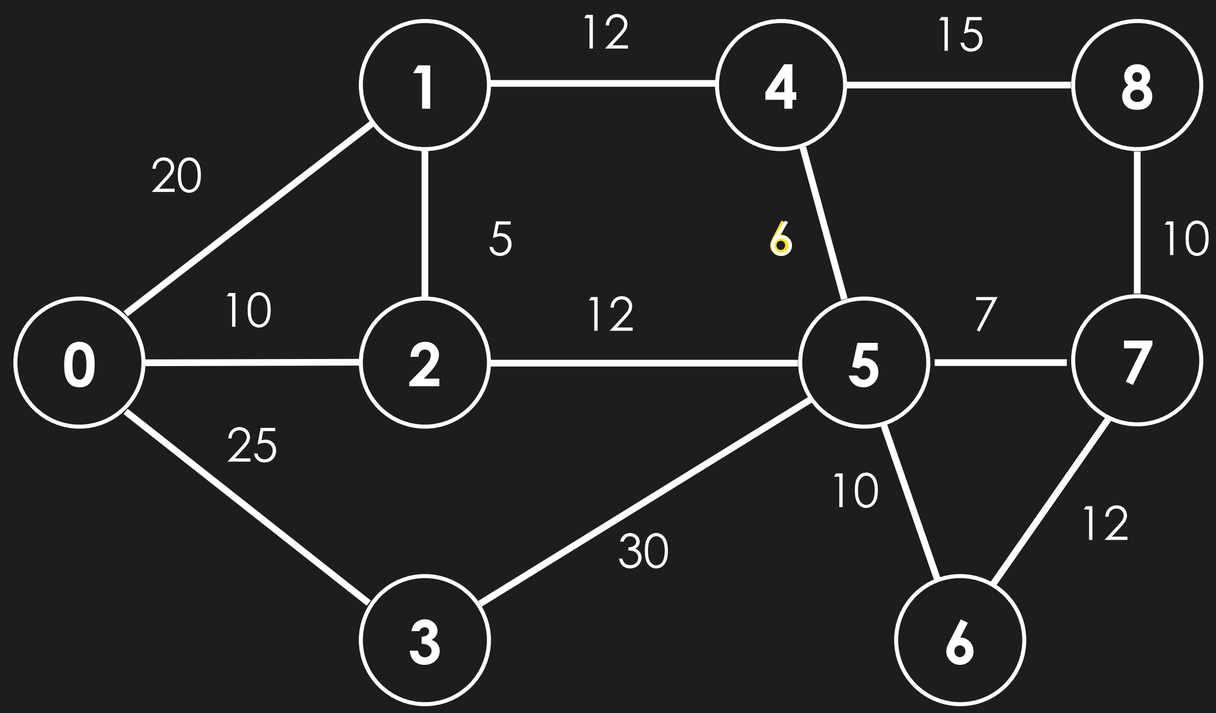
Minimum Spanning Trees

What is a spanning tree?

A spanning tree is a subgraph which includes the minimum number of edges possible such that all vertices in the graph are connected

What is a spanning tree?

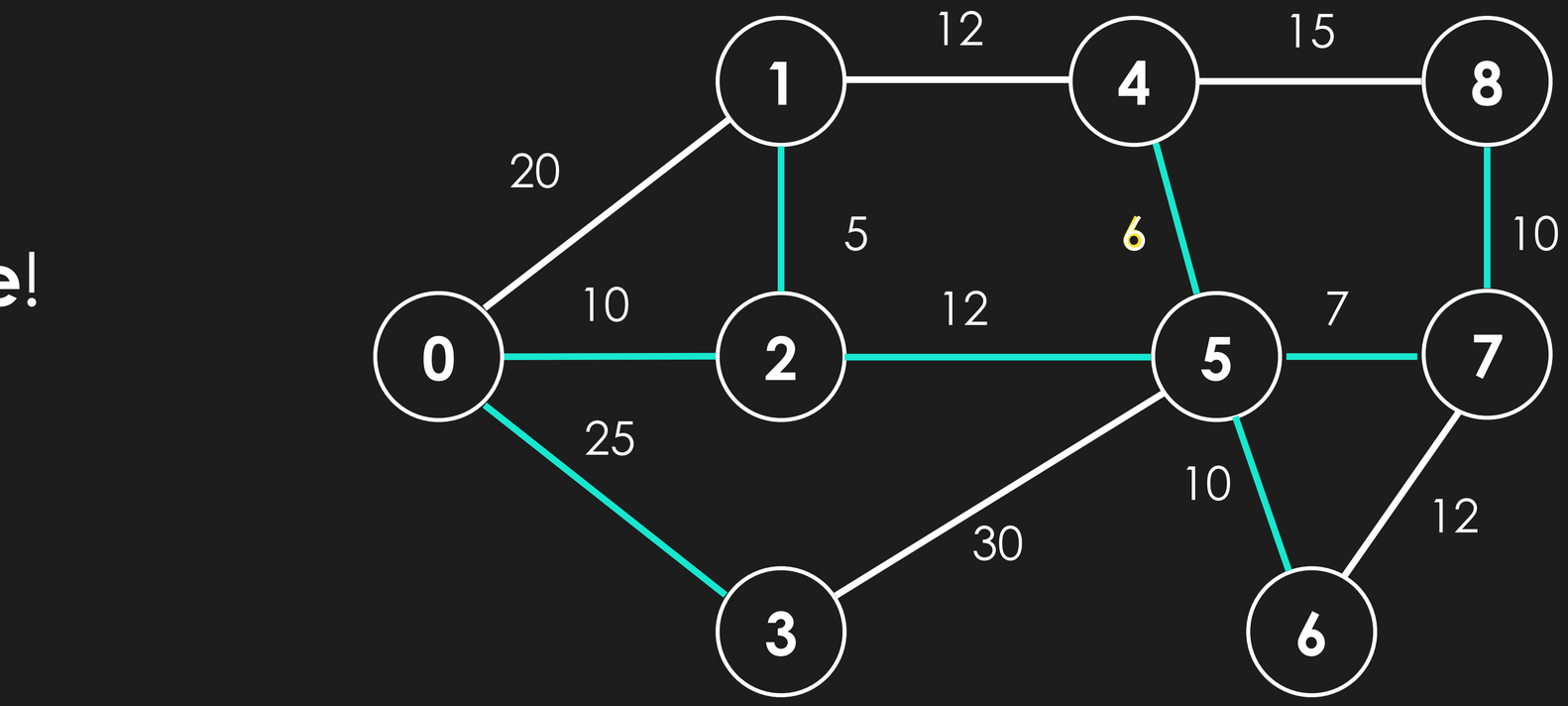
A spanning tree is a subgraph which includes the minimum number of edges possible such that all vertices in the graph are connected



What is a spanning tree?

A spanning tree is a subgraph which includes the minimum number of edges possible such that all vertices in the graph are connected

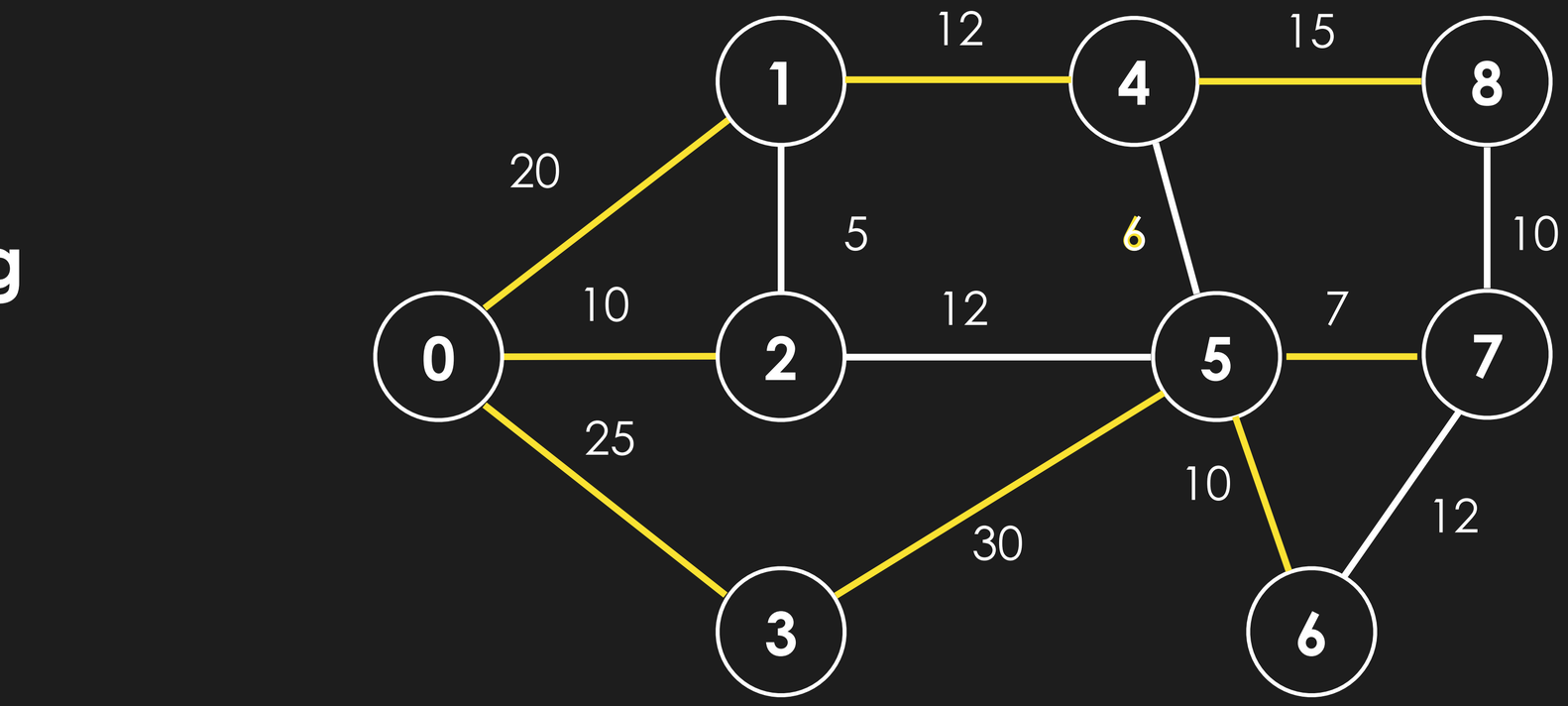
This is a **spanning tree**!



What is a spanning tree?

A spanning tree is a subgraph which includes the minimum number of edges possible such that all vertices in the graph are connected

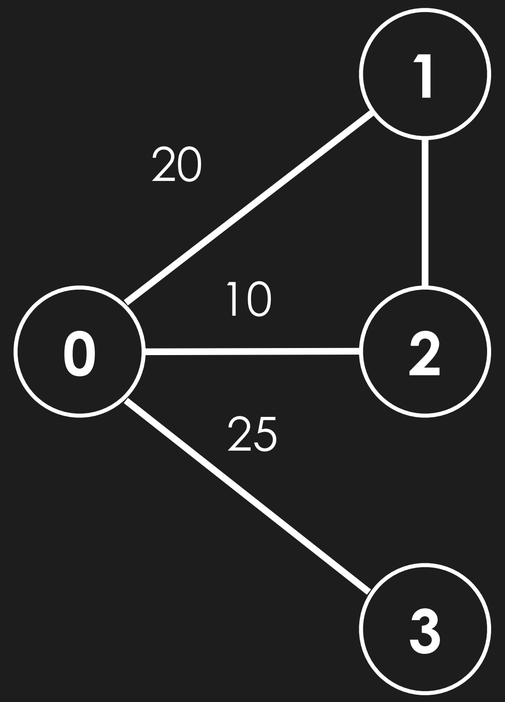
This is also a **spanning tree**!

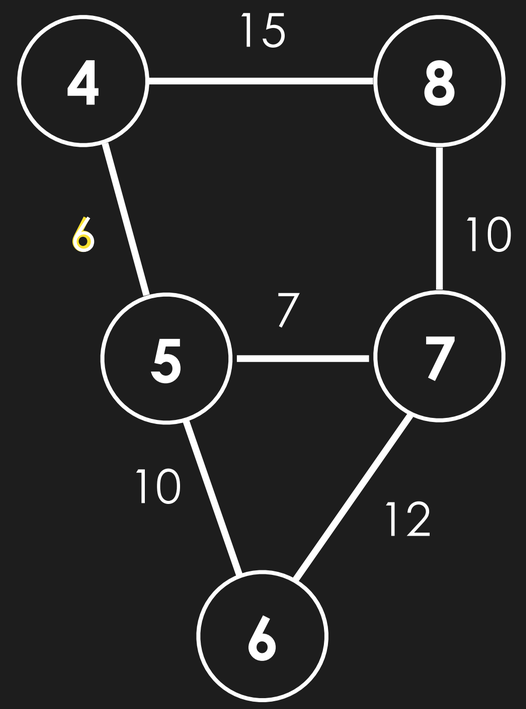


What is a spanning tree?

Spanning trees don't exist in graphs which are not **strongly connected** (every vertex has a path to every other vertex)

No edge from:
0, 1, 2, 3
to
4, 5, 6, 7, 8





Quiz: Why do you think spanning trees are useful?

Minimum Spanning Trees

Minimum spanning trees are spanning trees which have the **minimum sum of edge weights**

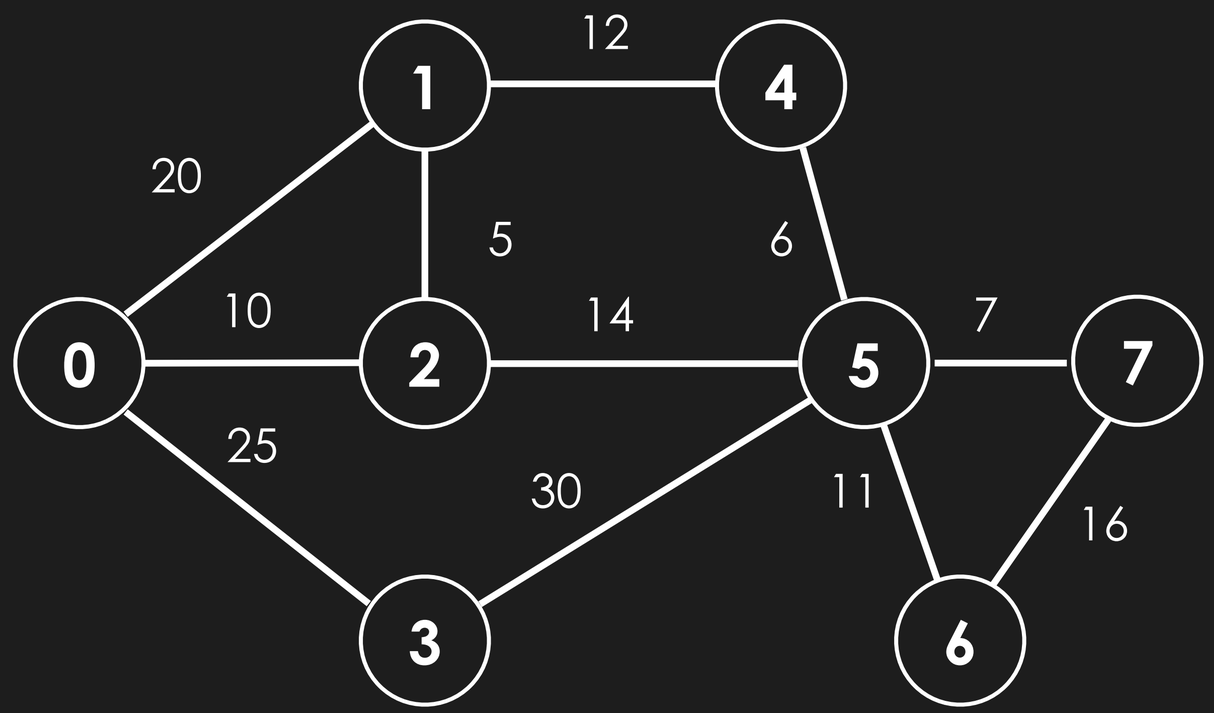
A graph can have **multiple minimum spanning trees**

How can we find a minimum spanning tree (MST) ?

MST Algorithm

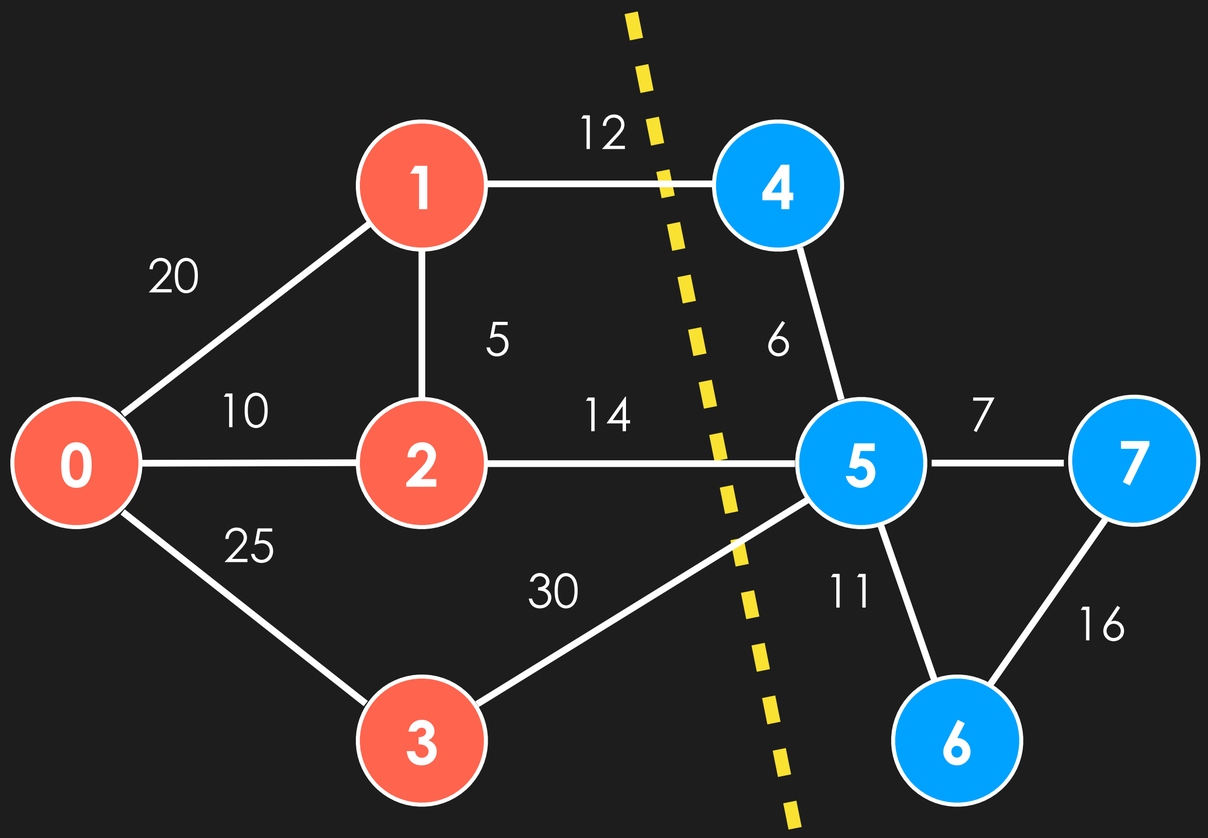
Definitions

A. **Cut:** A cut in a graph is a partition of its vertices into 2 **non empty sets**



Definitions

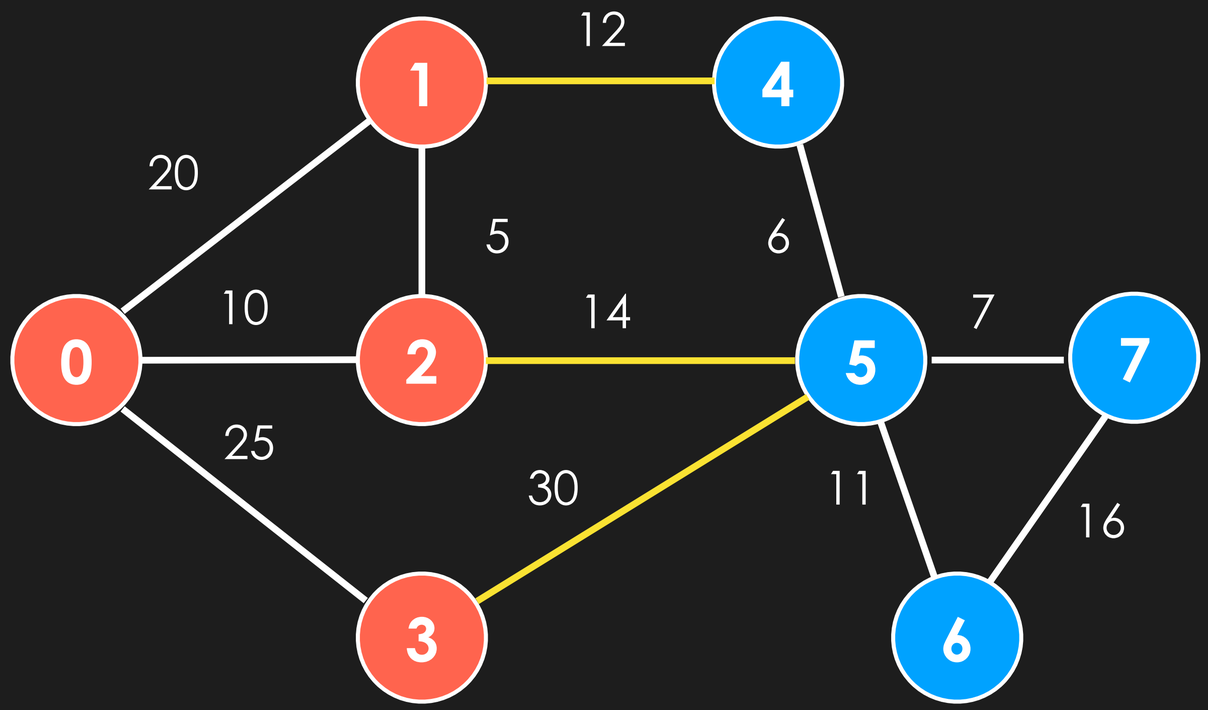
A. **Cut:** A cut in a graph is a partition of its vertices into 2 **non empty sets**



Definitions

A. **Cut:** A cut in a graph is a partition of its vertices into 2 **non empty sets**

B. For any **cut**, there are a **set of edges (let's call them cut edges)** that connect a vertex from **set A** to **set B**

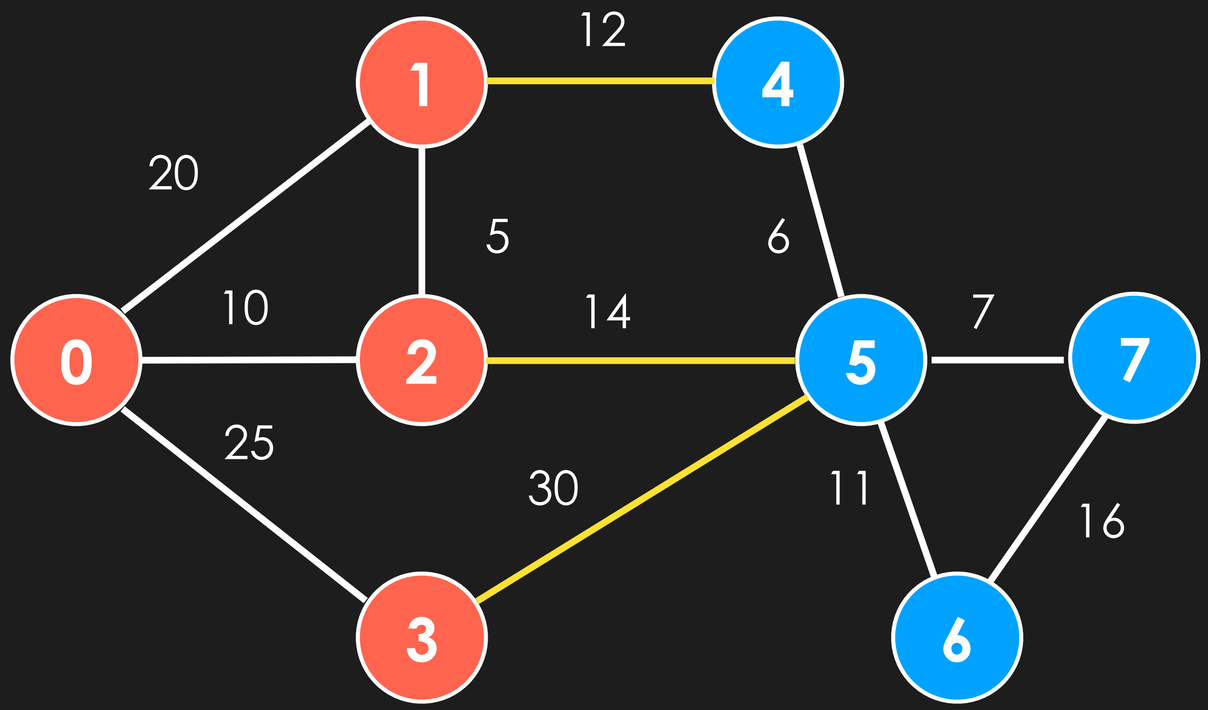


Definitions

A. **Cut:** A cut in a graph is a partition of its vertices into 2 **non empty sets**

B. For any **cut**, there are a **set of edges (let's call them cut edges)** that connect a vertex from **set A** to **set B**

C. For a set of **cut edges**, the **cut edge** with the smallest weight belongs to the **MST**

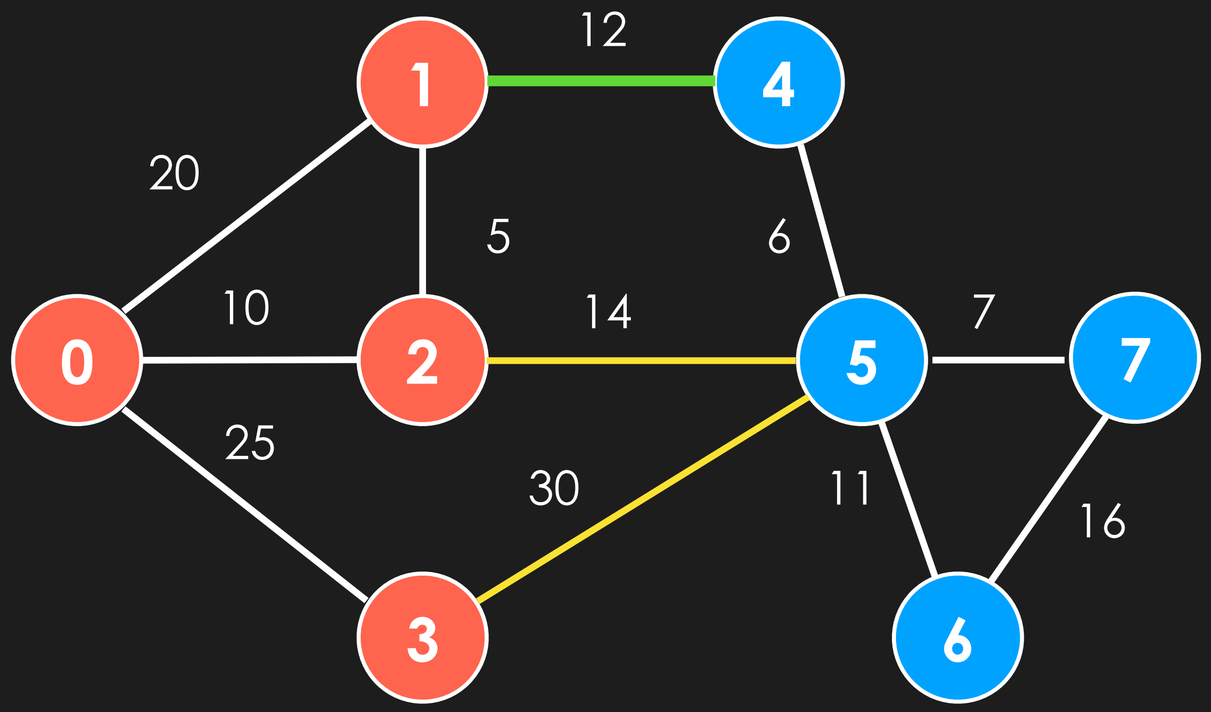


Definitions

A. **Cut:** A cut in a graph is a partition of its vertices into 2 **non empty sets**

B. For any **cut**, there are a **set of edges (let's call them cut edges)** that connect a vertex from **set A** to **set B**

C. For a set of **cut edges**, the **cut edge** with the **smallest weight** belongs to the **MST**



1 - 4 belongs to the MST!

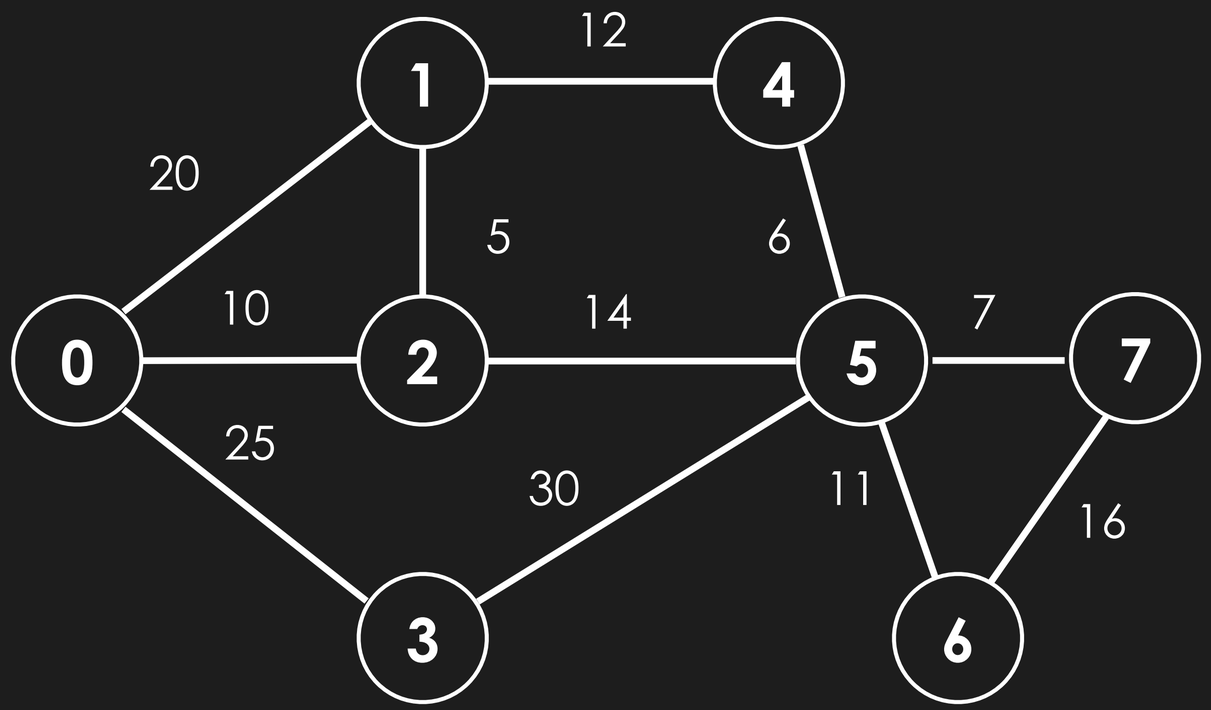
How can we use this idea to determine the MST?

