# Data Structures and Algorithms

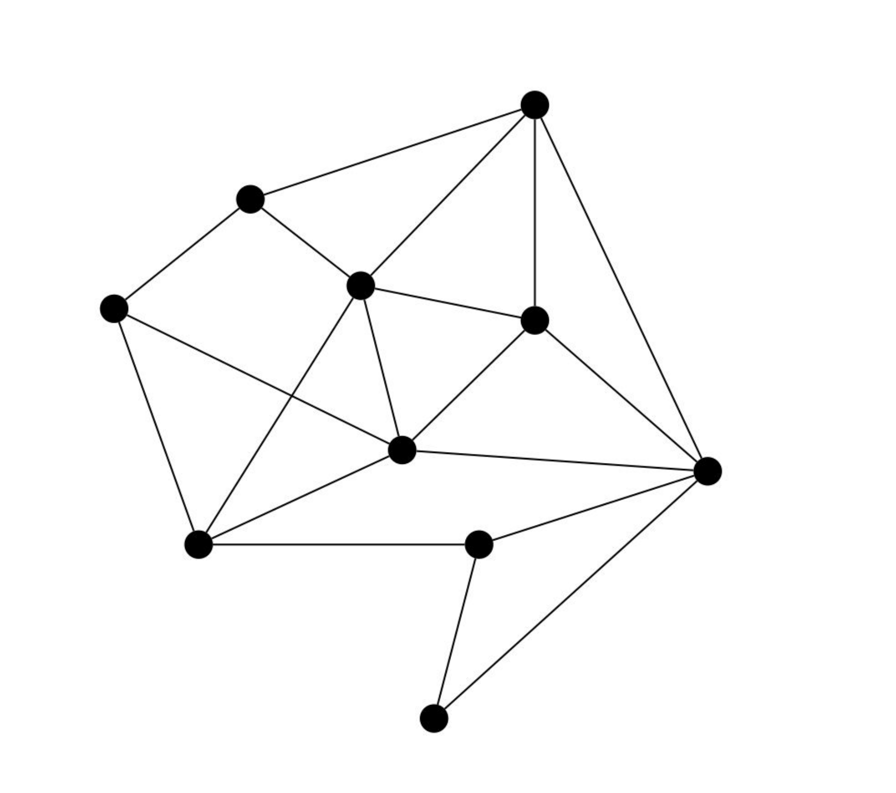
Lab 11
Minimum Spanning Tree. Kruskal’s algorithm

# Agenda

- Recap
- Kruskal’s algorithm theory
- Overview of implementations for
  - Prim’s algorithm
  - Kruskal’s algorithm
- Live Coding session

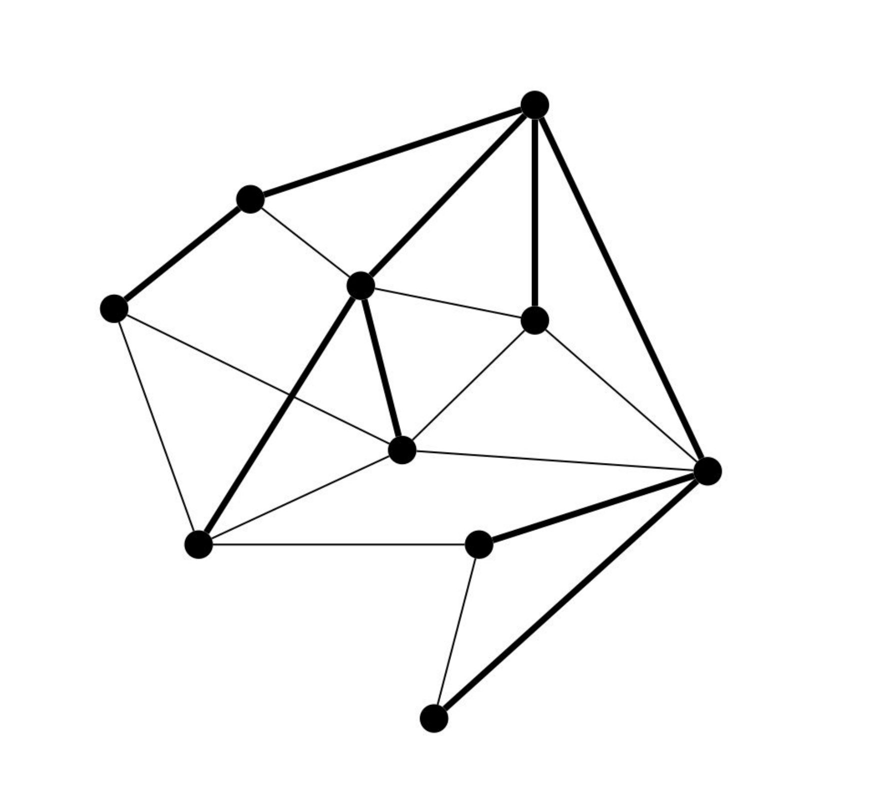
# Minimum spanning tree

- What is MST?



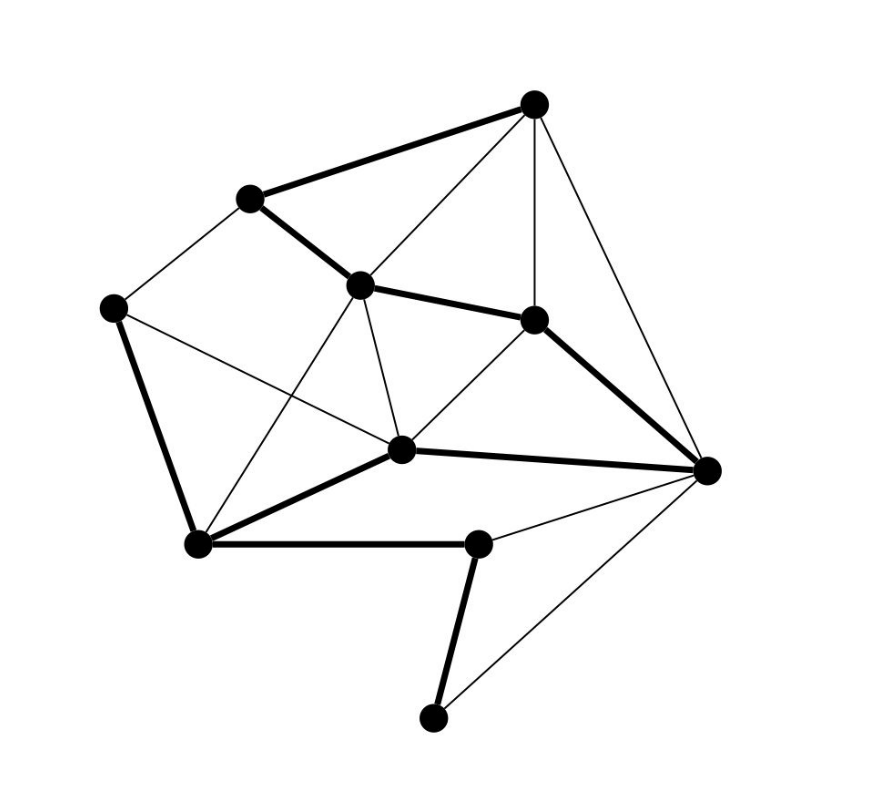
# Minimum spanning tree

- What is MST?



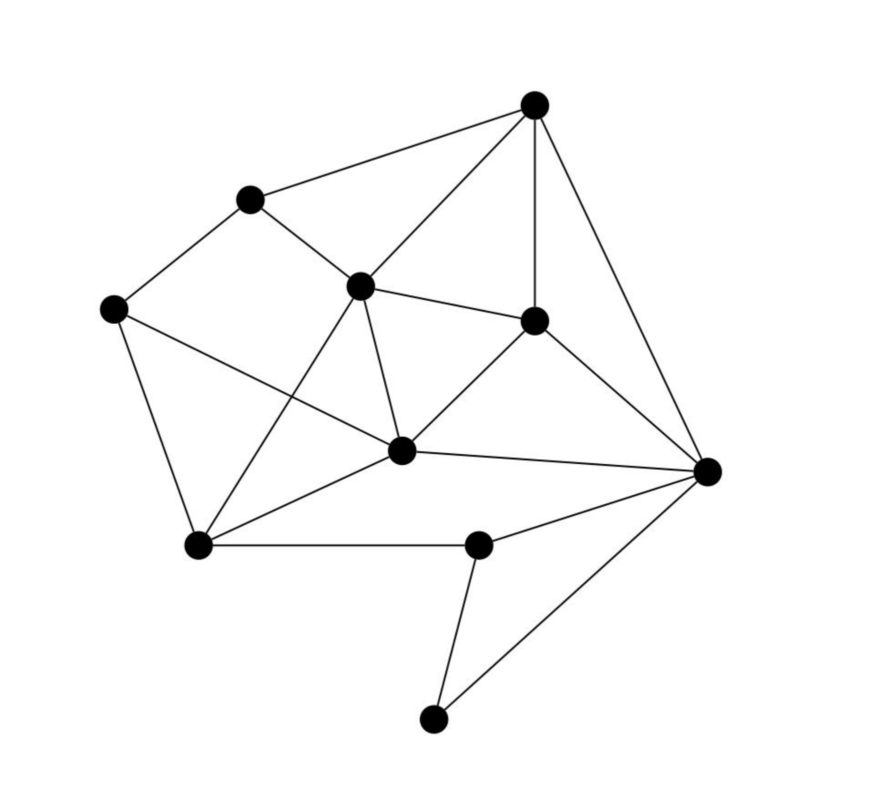
# Minimum spanning tree

- What is MST?
- *graph such that it has the same vertices with the minimum number of edges to connect them*



