# Minimum Spanning Trees

CSE 2320 – Algorithms and Data Structures
Alexandra Stefan
University of Texas at Arlington

These slides are based on CLRS and “Algorithms in C” by R. Sedgewick

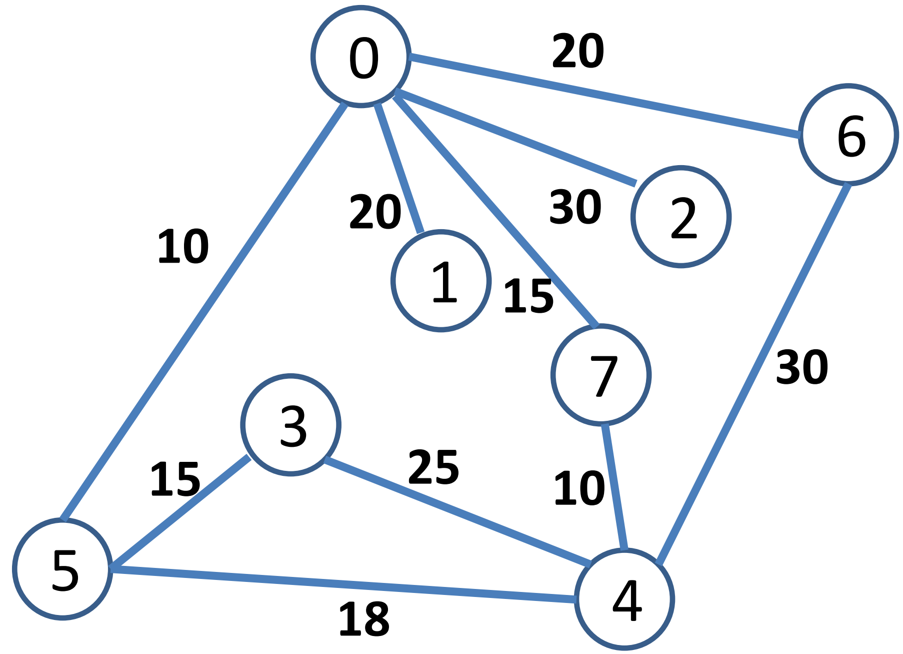
Last updated: 5/6/18

# Weighted Graphs: G,w

Each edge has a weight.

Examples:

- A transportation network (roads, railroads, subway). The weight of each road can be:
  - Length.
  - Expected time to traverse.
  - Expected cost to build.
- A computer network - the weight of each edge (direct link) can be:
  - Latency.
  - Expected cost to build.




Problem: find edges that connect all nodes with minimum total cost.

E.g. , you want to connect all cities to minimize highway cost, but do not care about duration to get from one to the other (e.g. ok if route from A to B goes through most of the other cities).

Solution: Minimum Spanning Tree (MST)

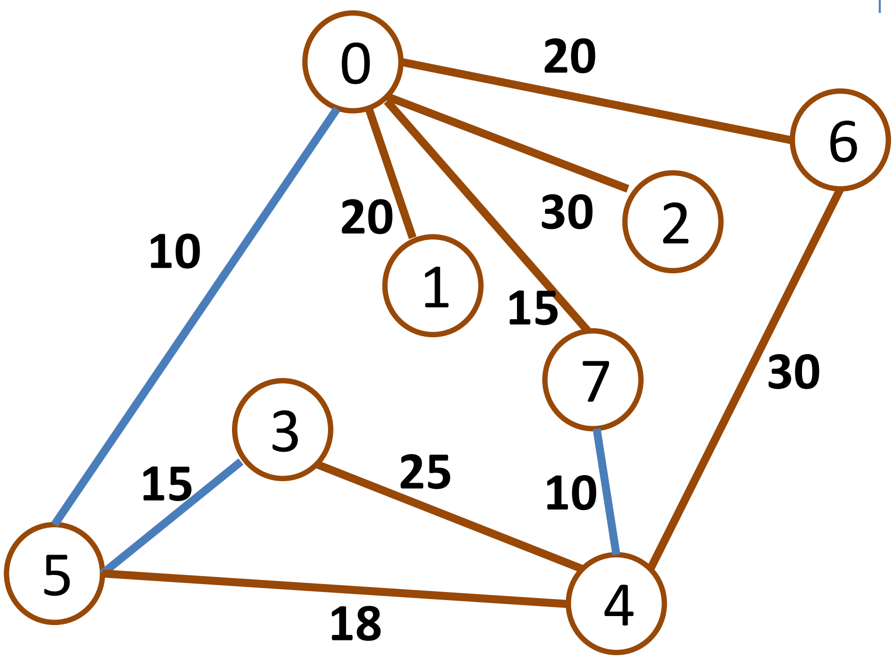
# Spanning Tree

- A spanning tree is a tree that connects all vertices of the graph.

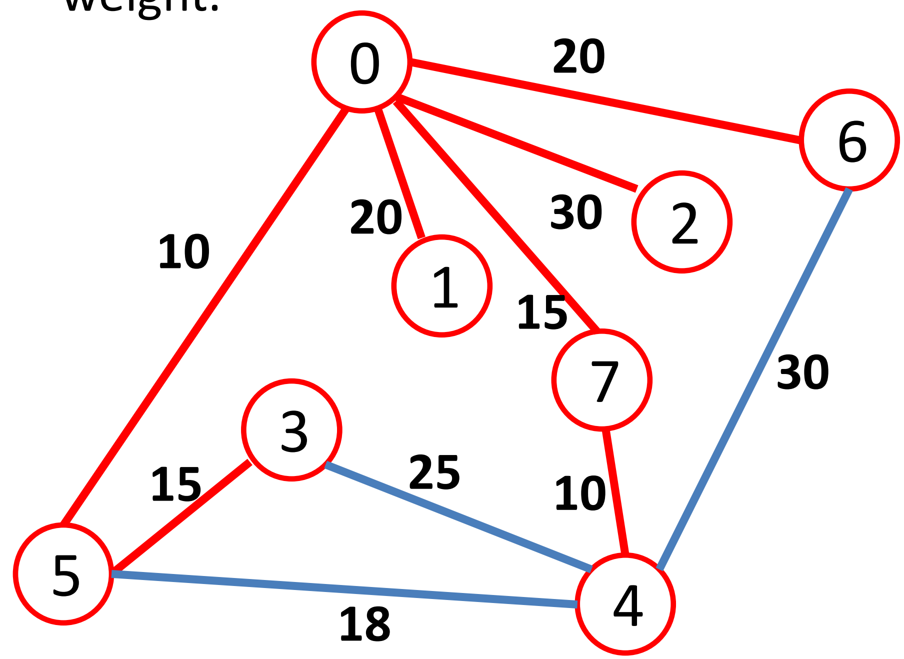
- The weight/cost of a spanning tree is the sum of weights of its edges.