1. Find a **cut** which contains no **green edge** (green indicates edge has been added to MST)

2. **Add** the min weight cut edge to the MST (mark edge green)

3. **Repeat** until V - 1 edges are added to the MST

MST

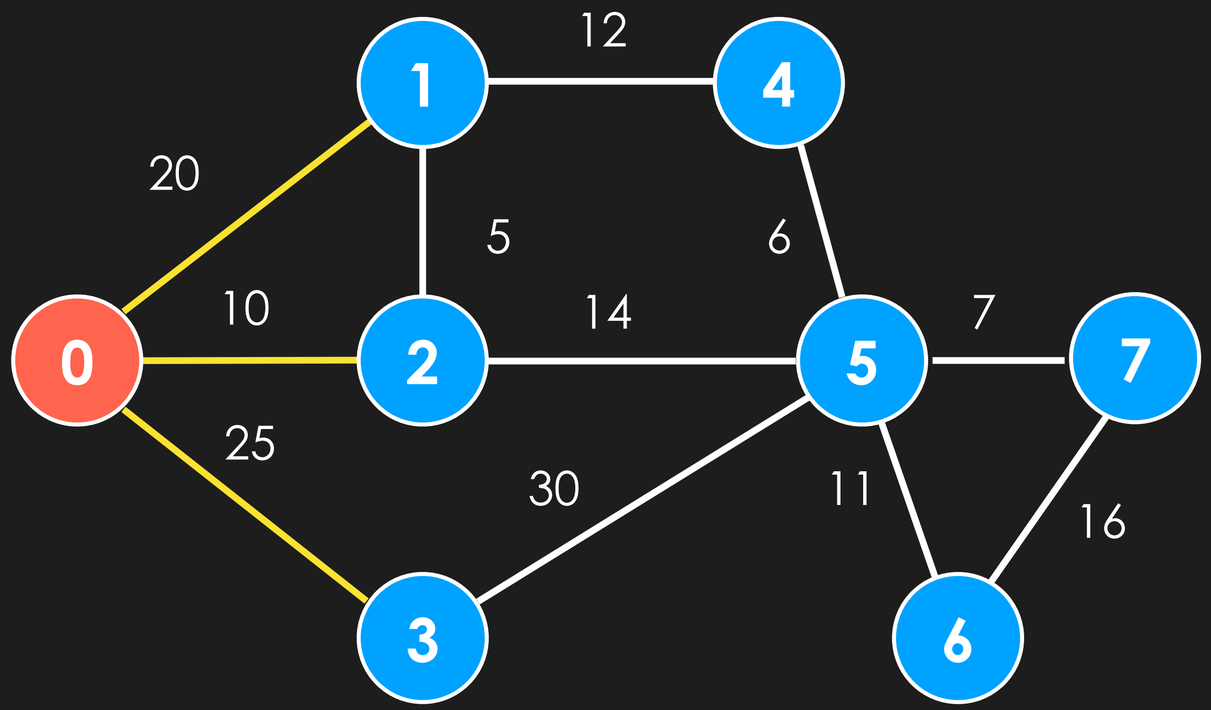


1. Find a **cut** which contains no **green edge** (green indicates edge has been added to MST)

2. **Add** the min weight cut edge to the MST (mark edge green)

3. **Repeat** until V - 1 edges are added to the MST

MST



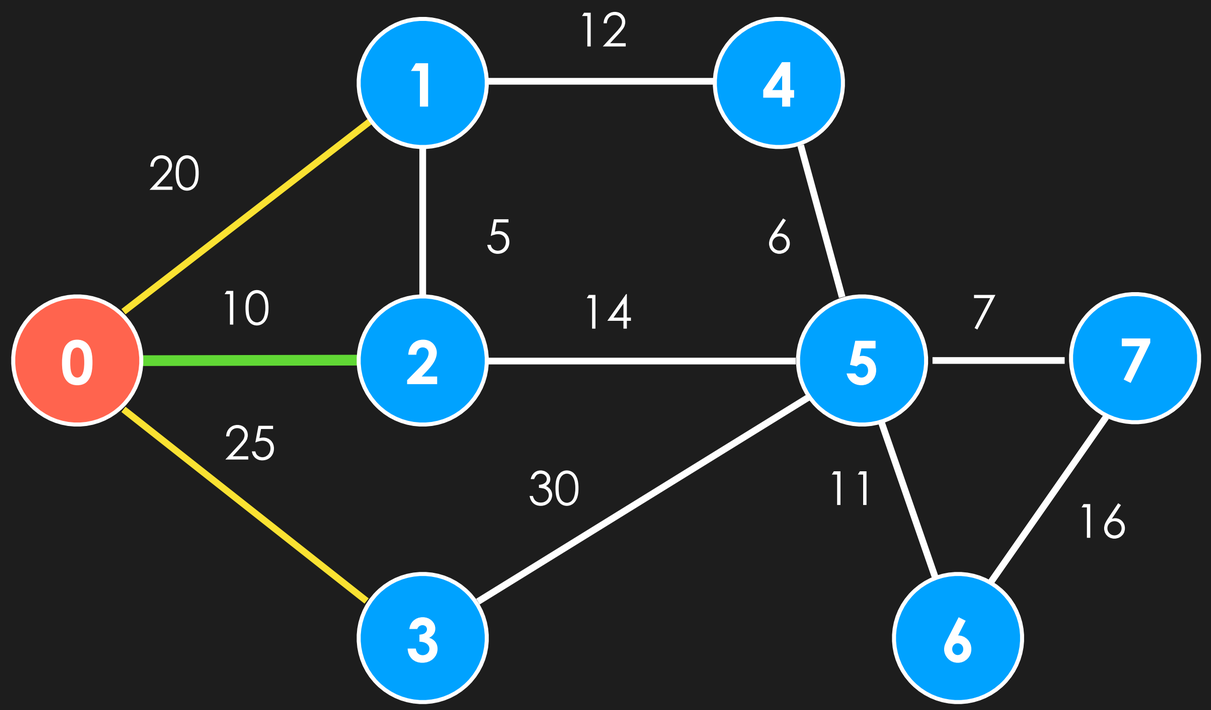
1. Find a **cut** which contains no **green edge** (green indicates edge has been added to MST)

2. **Add** the min weight cut edge to the MST (mark edge green)

3. **Repeat** until V - 1 edges are added to the MST

MST

0 - 2



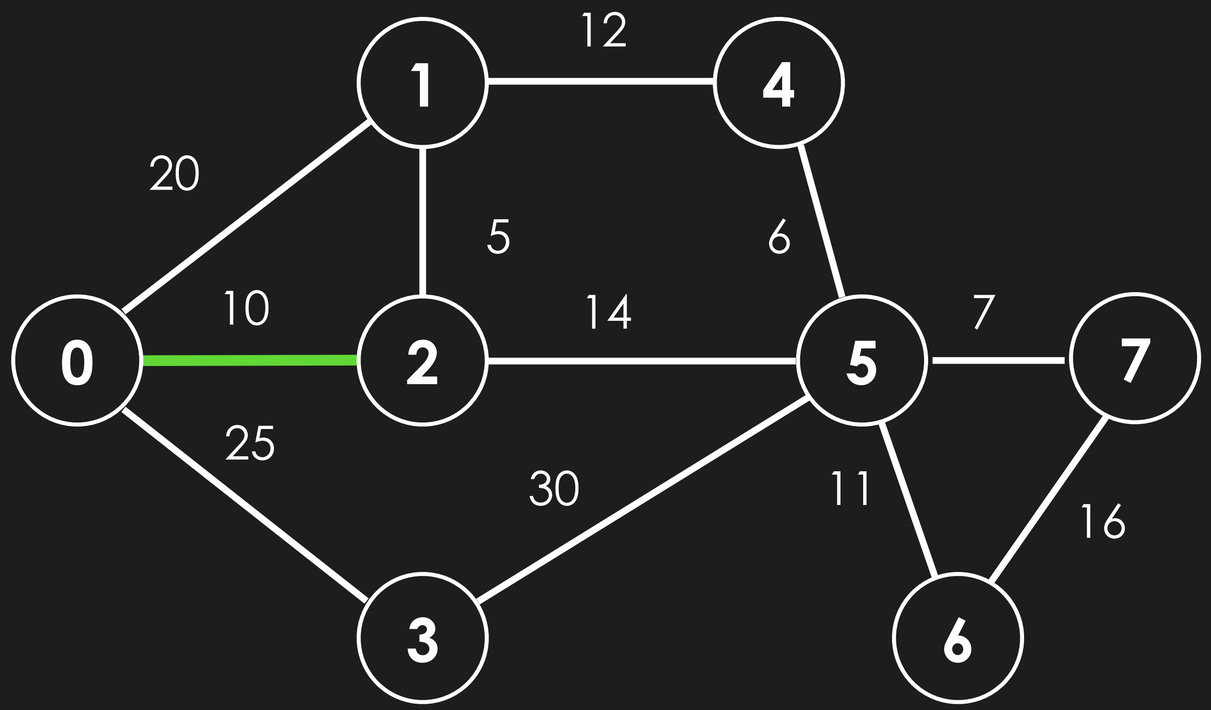
1. Find a **cut** which contains no **green edge** (green indicates edge has been added to MST)

2. **Add** the min weight cut edge to the MST (mark edge green)

3. **Repeat** until V - 1 edges are added to the MST

MST

0 - 2



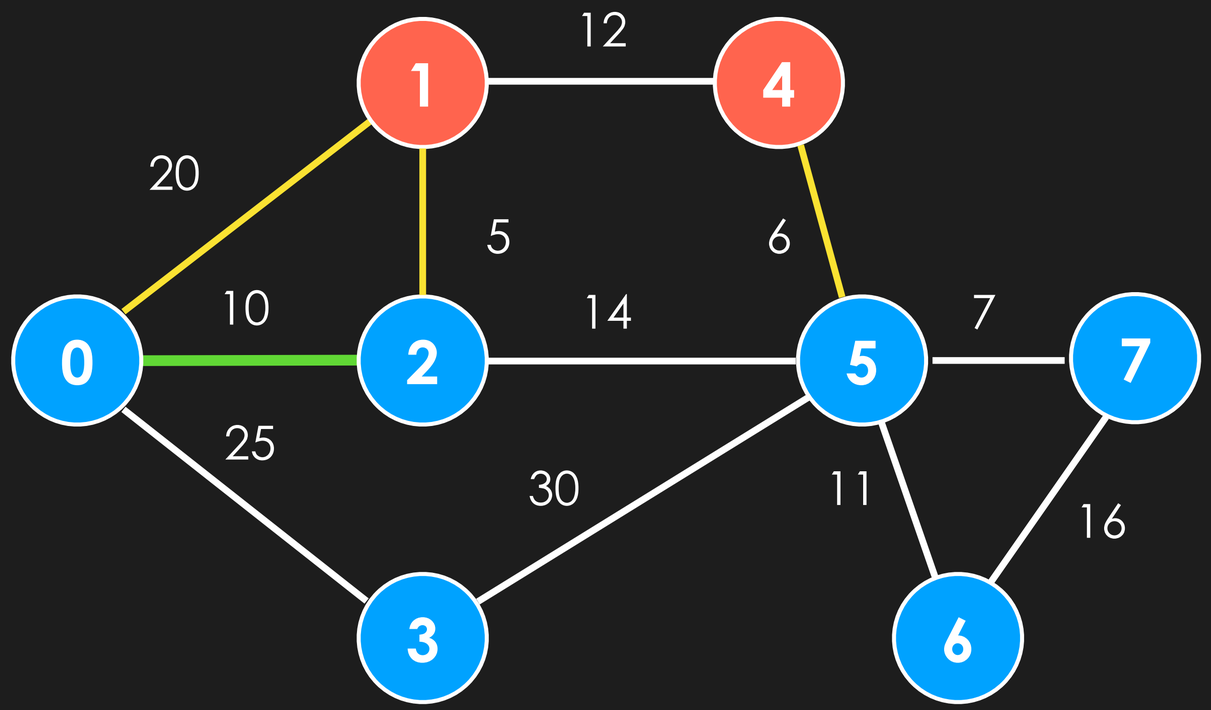
1. Find a **cut** which contains no **green edge** (green indicates edge has been added to MST)

2. **Add** the min weight cut edge to the MST (mark edge green)

3. **Repeat** until V - 1 edges are added to the MST

MST

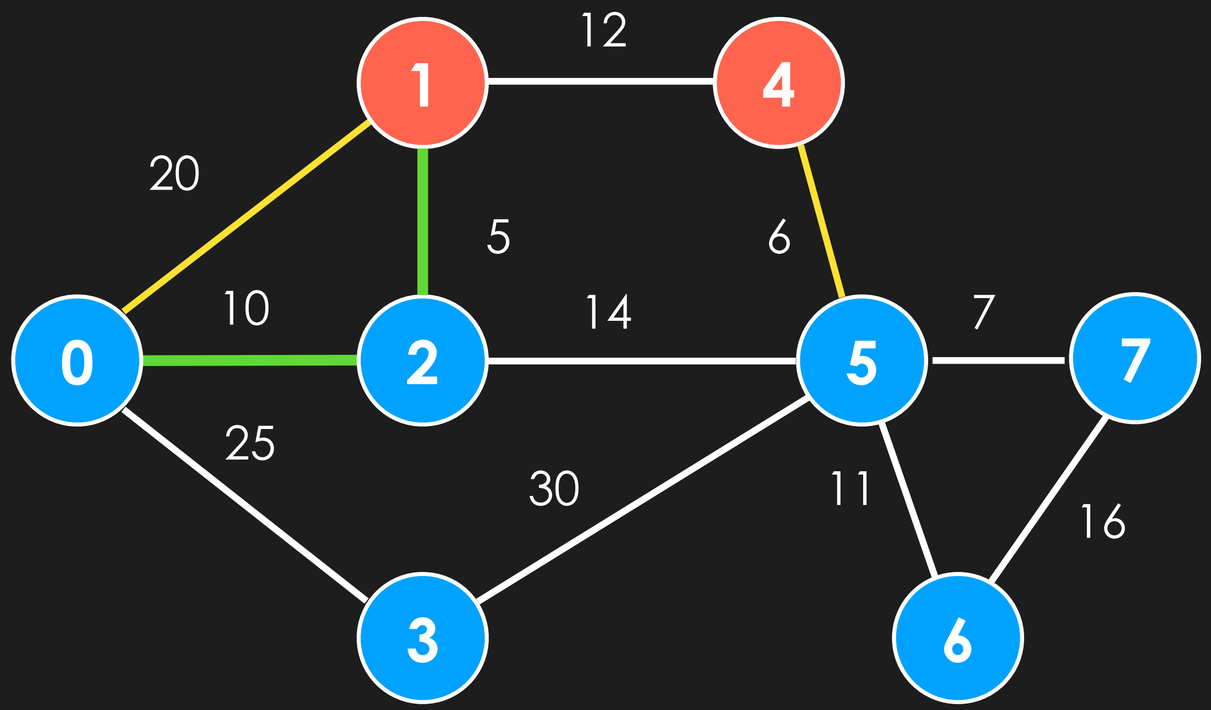
0 - 2



1. Find a **cut** which contains no **green edge** (green indicates edge has been added to MST)

2. **Add** the min weight cut edge to the MST (mark edge green)

3. **Repeat** until V - 1 edges are added to the MST



MST

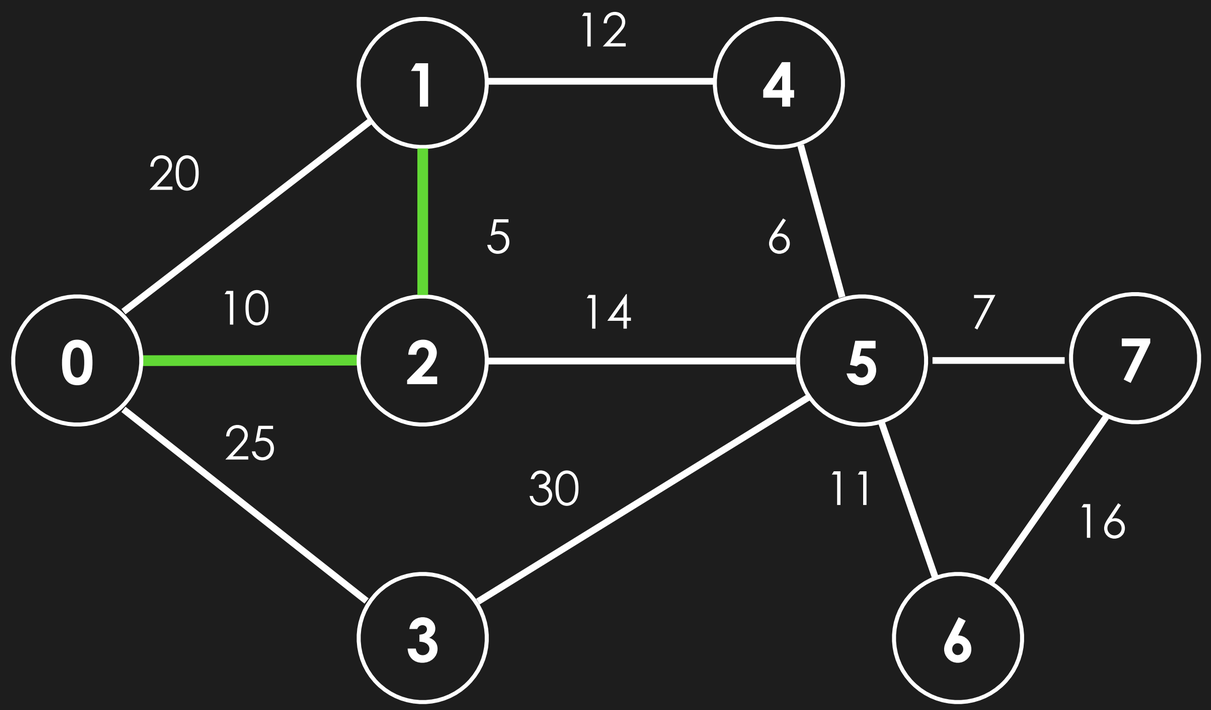
0 - 2

1 - 2

1. Find a **cut** which contains no **green edge** (green indicates edge has been added to MST)

2. **Add** the min weight cut edge to the MST (mark edge green)

3. **Repeat** until V - 1 edges are added to the MST



MST

0 - 2

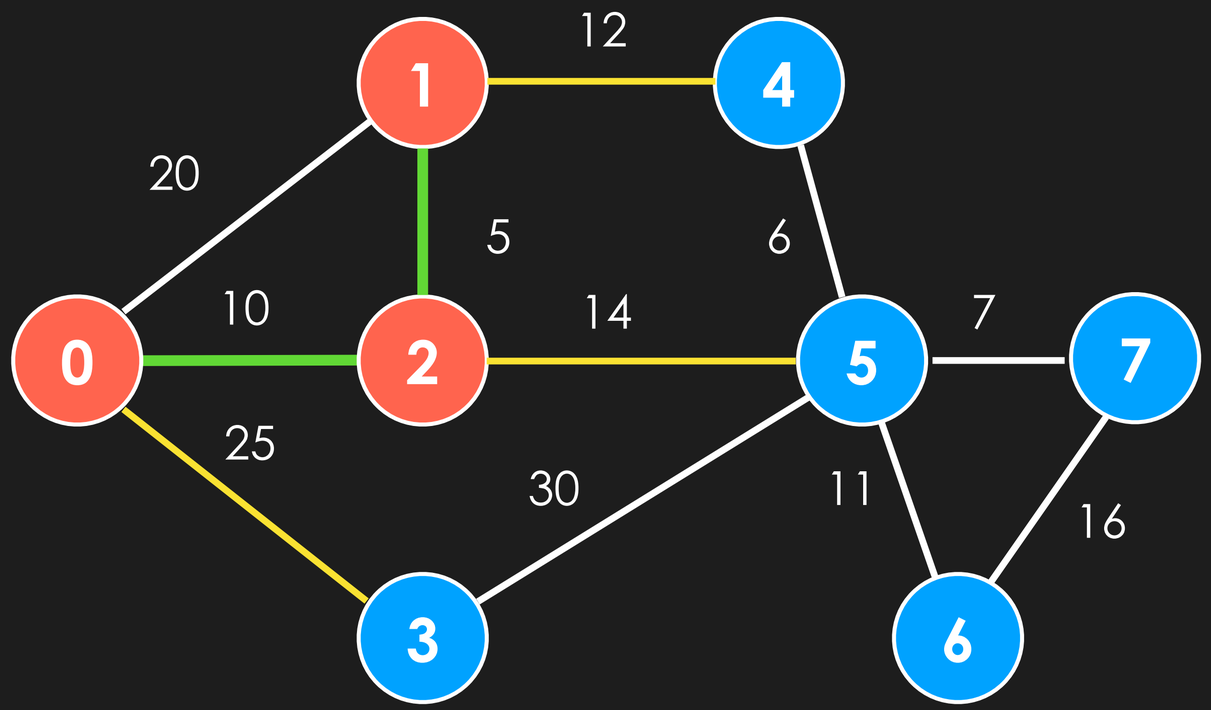
1 - 2

CodeIT

1. Find a **cut** which contains no **green edge** (green indicates edge has been added to MST)

2. **Add** the min weight cut edge to the MST (mark edge green)

3. **Repeat** until V - 1 edges are added to the MST



MST

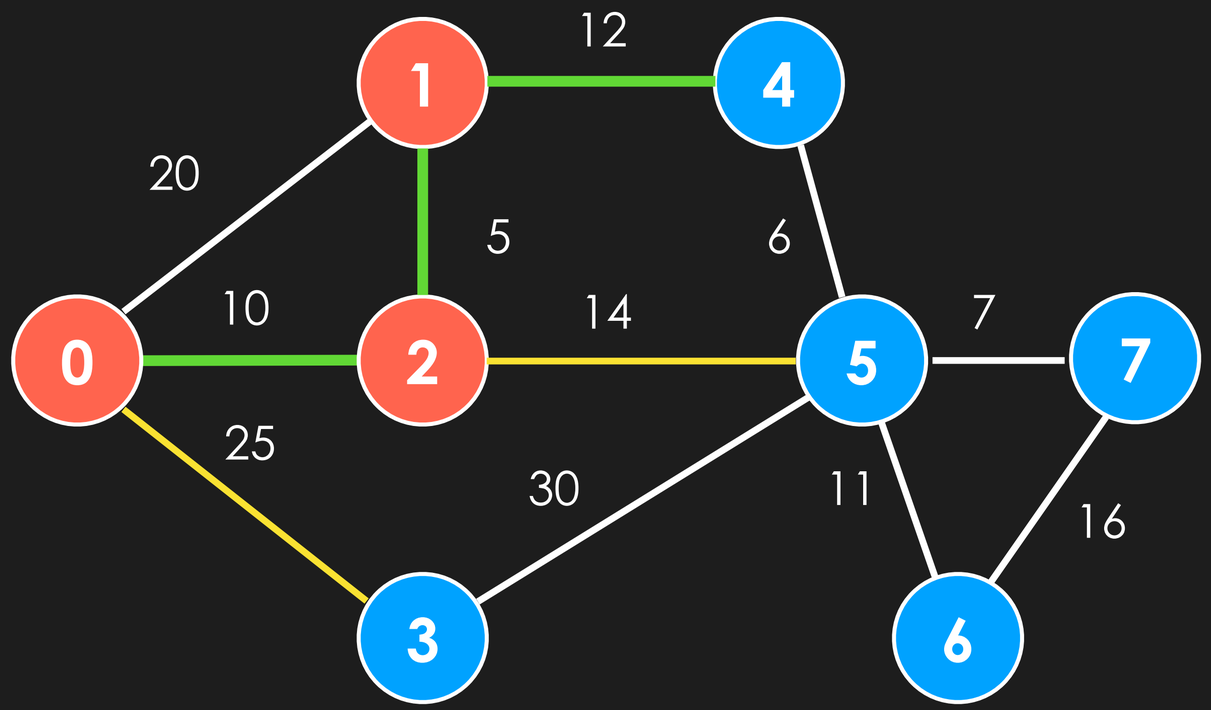
0 - 2

1 - 2

1. Find a **cut** which contains no **green edge** (green indicates edge has been added to MST)

2. **Add** the min weight cut edge to the MST (mark edge green)

3. **Repeat** until V - 1 edges are added to the MST



MST

0 - 2

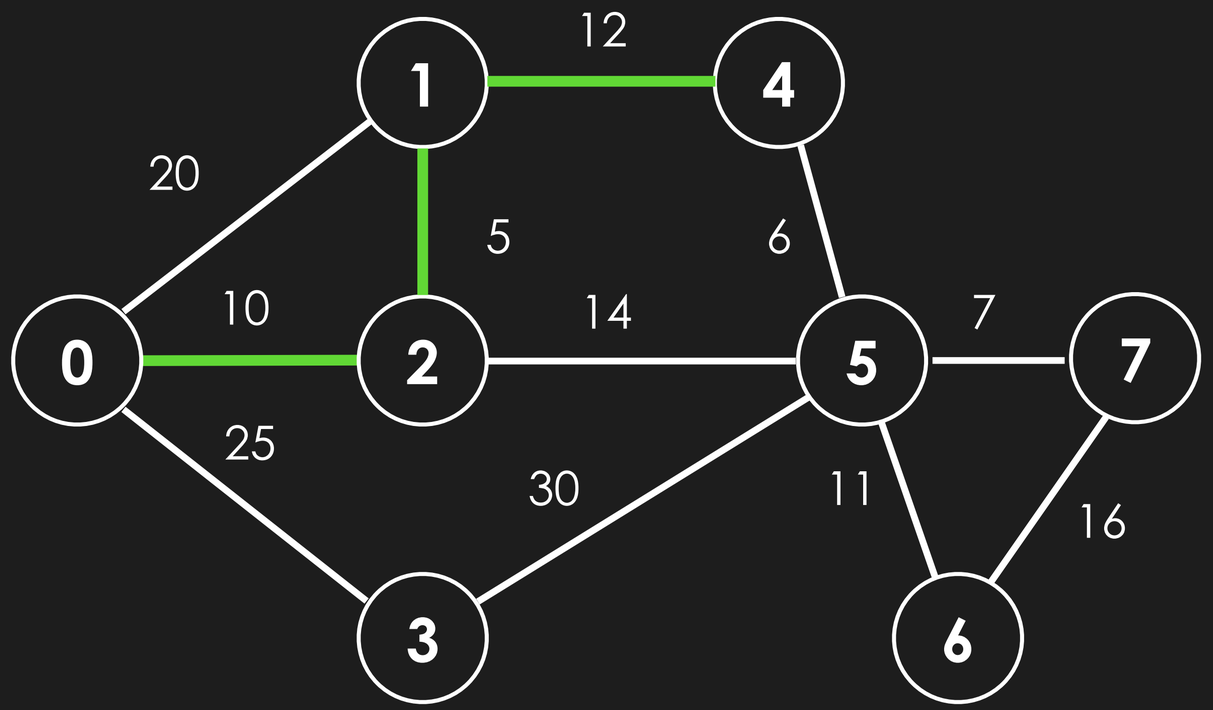
1 - 2

1 - 4

1. Find a **cut** which contains no **green edge** (green indicates edge has been added to MST)

2. **Add** the min weight cut edge to the MST (mark edge green)

3. **Repeat** until V - 1 edges are added to the MST



MST

0 - 2

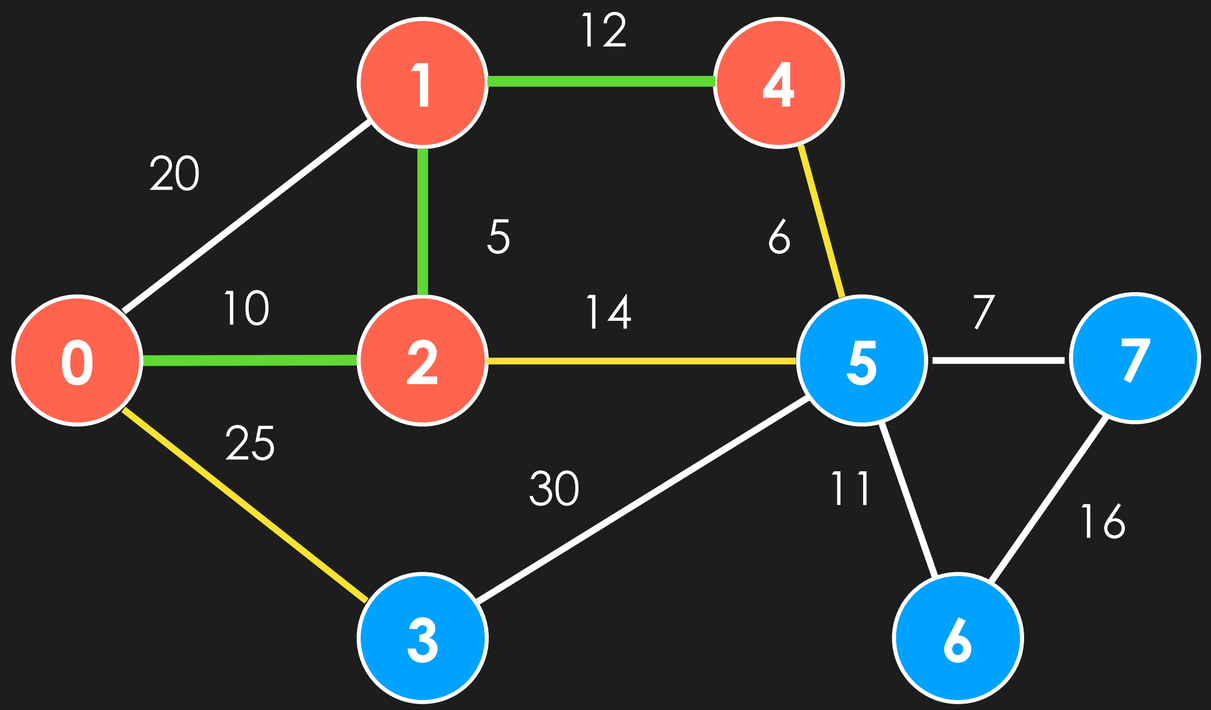
1 - 2

1 - 4

1. Find a **cut** which contains no **green edge** (green indicates edge has been added to MST)

2. **Add** the min weight cut edge to the MST (mark edge green)

3. **Repeat** until V - 1 edges are added to the MST



MST

0 - 2

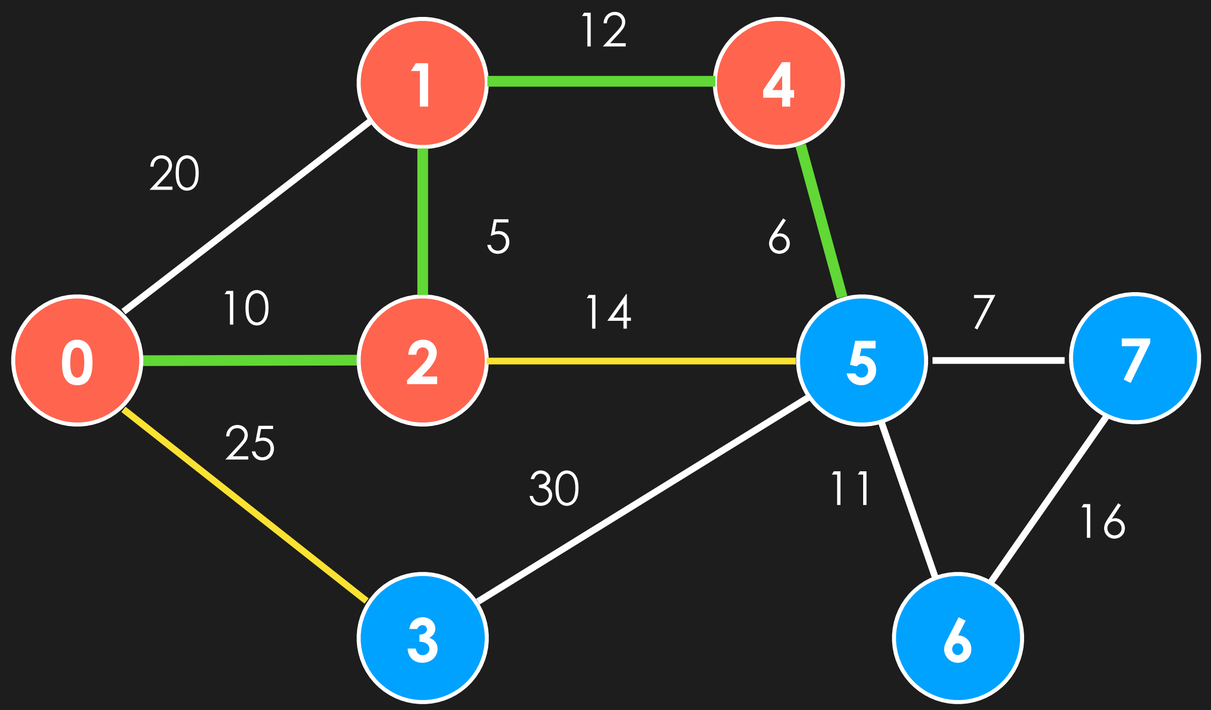
1 - 2

1 - 4

1. Find a **cut** which contains no **green edge** (green indicates edge has been added to MST)

2. **Add** the min weight cut edge to the MST (mark edge green)