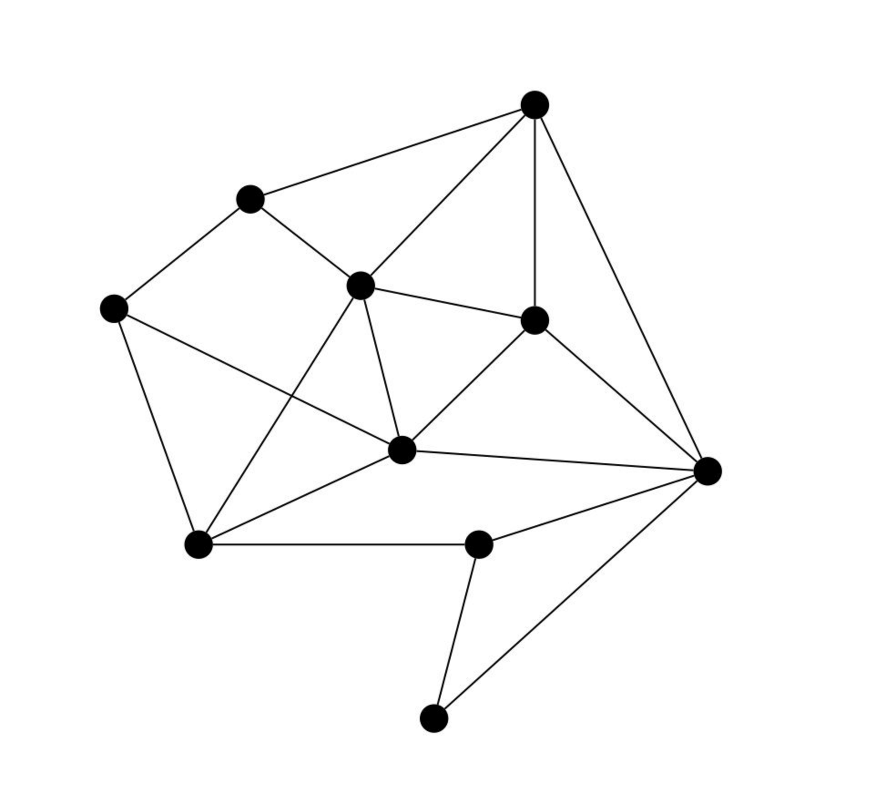
# Minimum spanning tree

- What is MST?
- What is special about MSTs for unweighted graphs?



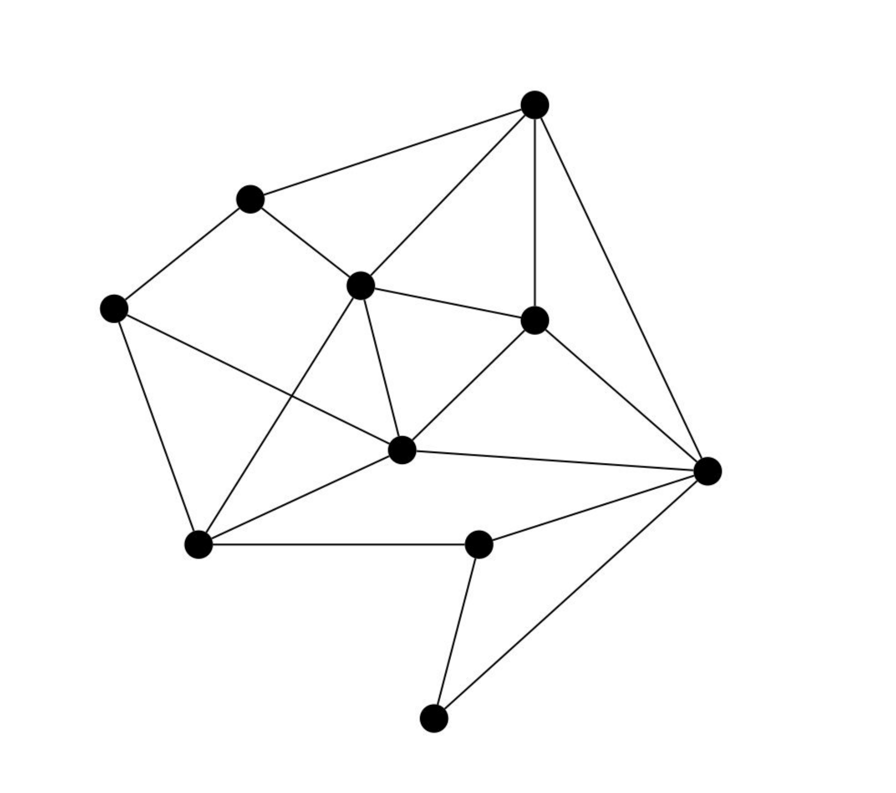
# Minimum spanning tree

- What is MST?
- What is special about MSTs for unweighted graphs?
- *For unweighted graphs, every spanning tree is the MST*



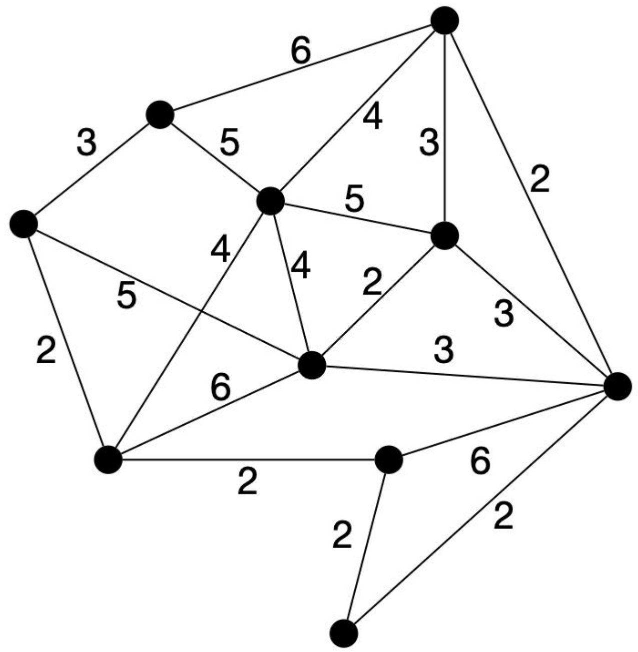
# Minimum spanning tree

- What is MST?
- What is special about MSTs for unweighted graphs?
- How to find an MST for an unweighted graph?



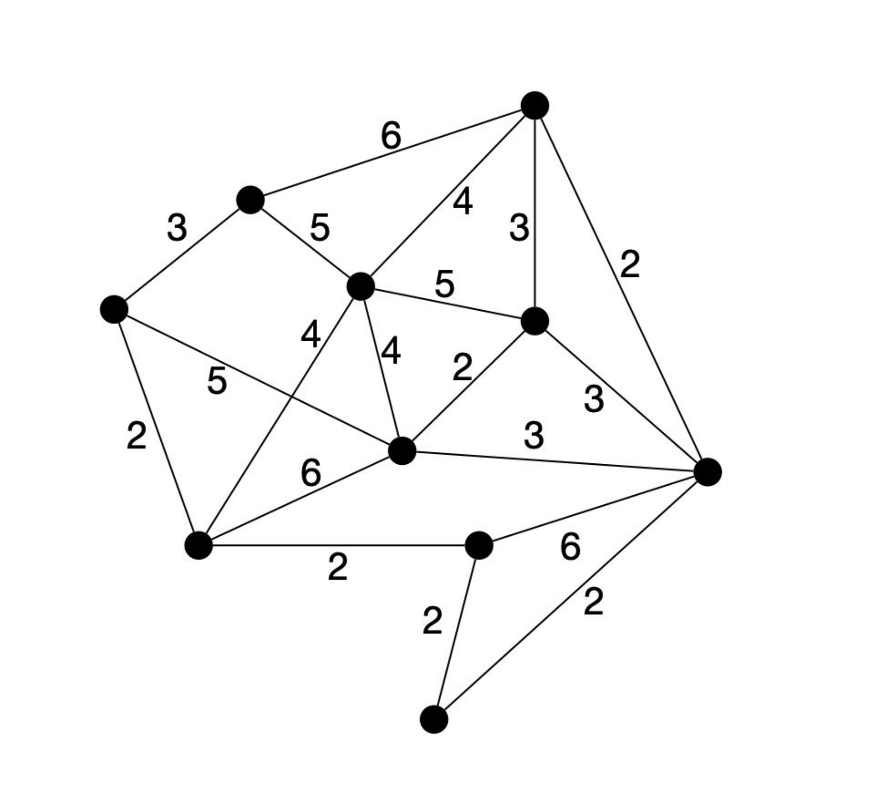
# Minimum spanning tree

- What is MST?
- What is special about MSTs for unweighted graphs?
- How to find an MST for an unweighted graph?
- *DFS and BFS can be used*