## - Minimum spanning tree (MST)

- Is a **Spanning Tree**: connects all vertices of the graph.
- Has the **smallest total weight** of edges.
- It is **not unique**: Two different spanning trees may have the (same) minimum weight.

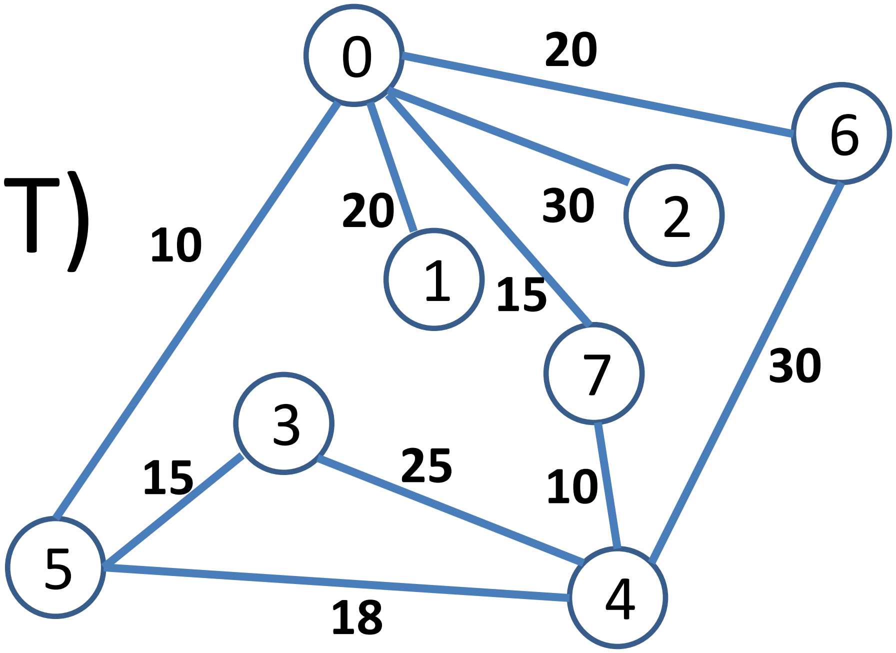



Weight:20+15+30+20+30+25+18 = 158

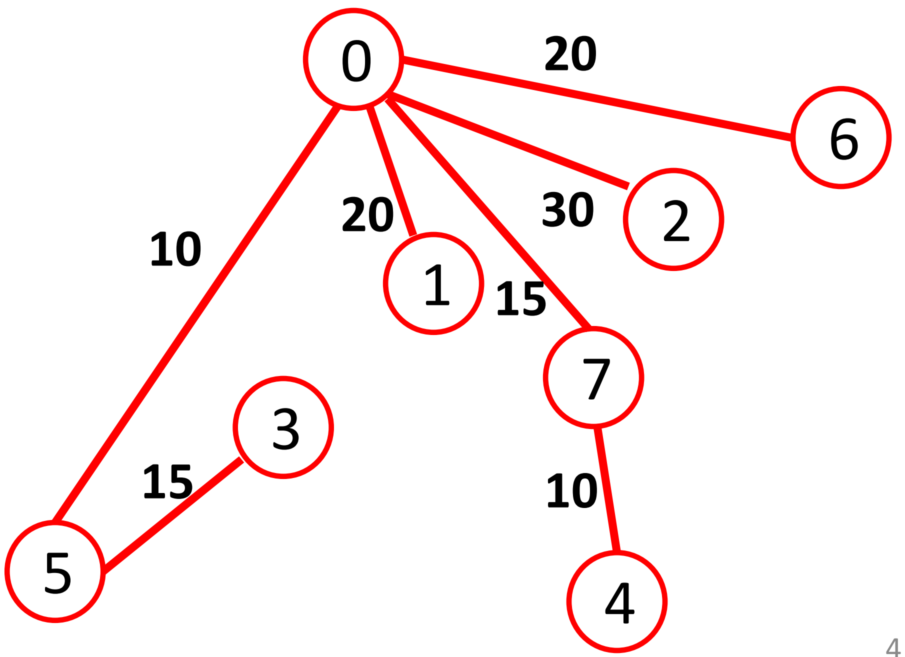



Weight:10+20+15+30+20+10+15 = 120

# Minimum-Cost Spanning Tree (MST)




- Assume that the graph is:
  - connected
  - undirected
  - edges can have negative weights.




- Warning: later in the course (when we discuss Dijkstra's algorithm) we will make the opposite assumptions:
  - Allow directed graphs.
  - Not allow negative weights.

# Kruskal's Algorithm

# Kruskal's Algorithm

- Uses a 'forest' (a set of trees).
  - Initially, each vertex in the graph is its own tree.
  - Keep merging trees together, until end up with a single tree.
    - Pick the smallest edge that connects two different trees.
- Time complexity: O(ElgV)  Note: ElgE = $O(E\lg E^2)$ = O(2ElgV) = O(ElgV)

  Depends on: 1. *Sort edges* (with what method?) or use a *Min-Heap*? 2. *Find-Set* and *Union*=> O(lgV) (with union-by-rank or weighted-union) – See the Union-Find slides for more information.

```
MST_Kruskal(G,w)  // N = |V|      -----> O(ElgV)
   A = empty set of edges
   int id[N], sz[N]
Θ(V) <-- For v = 0 -> N-1
      id(v) = v;  sz(v)=1
(mergesort)
Θ(ElgE) <-- Sort edges of G in increasing order of weight
O(ElgV)
   For each edge (u,v) in increasing order of weight  ---> O(E)
      if Find_Set(u,id) == Find_Set(v,id)  ------> Θ(lgV)
         add edge (u,v) to A
         union(u,v,id,sz)      ------> Θ(lgV)      (Θ(1) when given the representatives)
   return A
```

# Kruskal's Algorithm

*Uses a 'forest' (a set of trees).*

- *Initially, each vertex in the graph is its own tree.*
- *Keep merging trees together, until end up with a single tree.*
    - *Pick the smallest edge that connects two different trees*

- The abstract description is simple, but the implementation affects the runtime.
    - How to maintain the forest
        - See the Union-Find algorithm.
    - How to find the smallest edge connecting two trees:
        - Sort edges: Y/N?
        - Put edges in a min-heap?