3. **Repeat** until V - 1 edges are added to the MST



MST

0 - 2

1 - 2

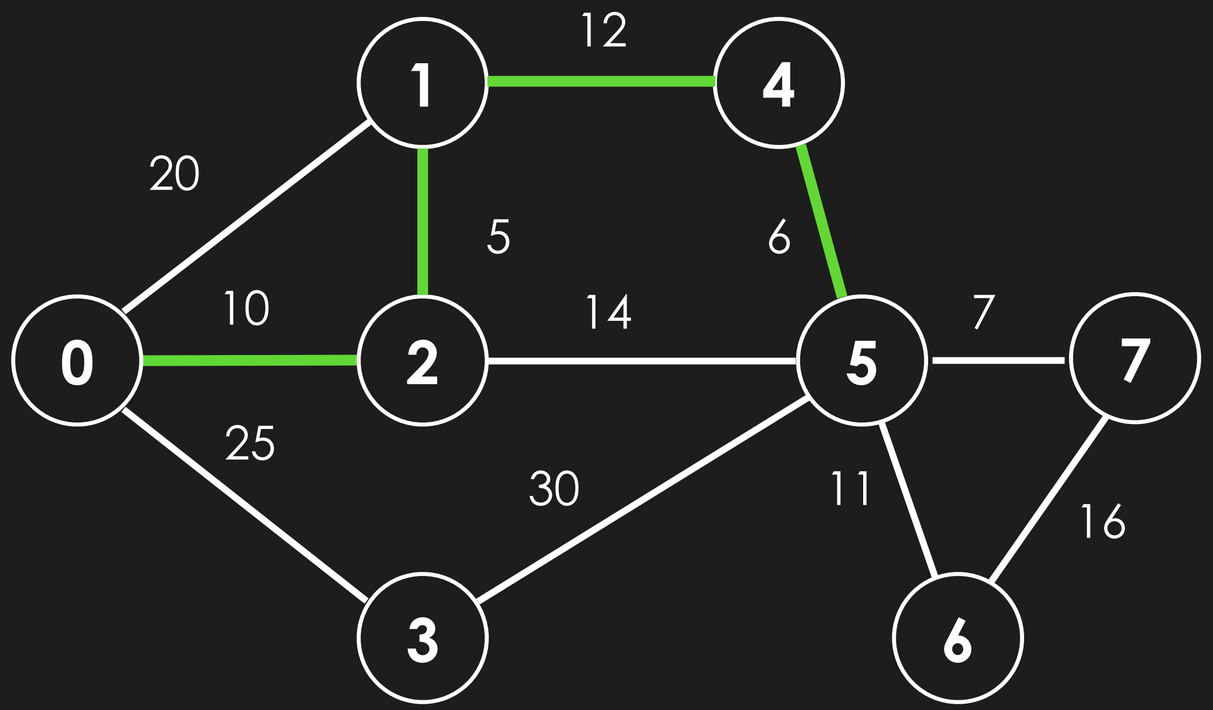
1 - 4

4 - 5

1. Find a **cut** which contains no **green edge** (green indicates edge has been added to MST)

2. **Add** the min weight cut edge to the MST (mark edge green)

3. **Repeat** until V - 1 edges are added to the MST



MST

0 - 2

1 - 2

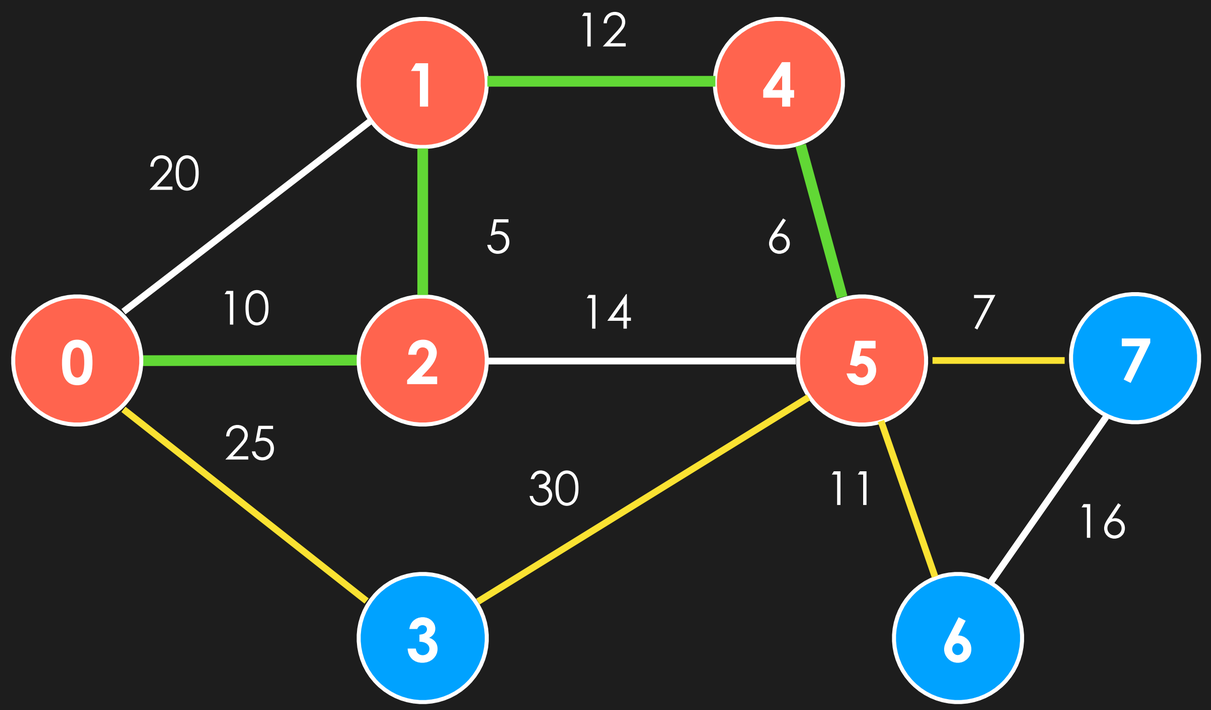
1 - 4

4 - 5

1. Find a **cut** which contains no **green edge** (green indicates edge has been added to MST)

2. **Add** the min weight cut edge to the MST (mark edge green)

3. **Repeat** until V - 1 edges are added to the MST



MST

0 - 2

1 - 2

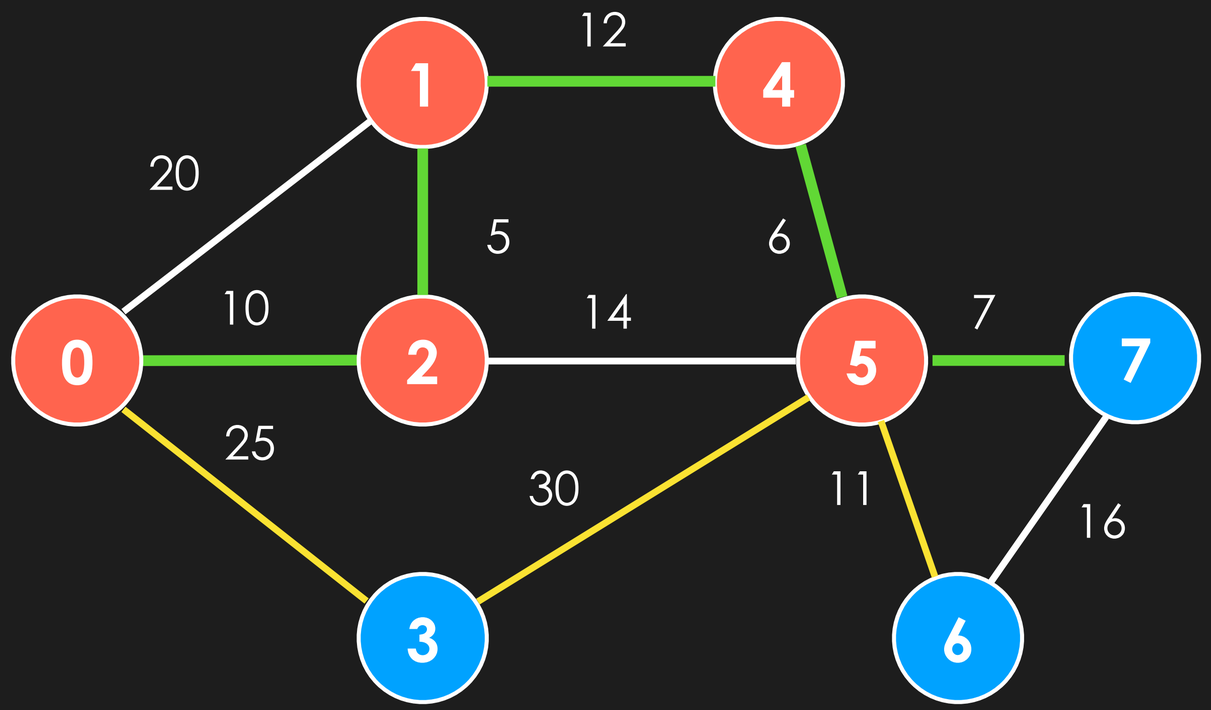
1 - 4

4 - 5

1. Find a **cut** which contains no **green edge** (green indicates edge has been added to MST)

2. **Add** the min weight cut edge to the MST (mark edge green)

3. **Repeat** until V - 1 edges are added to the MST



MST

0 - 2

1 - 2

1 - 4

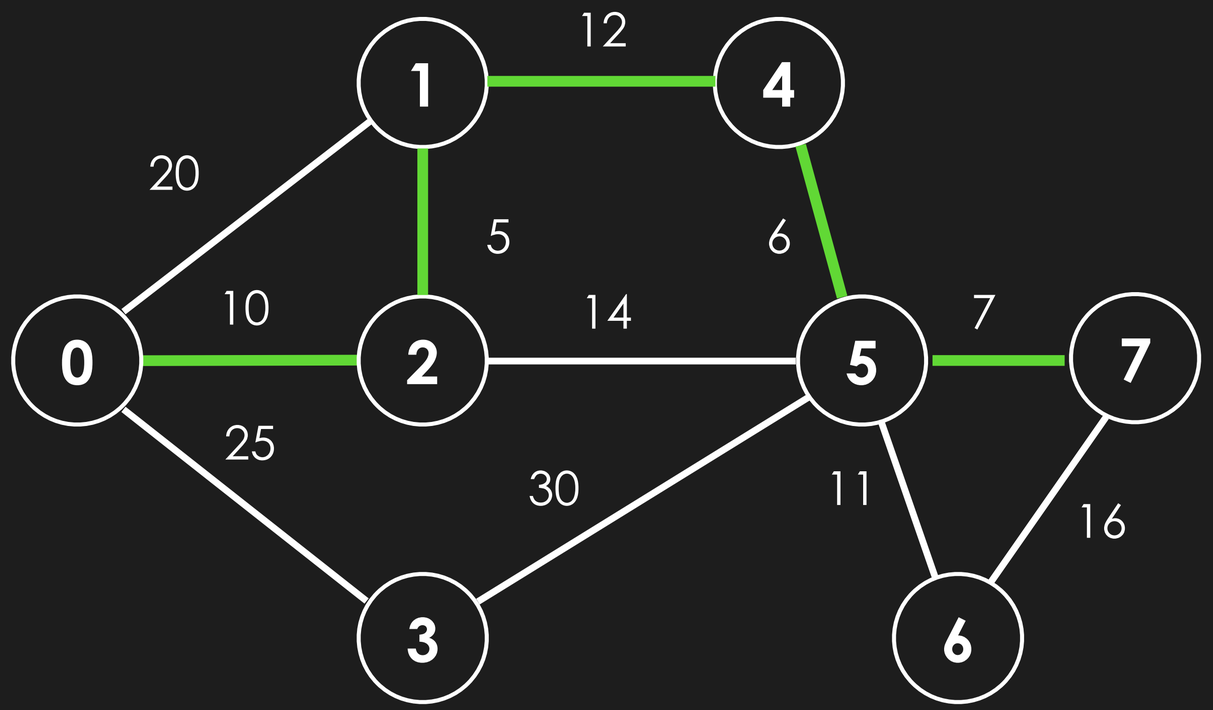
4 - 5

5 - 7

1. Find a **cut** which contains no **green edge** (green indicates edge has been added to MST)

2. **Add** the min weight cut edge to the MST (mark edge green)

3. **Repeat** until V - 1 edges are added to the MST



MST

0 - 2

1 - 2

1 - 4

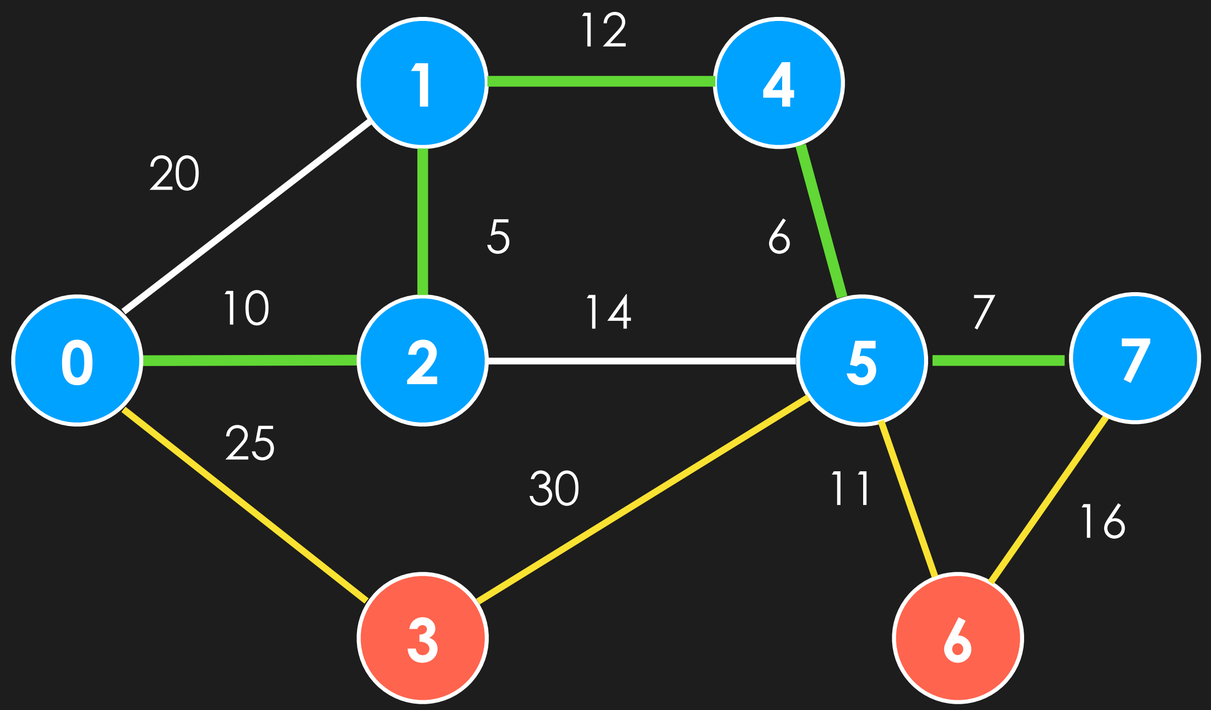
4 - 5

5 - 7

1. Find a **cut** which contains no **green edge** (green indicates edge has been added to MST)

2. **Add** the min weight cut edge to the MST (mark edge green)

3. **Repeat** until V - 1 edges are added to the MST



MST

0 - 2

1 - 2

1 - 4

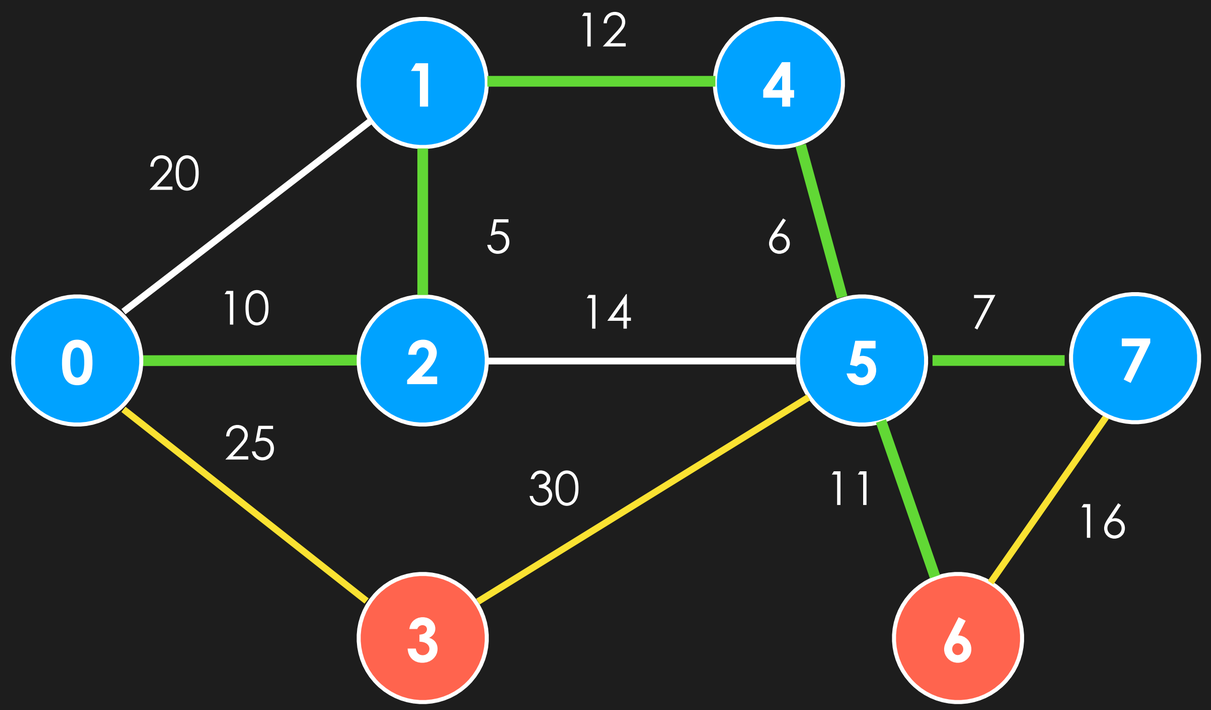
4 - 5

5 - 7

1. Find a **cut** which contains no **green edge** (green indicates edge has been added to MST)

2. **Add** the min weight cut edge to the MST (mark edge green)

3. **Repeat** until V - 1 edges are added to the MST



MST

0 - 2

1 - 2

1 - 4

4 - 5

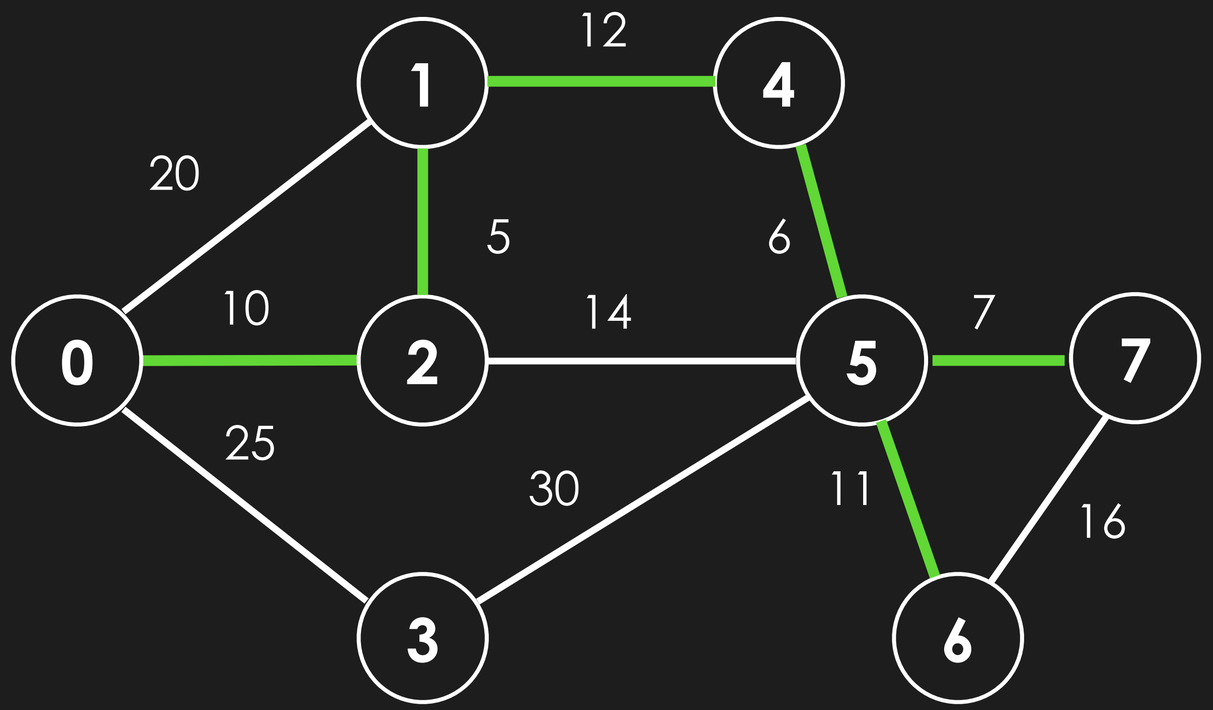
5 - 7

6 - 5

1. Find a **cut** which contains no **green edge** (green indicates edge has been added to MST)

2. **Add** the min weight cut edge to the MST (mark edge green)

3. **Repeat** until V - 1 edges are added to the MST



MST

0 - 2

1 - 2

1 - 4

4 - 5

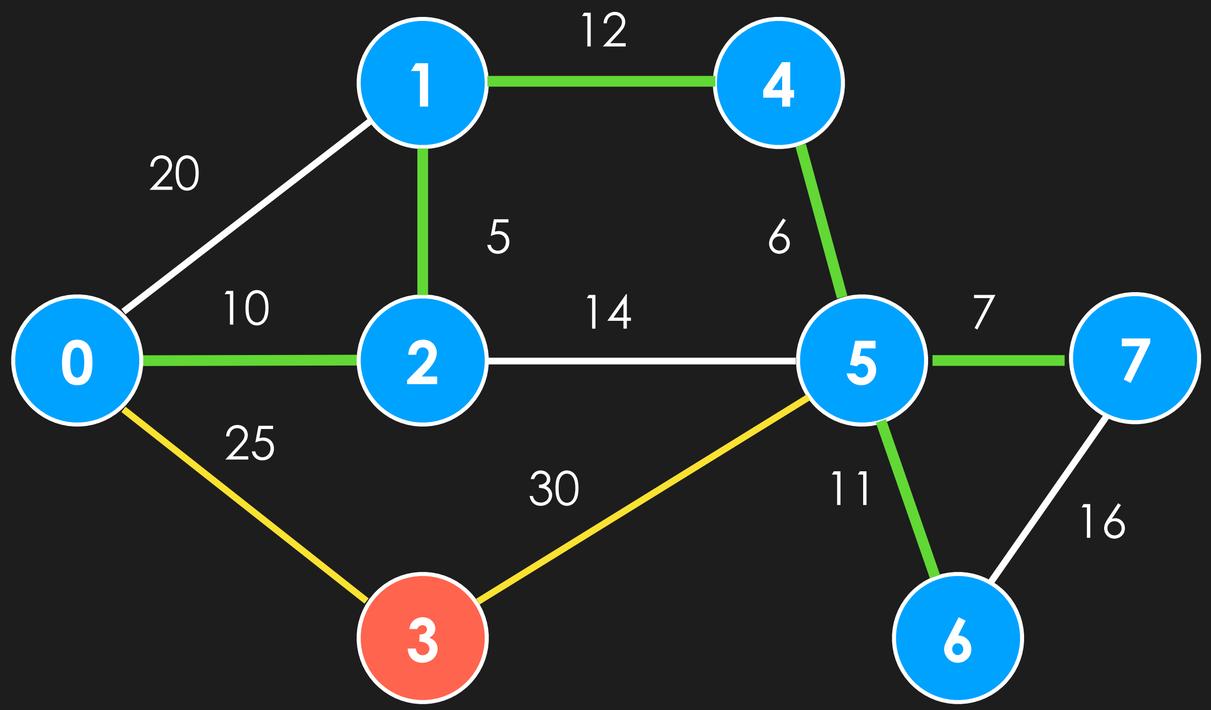
5 - 7

6 - 5

1. Find a **cut** which contains no **green edge** (green indicates edge has been added to MST)

2. **Add** the min weight cut edge to the MST (mark edge green)

3. **Repeat** until V - 1 edges are added to the MST



MST

0 - 2

1 - 2

1 - 4

4 - 5

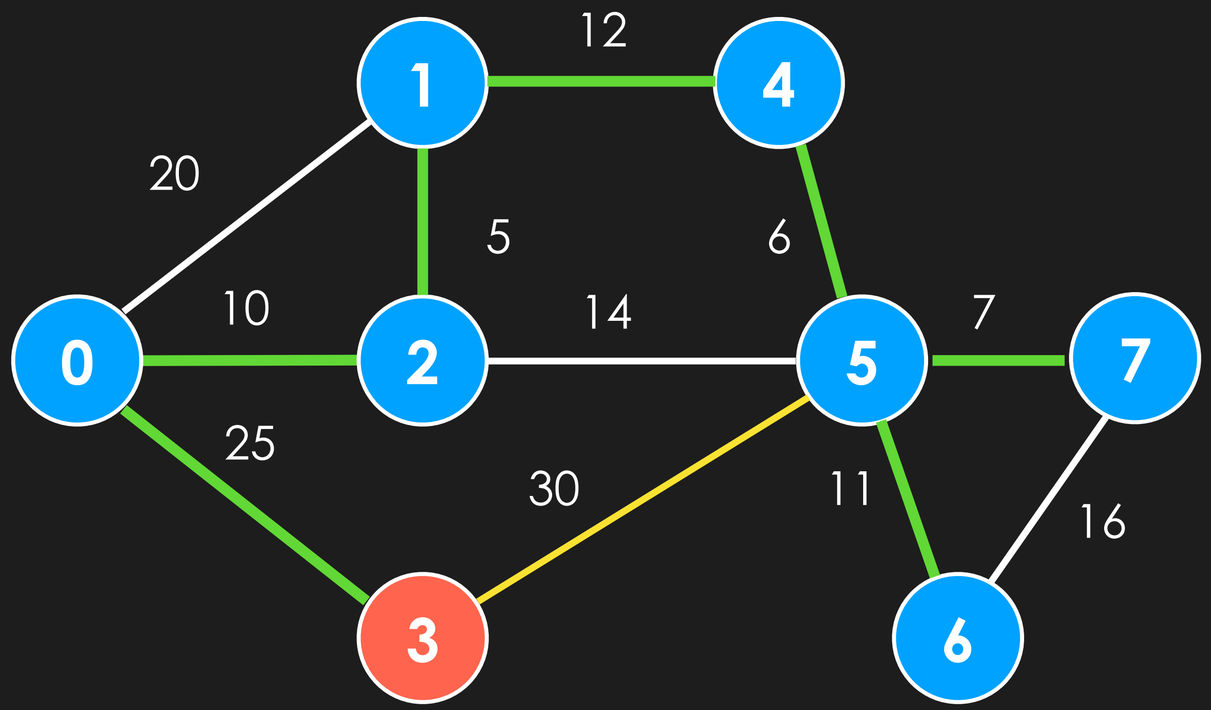
5 - 7

6 - 5

1. Find a **cut** which contains no **green edge** (green indicates edge has been added to MST)

2. **Add** the min weight cut edge to the MST (mark edge green)

3. **Repeat** until V - 1 edges are added to the MST



MST

0 - 2

1 - 2

1 - 4

4 - 5

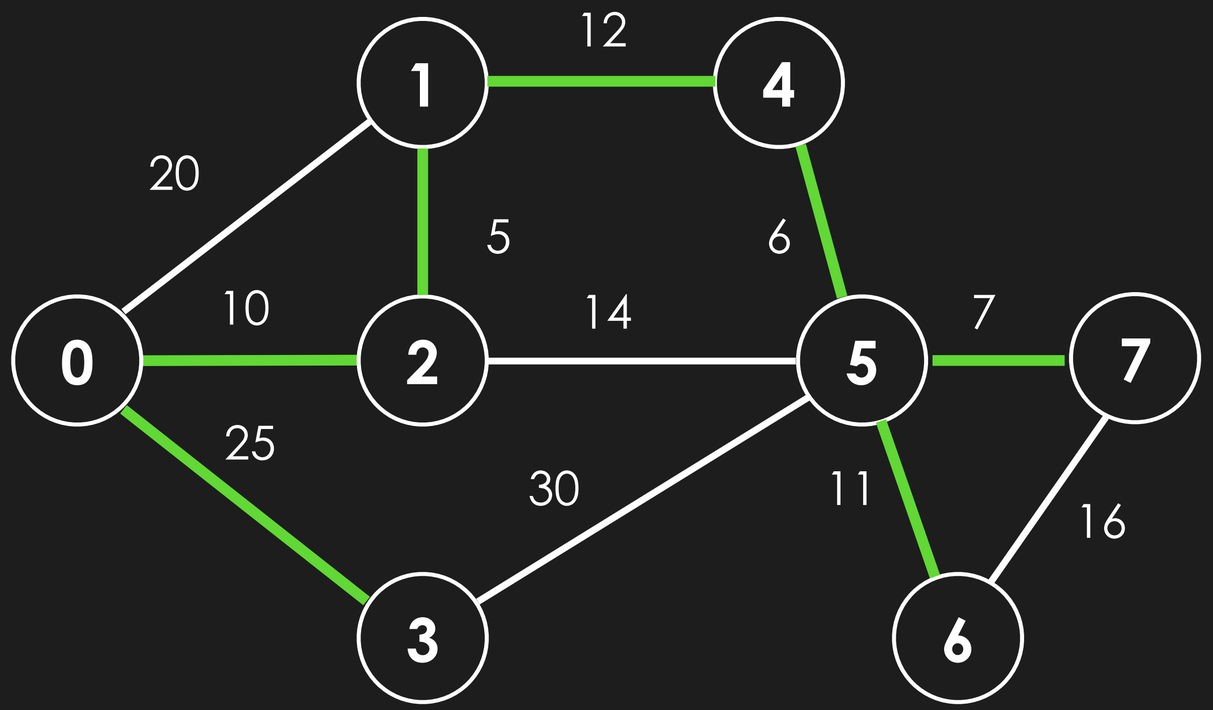
5 - 7

6 - 5

3 - 0

This is called a **greedy algorithm** because:

At each step, we are selecting the **best option** (min weight cut edge), aka **local optimum**, in hopes that it leads to the best overall solution (MST), aka **global optimum**



MST

0 - 2

1 - 2

1 - 4

4 - 5

5 - 7

6 - 5

3 - 0

How do we implement this algorithm?

How do we implement this algorithm?