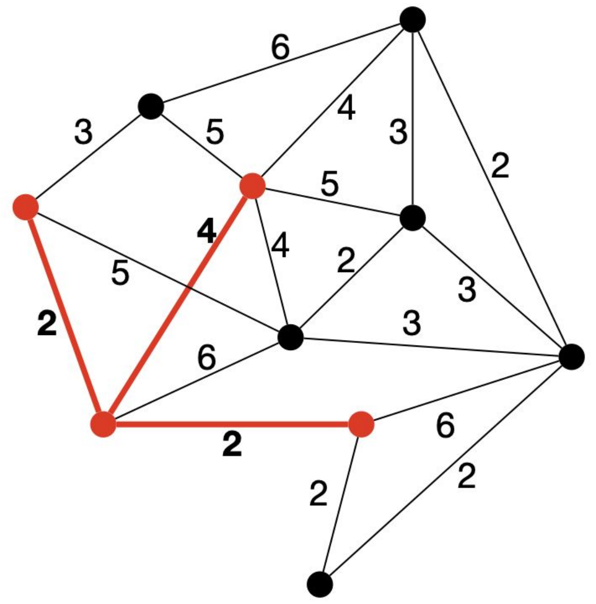
# Minimum spanning tree

- What is MST?
- What is special about MSTs for unweighted graphs?
- How to find an MST for an unweighted graph?
- What is Prim's algorithm?



# Minimum spanning tree

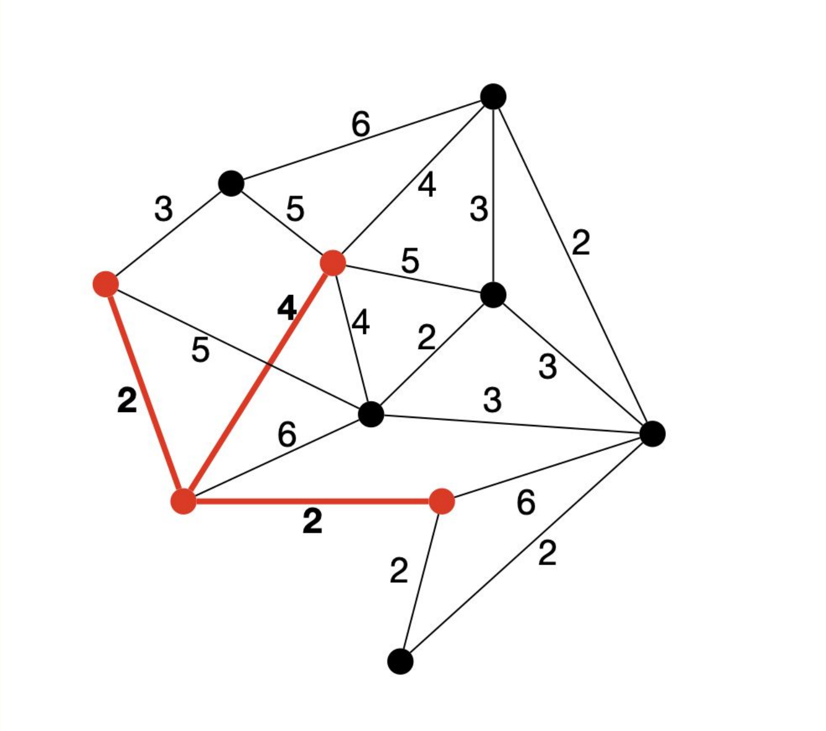
- What is MST?
- What is special about MSTs for unweighted graphs?
- How to find an MST for an unweighted graph?
- What is Prim's algorithm?




1. *Initialize the minimum spanning tree with a vertex chosen at random.*
2. *Find all the edges that connect the tree to new vertices, find the minimum and add it to the tree*
3. *Keep repeating step 2 until we get a minimum spanning tree*

# Minimum spanning tree

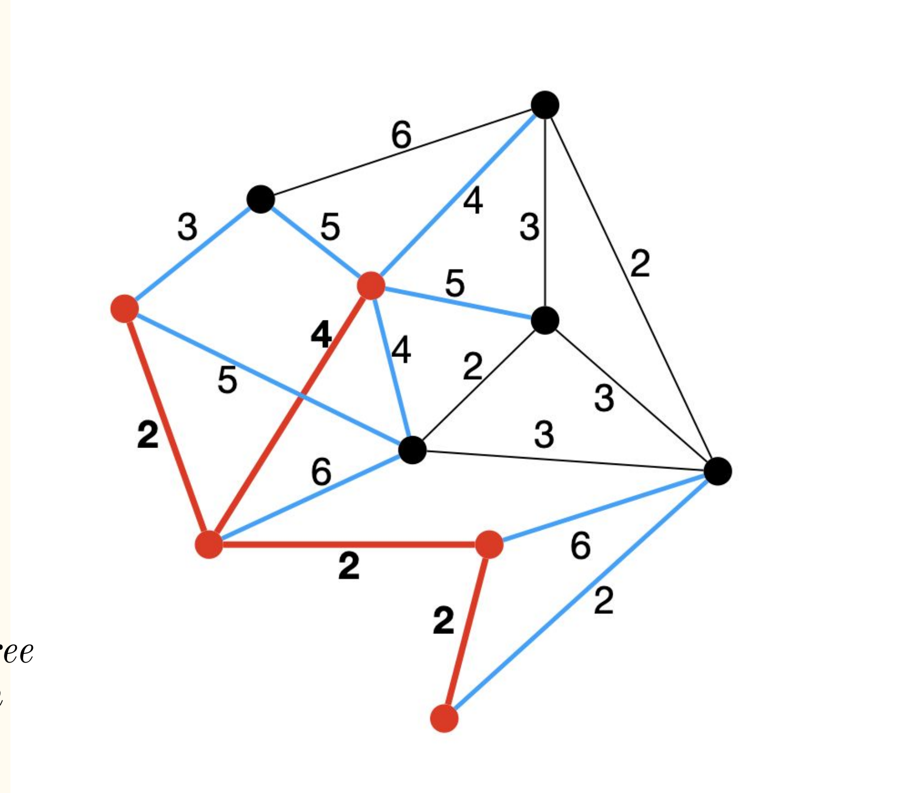
- What is MST?
- What is special about MSTs for unweighted graphs?
- How to find an MST for an unweighted graph?
- What is Prim's algorithm?




1. *Initialize the minimum spanning tree with a vertex chosen at random.*
2. *Find all the edges that connect the tree to new vertices, find the minimum and add it to the tree*
3. *Keep repeating step 2 until we get a minimum spanning tree*

# Minimum spanning tree

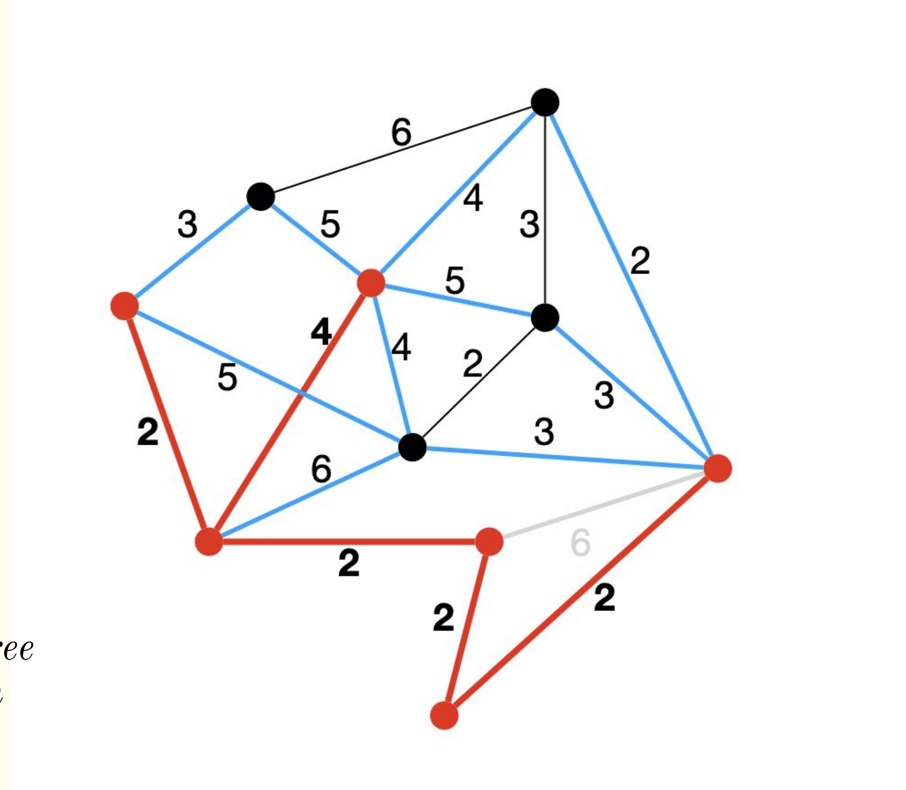
- What is MST?
- What is special about MSTs for unweighted graphs?
- How to find an MST for an unweighted graph?
- What is Prim's algorithm?