# Kruskal's Algorithm

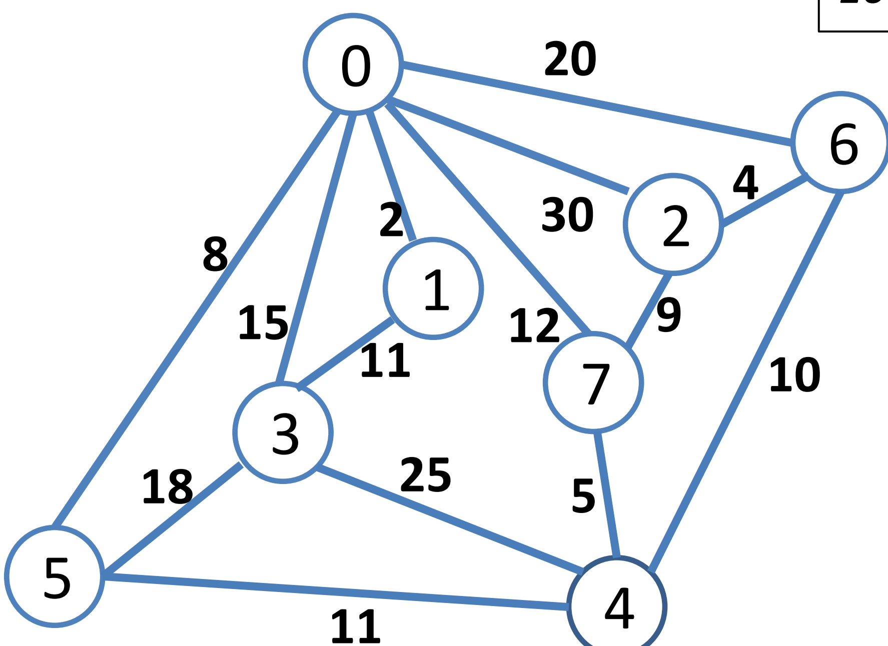
Idea:

- Initially, each vertex in the graph is its own tree.

- Keep merging trees together, until end up with a single tree (pick the smallest edge connecting different trees).

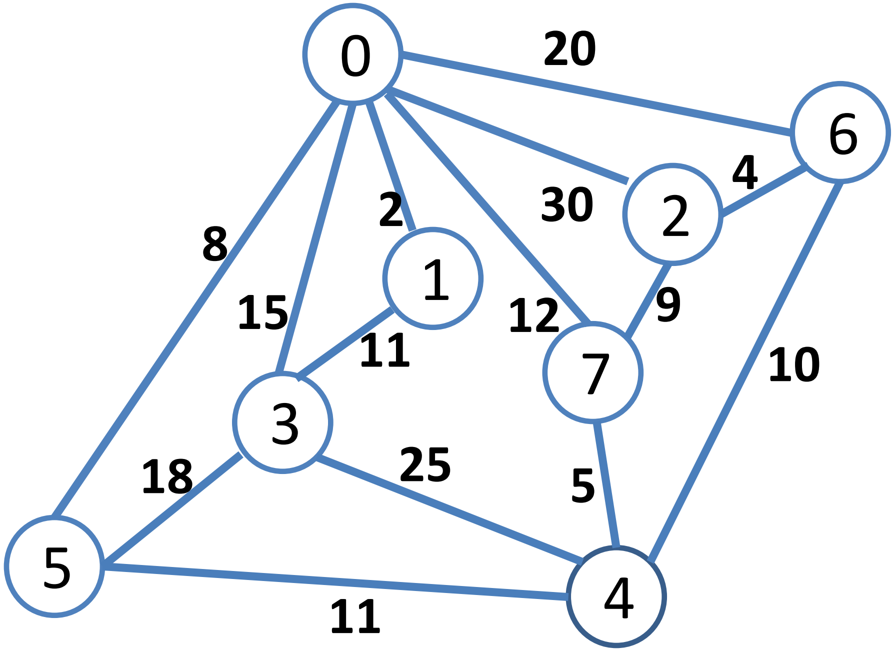
See Union-Find slides as well.

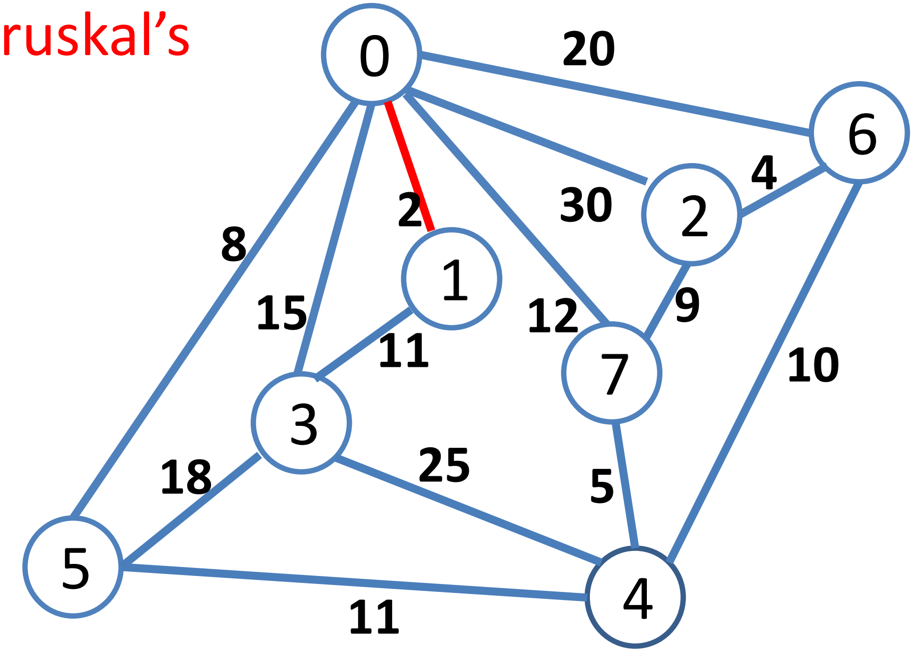
```
MST_Kruskal(G,w)  // N = |V|
A = empty set of edges
int id[N], sz[N]
For v = 0 -> N-1
     id(v) = v;  sz(v)=1
Sort edges of G in increasing order of weight
For each edge (u,v) in increasing order of weight
     if Find_Set(u,id) == Find_Set(v,id)
          add edge (u,v) to A
          union(u,v,id,sz)
return A
```