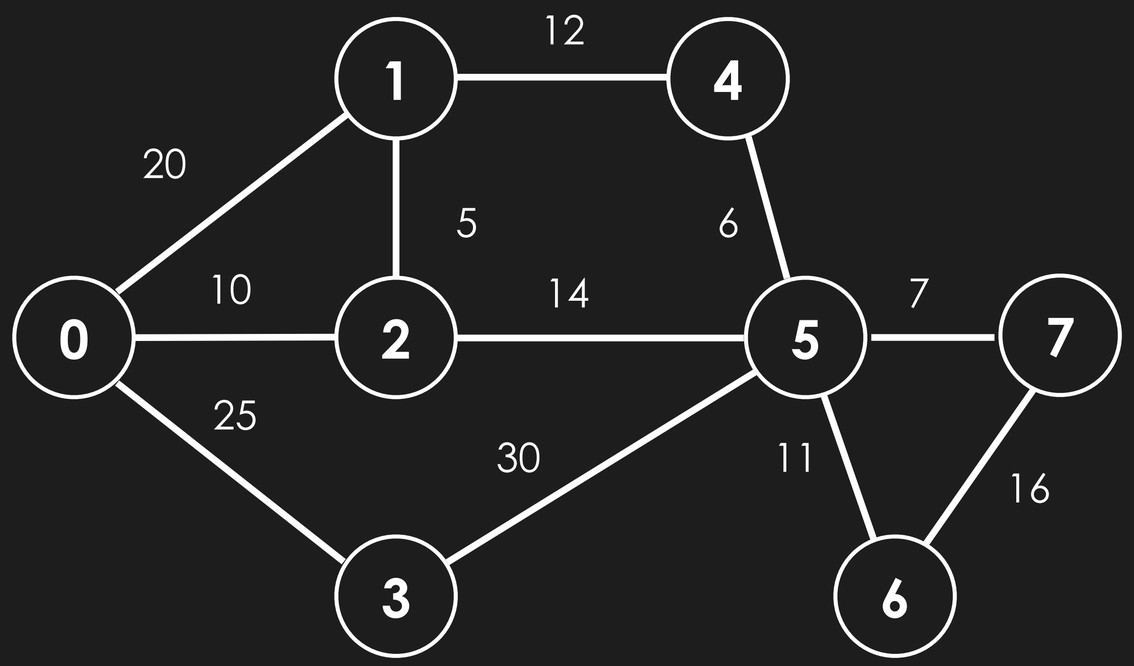
The problem comes in **finding a cut** in the graph. This is increasingly **problematic** as more edges in the MST are found (marked green)

There are two classic algorithms that solve this problem:

1. **Prim's** algorithm (To be learnt)
2. **Kruskal's** algorithm

Prim's Algorithm

1. Start at **vertex 0** (or any random vertex), and add all adjacent edges to **pq**



pq

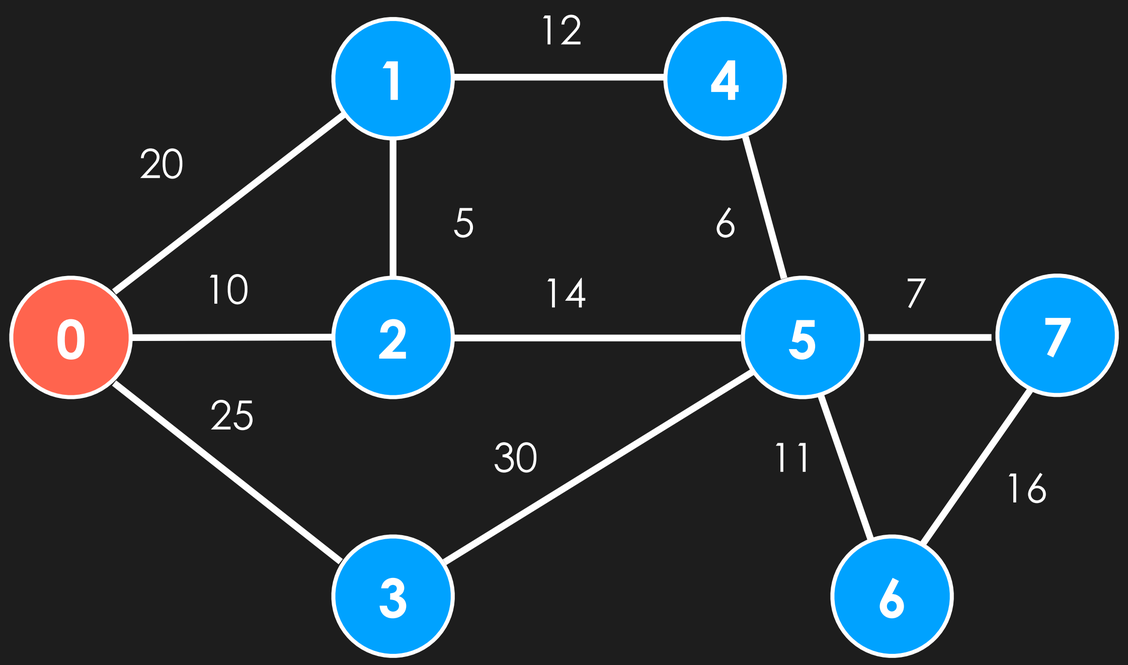
edge	weight

mstEdges

inMST

0	FALSE
1	FALSE
2	FALSE
3	FALSE
4	FALSE
5	FALSE
6	FALSE
7	FALSE

1. Start at **vertex 0** (or any random vertex), and add all adjacent edges to **pq**



pq

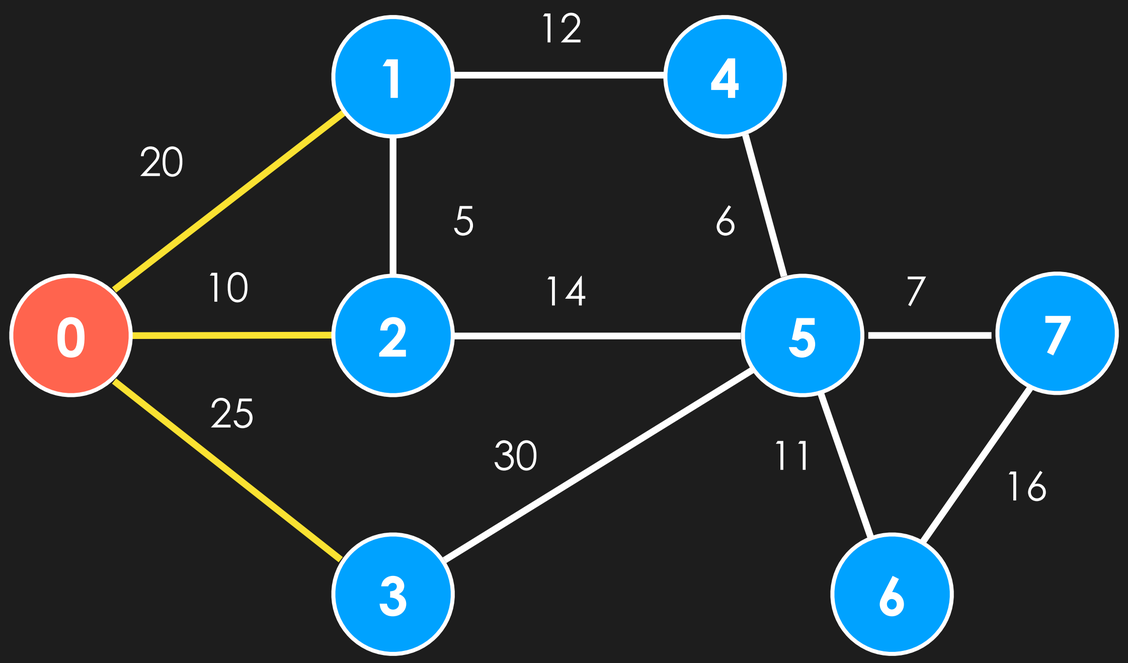
edge	weight

mstEdges

inMST

0	TRUE
1	FALSE
2	FALSE
3	FALSE
4	FALSE
5	FALSE
6	FALSE
7	FALSE

1. Start at **vertex 0** (or any random vertex), and add all adjacent edges to **pq**



pq

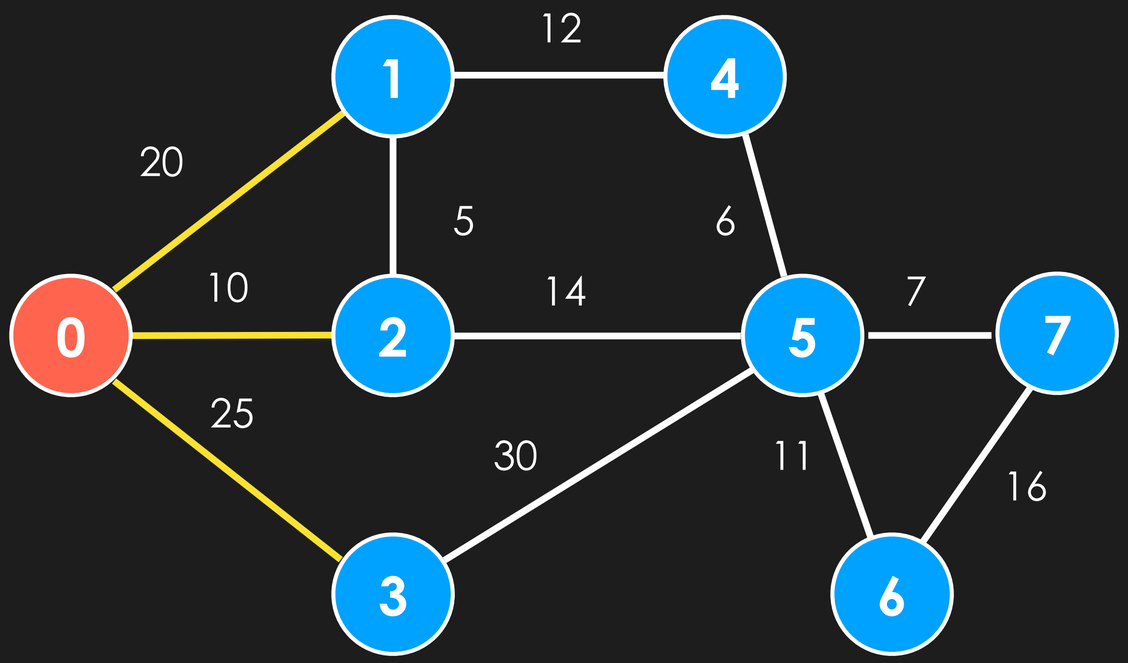
edge	weight
0 - 2	10
0 - 1	20
0 - 3	25

mstEdges

inMST

0	TRUE
1	FALSE
2	FALSE
3	FALSE
4	FALSE
5	FALSE
6	FALSE
7	FALSE

1. Start at **vertex 0** (or any random vertex), and add all adjacent edges to **pq**
2. Get **min edge** from pq and add to **mstEdges** if dest is not **inMST** **(means edge has not yet been added)**



pq

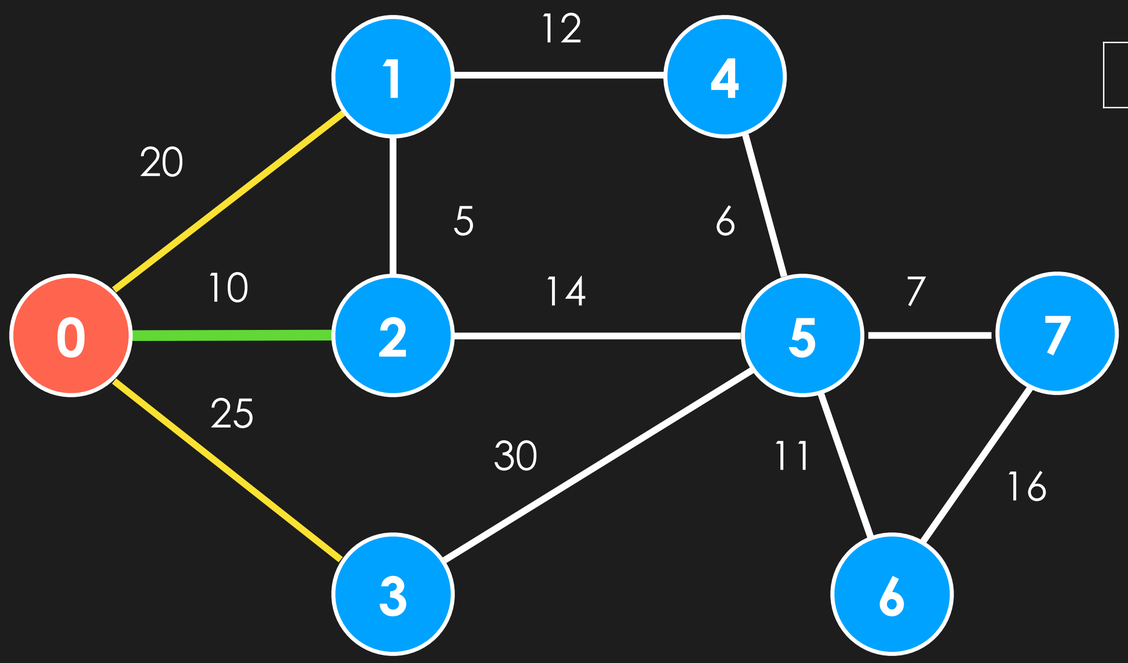
edge	weight
0 - 2	10
0 - 1	20
0 - 3	25

mstEdges

inMST

0	TRUE
1	FALSE
2	FALSE
3	FALSE
4	FALSE
5	FALSE
6	FALSE
7	FALSE

1. Start at **vertex 0** (or any random vertex), and add all adjacent edges to **pq**
2. Get **min edge** from pq and add to **mstEdges** if dest is not **inMST** **(means edge has not yet been added)**



0 - 2	10

pq

edge	weight
0 - 1	20
0 - 3	25

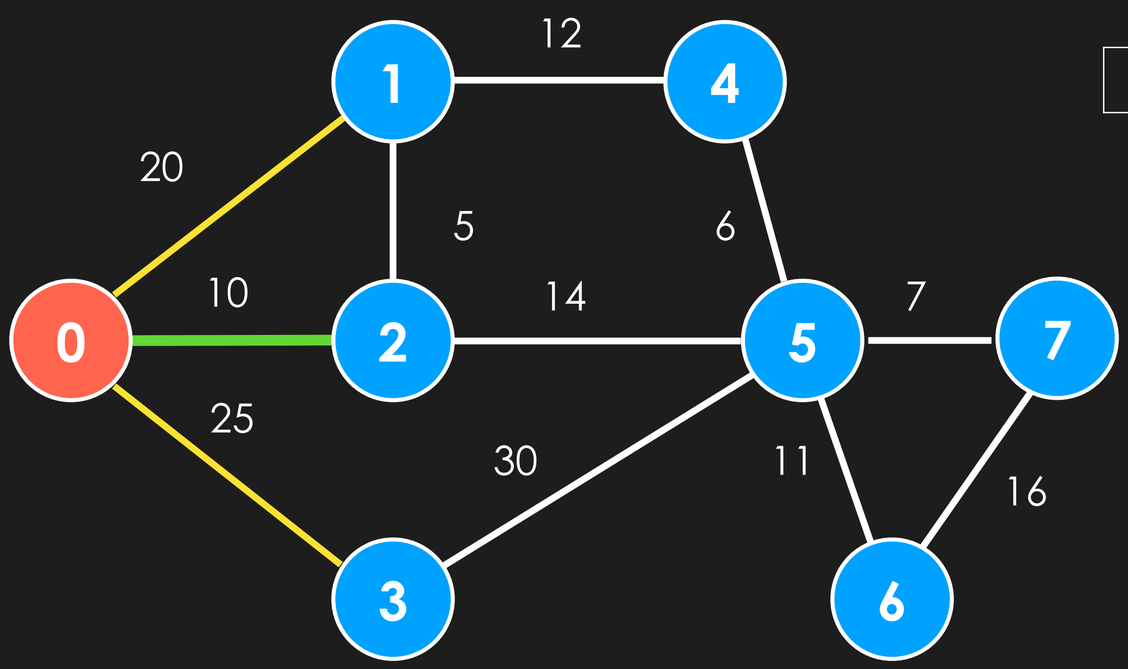
mstEdges

0 - 2

inMST

0	TRUE
1	FALSE
2	FALSE
3	FALSE
4	FALSE
5	FALSE
6	FALSE
7	FALSE

1. Start at **vertex 0** (or any random vertex), and add all adjacent edges to **pq**
2. Get **min edge** from pq and add to **mstEdges** if dest is not **inMST** **(means edge has not yet been added)**
3. For vertices in min edge, add to **inMST** and add adjacent edges to **pq if not already done so**



0 - 2	10

pq

edge	weight
0 - 1	20
0 - 3	25

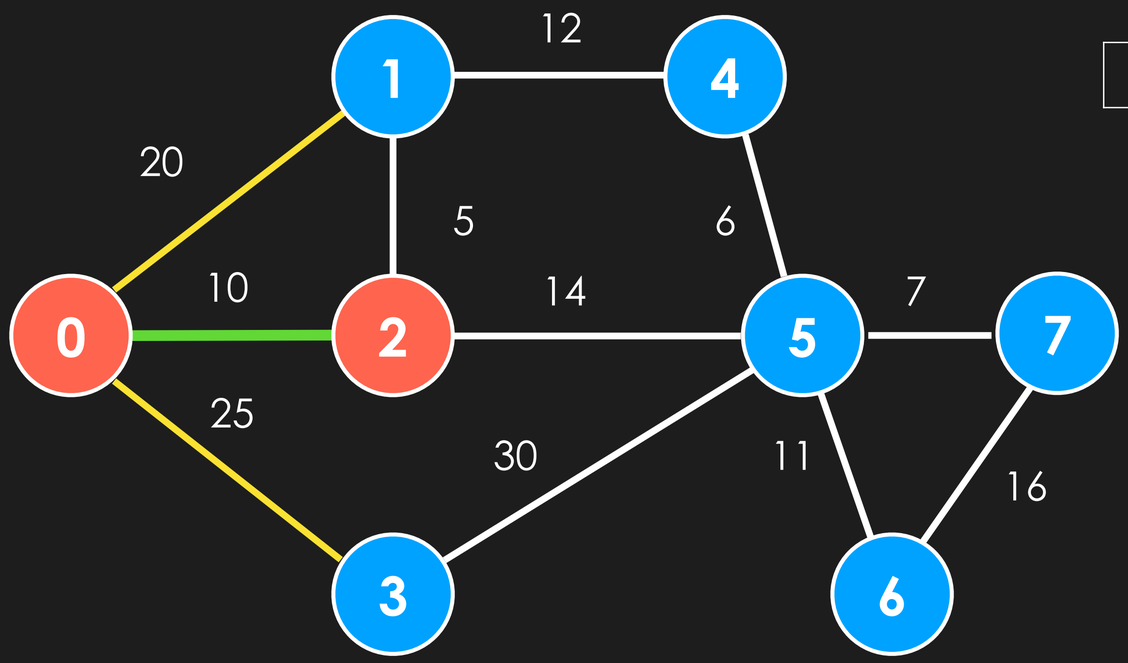
mstEdges

0 - 2

inMST

0	TRUE
1	FALSE
2	FALSE
3	FALSE
4	FALSE
5	FALSE
6	FALSE
7	FALSE

1. Start at **vertex 0** (or any random vertex), and add all adjacent edges to **pq**
2. Get **min edge** from pq and add to **mstEdges** if dest is not **inMST** **(means edge has not yet been added)**
3. For vertices in min edge, add to **inMST** and add adjacent edges to **pq if not already done so**



0 - 2	10

pq

edge	weight
0 - 1	20
0 - 3	25

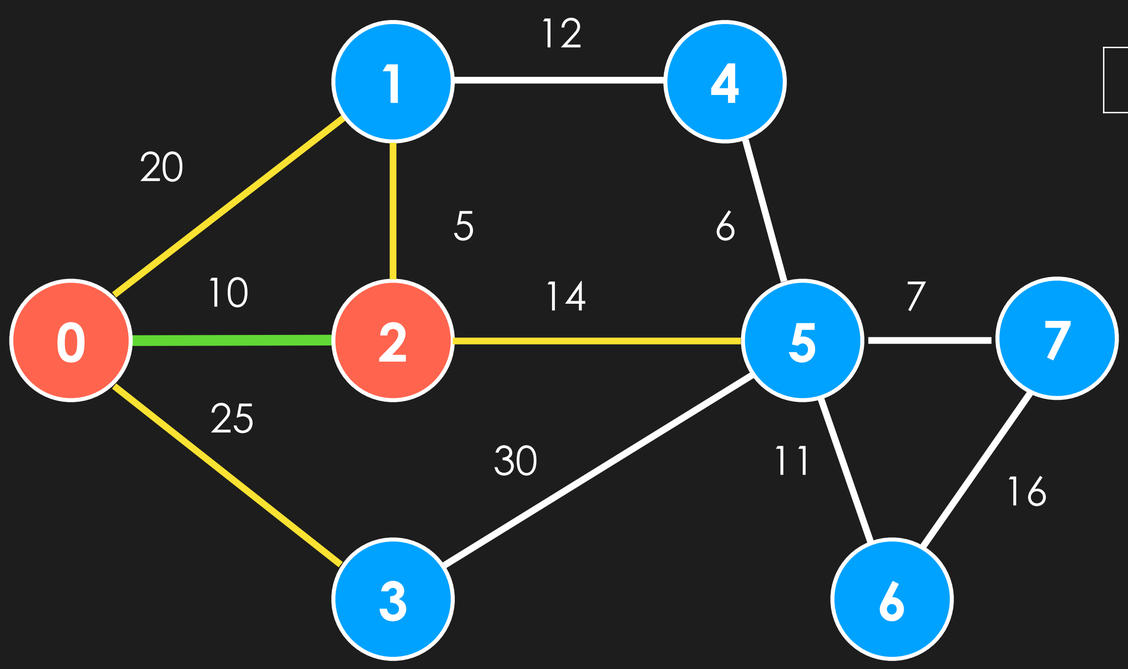
mstEdges

0 - 2

inMST

0	**TRUE**
1	FALSE
2	**TRUE**
3	FALSE
4	FALSE
5	FALSE
6	FALSE
7	FALSE

1. Start at **vertex 0** (or any random vertex), and add all adjacent edges to **pq**
2. Get **min edge** from pq and add to **mstEdges** if dest is not **inMST** **(means edge has not yet been added)**
3. For vertices in min edge, add to **inMST** and add adjacent edges to **pq if not already done so**



0 - 2	10

pq

edge	weight
2 - 1	**5**
2 - 5	**14**
0 - 1	20
0 - 3	25

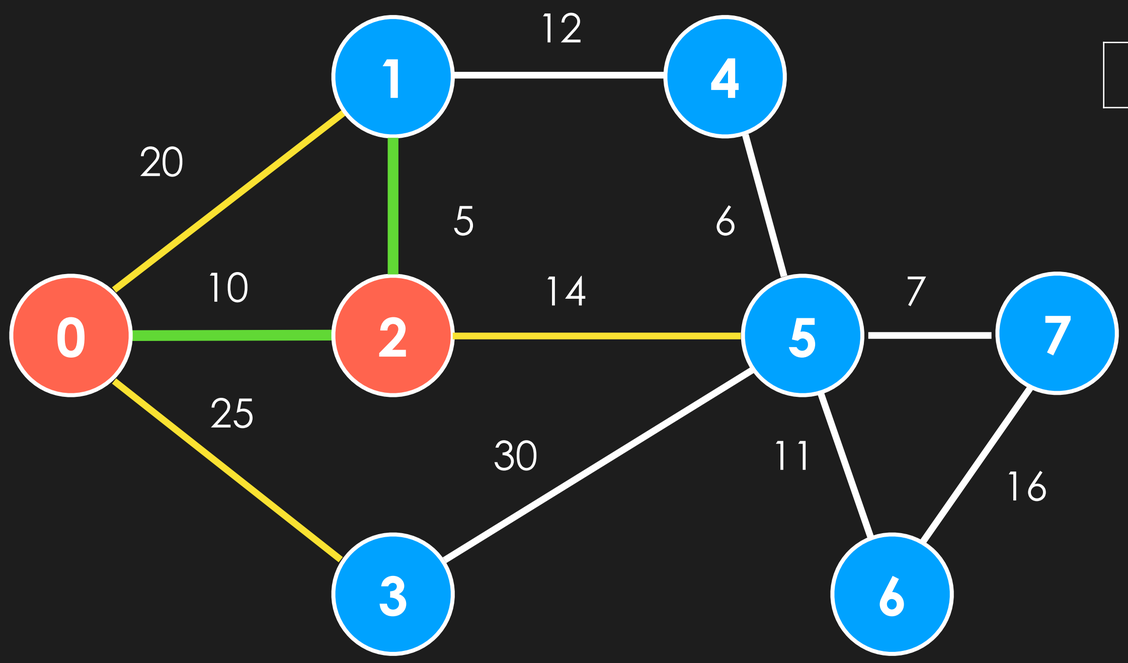
mstEdges

0 - 2

inMST

0	**TRUE**
1	FALSE
2	**TRUE**
3	FALSE
4	FALSE
5	FALSE
6	FALSE
7	FALSE

1. Start at **vertex 0** (or any random vertex), and add all adjacent edges to **pq**
2. Get **min edge** from pq and add to **mstEdges** if dest is not **inMST** **(means edge has not yet been added)**
3. For vertices in min edge, add to **inMST** and add adjacent edges to **pq if not already done so**



2 - 1	5

pq

edge	weight
2 - 5	14
0 - 1	20
0 - 3	25

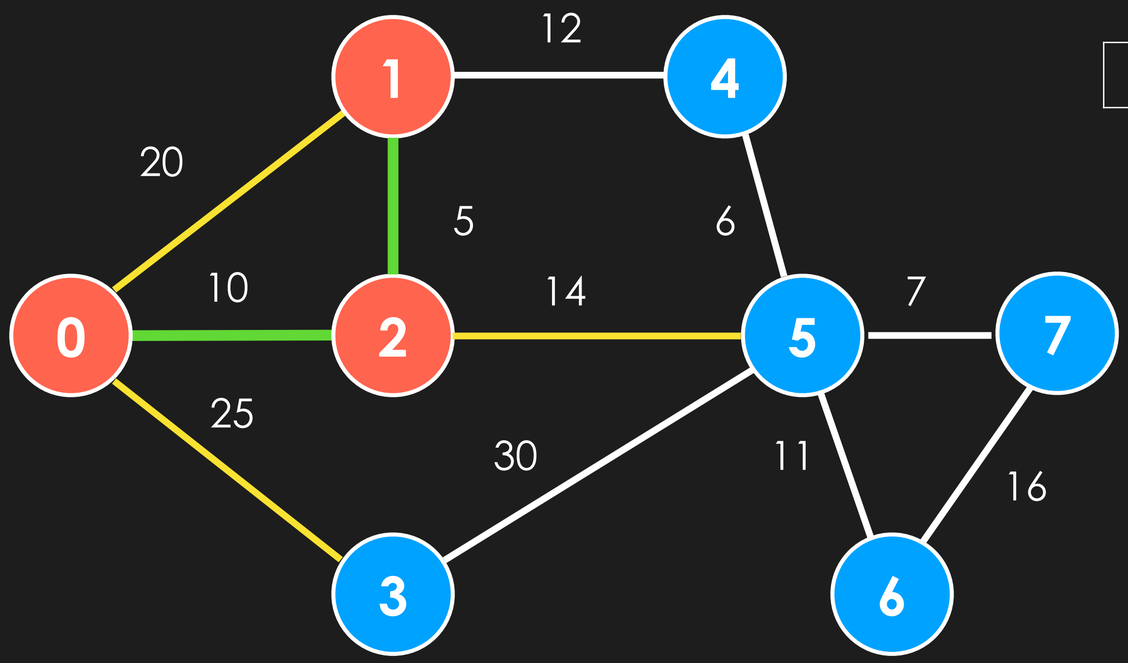
mstEdges

0 - 2
2 - 1

inMST

0	TRUE
1	FALSE
2	TRUE
3	FALSE
4	FALSE
5	FALSE
6	FALSE
7	FALSE

1. Start at **vertex 0** (or any random vertex), and add all adjacent edges to **pq**
2. Get **min edge** from pq and add to **mstEdges** if dest is not **inMST** **(means edge has not yet been added)**
3. For vertices in min edge, add to **inMST** and add adjacent edges to **pq if not already done so**



2 - 1	5

pq

edge	weight
2 - 5	14
0 - 1	20
0 - 3	25

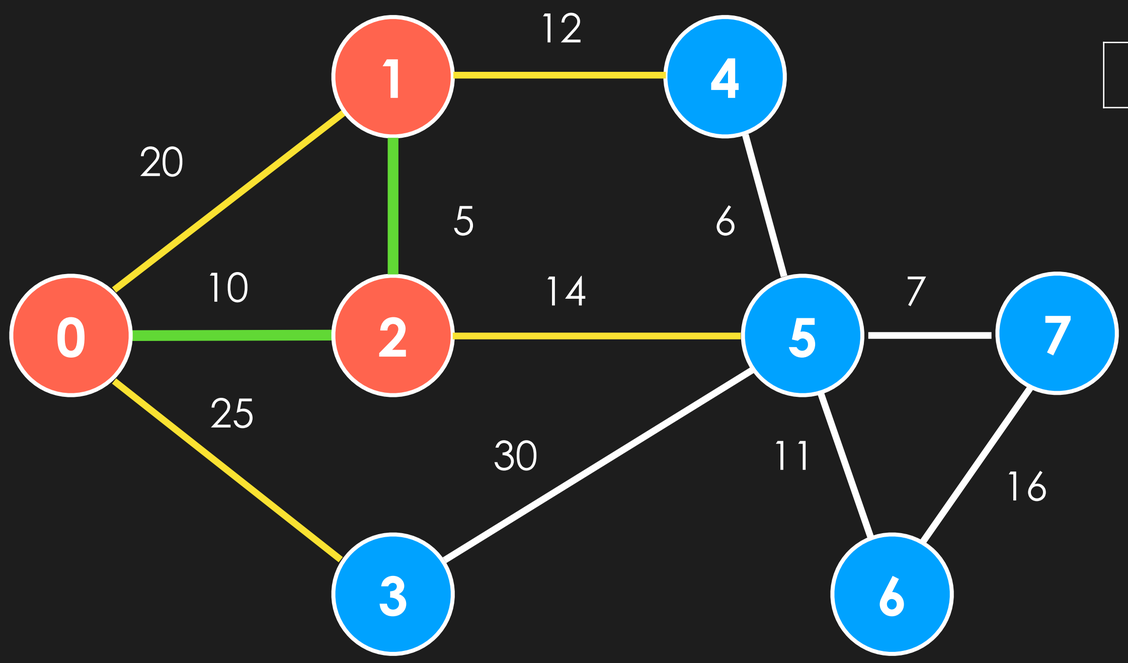
mstEdges

0 - 2
2 - 1

inMST

0	TRUE
1	TRUE
2	TRUE
3	FALSE
4	FALSE
5	FALSE
6	FALSE
7	FALSE

1. Start at **vertex 0** (or any random vertex), and add all adjacent edges to **pq**
2. Get **min edge** from pq and add to **mstEdges** if dest is not **inMST (means edge has not yet been added)**
3. For vertices in min edge, add to **inMST** and add adjacent edges to **pq if not already done so**



2 - 1	5

pq

edge	weight
1 - 4	**12**
2 - 5	14
0 - 1	20
0 - 3	25

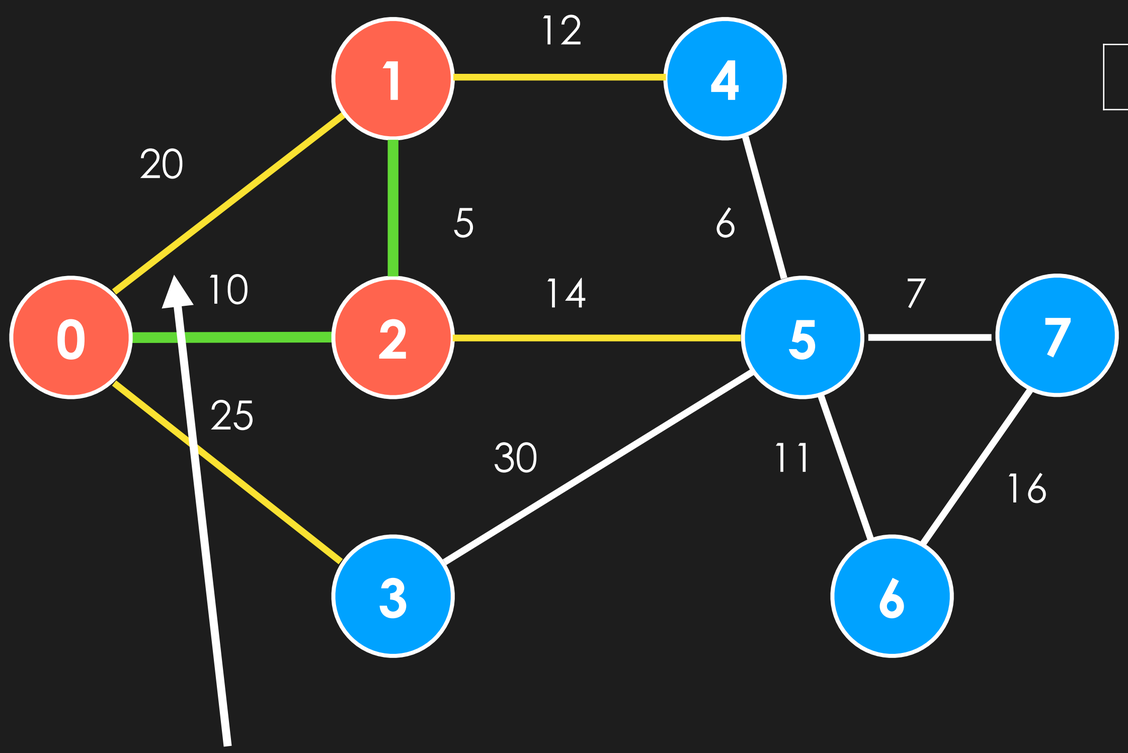
mstEdges

0 - 2
2 - 1

inMST

0	**TRUE**
1	**TRUE**
2	**TRUE**
3	FALSE
4	FALSE
5	FALSE
6	FALSE
7	FALSE

1. Start at **vertex 0** (or any random vertex), and add all adjacent edges to **pq**
2. Get **min edge** from pq and add to **mstEdges** if dest is not **inMST** **(means edge has not yet been added)**
3. For vertices in min edge, add to **inMST** and add adjacent edges to **pq if not already done so**



2 - 1	5

pq

edge	weight
1 - 4	**12**
2 - 5	14
0 - 1	20
0 - 3	25

mstEdges

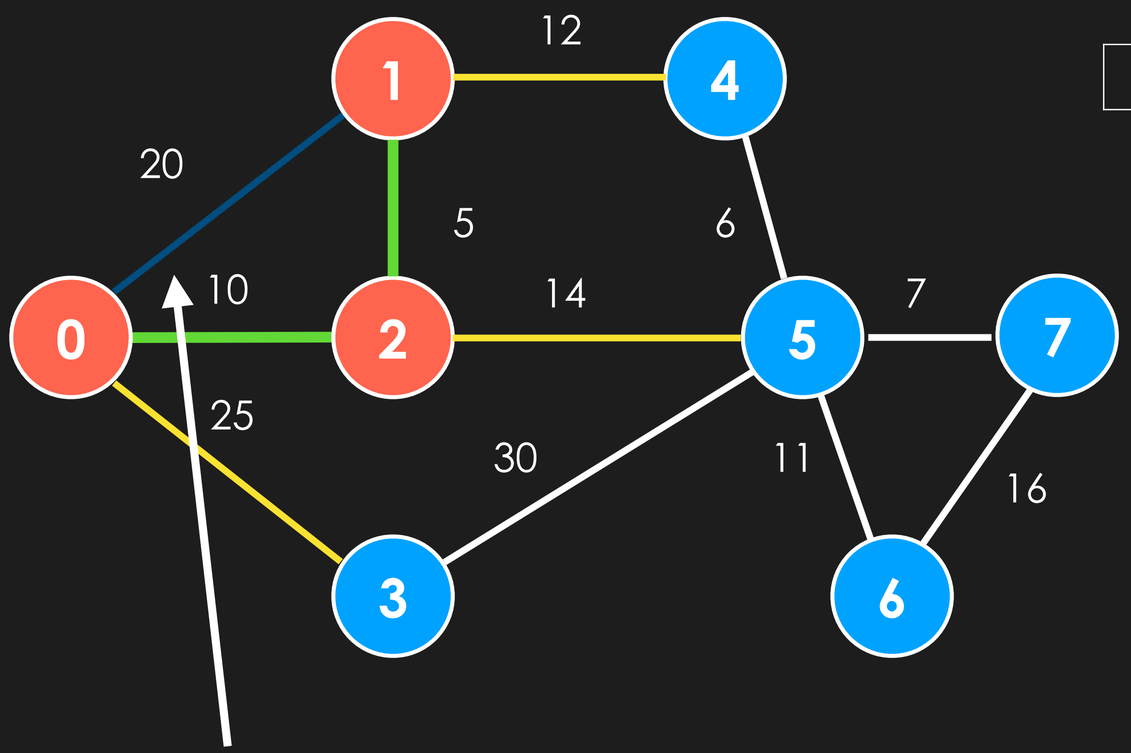
0 - 2
2 - 1

inMST

0	**TRUE**
1	**TRUE**
2	**TRUE**
3	FALSE
4	FALSE
5	FALSE
6	FALSE
7	FALSE

Note: As edges are added to the pq, some edges become **obsolete**!