1. *Initialize the minimum spanning tree with a vertex chosen at random.*
2. *Find all the edges that connect the tree to new vertices, find the minimum and add it to the tree*
3. *Keep repeating step 2 until we get a minimum spanning tree*

# Minimum spanning tree

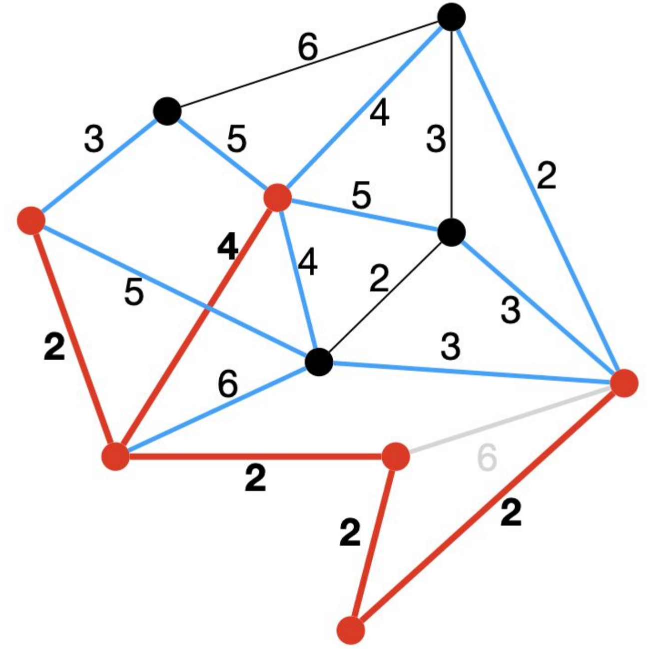
- What is MST?
- What is special about MSTs for unweighted graphs?
- How to find an MST for an unweighted graph?
- What is Prim's algorithm?




1. *Initialize the minimum spanning tree with a vertex chosen at random.*
2. *Find all the edges that connect the tree to new vertices, find the minimum and add it to the tree*
3. *Keep repeating step 2 until we get a minimum spanning tree*

# Minimum spanning tree

- What is MST?
- What is special about MSTs for unweighted graphs?
- How to find an MST for an unweighted graph?
- What is Prim's algorithm?



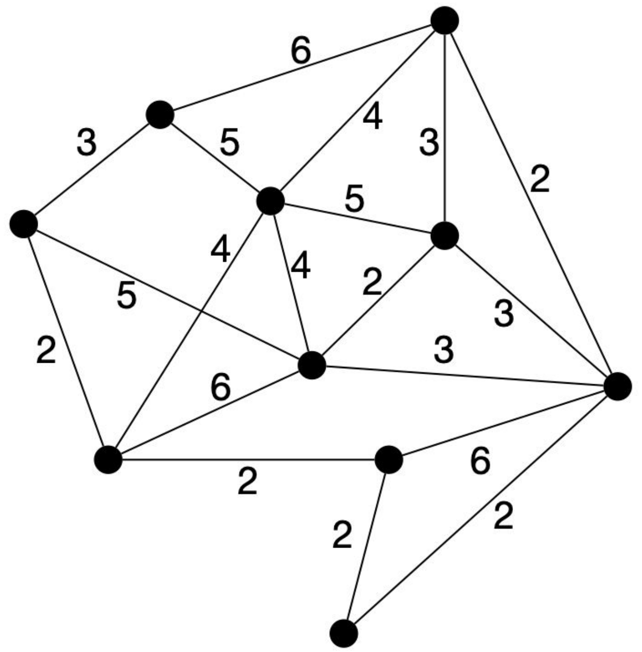

**Idea:** Add one vertex at a time
**Implementation:** Priority Queue of edges

# Kruskal’s algorithm

[idea]

# Kruskal's algorithm

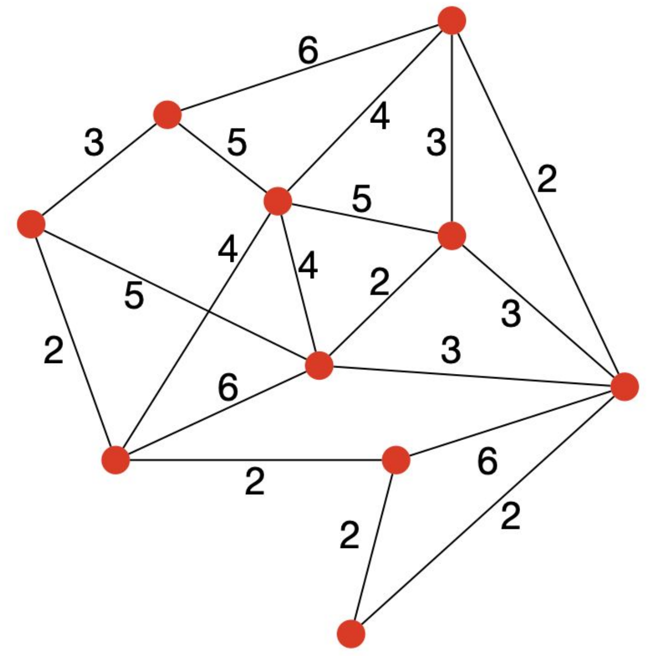
1. Sort all the edges in non-decreasing order of their weight.



# Kruskal's algorithm

1. Sort all the edges in non-decreasing order of their weight.

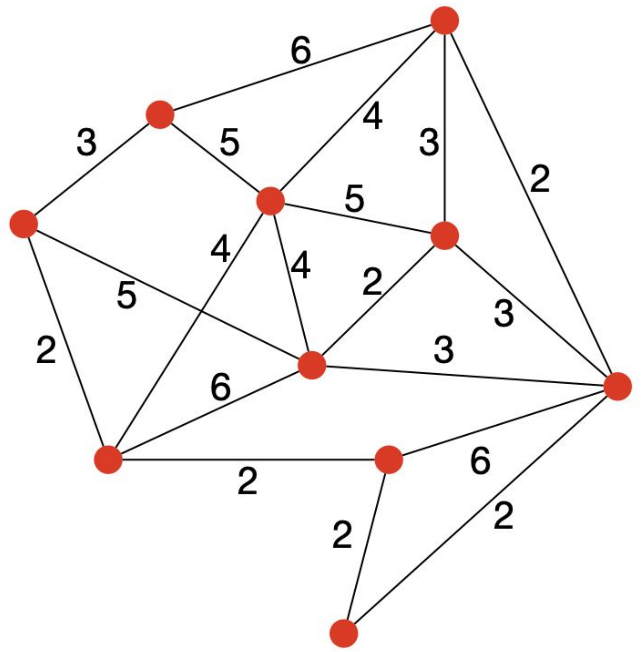
2. Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If cycle is not formed, include this edge. Else, discard it.



# Kruskal’s algorithm

1. Sort all the edges in non-decreasing order of their weight.

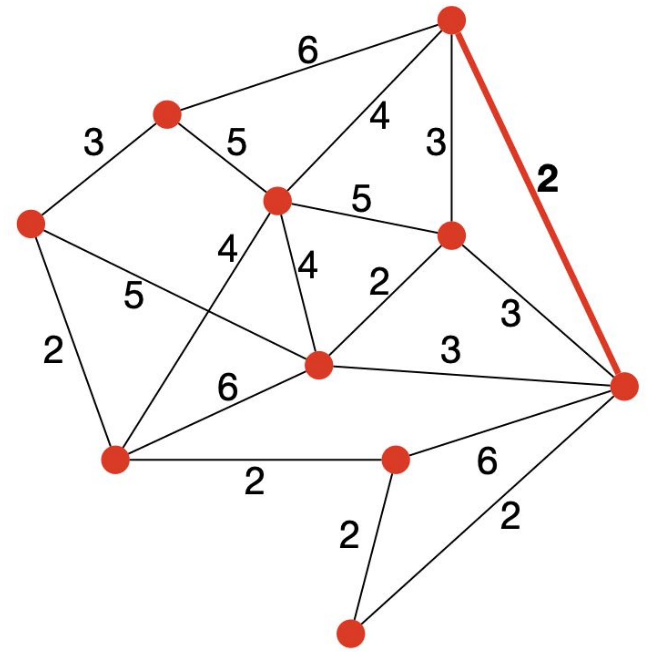
2. Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If cycle is not formed, include this edge. Else, discard it.