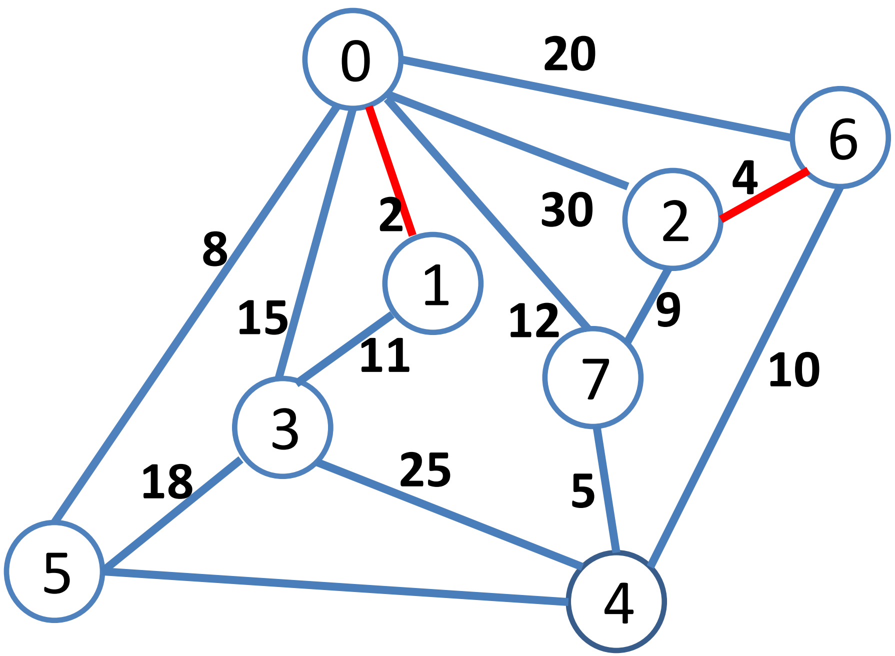


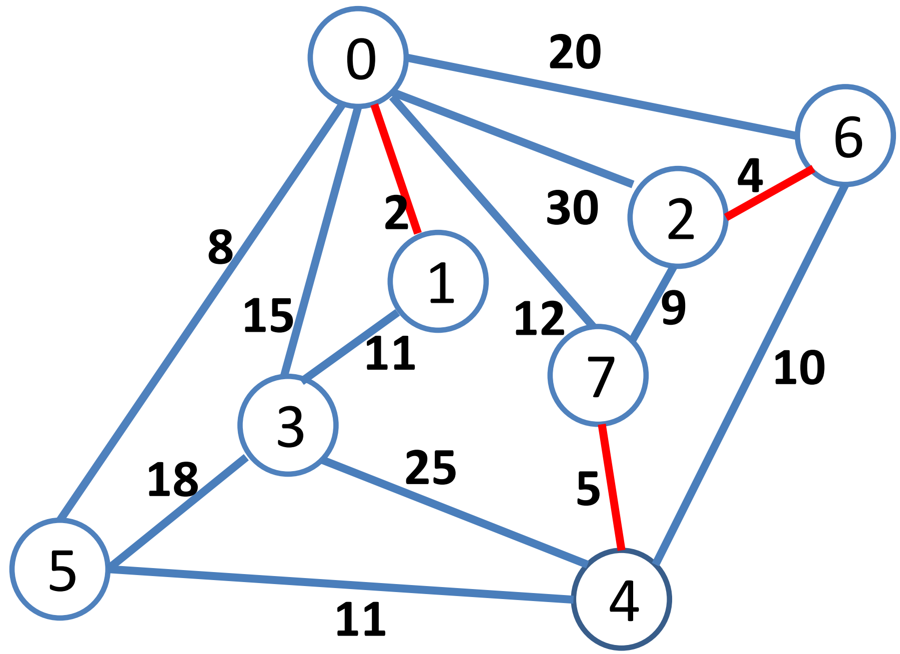

| u, v, weight |
|---|
| |
| |
| |
| |
| |
| |
| |

# Kruskal’s

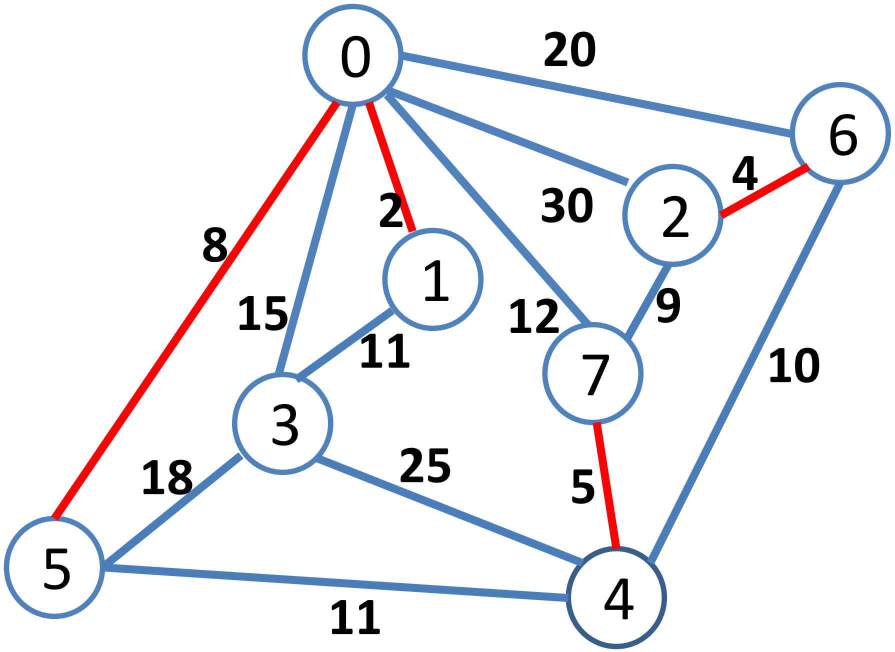

0
1
2
3
4
5
6
7
20
30
4
2
8
15
11
12
9
10
18
25
5
11

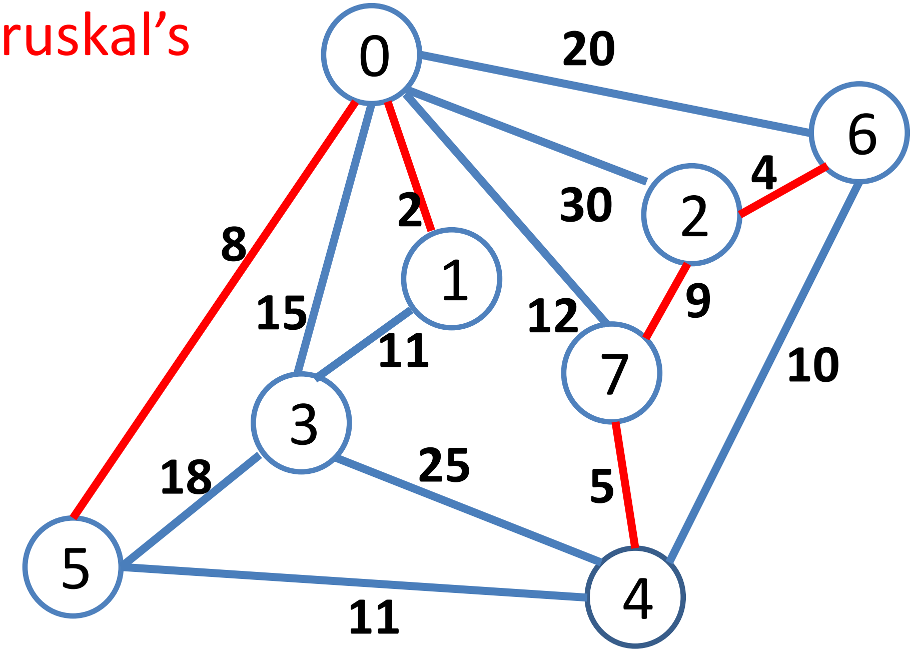