1. Start at **vertex 0** (or any random vertex), and add all adjacent edges to **pq**
2. Get **min edge** from pq and add to **mstEdges** if dest is not **inMST** **(means edge has not yet been added)**
3. For vertices in min edge, add to **inMST** and add adjacent edges to **pq if not already done so**



2 - 1	5

pq

edge	weight
1 - 4	**12**
2 - 5	14
0 - 1	20
0 - 3	25

mstEdges

0 - 2
2 - 1

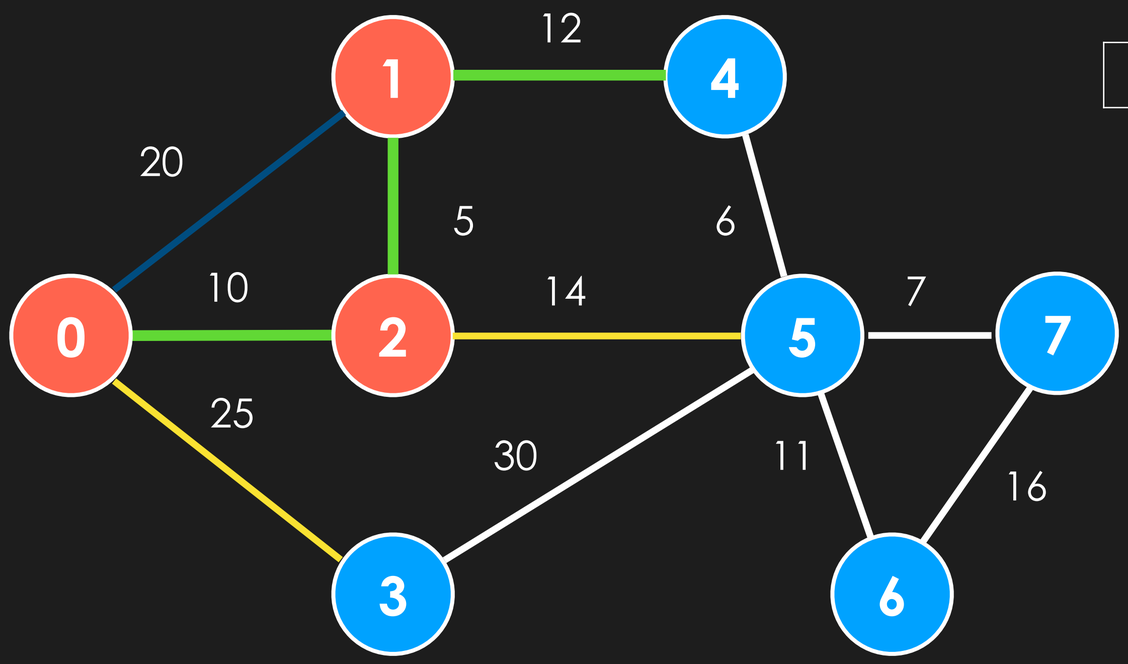
inMST

0	**TRUE**
1	**TRUE**
2	**TRUE**
3	FALSE
4	FALSE
5	FALSE
6	FALSE
7	FALSE

Note: As edges are added to the pq, some edges become **obsolete**!

We have found a better edge **2 - 1** already! We will mark obsolete edges as **dark blue as they are no longer relevant to cuts**

1. Start at **vertex 0** (or any random vertex), and add all adjacent edges to **pq**
2. Get **min edge** from pq and add to **mstEdges** if dest is not **inMST** **(means edge has not yet been added)**
3. For vertices in min edge, add to **inMST** and add adjacent edges to **pq if not already done so**



1 - 4	12

pq

edge	weight
2 - 5	14
0 - 1	20
0 - 3	25

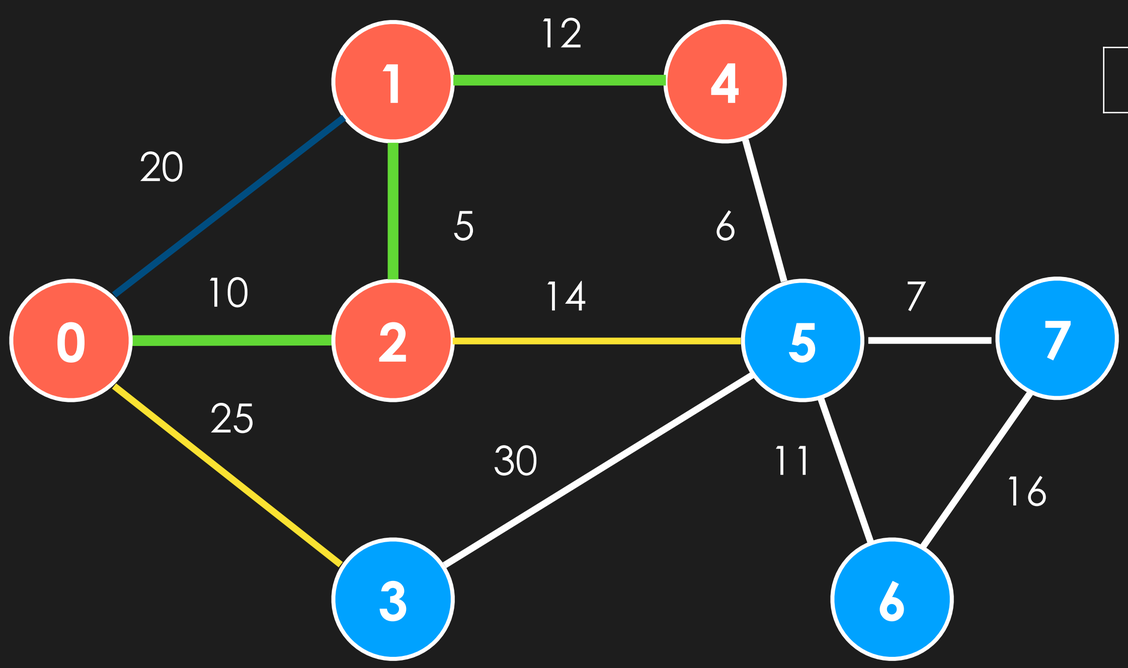
mstEdges

0 - 2
2 - 1
1 - 4

inMST

0	TRUE
1	TRUE
2	TRUE
3	FALSE
4	FALSE
5	FALSE
6	FALSE
7	FALSE

1. Start at **vertex 0** (or any random vertex), and add all adjacent edges to **pq**
2. Get **min edge** from pq and add to **mstEdges** if dest is not **inMST** **(means edge has not yet been added)**
3. For vertices in min edge, add to **inMST** and add adjacent edges to **pq if not already done so**



1 - 4	12

pq

edge	weight
2 - 5	14
0 - 1	20
0 - 3	25

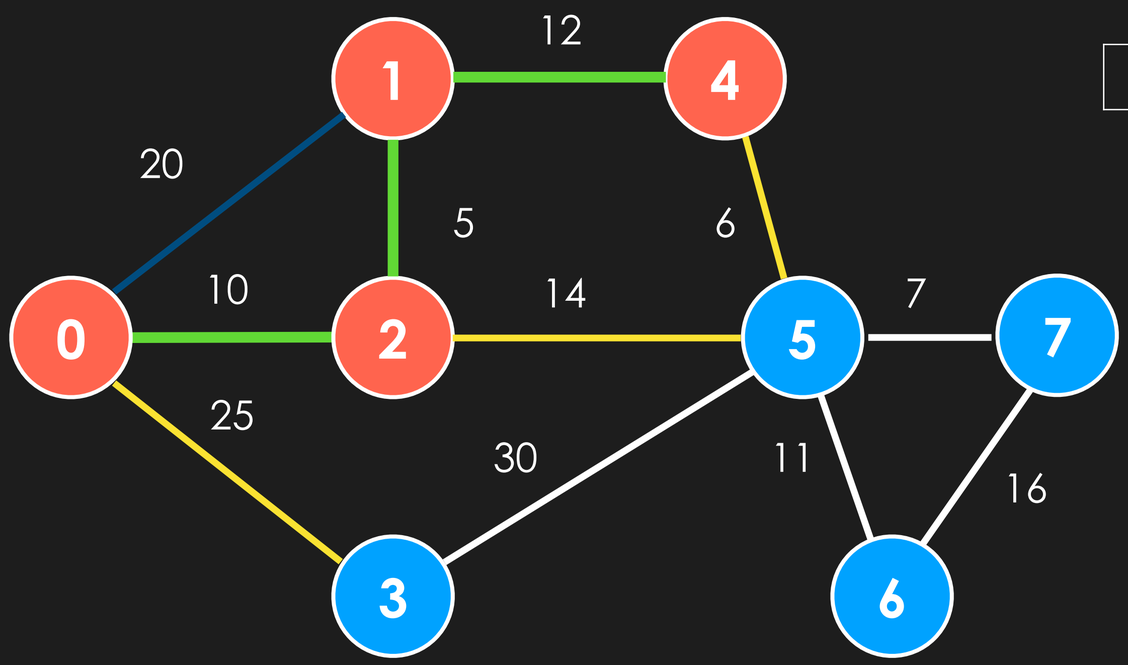
mstEdges

0 - 2
2 - 1
1 - 4

inMST

0	TRUE
1	TRUE
2	TRUE
3	FALSE
4	TRUE
5	FALSE
6	FALSE
7	FALSE

1. Start at **vertex 0** (or any random vertex), and add all adjacent edges to **pq**
2. Get **min edge** from pq and add to **mstEdges** if dest is not **inMST** **(means edge has not yet been added)**
3. For vertices in min edge, add to **inMST** and add adjacent edges to **pq if not already done so**



1 - 4	12

pq

edge	weight
4 - 5	**6**
2 - 5	14
0 - 1	20
0 - 3	25

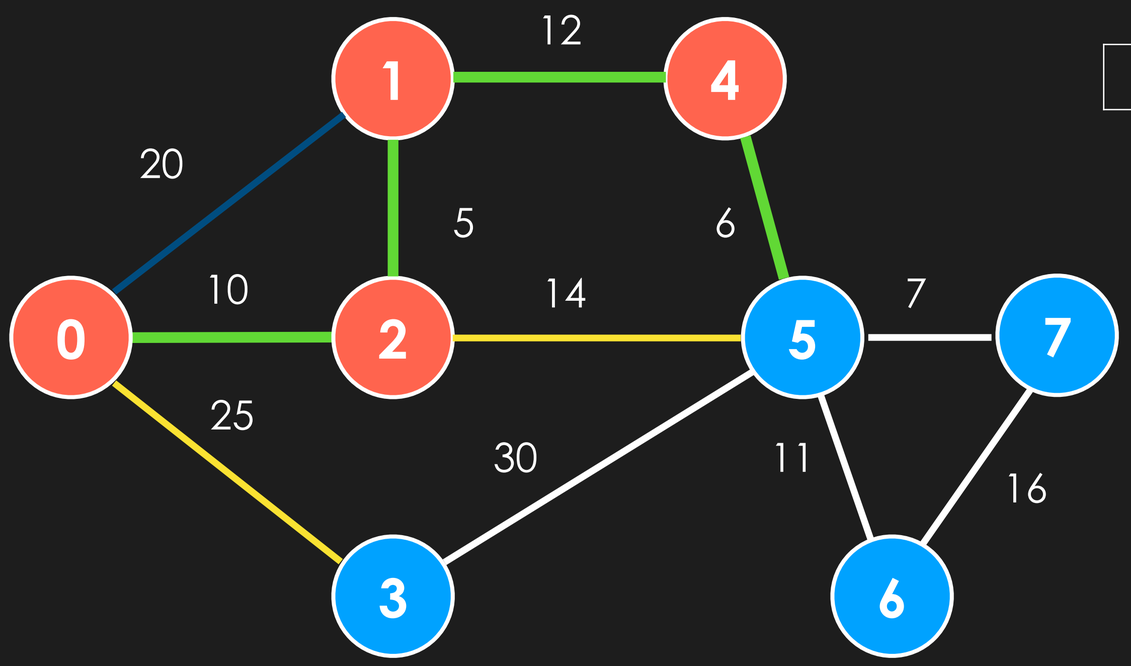
mstEdges

0 - 2
2 - 1
1 - 4

inMST

0	**TRUE**
1	**TRUE**
2	**TRUE**
3	FALSE
4	**TRUE**
5	FALSE
6	FALSE
7	FALSE

1. Start at **vertex 0** (or any random vertex), and add all adjacent edges to **pq**
2. Get **min edge** from pq and add to **mstEdges** if dest is not **inMST** **(means edge has not yet been added)**
3. For vertices in min edge, add to **inMST** and add adjacent edges to **pq if not already done so**



4 - 5	6

pq

edge	weight
2 - 5	14
0 - 1	20
0 - 3	25

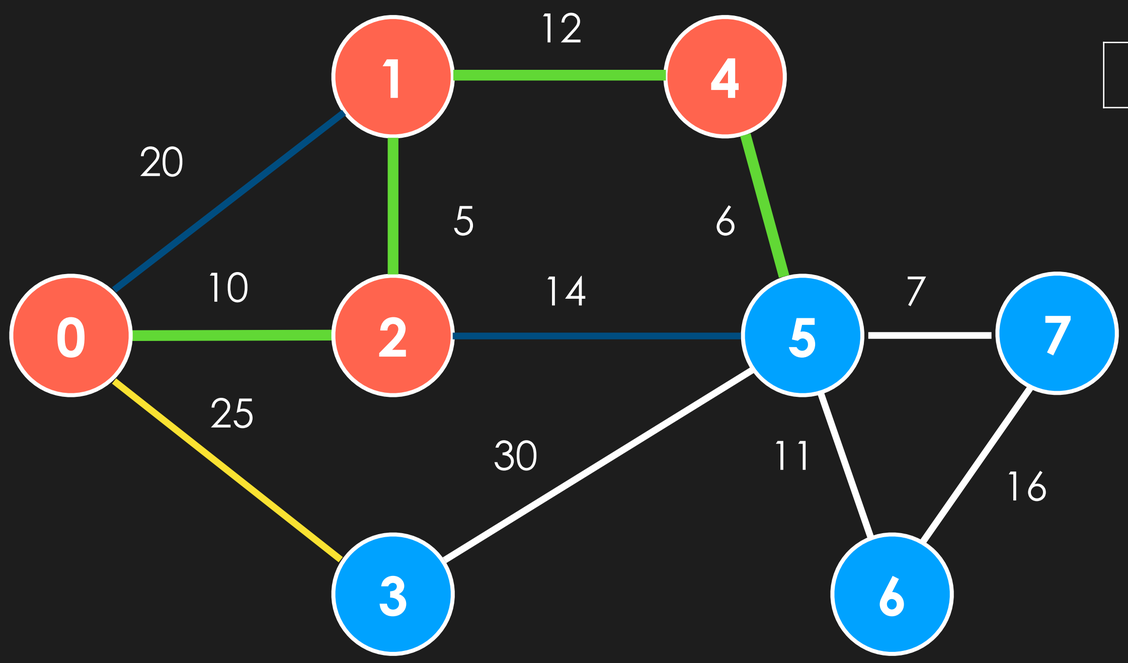
mstEdges

0 - 2
2 - 1
1 - 4
4 - 5

inMST

0	TRUE
1	TRUE
2	TRUE
3	FALSE
4	TRUE
5	FALSE
6	FALSE
7	FALSE

1. Start at **vertex 0** (or any random vertex), and add all adjacent edges to **pq**
2. Get **min edge** from pq and add to **mstEdges** if dest is not **inMST** **(means edge has not yet been added)**
3. For vertices in min edge, add to **inMST** and add adjacent edges to **pq if not already done so**



4 - 5	6

pq

edge	weight
2 - 5	14
0 - 1	20
0 - 3	25

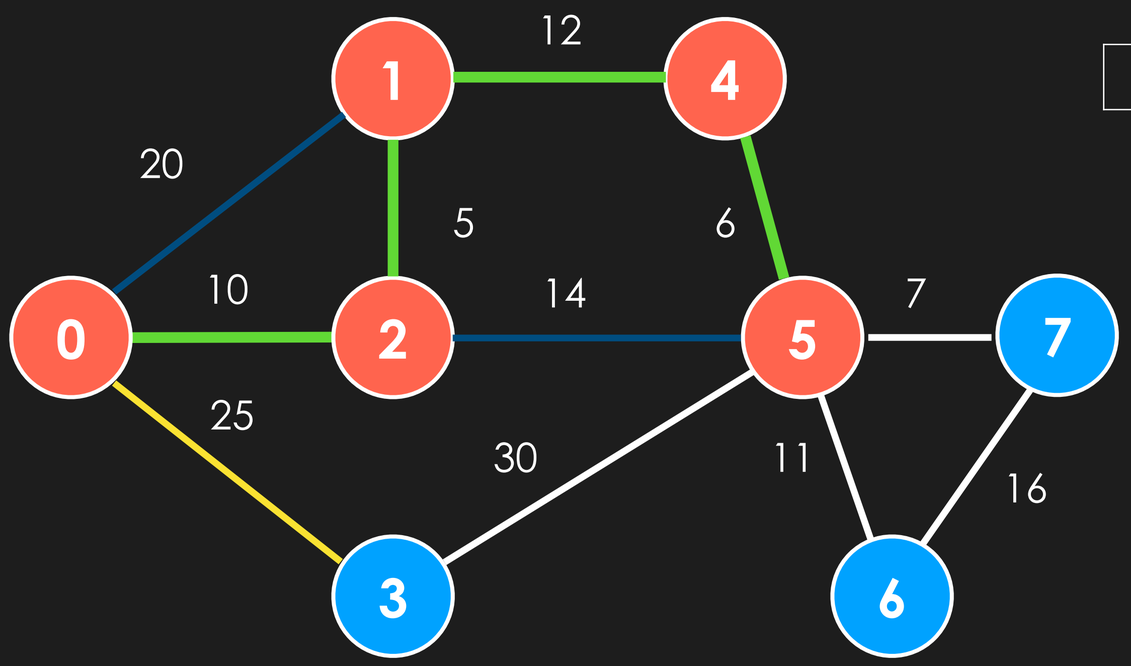
mstEdges

0 - 2
2 - 1
1 - 4
4 - 5

inMST

0	**TRUE**
1	**TRUE**
2	**TRUE**
3	FALSE
4	**TRUE**
5	FALSE
6	FALSE
7	FALSE

1. Start at **vertex 0** (or any random vertex), and add all adjacent edges to **pq**
2. Get **min edge** from pq and add to **mstEdges** if dest is not **inMST** **(means edge has not yet been added)**
3. For vertices in min edge, add to **inMST** and add adjacent edges to **pq if not already done so**



4 - 5	6

pq

edge	weight
2 - 5	14
0 - 1	20
0 - 3	25

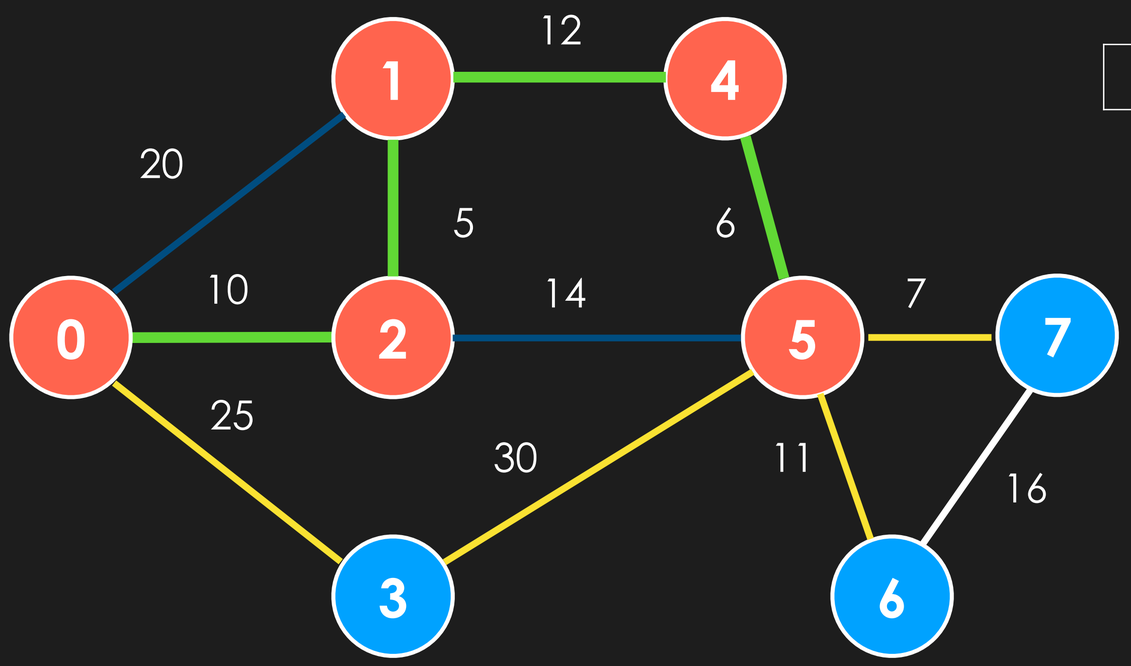
mstEdges

0 - 2
2 - 1
1 - 4
4 - 5

inMST

0	TRUE
1	TRUE
2	TRUE
3	FALSE
4	TRUE
5	TRUE
6	FALSE
7	FALSE

1. Start at **vertex 0** (or any random vertex), and add all adjacent edges to **pq**
2. Get **min edge** from pq and add to **mstEdges** if dest is not **inMST** **(means edge has not yet been added)**
3. For vertices in min edge, add to **inMST** and add adjacent edges to **pq if not already done so**



4 - 5	6

pq

edge	weight
5 - 7	**7**
5 - 6	**11**
2 - 5	14
0 - 1	20
0 - 3	25
5 - 3	**30**

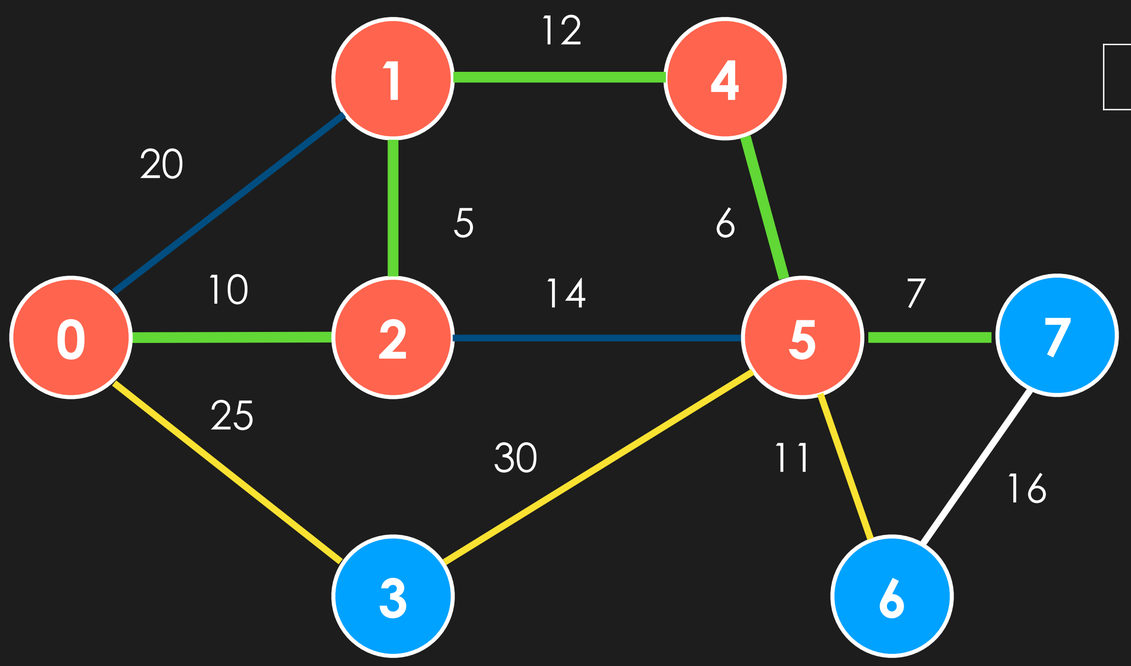
mstEdges

0 - 2
2 - 1
1 - 4
4 - 5

inMST

0	**TRUE**
1	**TRUE**
2	**TRUE**
3	FALSE
4	**TRUE**
5	**TRUE**
6	FALSE
7	FALSE

1. Start at **vertex 0** (or any random vertex), and add all adjacent edges to **pq**
2. Get **min edge** from pq and add to **mstEdges** if dest is not **inMST** **(means edge has not yet been added)**
3. For vertices in min edge, add to **inMST** and add adjacent edges to **pq if not already done so**



5 - 7	7

pq

edge	weight
5 - 6	11
2 - 5	14
0 - 1	20
0 - 3	25
5 - 3	30

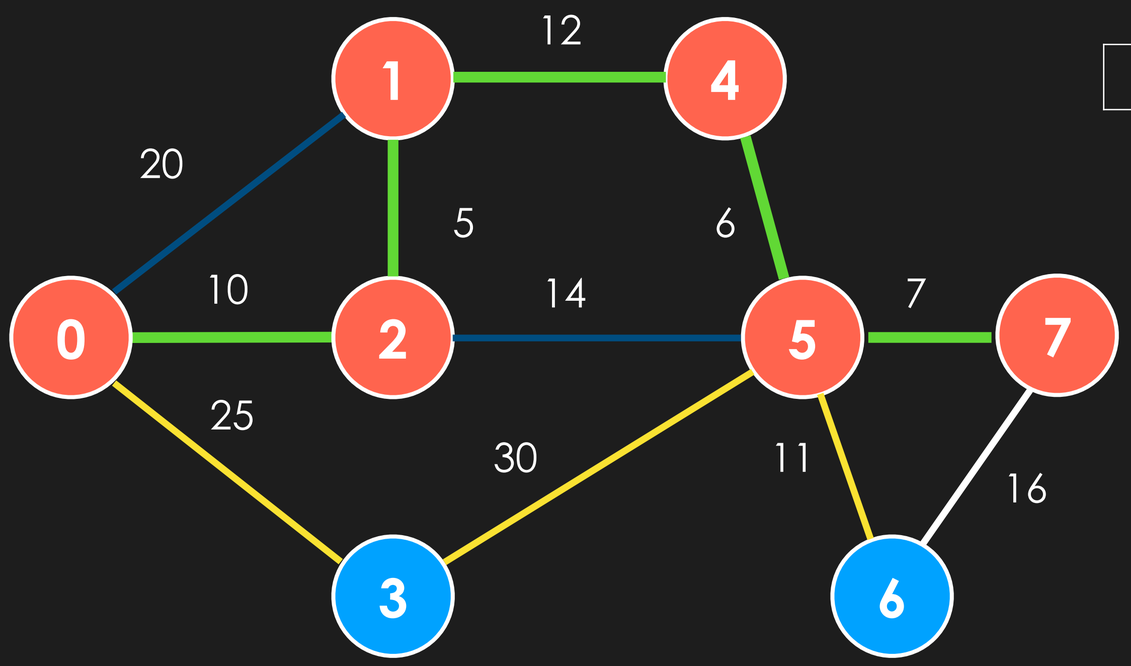
mstEdges

0 - 2
2 - 1
1 - 4
4 - 5
5 - 7

inMST

0	TRUE
1	TRUE
2	TRUE
3	FALSE
4	TRUE
5	TRUE
6	FALSE
7	FALSE

1. Start at **vertex 0** (or any random vertex), and add all adjacent edges to **pq**
2. Get **min edge** from pq and add to **mstEdges** if dest is not **inMST** **(means edge has not yet been added)**
3. For vertices in min edge, add to **inMST** and add adjacent edges to **pq if not already done so**