# Kruskal’s algorithm

1. Sort all the edges in non-decreasing order of their weight.

2. Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If cycle is not formed, include this edge. Else, discard it.

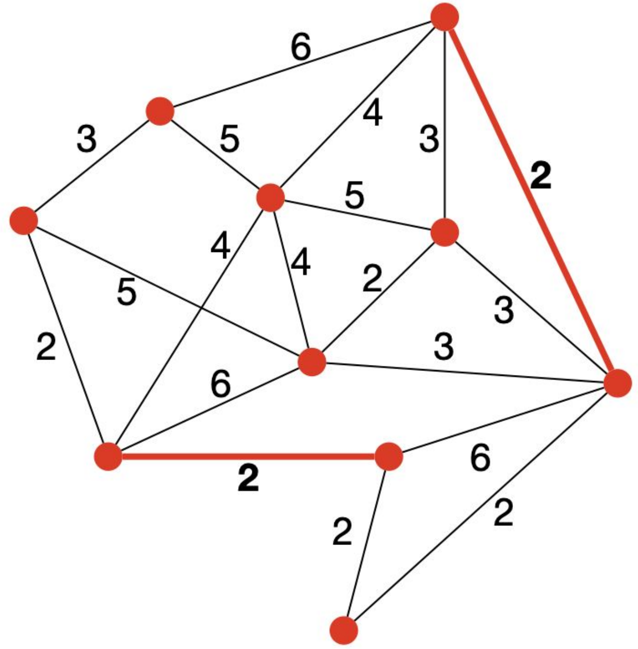


# Kruskal's algorithm

1. Sort all the edges in non-decreasing order of their weight.

2. Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If cycle is not formed, include this edge. Else, discard it.

3. Repeat step#2 until there are (V-1) edges in the spanning tree

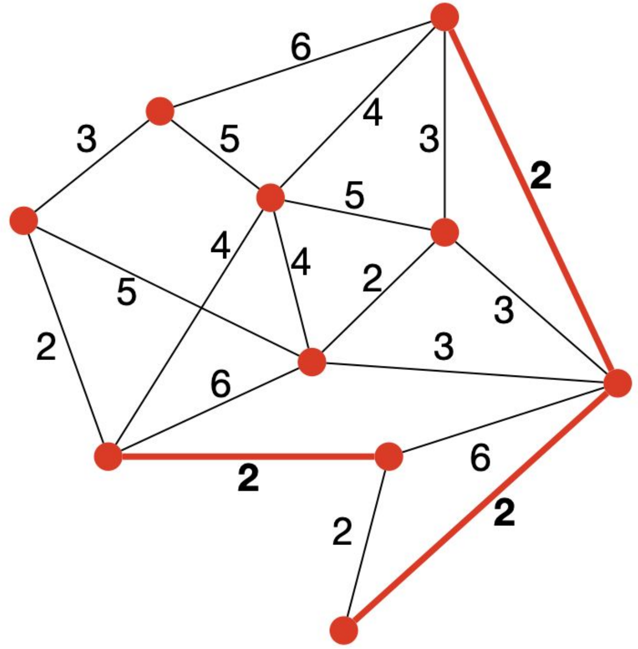


# Kruskal’s algorithm

1. Sort all the edges in non-decreasing order of their weight.

2. Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If cycle is not formed, include this edge. Else, discard it.

3. Repeat step#2 until there are (V-1) edges in the spanning tree

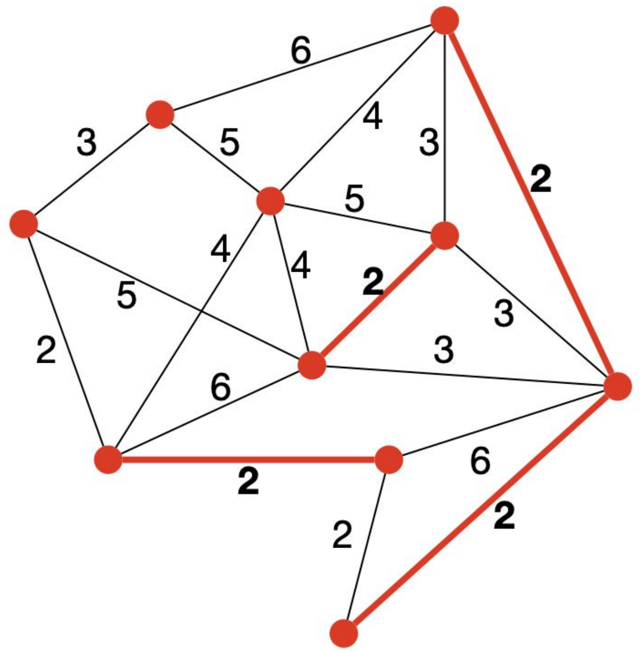


# Kruskal’s algorithm

1. Sort all the edges in non-decreasing order of their weight.

2. Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If cycle is not formed, include this edge. Else, discard it.

3. Repeat step#2 until there are (V-1) edges in the spanning tree

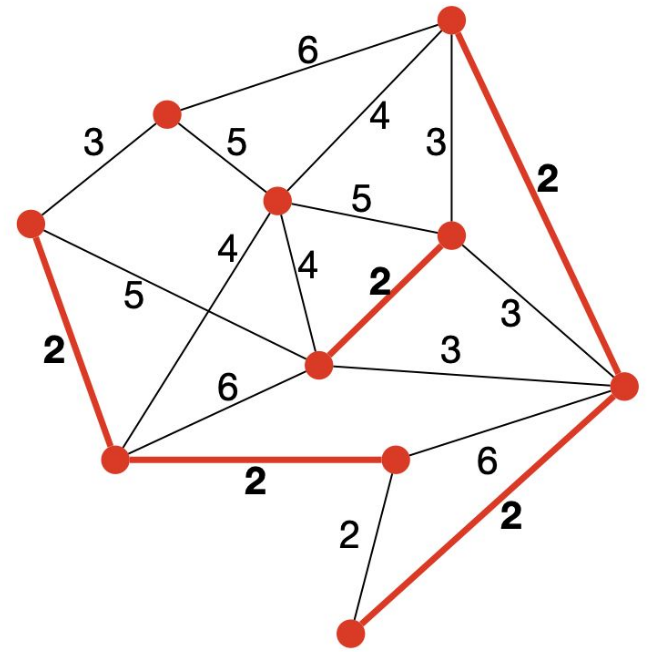


# Kruskal's algorithm

1. Sort all the edges in non-decreasing order of their weight.

2. Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If cycle is not formed, include this edge. Else, discard it.

3. Repeat step#2 until there are (V-1) edges in the spanning tree

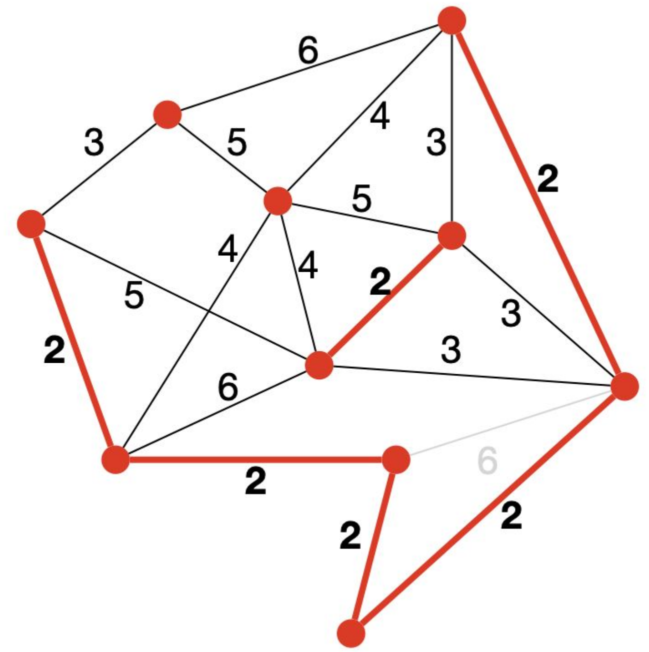


# Kruskal's algorithm

1. Sort all the edges in non-decreasing order of their weight.

2. Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If cycle is not formed, include this edge. Else, discard it.