0
1
2
3
4
5
6
7
20
30
4
2
8
15
11
12
9
10
18
25
5
11

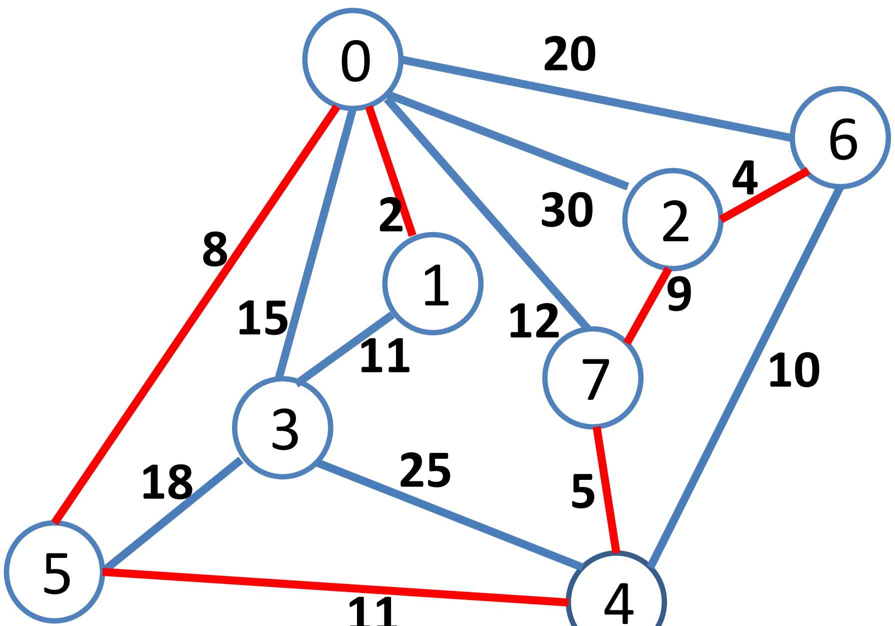

0
1
2
3
4
5
6
7
20
30
4
2
8
15
11
12
9
10
18
25
5

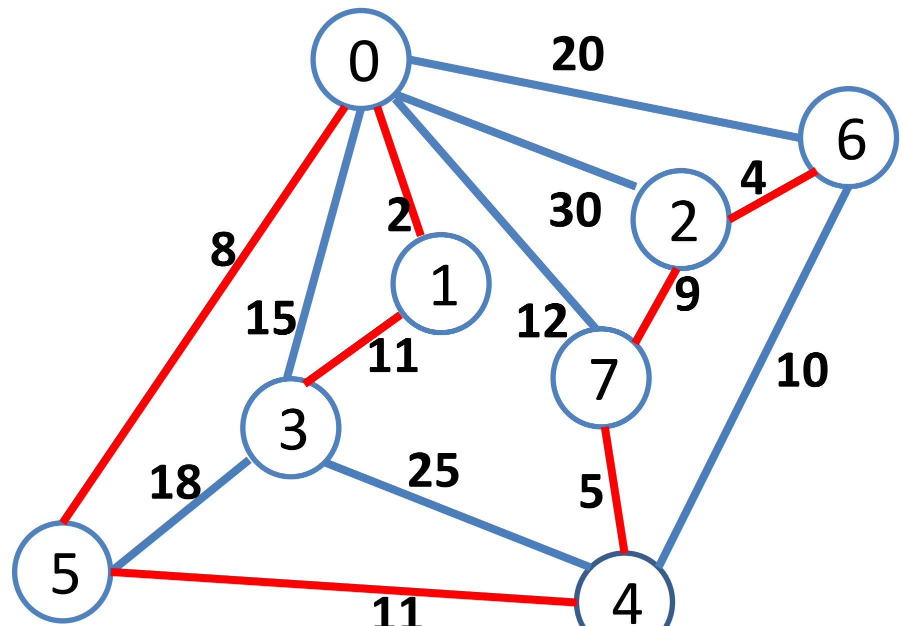

0
1
2
3
4
5
6
7
20
30
4
2
8
15
11
12
9
10
18
25
5
11

# Kruskal's



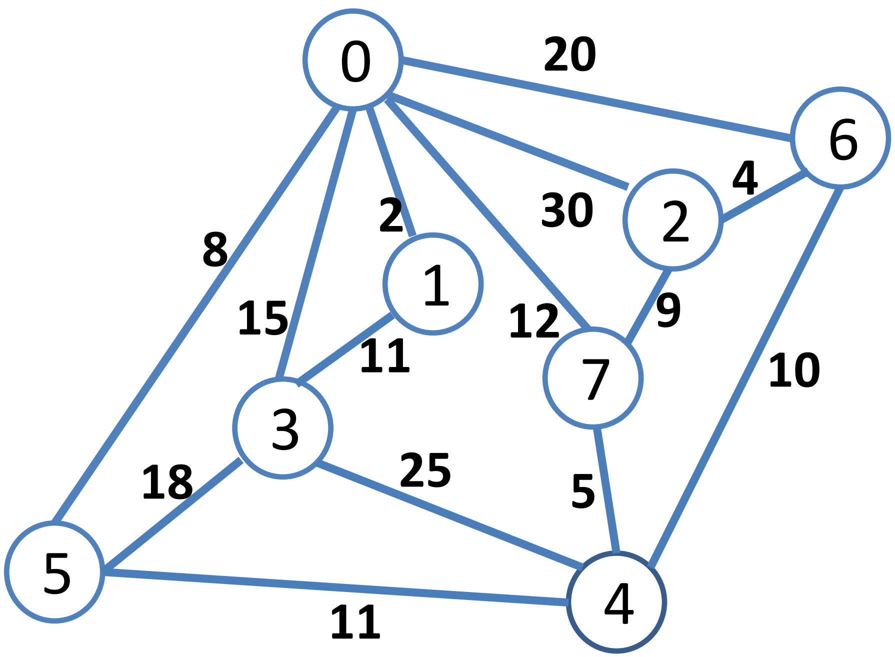