5 - 7	7

pq

edge	weight
5 - 6	11
2 - 5	14
0 - 1	20
0 - 3	25
5 - 3	30

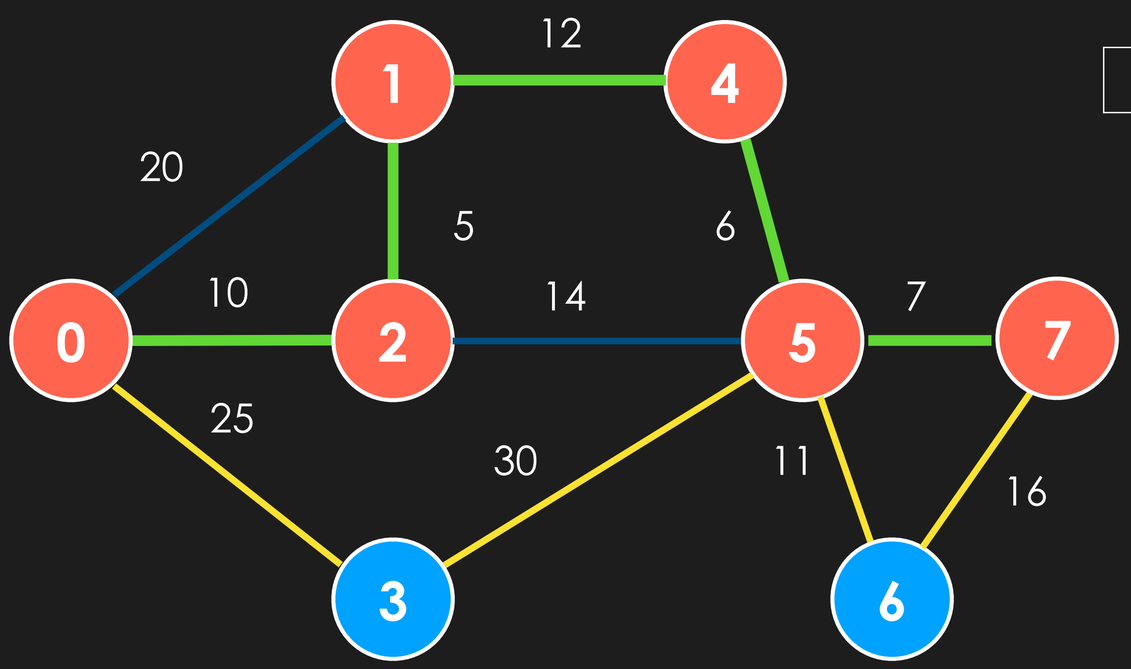
mstEdges

0 - 2
2 - 1
1 - 4
4 - 5
5 - 7

inMST

0	TRUE
1	TRUE
2	TRUE
3	FALSE
4	TRUE
5	TRUE
6	FALSE
7	TRUE

1. Start at **vertex 0** (or any random vertex), and add all adjacent edges to **pq**
2. Get **min edge** from pq and add to **mstEdges** if dest is not **inMST** **(means edge has not yet been added)**
3. For vertices in min edge, add to **inMST** and add adjacent edges to **pq if not already done so**



5 - 7	7

pq

edge	weight
5 - 6	11
2 - 5	14
7 - 6	**16**
0 - 1	20
0 - 3	25
5 - 3	30

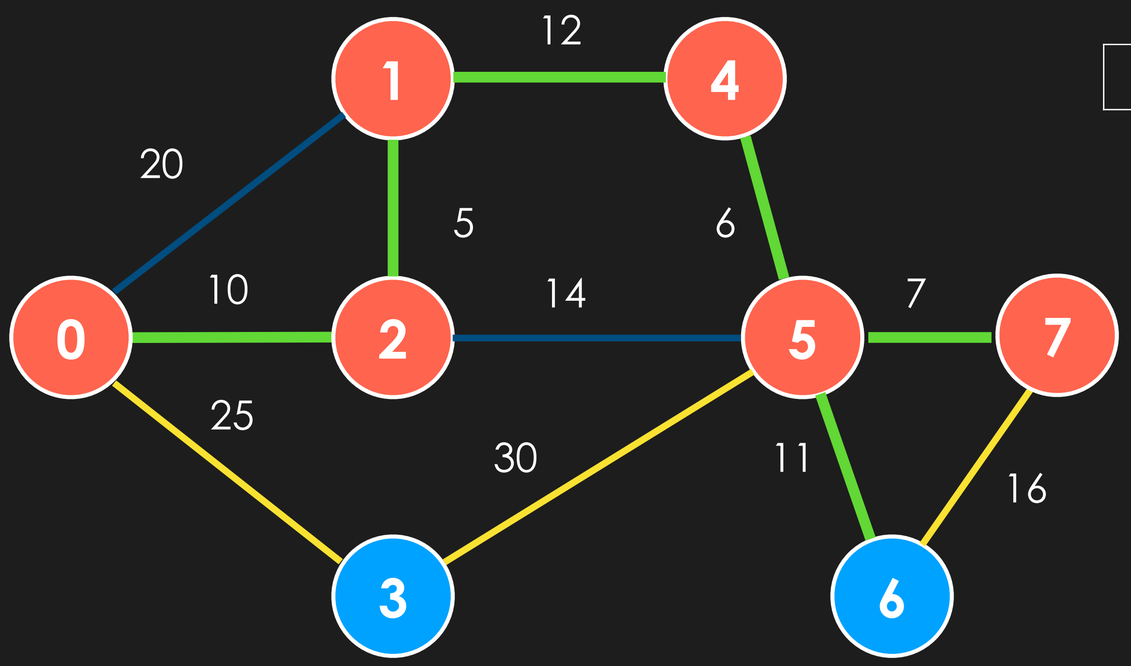
mstEdges

0 - 2
2 - 1
1 - 4
4 - 5
5 - 7

inMST

0	**TRUE**
1	**TRUE**
2	**TRUE**
3	FALSE
4	**TRUE**
5	**TRUE**
6	FALSE
7	**TRUE**

1. Start at **vertex 0** (or any random vertex), and add all adjacent edges to **pq**
2. Get **min edge** from pq and add to **mstEdges** if dest is not **inMST** **(means edge has not yet been added)**
3. For vertices in min edge, add to **inMST** and add adjacent edges to **pq if not already done so**



5 - 6	11

pq

edge	weight
2 - 5	14
7 - 6	16
0 - 1	20
0 - 3	25
5 - 3	30

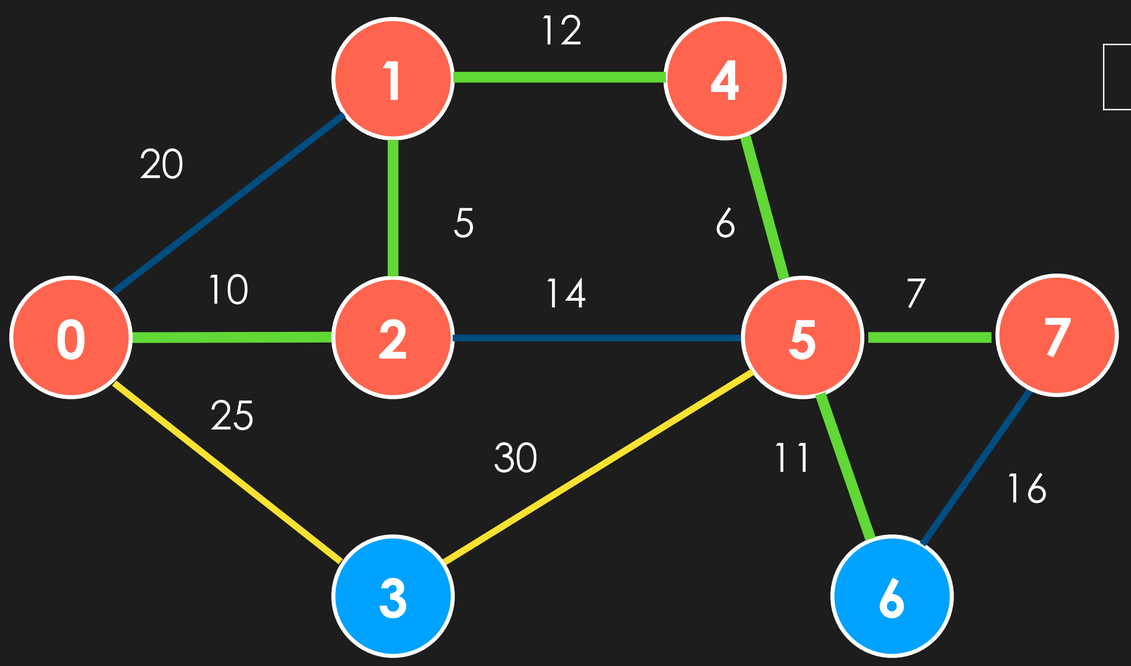
mstEdges

0 - 2
2 - 1
1 - 4
4 - 5
5 - 7
5 - 6

inMST

0	TRUE
1	TRUE
2	TRUE
3	FALSE
4	TRUE
5	TRUE
6	FALSE
7	TRUE

1. Start at **vertex 0** (or any random vertex), and add all adjacent edges to **pq**
2. Get **min edge** from pq and add to **mstEdges** if dest is not **inMST** **(means edge has not yet been added)**
3. For vertices in min edge, add to **inMST** and add adjacent edges to **pq if not already done so**



5 - 6	11

pq

edge	weight
2 - 5	14
7 - 6	16
0 - 1	20
0 - 3	25
5 - 3	30

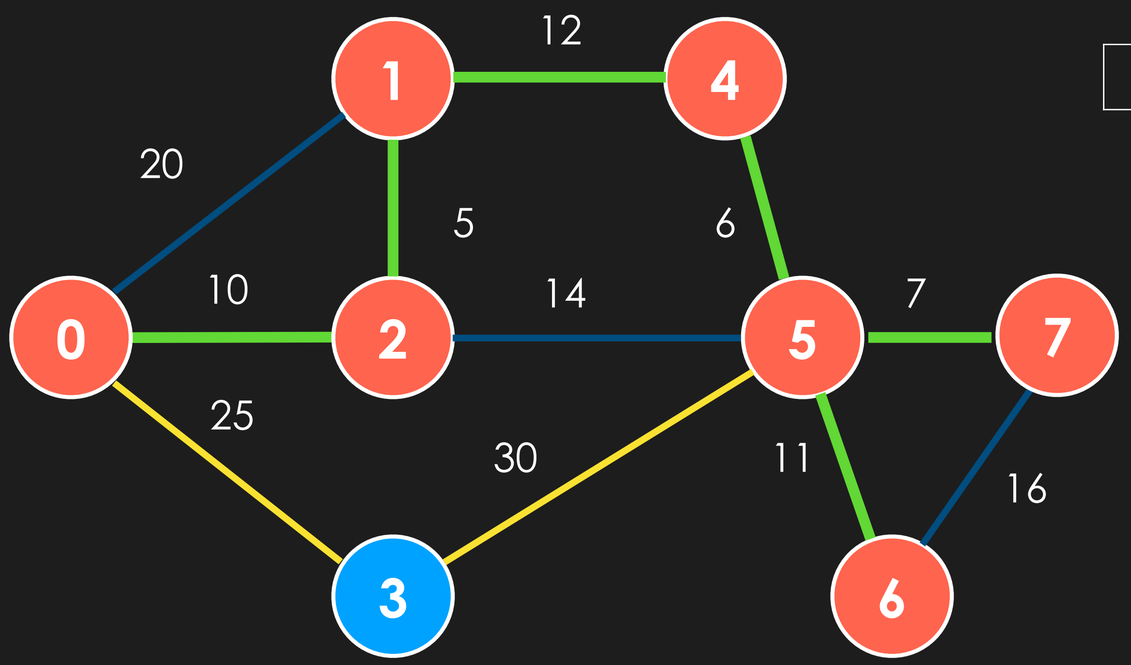
mstEdges

0 - 2
2 - 1
1 - 4
4 - 5
5 - 7
5 - 6

inMST

0	TRUE
1	TRUE
2	TRUE
3	FALSE
4	TRUE
5	TRUE
6	FALSE
7	TRUE

1. Start at **vertex 0** (or any random vertex), and add all adjacent edges to **pq**
2. Get **min edge** from pq and add to **mstEdges** if dest is not **inMST** **(means edge has not yet been added)**
3. For vertices in min edge, add to **inMST** and add adjacent edges to **pq if not already done so**



5 - 6	11

pq

edge	weight
2 - 5	14
7 - 6	16
0 - 1	20
0 - 3	25
5 - 3	30

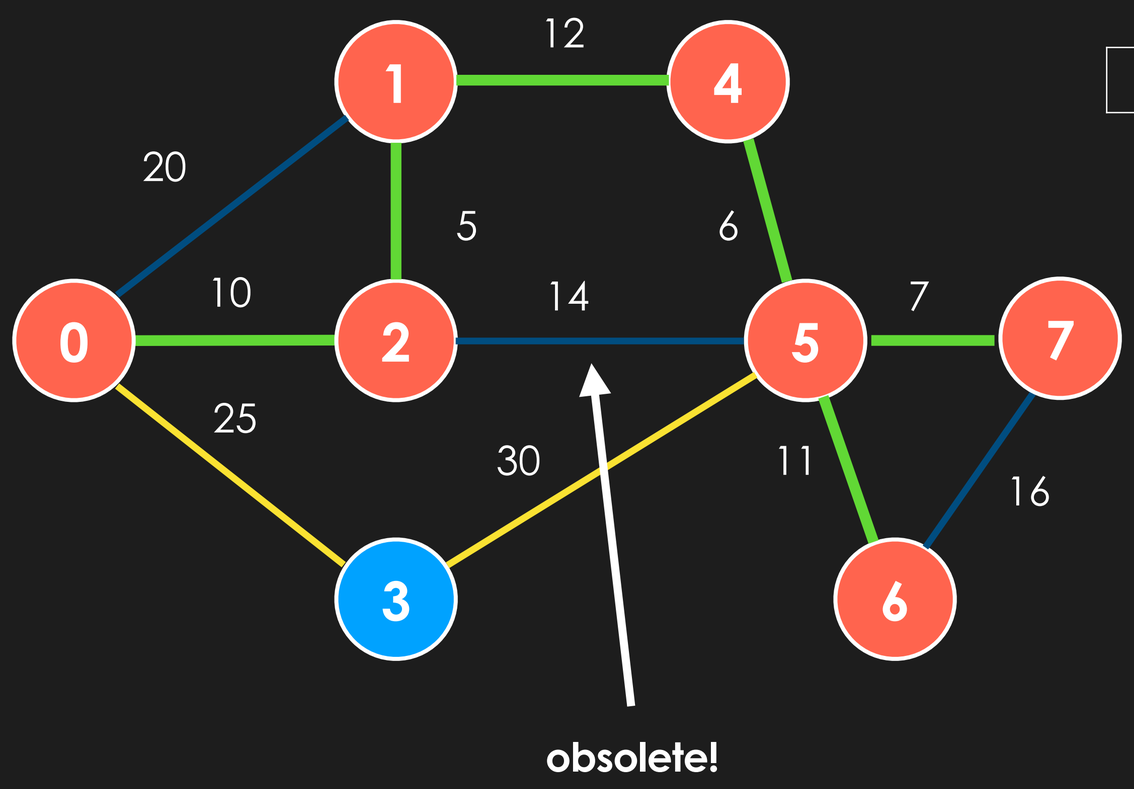
mstEdges

0 - 2
2 - 1
1 - 4
4 - 5
5 - 7
5 - 6

inMST

0	TRUE
1	TRUE
2	TRUE
3	FALSE
4	TRUE
5	TRUE
6	TRUE
7	TRUE

1. Start at **vertex 0** (or any random vertex), and add all adjacent edges to **pq**
2. Get **min edge** from pq and add to **mstEdges** if dest is not **inMST** **(means edge has not yet been added)**
3. For vertices in min edge, add to **inMST** and add adjacent edges to **pq if not already done so**



2 - 5	14

pq

edge	weight
7 - 6	16
0 - 1	20
0 - 3	25
5 - 3	30

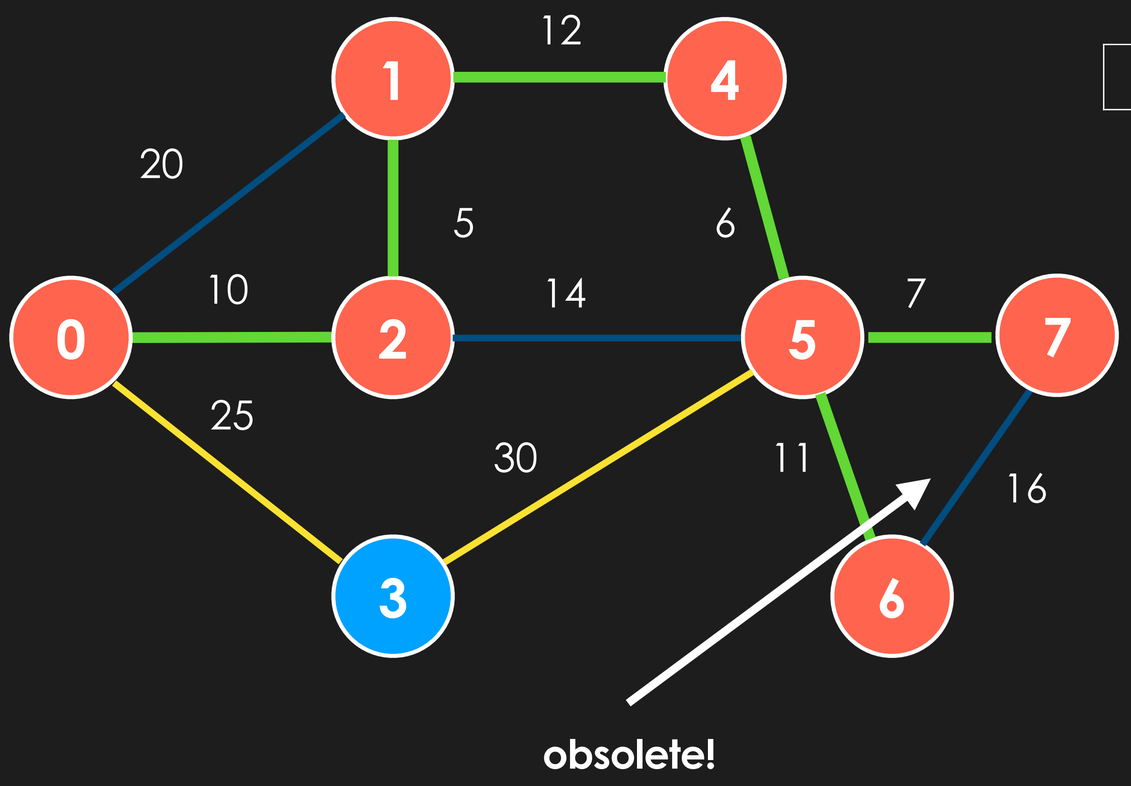
mstEdges

0 - 2
2 - 1
1 - 4
4 - 5
5 - 7
5 - 6

inMST

0	TRUE
1	TRUE
2	TRUE
3	FALSE
4	TRUE
5	TRUE
6	TRUE
7	TRUE

1. Start at **vertex 0** (or any random vertex), and add all adjacent edges to **pq**
2. Get **min edge** from pq and add to **mstEdges** if dest is not **inMST** **(means edge has not yet been added)**
3. For vertices in min edge, add to **inMST** and add adjacent edges to **pq if not already done so**



7 - 6	16

pq

edge	weight
0 - 1	20
0 - 3	25
5 - 3	30

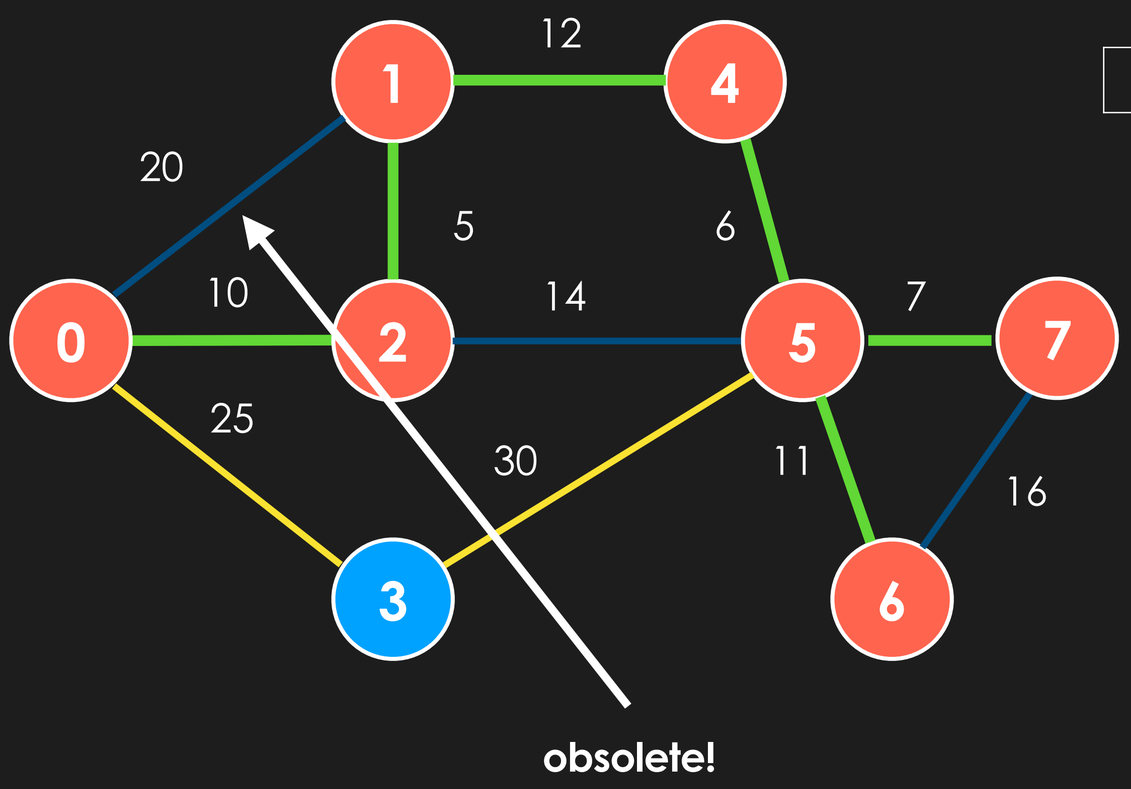
mstEdges

0 - 2
2 - 1
1 - 4
4 - 5
5 - 7
5 - 6

inMST

0	TRUE
1	TRUE
2	TRUE
3	FALSE
4	TRUE
5	TRUE
6	TRUE
7	TRUE

1. Start at **vertex 0** (or any random vertex), and add all adjacent edges to **pq**
2. Get **min edge** from pq and add to **mstEdges** if dest is not **inMST** **(means edge has not yet been added)**
3. For vertices in min edge, add to **inMST** and add adjacent edges to **pq if not already done so**



0 - 1	20

pq

edge	weight
0 - 3	25
5 - 3	30

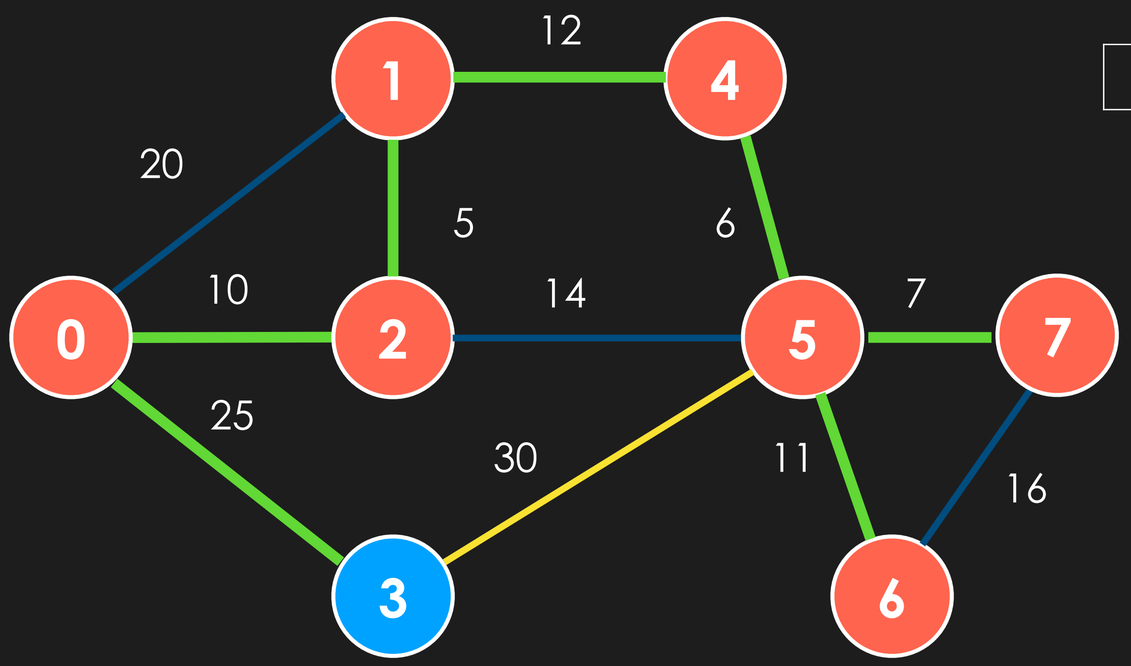
mstEdges

0 - 2
2 - 1
1 - 4
4 - 5
5 - 7
5 - 6

inMST

0	TRUE
1	TRUE
2	TRUE
3	FALSE
4	TRUE
5	TRUE
6	TRUE
7	TRUE

1. Start at **vertex 0** (or any random vertex), and add all adjacent edges to **pq**
2. Get **min edge** from pq and add to **mstEdges** if dest is not **inMST** **(means edge has not yet been added)**
3. For vertices in min edge, add to **inMST** and add adjacent edges to **pq if not already done so**



0 - 3	25

pq

edge	weight
5 - 3	30

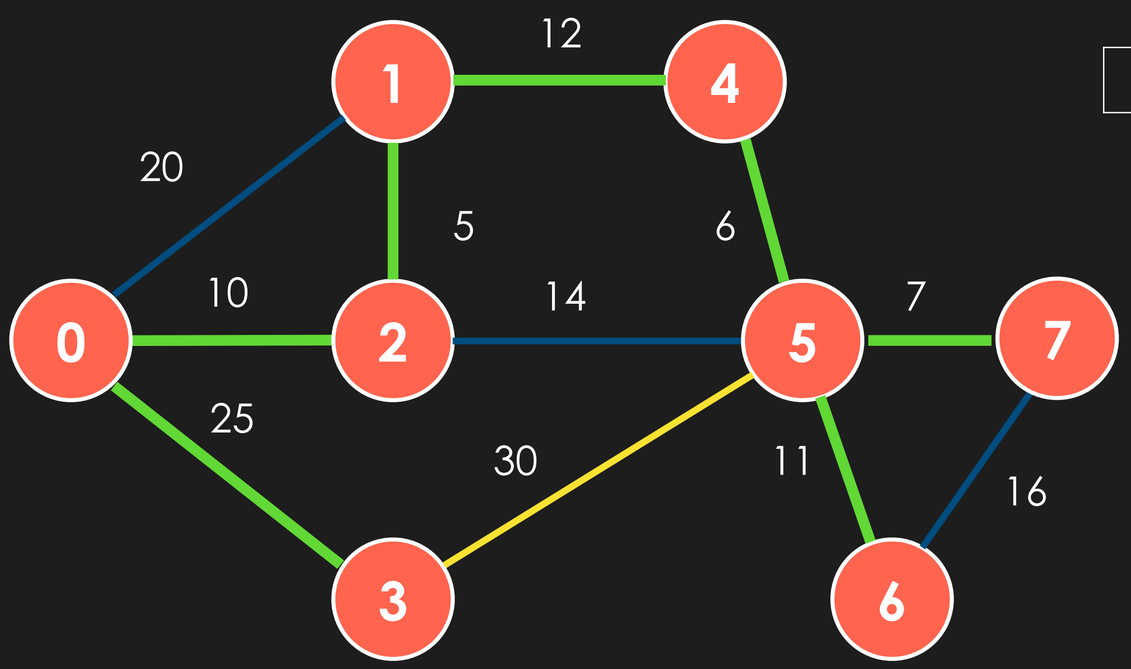
mstEdges

0 - 2
2 - 1
1 - 4
4 - 5
5 - 7
5 - 6
0 - 3

inMST

0	TRUE
1	TRUE
2	TRUE
3	FALSE
4	TRUE
5	TRUE
6	TRUE
7	TRUE

1. Start at **vertex 0** (or any random vertex), and add all adjacent edges to **pq**
2. Get **min edge** from pq and add to **mstEdges** if dest is not **inMST** **(means edge has not yet been added)**
3. For vertices in min edge, add to **inMST** and add adjacent edges to **pq if not already done so**



0 - 3	25

pq

edge	weight
5 - 3	30

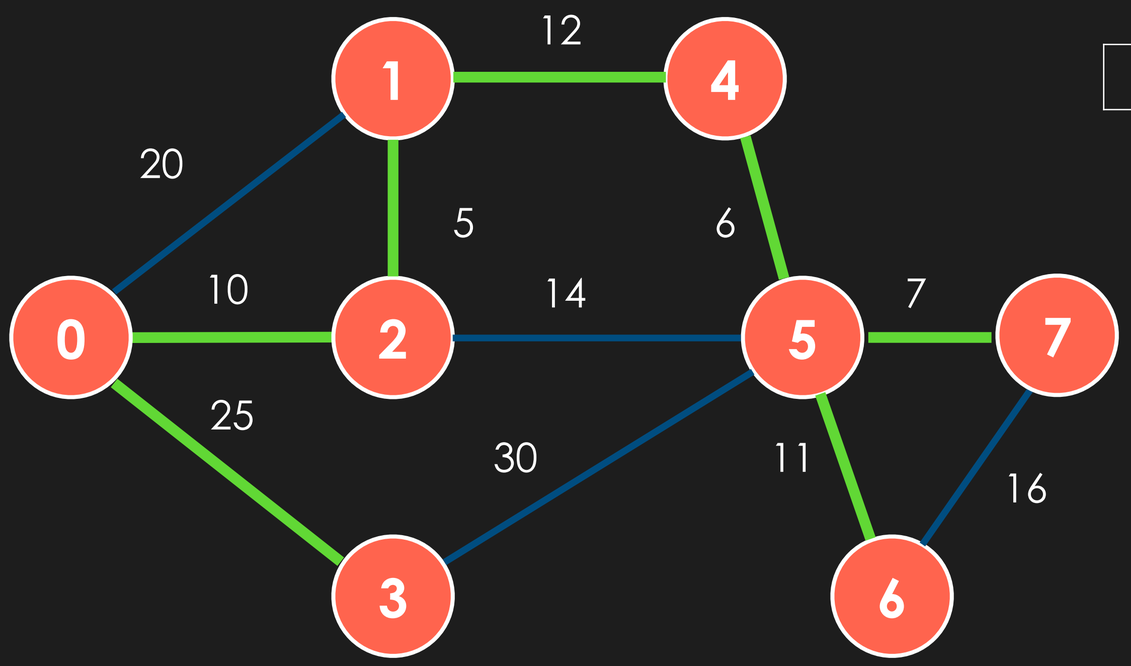
mstEdges

0 - 2
2 - 1
1 - 4
4 - 5
5 - 7
5 - 6
0 - 3

inMST

0	TRUE
1	TRUE
2	TRUE
3	TRUE
4	TRUE
5	TRUE
6	TRUE
7	TRUE

1. Start at **vertex 0** (or any random vertex), and add all adjacent edges to **pq**
2. Get **min edge** from pq and add to **mstEdges** if dest is not **inMST** **(means edge has not yet been added)**
3. For vertices in min edge, add to **inMST** and add adjacent edges to **pq if not already done so**



0 - 3	25

pq

edge	weight
5 - 3	30

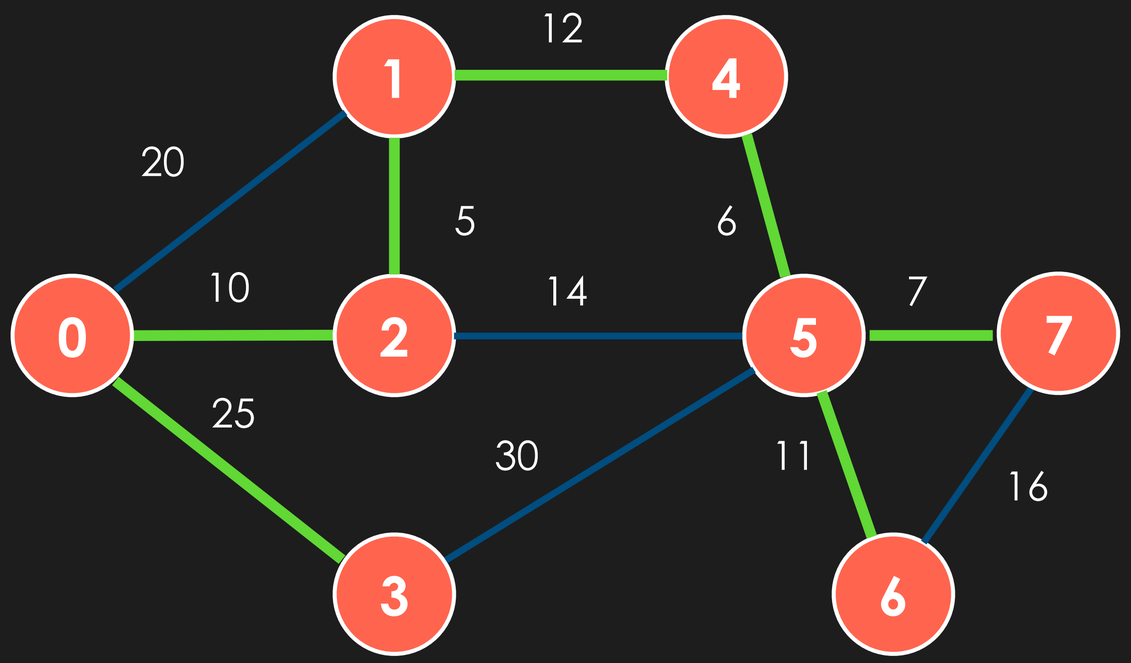
mstEdges

0 - 2
2 - 1
1 - 4
4 - 5
5 - 7
5 - 6
0 - 3

inMST

0	TRUE
1	TRUE
2	TRUE
3	TRUE
4	TRUE
5	TRUE
6	TRUE
7	TRUE

Now we have our MST!



mstEdges

0 - 2
2 - 1
1 - 4
4 - 5
5 - 7
5 - 6
0 - 3

Implementation of Prim's Algorithm

LazyPrimMST

```
def LazyPrimMST(graph: WeightedGraph):
    V = len(graph.adjList)

    inMST = [False] * V
    mstEdges = []

    pq = MinHeap(V ** 2)

    for v in range(V):
        if not inMST[v]:
            prim(graph, v, pq, inMST, mstEdges)

    return edges
```

LazyPrimMST

```
def LazyPrimMST(graph: WeightedGraph):
    V = len(graph.adjList)

    inMST = [False] * V
    mstEdges = []

    pq = MinHeap(V ** 2)

    for v in range(V):
        if not inMST[v]:
            prim(graph, v, pq, inMST, mstEdges)

    return edges
```

This step is performed under the assumption that the graph is not strongly connected and therefore we are searching for **Minimum Spanning Forest**

prim

```
def prim(graph, s, pq, inMST, mstEdges):

    addEdgesToPQ(graph, s, pq, inMST)

    while (pq.size != 0):
        edge = pq.getMin().key

        if inMST[edge.dest]:
            continue

        mstEdges.append(edge)
        addEdgesToPQ(graph, edge.dest, pq, inMST)
```

addEdgesToPQ

```
def addEdgesToPQ(graph, s, pq, inMST):
    inMST[s] = True
    for edge in graph.adjList[s]:
        pq.insert(edge, edge.weight)
```

Time complexity of Prim's Algorithm

Time complexity of Prim's Algorithm

We perform **getMin()** from **pq** at most **E times**

We perform **insert()** into **pq** at most **E times**

Since our **pq** has at most E items, the time complexity for **getMin()** and **insert()** is **logE**

Thus, the time complexity for computing our **MST** is **ElogE**

In fact, we can alter our algorithm to store vertices (similar to Dijkstra's algo) to achieve a time complexity of E logV.