3. Repeat step#2 until there are (V-1) edges in the spanning tree

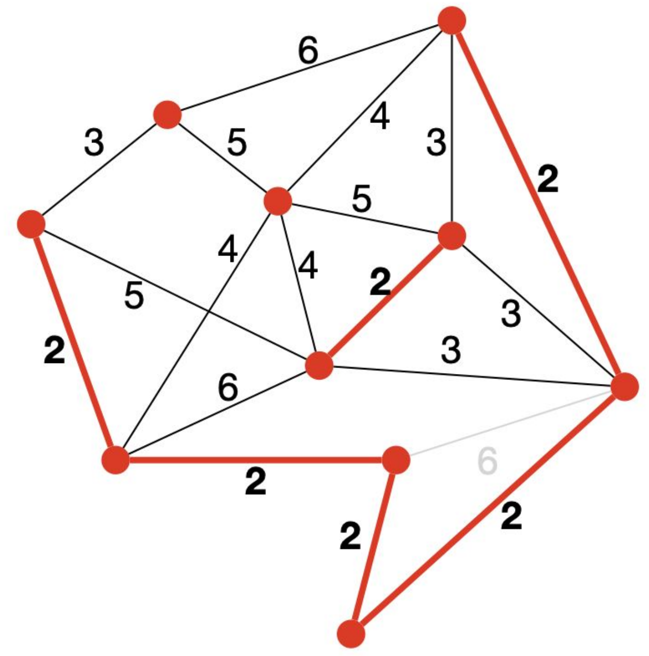


# Kruskal's algorithm

1. Sort all the edges in non-decreasing order of their weight.

2. Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If cycle is not formed, include this edge. Else, discard it.

3. Repeat step#2 until there are (V-1) edges in the spanning tree

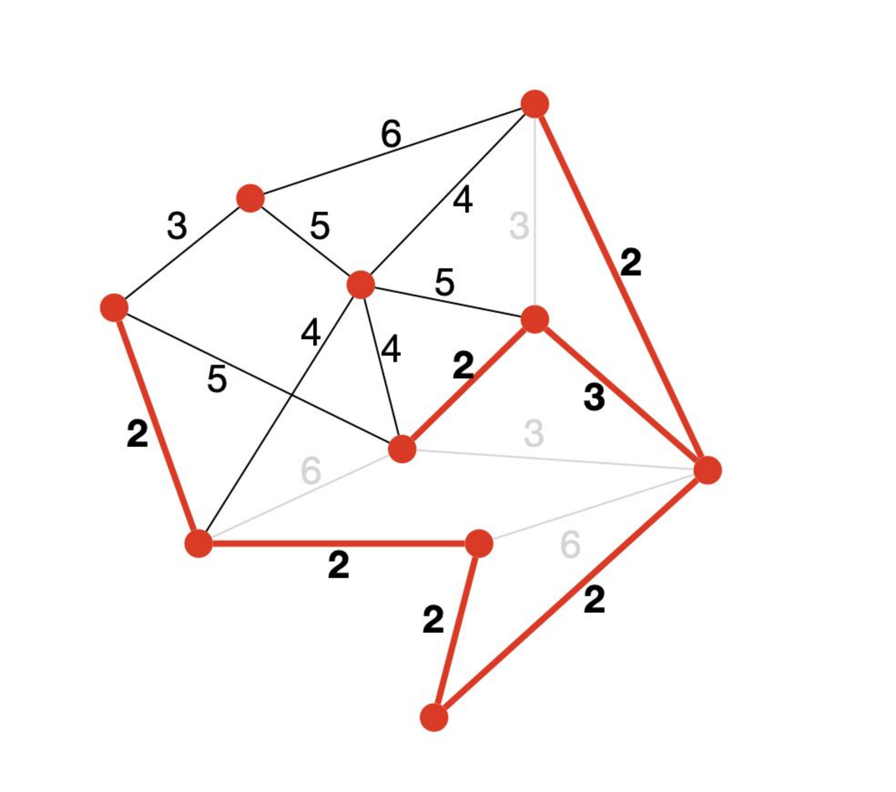


# Kruskal's algorithm

1. Sort all the edges in non-decreasing order of their weight.

2. Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If cycle is not formed, include this edge. Else, discard it.

3. Repeat step#2 until there are (V-1) edges in the spanning tree

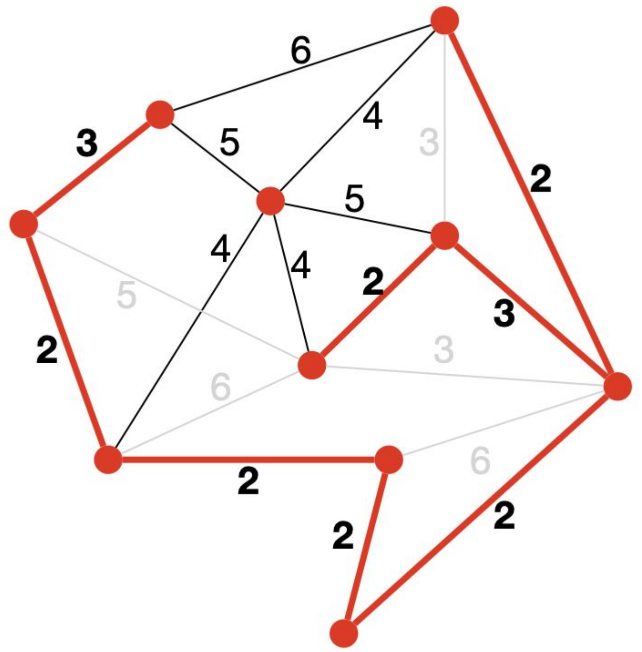


# Kruskal’s algorithm

1. Sort all the edges in non-decreasing order of their weight.

2. Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If cycle is not formed, include this edge. Else, discard it.

3. Repeat step#2 until there are (V-1) edges in the spanning tree

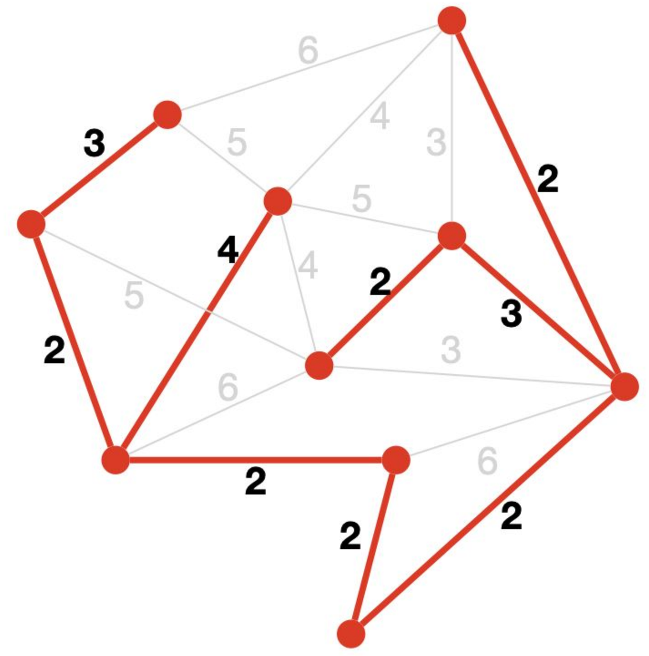


# Kruskal's algorithm

1. Sort all the edges in non-decreasing order of their weight.

2. Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If cycle is not formed, include this edge. Else, discard it.