Edge (4,6,10) was not picked b.c. it makes a cycle.

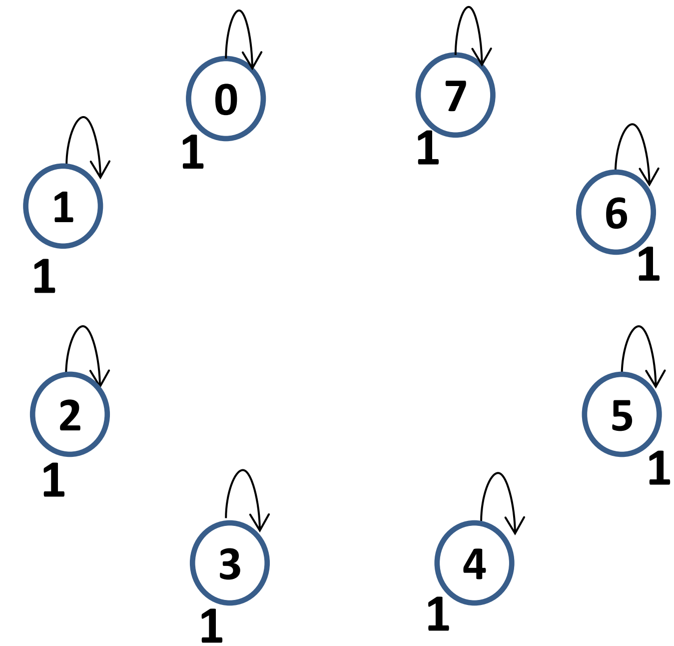
# Kruskal’s Algorithm and the Union-Find Structure

- Note the Union-Find method is under the “Data Structures for Disjoint Sets” in CLRS, Chapter 21, page 561,