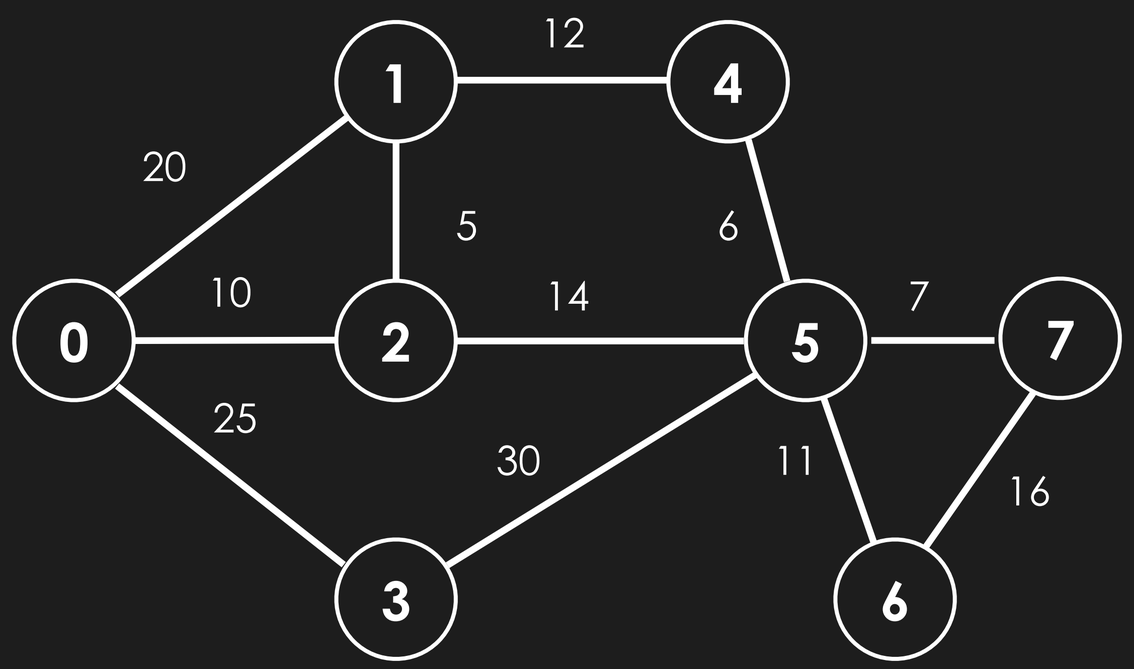
Note: E logV is faster in graphs which are sparse (less edges)

Prim's algorithm in ElogV

If we use a **pq** with **decrease key** operation, then we can keep vertices as keys in our pq and compute decrease key every time we encounter a smaller edge to that vertice!

This way, our **pq** has at most V items, and thus **insert** / **getMin** will take at most **logV**

1. Set distTo vertex 0 as 0 and insert into **pq**

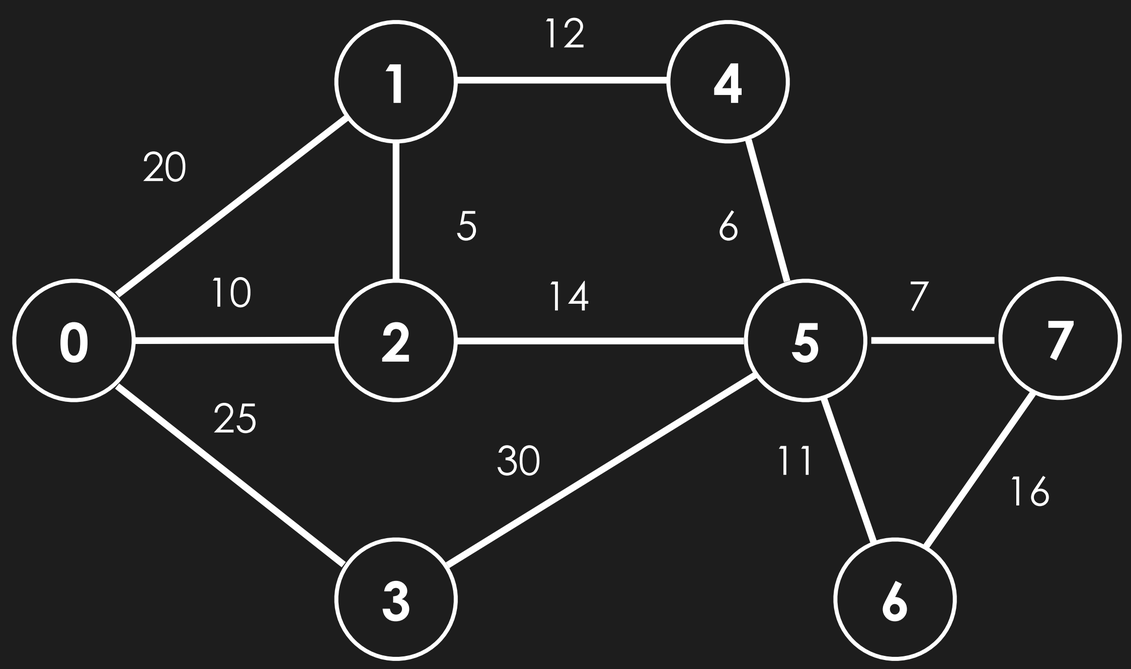


pq

vertex	distTo

	distTo		**edgeTo**		**inMST**
0	None	**0**	-1	**0**	FALSE
1	None	**1**	-1	**1**	FALSE
2	None	**2**	-1	**2**	FALSE
3	None	**3**	-1	**3**	FALSE
4	None	**4**	-1	**4**	FALSE
5	None	**5**	-1	**5**	FALSE
6	None	**6**	-1	**6**	FALSE
7	None	**7**	-1	**7**	FALSE

1. Set distTo vertex 0 as 0 and insert into **pq**



pq

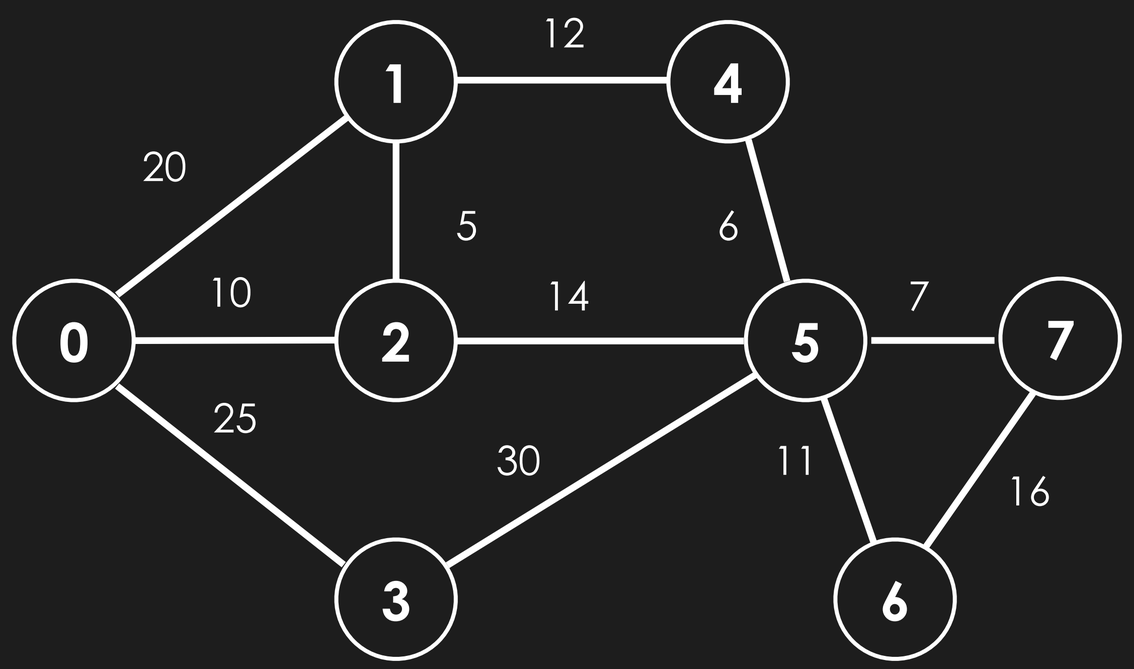
vertex	distTo
0	0

	distTo
0	0
1	None
2	None
3	None
4	None
5	None
6	None
7	None

	edgeTo
0	-1
1	-1
2	-1
3	-1
4	-1
5	-1
6	-1
7	-1

	inMST
0	FALSE
1	FALSE
2	FALSE
3	FALSE
4	FALSE
5	FALSE
6	FALSE
7	FALSE

1. Set distTo vertex 0 as 0 and insert into **pq**
2. While **pq** is not empty, remove minimum, and add adjacent vertices
    1. if not **inMST**
    2. if edge weight < current distTo adjacent vertex



pq

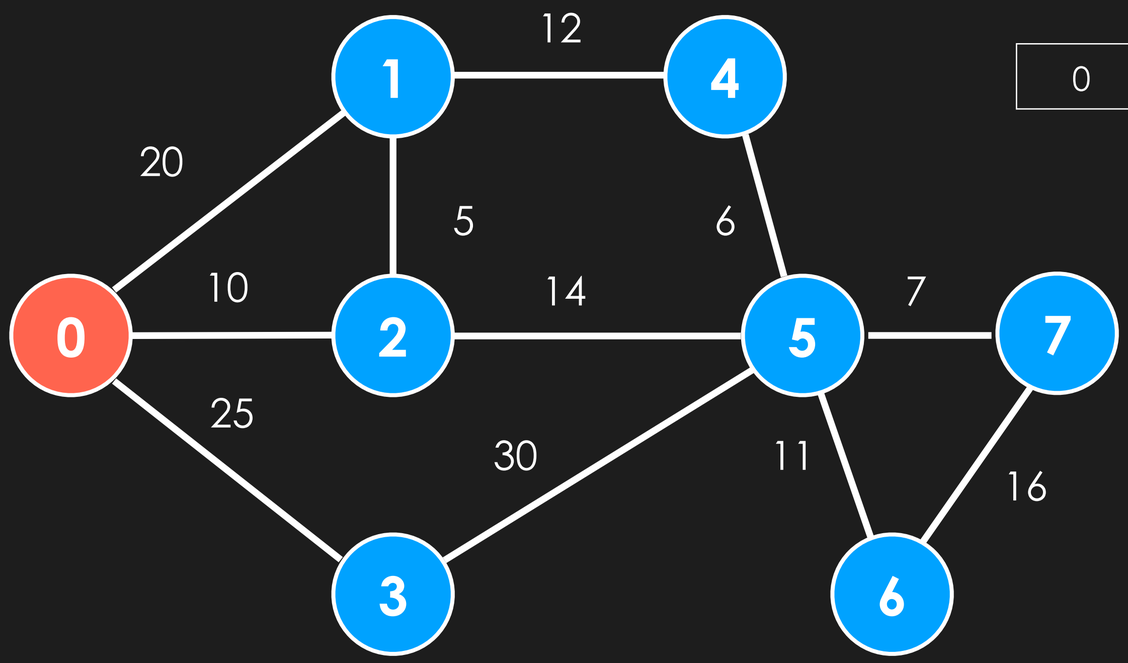
vertex	distTo
0	0

	distTo
0	0
1	None
2	None
3	None
4	None
5	None
6	None
7	None

	edgeTo
0	-1
1	-1
2	-1
3	-1
4	-1
5	-1
6	-1
7	-1

	inMST
0	FALSE
1	FALSE
2	FALSE
3	FALSE
4	FALSE
5	FALSE
6	FALSE
7	FALSE

1. Set distTo vertex 0 as 0 and insert into **pq**
2. While **pq** is not empty, remove minimum, and add adjacent vertices
    1. if not **inMST**
    2. if edge weight < current distTo adjacent vertex



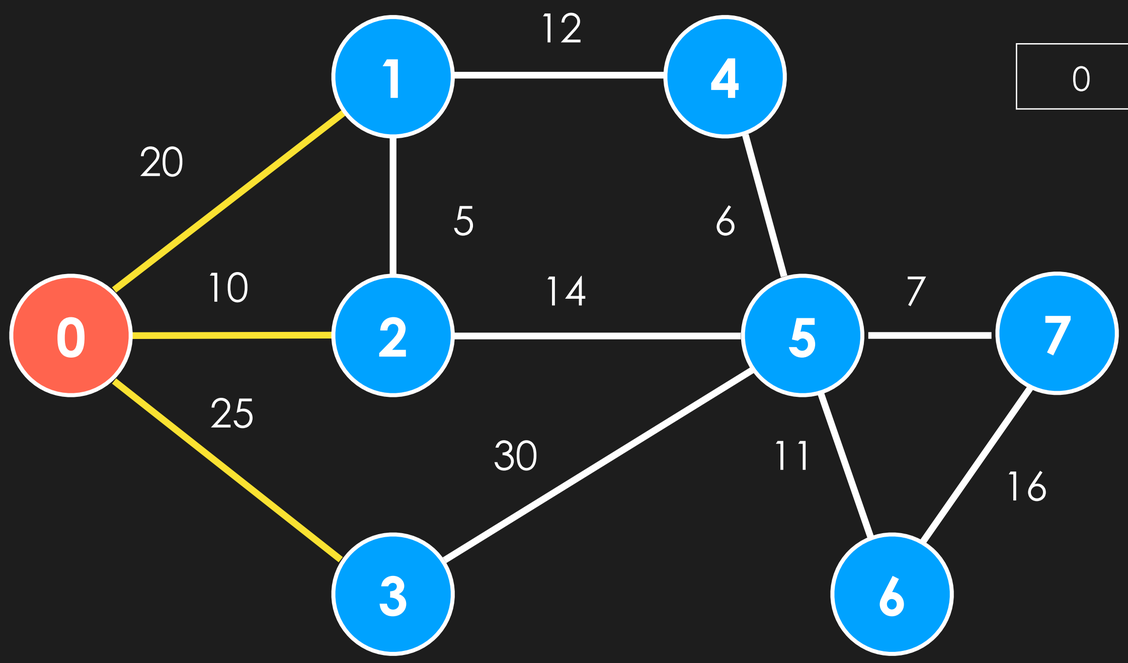
pq	
vertex	distTo
0	0

	distTo
0	0
1	None
2	None
3	None
4	None
5	None
6	None
7	None

	edgeTo
0	-1
1	-1
2	-1
3	-1
4	-1
5	-1
6	-1
7	-1

	inMST
0	TRUE
1	FALSE
2	FALSE
3	FALSE
4	FALSE
5	FALSE
6	FALSE
7	FALSE

1. Set distTo vertex 0 as 0 and insert into **pq**
2. While **pq** is not empty, remove minimum, and add adjacent vertices
    1. if not **inMST**
    2. if edge weight < current distTo adjacent vertex



0	0

pq

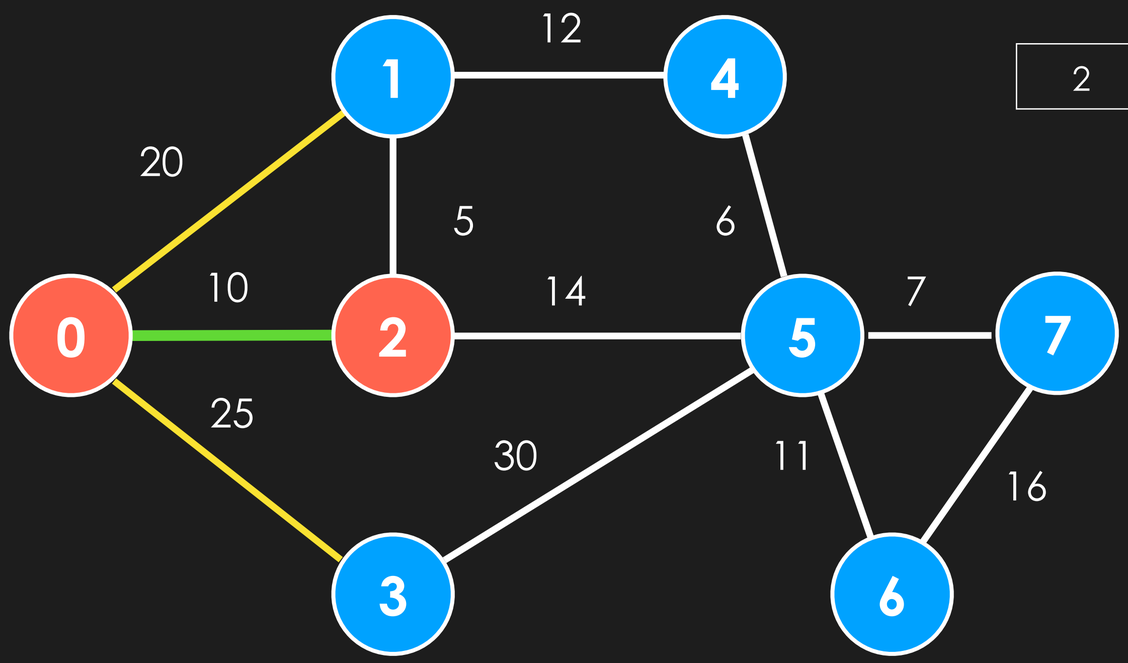
vertex	distTo
2	10
1	20
3	25

	distTo
0	0
1	20
2	10
3	25
4	None
5	None
6	None
7	None

	edgeTo
0	-1
1	0 - 1
2	0 - 2
3	0 - 3
4	-1
5	-1
6	-1
7	-1

	inMST
0	TRUE
1	FALSE
2	FALSE
3	FALSE
4	FALSE
5	FALSE
6	FALSE
7	FALSE

1. Set distTo vertex 0 as 0 and insert into **pq**
2. While **pq** is not empty, remove minimum, and add adjacent vertices
    1. if not **inMST**
    2. if edge weight < current distTo adjacent vertex



2	10

pq

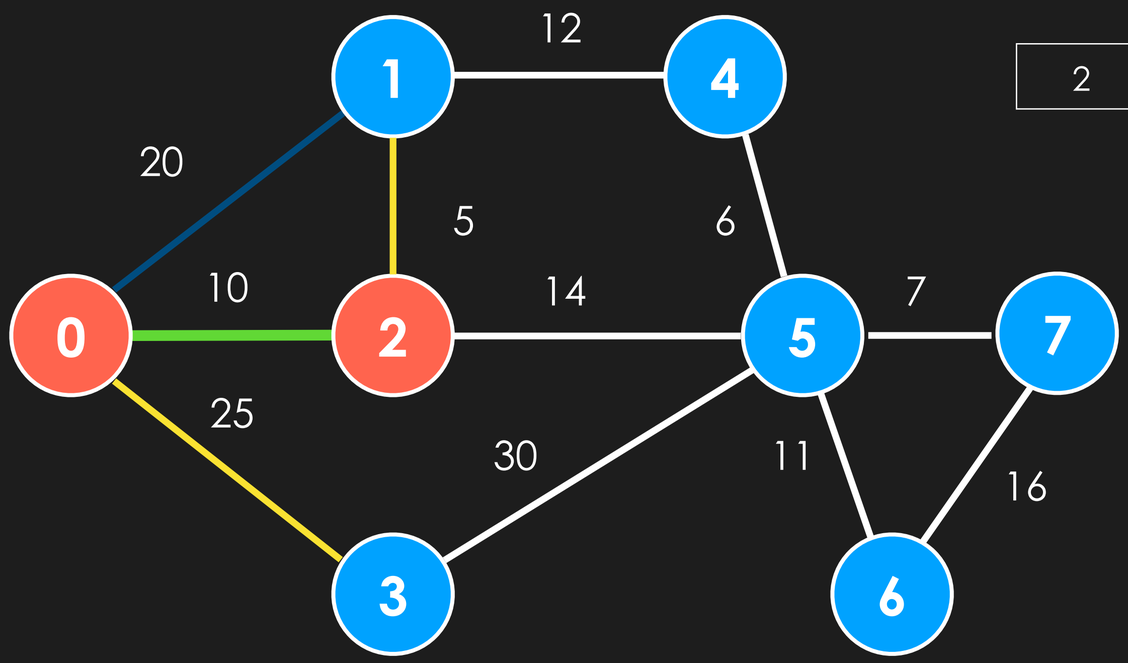
vertex	distTo
1	20
3	25

	distTo
0	0
1	20
2	10
3	25
4	None
5	None
6	None
7	None

	edgeTo
0	-1
1	0 - 1
2	0 - 2
3	0 - 3
4	-1
5	-1
6	-1
7	-1

	inMST
0	TRUE
1	FALSE
2	TRUE
3	FALSE
4	FALSE
5	FALSE
6	FALSE
7	FALSE

1. Set distTo vertex 0 as 0 and insert into **pq**
2. While **pq** is not empty, remove minimum, and add adjacent vertices
    1. if not **inMST**
    2. if edge weight < current distTo adjacent vertex



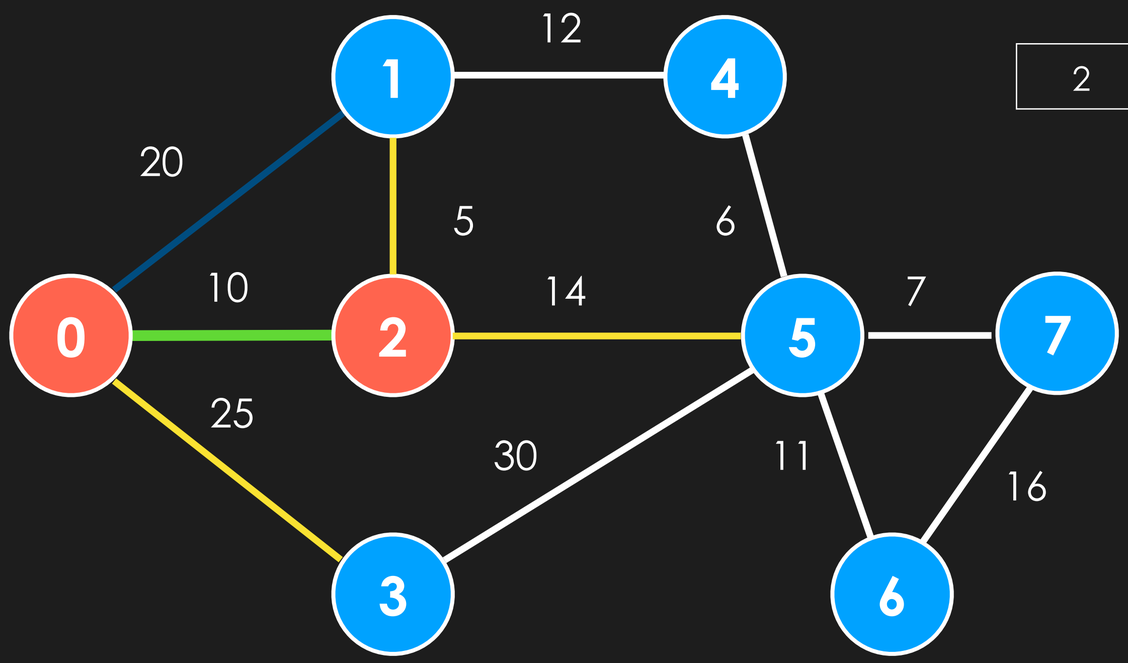
2	10

pq

vertex	distTo
1	**5**
3	25

	distTo	**edgeTo**	**inMST**
0	0	-1	**TRUE**
1	**5**	**2 - 1**	FALSE
2	10	**0 - 2**	**TRUE**
3	25	0 - 3	FALSE
4	None	-1	FALSE
5	None	-1	FALSE
6	None	-1	FALSE
7	None	-1	FALSE

1. Set distTo vertex 0 as 0 and insert into **pq**
2. While **pq** is not empty, remove minimum, and add adjacent vertices
    1. if not **inMST**
    2. if edge weight < current distTo adjacent vertex



2	10

pq

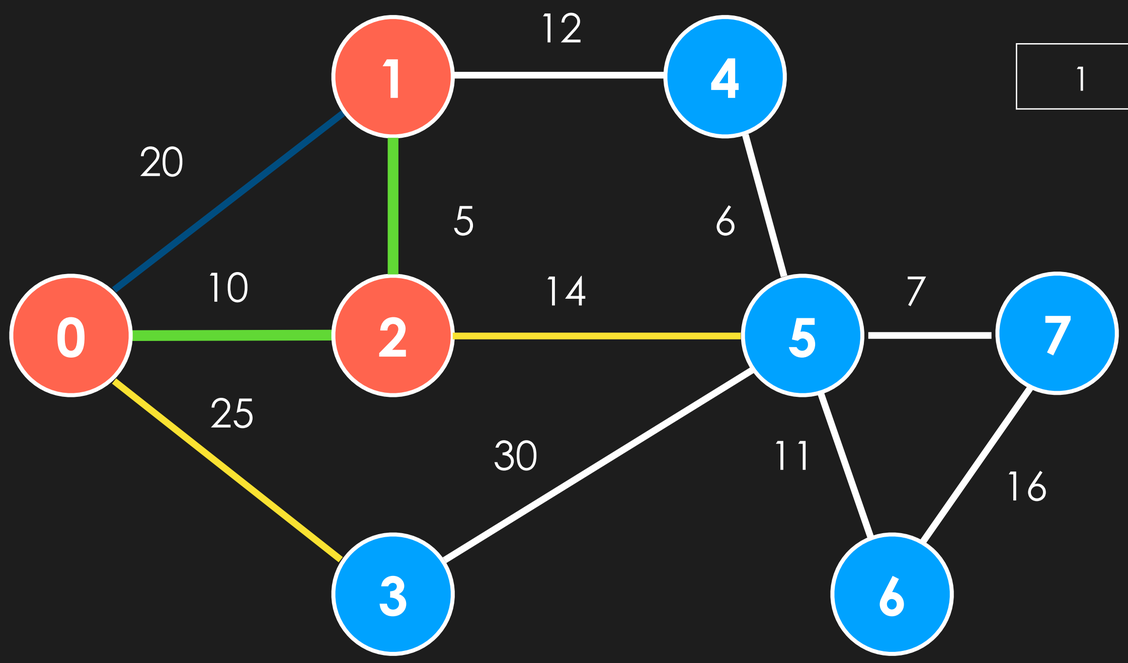
vertex	distTo
1	**5**
5	**14**
3	25

	distTo
0	0
1	**5**
2	10
3	25
4	None
5	**14**
6	None
7	None

	edgeTo
0	-1
1	**2 - 1**
2	**0 - 2**
3	0 - 3
4	-1
5	**2 - 5**
6	-1
7	-1

	inMST
0	**TRUE**
1	FALSE
2	**TRUE**
3	FALSE
4	FALSE
5	FALSE
6	FALSE
7	FALSE

1. Set distTo vertex 0 as 0 and insert into **pq**
2. While **pq** is not empty, remove minimum, and add adjacent vertices
    1. if not **inMST**
    2. if edge weight < current distTo adjacent vertex



1	5

pq

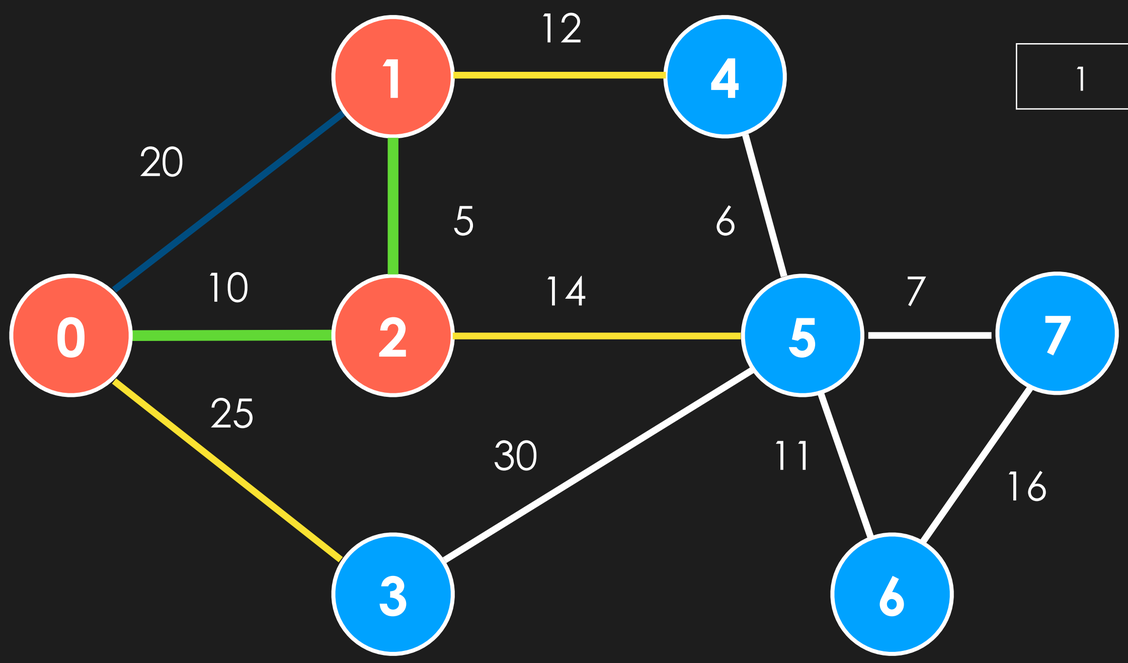
vertex	distTo
5	14
3	25

	distTo
0	0
1	5
2	10
3	25
4	None
5	14
6	None
7	None

	edgeTo
0	-1
1	2 - 1
2	0 - 2
3	0 - 3
4	-1
5	2 - 5
6	-1
7	-1

	inMST
0	TRUE
1	TRUE
2	TRUE
3	FALSE
4	FALSE
5	FALSE
6	FALSE
7	FALSE

1. Set distTo vertex 0 as 0 and insert into **pq**
2. While **pq** is not empty, remove minimum, and add adjacent vertices
    1. if not **inMST**
    2. if edge weight < current distTo adjacent vertex