3. Repeat step#2 until there are (V-1) edges in the spanning tree

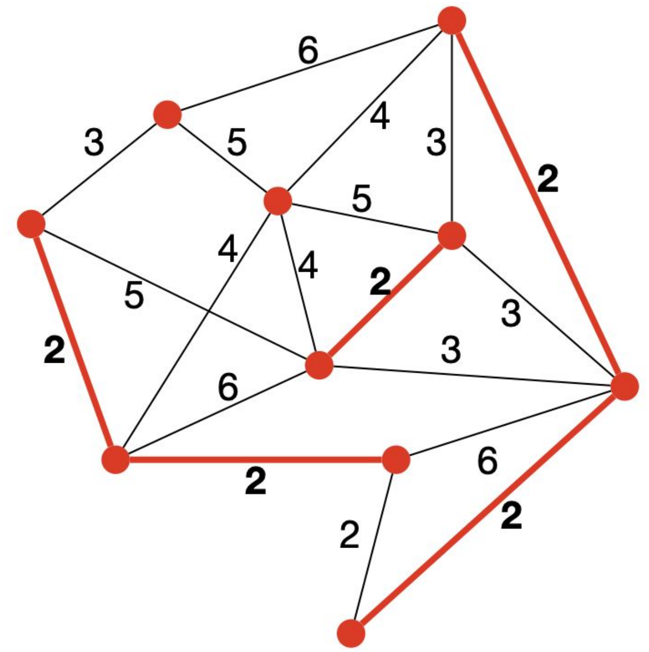


# Kruskal’s algorithm

## [implementation]

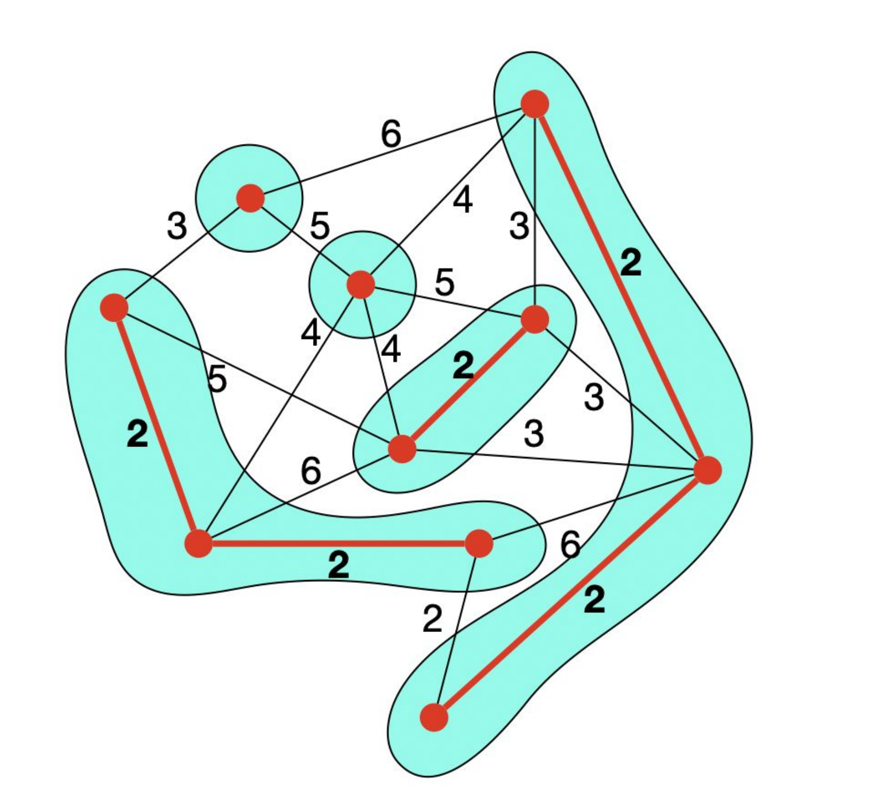
# Kruskal's algorithm

- How can we keep track of which vertices belong to which trees?



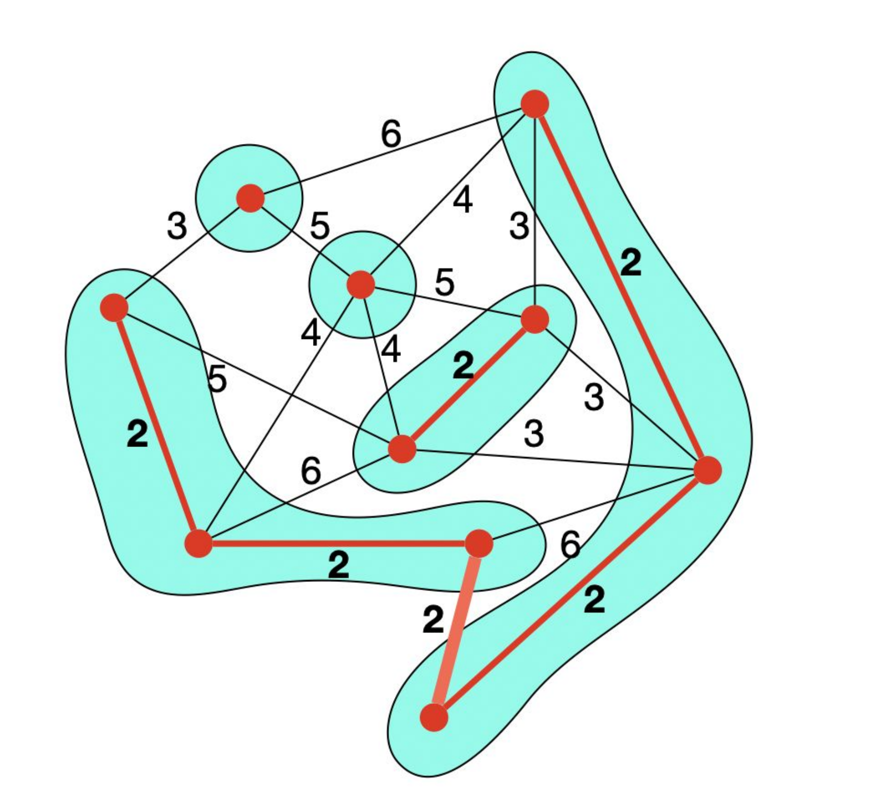
# Kruskal's algorithm

- How can we keep track of which vertices belong to which trees?
- At any point in time the set of all vertices is split into a disjoint set of trees.



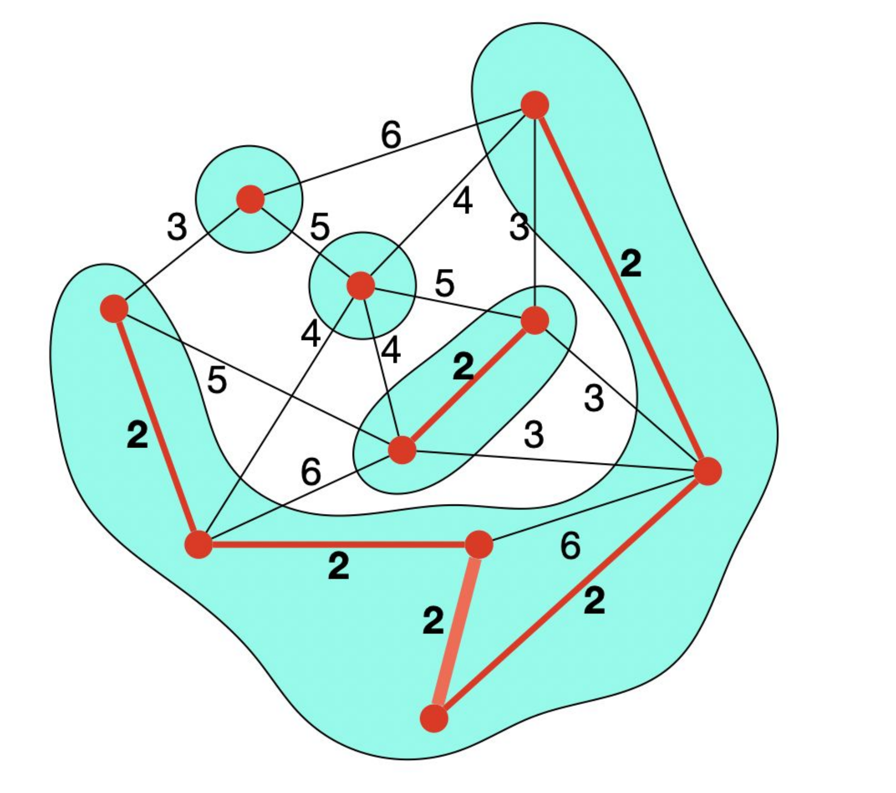
# Kruskal’s algorithm

- How can we keep track of which vertices belong to which trees?
- At any point in time the set of all vertices is split into a disjoint set of trees.
- When we connect two trees, we merge (union) the disjoint sets of trees.



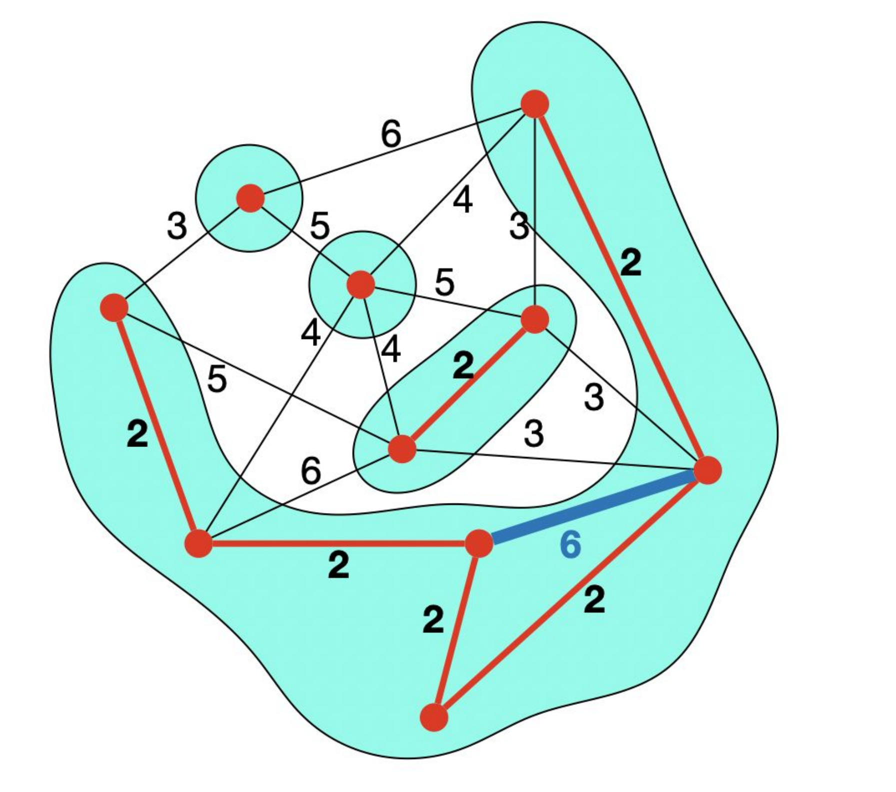
# Kruskal's algorithm

- How can we keep track of which vertices belong to which trees?
- At any point in time the set of all vertices is split into a disjoint set of trees.
- When we connect two trees, we merge (union) the disjoint sets of trees.