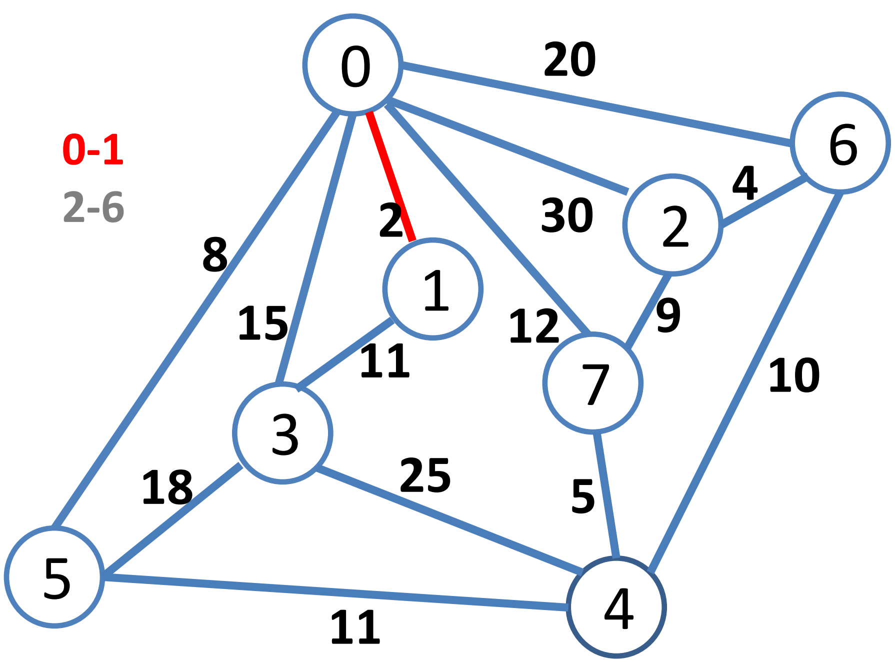
# Kruskal & Union Find



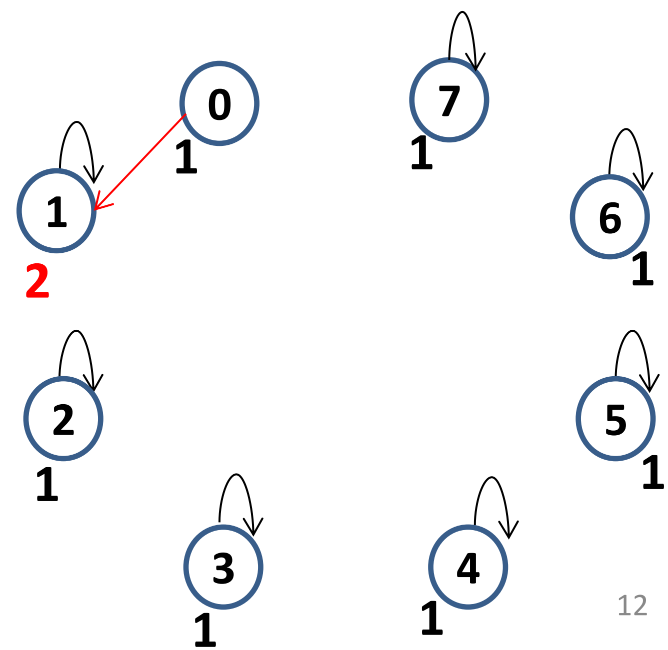

| idx | Id | Sz |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 1 | 1 |
| 2 | 2 | 1 |
| 3 | 3 | 1 |
| 4 | 4 | 1 |
| 5 | 5 | 1 |
| 6 | 6 | 1 |
| 7 | 7 | 1 |



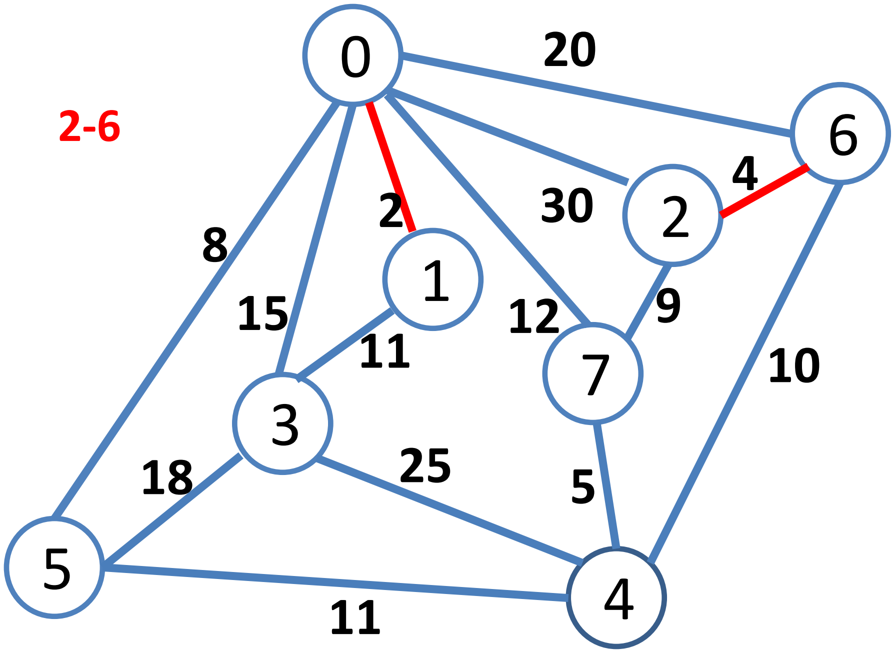

0-1
2-6



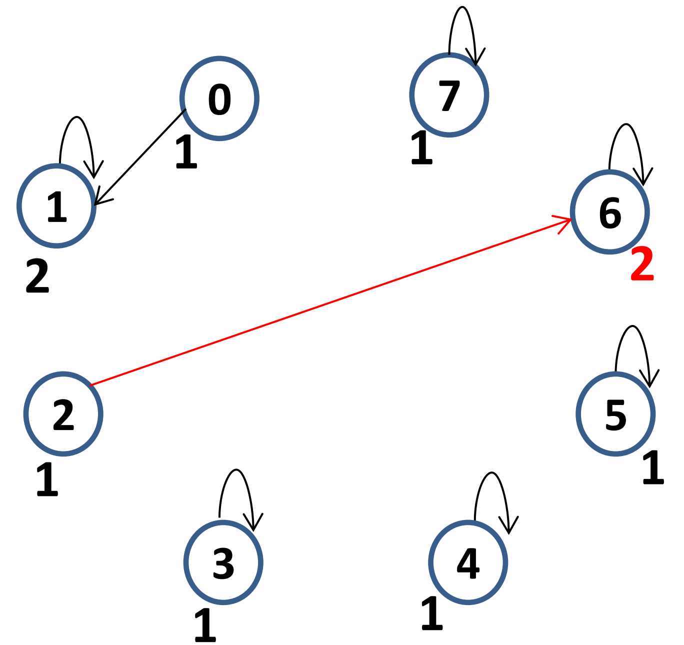

| idx | Id | Sz |
|---|---|---|
| 0 | 1 | 1 |
| 1 | 1 | 2 |
| 2 | 2 | 1 |
| 3 | 3 | 1 |
| 4 | 4 | 1 |
| 5 | 5 | 1 |
| 6 | 6 | 1 |
| 7 | 7 | 1 |