1	5

pq

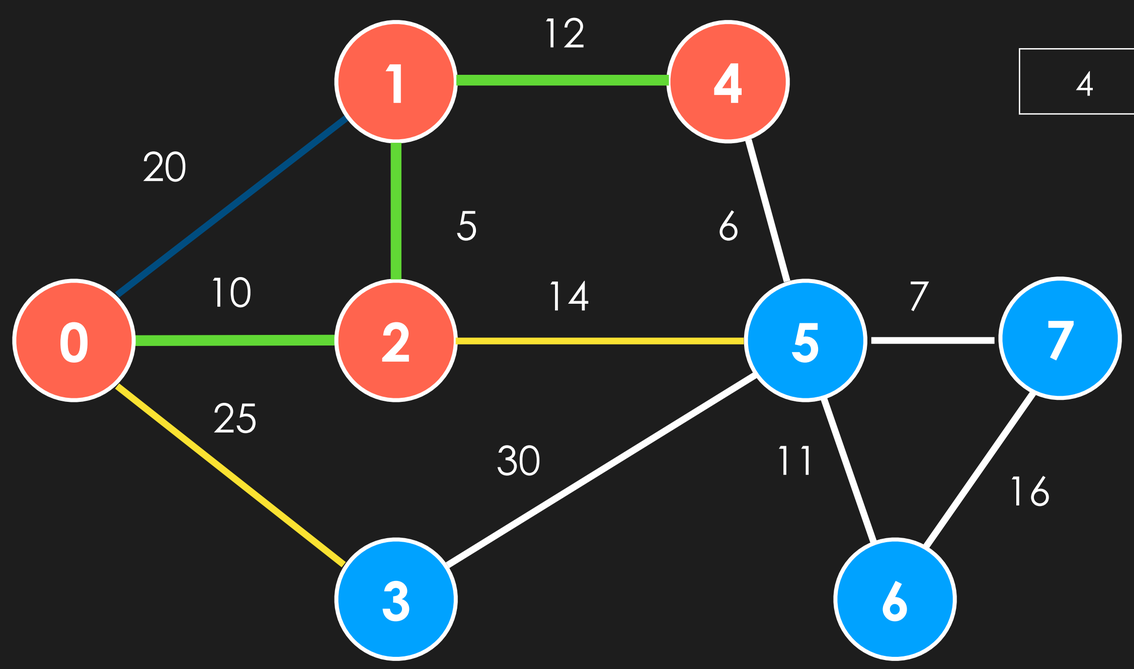
vertex	distTo
4	12
5	14
3	25

	distTo
0	0
1	5
2	10
3	25
4	12
5	14
6	None
7	None

	edgeTo
0	-1
1	2 - 1
2	0 - 2
3	0 - 3
4	1 - 4
5	2 - 5
6	-1
7	-1

	inMST
0	TRUE
1	TRUE
2	TRUE
3	FALSE
4	FALSE
5	FALSE
6	FALSE
7	FALSE

1. Set distTo vertex 0 as 0 and insert into **pq**
2. While **pq** is not empty, remove minimum, and add adjacent vertices
    1. if not **inMST**
    2. if edge weight < current distTo adjacent vertex



4	12

pq

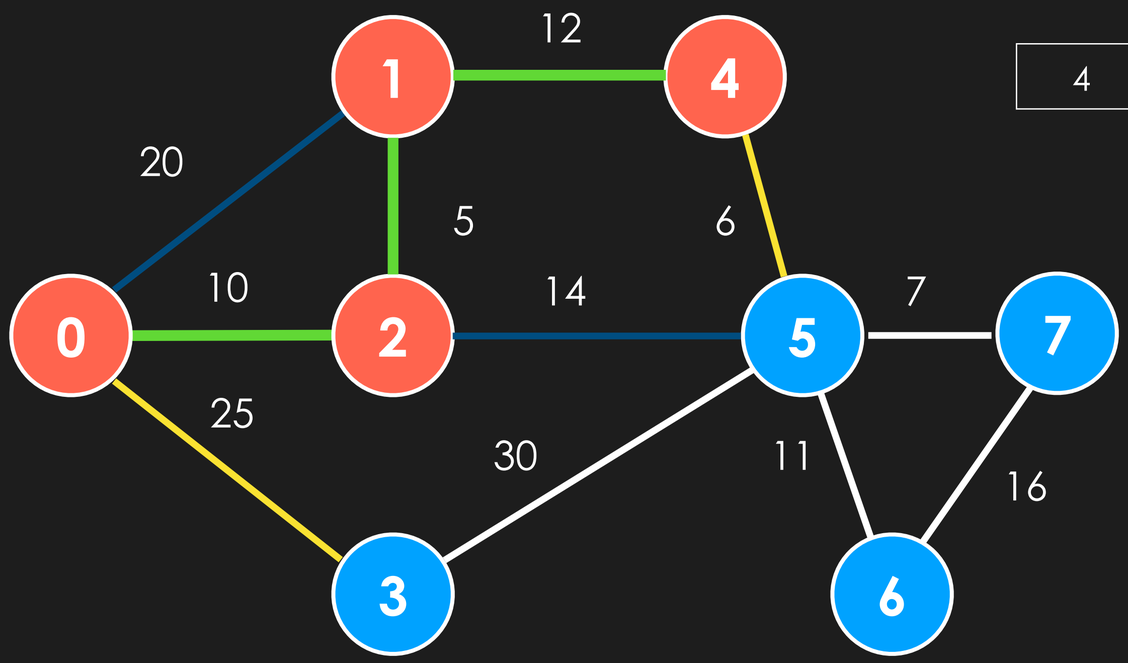
vertex	distTo
5	14
3	25

	distTo
0	0
1	5
2	10
3	25
4	12
5	14
6	None
7	None

	edgeTo
0	-1
1	2 - 1
2	0 - 2
3	0 - 3
4	1 - 4
5	2 - 5
6	-1
7	-1

	inMST
0	TRUE
1	TRUE
2	TRUE
3	FALSE
4	TRUE
5	FALSE
6	FALSE
7	FALSE

1. Set distTo vertex 0 as 0 and insert into **pq**
2. While **pq** is not empty, remove minimum, and add adjacent vertices
    1. if not **inMST**
    2. if edge weight < current distTo adjacent vertex



4	12

pq

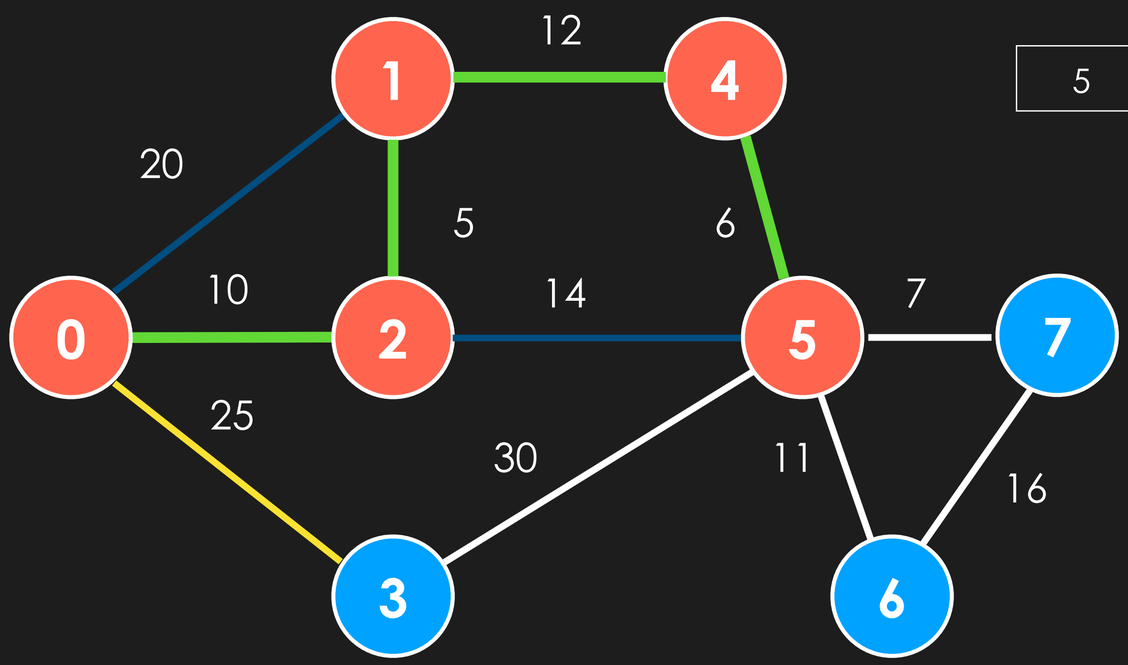
vertex	distTo
5	6
3	25

	distTo
0	0
1	5
2	10
3	25
4	12
5	6
6	None
7	None

	edgeTo
0	-1
1	2 - 1
2	0 - 2
3	0 - 3
4	1 - 4
5	4 - 5
6	-1
7	-1

	inMST
0	TRUE
1	TRUE
2	TRUE
3	FALSE
4	TRUE
5	FALSE
6	FALSE
7	FALSE

1. Set distTo vertex 0 as 0 and insert into **pq**
2. While **pq** is not empty, remove minimum, and add adjacent vertices
    1. if not **inMST**
    2. if edge weight < current distTo adjacent vertex



5	6

pq

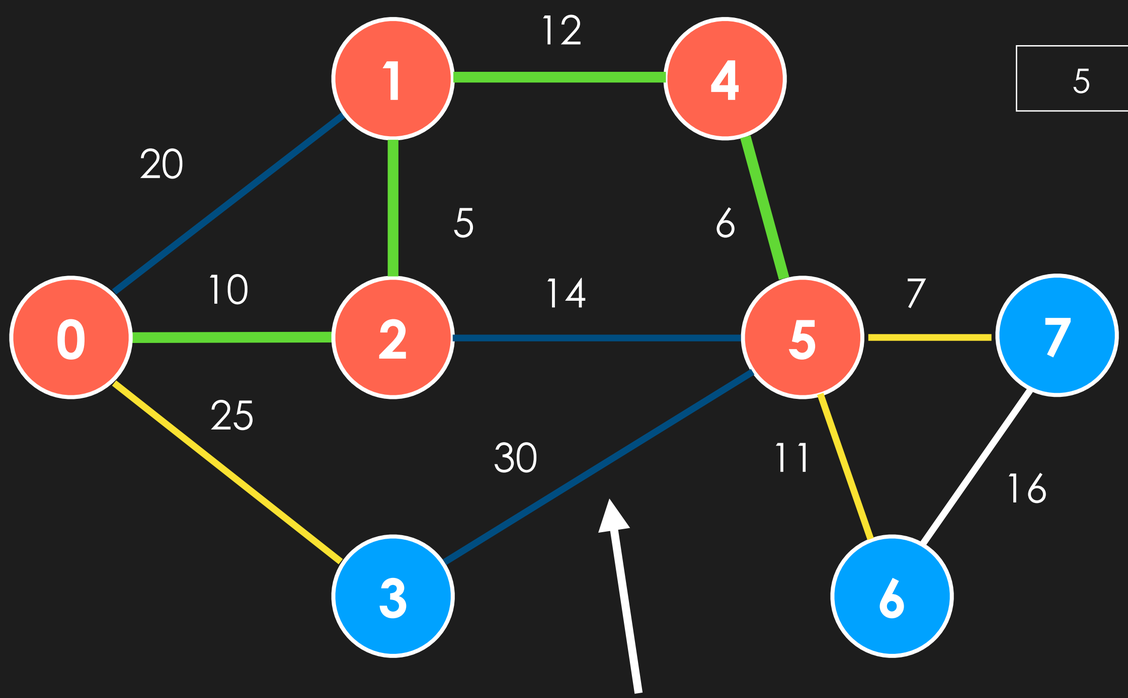
vertex	distTo
3	25

	distTo
0	0
1	5
2	10
3	25
4	12
5	6
6	None
7	None

	edgeTo
0	-1
1	2 - 1
2	0 - 2
3	0 - 3
4	1 - 4
5	4 - 5
6	-1
7	-1

	inMST
0	TRUE
1	TRUE
2	TRUE
3	FALSE
4	TRUE
5	TRUE
6	FALSE
7	FALSE

1. Set distTo vertex 0 as 0 and insert into **pq**
2. While **pq** is not empty, remove minimum, and add adjacent vertices
    1. if not **inMST**
    2. if edge weight < current distTo adjacent vertex



5	6

rejected because current distTo[3] < 30!

pq

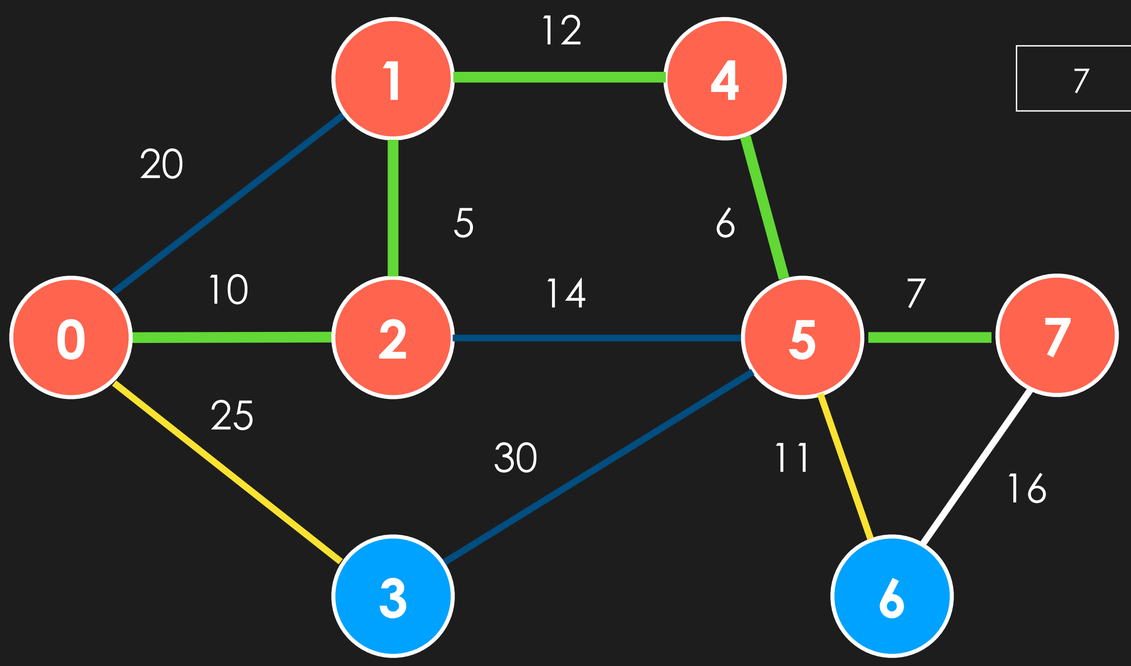
vertex	distTo
7	7
6	11
3	25

	distTo
0	0
1	5
2	10
3	25
4	12
5	6
6	11
7	7

	edgeTo
0	-1
1	2 - 1
2	0 - 2
3	0 - 3
4	1 - 4
5	4 - 5
6	5 - 6
7	5 - 7

	inMST
0	TRUE
1	TRUE
2	TRUE
3	FALSE
4	TRUE
5	TRUE
6	FALSE
7	FALSE

1. Set distTo vertex 0 as 0 and insert into **pq**
2. While **pq** is not empty, remove minimum, and add adjacent vertices
    1. if not **inMST**
    2. if edge weight < current distTo adjacent vertex



7	7

pq

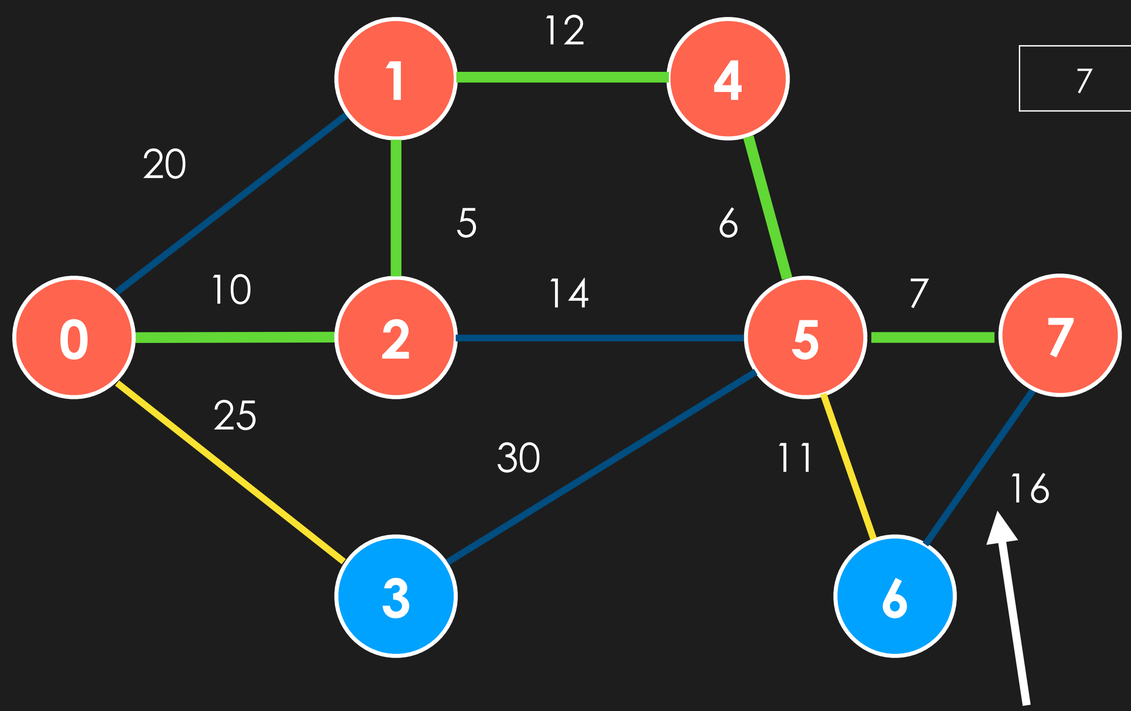
vertex	distTo
6	11
3	25

	distTo
0	0
1	5
2	10
3	25
4	12
5	6
6	11
7	7

	edgeTo
0	-1
1	2 - 1
2	0 - 2
3	0 - 3
4	1 - 4
5	4 - 5
6	5 - 6
7	5 - 7

	inMST
0	TRUE
1	TRUE
2	TRUE
3	FALSE
4	TRUE
5	TRUE
6	FALSE
7	TRUE

1. Set distTo vertex 0 as 0 and insert into **pq**
2. While **pq** is not empty, remove minimum, and add adjacent vertices
    1. if not **inMST**
    2. if edge weight < current distTo adjacent vertex



7	7

pq

vertex	distTo
6	11
3	25

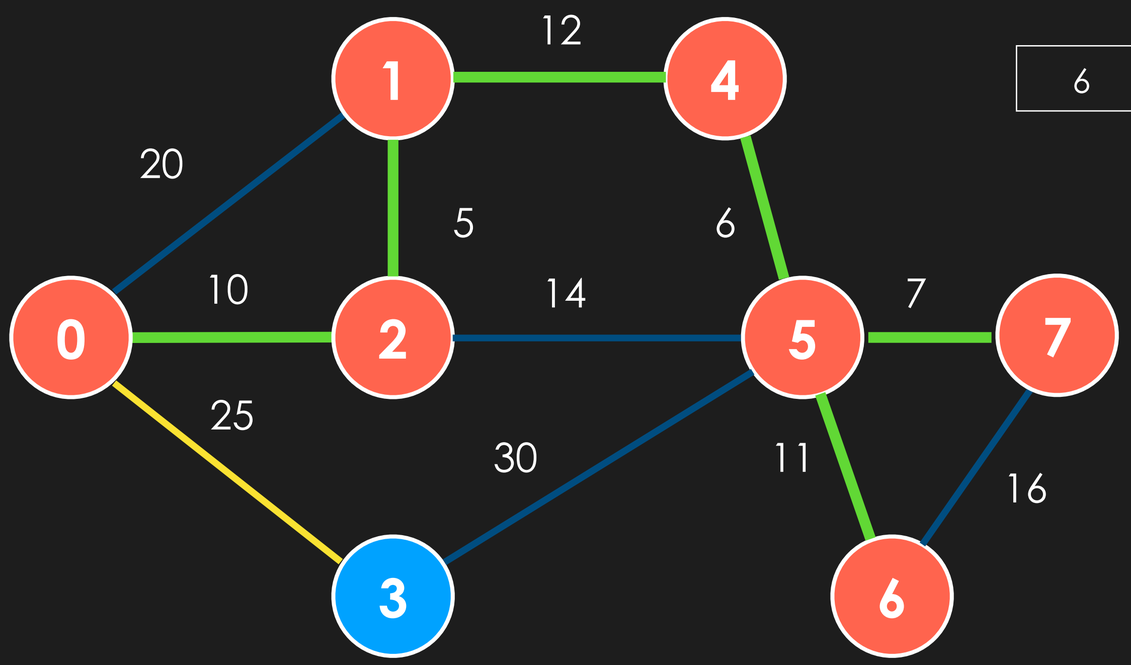
	distTo
0	0
1	5
2	10
3	25
4	12
5	6
6	11
7	7

	edgeTo
0	-1
1	2 - 1
2	0 - 2
3	0 - 3
4	1 - 4
5	4 - 5
6	5 - 6
7	5 - 7

	inMST
0	TRUE
1	TRUE
2	TRUE
3	FALSE
4	TRUE
5	TRUE
6	FALSE
7	TRUE

rejected because current distTo[6] < 16!

1. Set distTo vertex 0 as 0 and insert into **pq**
2. While **pq** is not empty, remove minimum, and add adjacent vertices
    1. if not **inMST**
    2. if edge weight < current distTo adjacent vertex



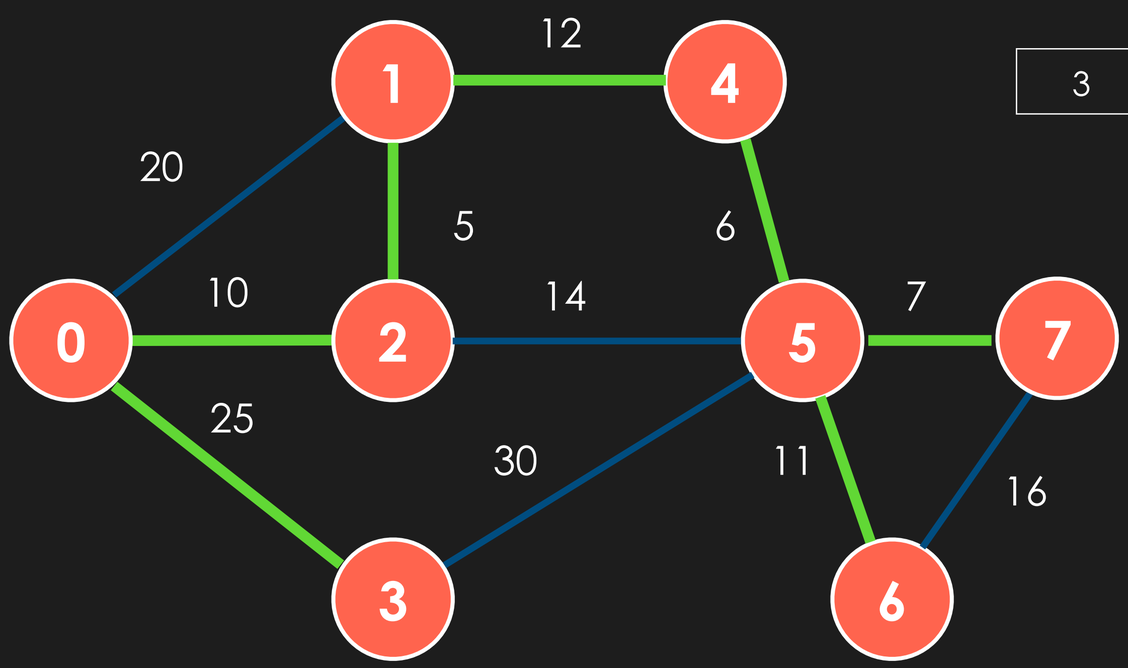
6	11

pq

vertex	distTo
3	25

	distTo	edgeTo	inMST
0	0	-1	TRUE
1	5	2 - 1	TRUE
2	10	0 - 2	TRUE
3	25	0 - 3	FALSE
4	12	1 - 4	TRUE
5	6	4 - 5	TRUE
6	11	5 - 6	TRUE
7	7	5 - 7	TRUE

1. Set distTo vertex 0 as 0 and insert into **pq**
2. While **pq** is not empty, remove minimum, and add adjacent vertices
    1. if not **inMST**
    2. if edge weight < current distTo adjacent vertex

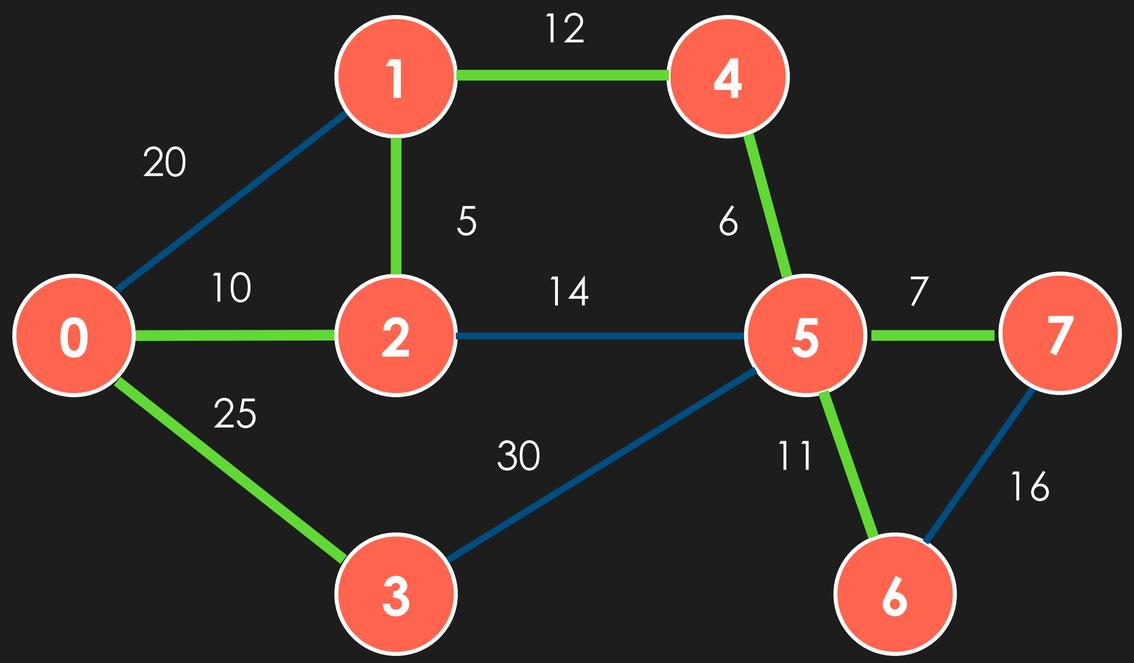


pq

vertex	distTo
3	25

	distTo	edgeTo	inMST
0	0	-1	TRUE
1	5	2 - 1	TRUE
2	10	0 - 2	TRUE
3	25	0 - 3	TRUE
4	12	1 - 4	TRUE
5	6	4 - 5	TRUE
6	11	5 - 6	TRUE
7	7	5 - 7	TRUE

Now we have our MST!



edgeTo

0	-1
1	2 - 1
2	0 - 2
3	0 - 3
4	1 - 4
5	4 - 5
6	5 - 6
7	5 - 7

PrimMST

```
def LazyPrimMST(graph: WeightedGraph):
    V = len(graph.adjList)

    edgeTo = [-1] * V
    distTo = [None] * V
    inMST = [False] * V

    pq = MinHeap(V)

    for v in range(V):
        if not inMST[v]:
            prim(graph, v, pq, distTo, edgeTo, inMST)

    return edgeTo
```

prim

```
def prim(graph, s, pq: MinHeap, distTo, edgeTo, inMST):
    distTo[s] = 0
    pq.insert(s, distTo[s])
    while (pq.size != 0):
        v = pq.getMin().key
        inMST[v] = True

        for edge in graph.adjList[v]:
            if inMST[edge.dest]:
                continue

            if distTo[edge.dest] == None or edge.weight < distTo[edge.dest]:
                distTo[edge.dest] = edge.weight
                edgeTo[edge.dest] = edge

                if edge.dest in pq.positions:
                    pq.decreaseKey(edge.dest, edge.weight)
                else:
                    pq.insert(edge.dest, edge.weight)
```

Alternatives to Prim's Algorithm

Kruskal's Algorithm

- Kruskal's algorithm, on a high level, sorts all edges by weights, and then adds them one by one while verifying that an MST is maintained
- Prim's algorithm works well for dense graphs while Kruskal's algorithm works better for sparse graphs

Applications of MST

Applications of MST

Design of telecommunication networks

Cluster analysis

Lab Session 2

- In this lab session, you will be implementing **prim.py**
- Your task is to implement **Prim's MST algorithm** for a **Weighted Undirected Graph** (you may assume that graphs being passed in only have a single MST)
- Your version of Prim can be lazy, in that you only need to achieve **ElogE time complexity**
- Your PrimMST function takes in a single argument: graph, which is a WeightedGraph object
- Your function should return a list of all MST edges (as WeightedEdge objects) in the MST
- The WeightedEdge & WeightedGraph class has been implemented for you, and functions as that shown in the lesson.
- The MinHeap class has been implemented for you, and contains methods as shown in lesson 2
- To test, run `**python utils/dijkstra_test.py**`

WeightedEdge & WeightedGraph

```
class WeightedEdge:
    def __init__(self, src, dest, weight):
        self.src = src
        self.dest = dest
        self.weight = weight

    def __str__(self):


class WeightedGraph:
    def __init__(self, V):
        self.adjList = [[] for i in range(V)]

    def addEdge(self, src, dest, weight):

    def printGraph(self):
```

MinHeap

```
class HeapItem:
    def __init__(self, key, value):
        self.key = key
        self.value = value


class MinHeap:
    def __init__(self, maxsize):
        self.maxsize = maxsize
        self.size = 0
        self.heap = [None] * (maxsize + 1)
        self.positions = {}

    def insert(self, newKey, newValue) -> None:

    def getMin(self) -> HeapItem:
```

Solution

```
def LazyPrimMST(graph: WeightedGraph):
    V = len(graph.adjList)
    inMST = [False] * V
    pq = MinHeap(V ** 2)
    mstEdges = []

    for v in range(V):
        if not inMST[v]:
            prim(graph, v, pq, inMST, mstEdges)

    return mstEdges
```

Solution

```
def prim(graph, s, pq: MinHeap, inMST, mstEdges):

    addEdgesToPQ(graph, s, pq, inMST)

    while (pq.size != 0):
        edge = pq.getMin().key

        if inMST[edge.dest]:
            continue

        mstEdges.append(edge)
        addEdgesToPQ(graph, edge.dest, pq, inMST)

def addEdgesToPQ(graph, s, pq, inMST):
    inMST[s] = True
    for edge in graph.adjList[s]:
        pq.insert(edge, edge.weight)
    return
```