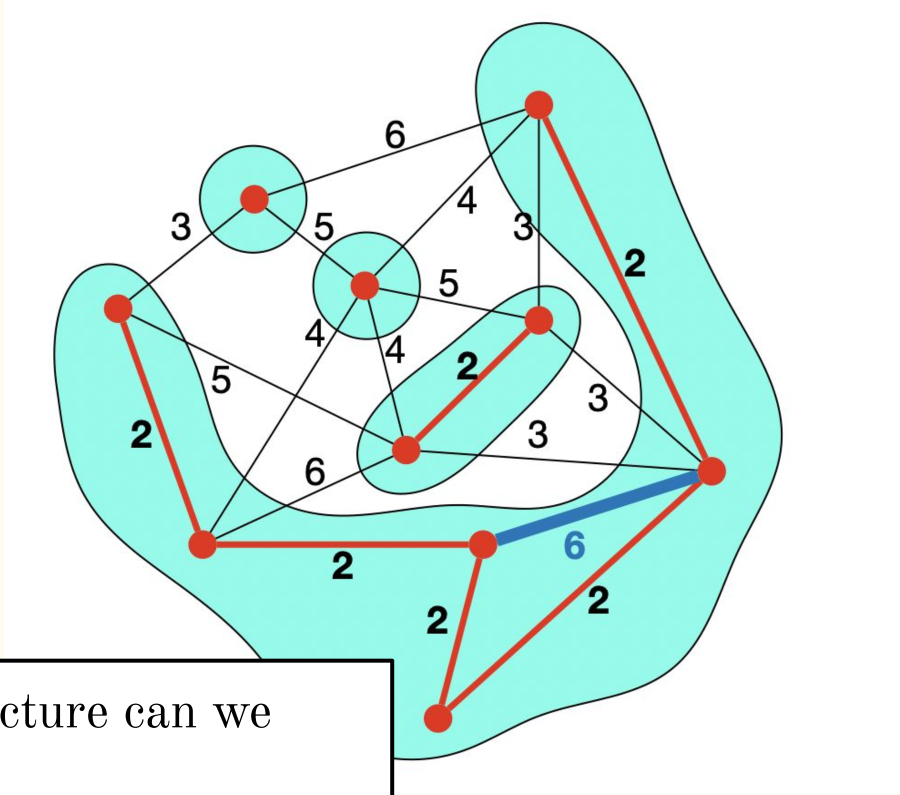
# Kruskal’s algorithm

- How can we keep track of which vertices belong to which trees?
- At any point in time the set of all vertices is split into a disjoint set of trees.
- When we connect two trees, we merge (union) the disjoint sets of trees.
- To avoid loops we need to understand if two vertices come from the same tree.



# Kruskal’s algorithm

- How can we keep track of which vertices belong to which trees?
- At any point in time the set of all vertices is split into a disjoint set of trees.
- When we connect two trees, we merge (union) the disjoint sets of trees.
- To avoid loops we need to understand if two vertices come from the same tree.




**Question:** What kind of data structure can we use?

# Disjoint Sets

```java
// A collection of integers from 0 to N split into disjoint sets.
interface DisjointIntSets {
    int find(int i);              // find representative of a subset
    void union(int i, int j);     // merge two disjoint subset
}
```

We can keep track of the subsets in a 1D array, let's call it ***L[].***

# Disjoint Sets

**Algorithm 1.** Possible implementation of *Union-Find*. $L$ is the *Union-Find* array, $a$ and $b$ are both array indexes and pixel identifiers.

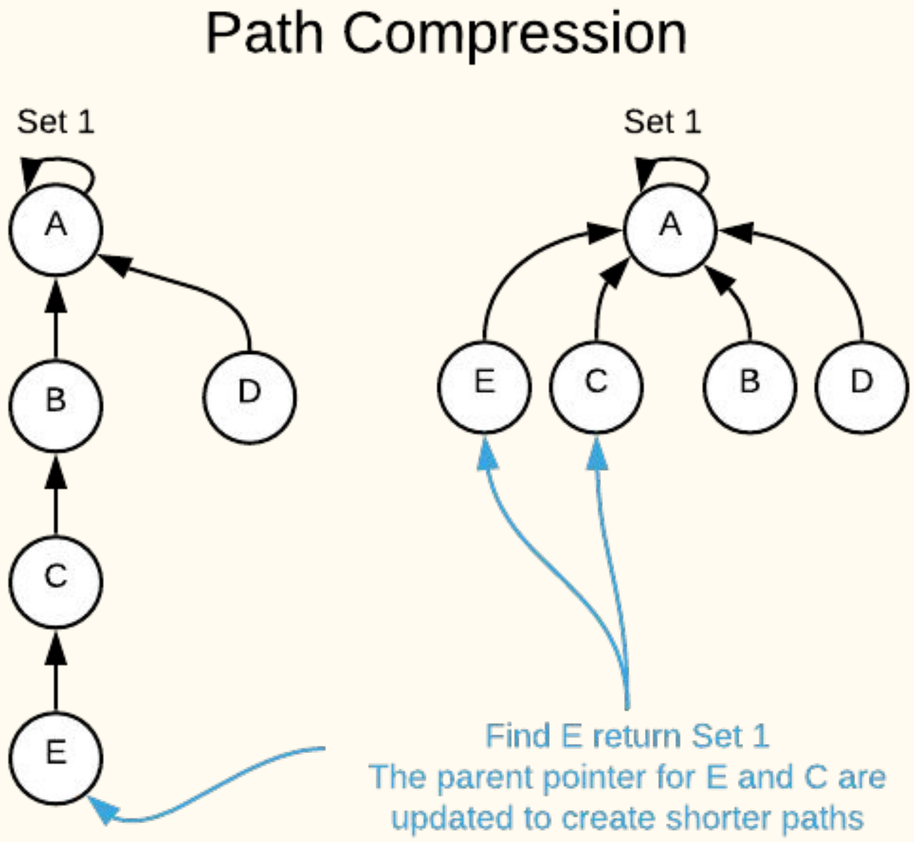
**function** FIND($L$, $a$)
  **while** $L[a] \neq a$ **do**
    $a \leftarrow L[a]$
  **return** $a$

**procedure** UNION($L$, $a$, $b$)
  $a \leftarrow \texttt{Find}(L, a)$
  $b \leftarrow \texttt{Find}(L, b)$
  **if** $a < b$ **then**
    $L[b] \leftarrow a$
  **else if** $b < a$ **then**
    $L[a] \leftarrow b$

```
Let there be 4 elements 0, 1, 2, 3

Initially, all elements are single element subsets.
0 1 2 3

Do Union(0, 1)
   1   2   3
  /
 0

Do Union(1, 2)
     2   3
    /
   1
  /
 0

Do Union(2, 3)
         3
        /
      2
     /
   1
  /
 0
```

```
0   1   2    <----- 2 is made parent of 1 (2 is now representative of subset {0, 1, 2}
1   2  -1
```

# Disjoint Sets



# Disjoint Sets (path compression)

```
MAKE-SET(x)
  x.p = x
  x.rank = 0

UNION(x, y)
  LINK(FIND-SET(x), FIND-SET(y))

LINK(x, y)
  if x.rank > y.rank
      y.p = x
  else x.p = y
      if x.rank == y.rank
          y.rank = y.rank + 1

FIND-SET(x)
  if x ≠ x.p
      x.p = FIND-SET(x.p)
  return x.p
```

```
Let us see the above example with union by rank
Initially, all elements are single element subsets.
0 1 2 3

Do Union(0, 1)
   1   2   3
  /
 0

Do Union(1, 2)
   1    3
 /  \
0    2

Do Union(2, 3)
    1
 /  |  \
0   2   3
```

# Live Coding

## Implementing Kruskal's algorithm

See You next week!