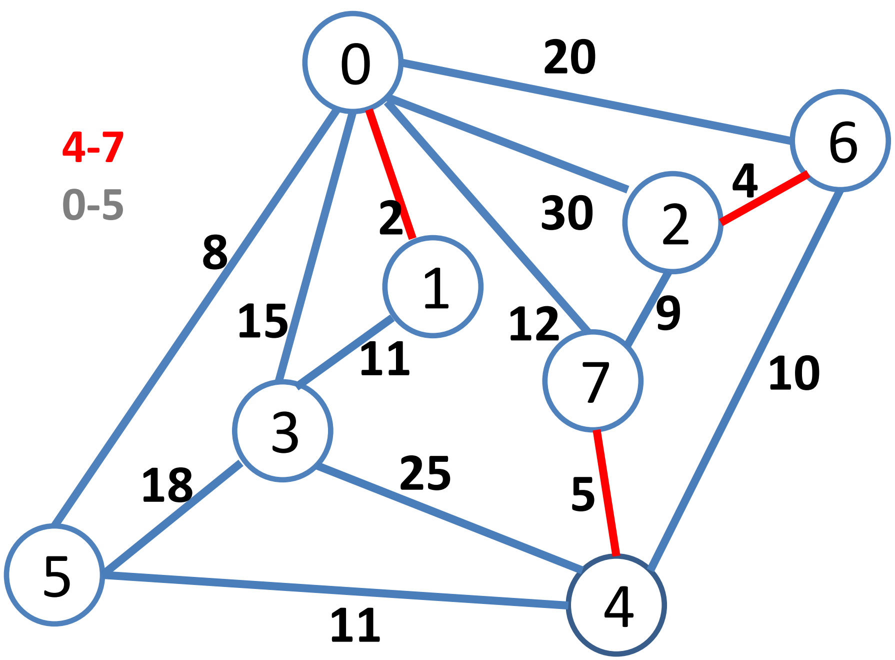
2-6



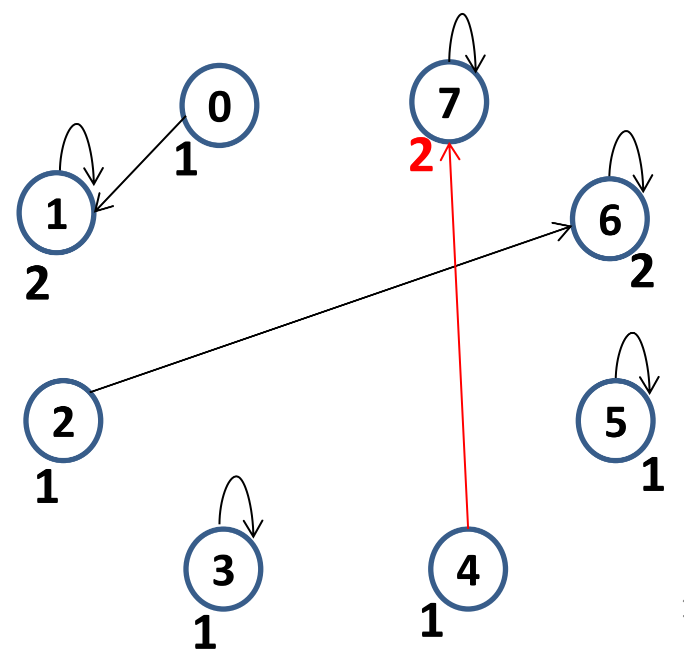

| idx | Id | Sz |
|---|---|---|
| 0 | 1 | 1 |
| 1 | 1 | 2 |
| 2 | 6 | 1 |
| 3 | 3 | 1 |
| 4 | 4 | 1 |
| 5 | 5 | 1 |
| 6 | 6 | 2 |
| 7 | 7 | 1 |



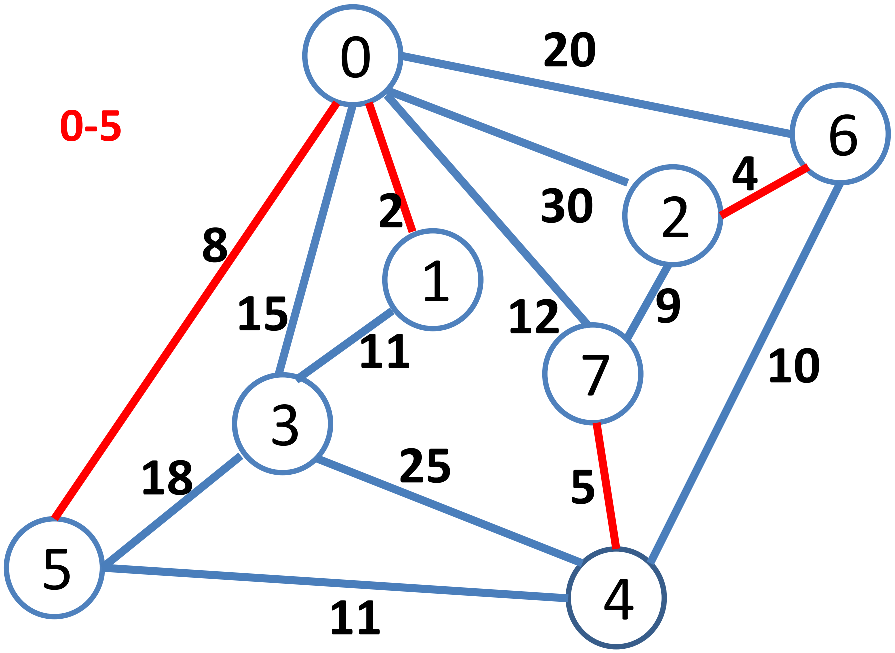

4-7
0-5



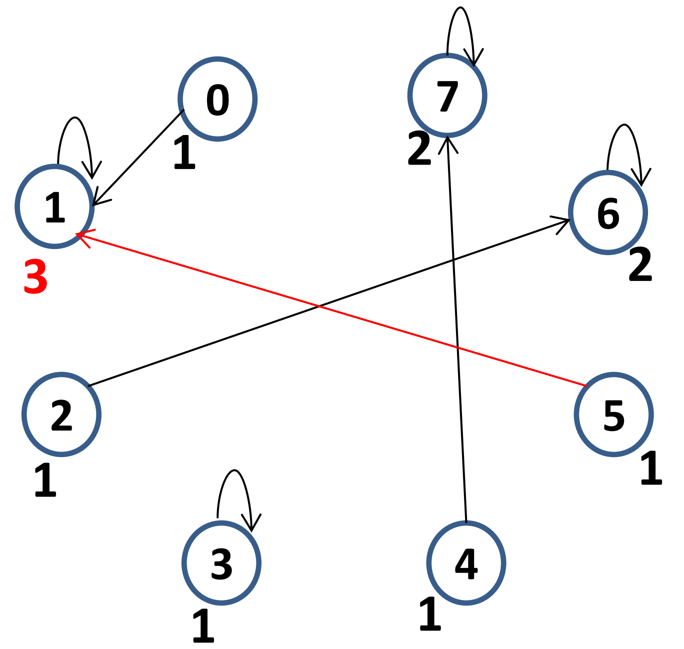

| idx | Id | Sz |
|---|---|---|
| 0 | 1 | 1 |
| 1 | 1 | 2 |
| 2 | 6 | 1 |
| 3 | 3 | 1 |
| 4 | 7 | 1 |
| 5 | 5 | 1 |
| 6 | 6 | 2 |
| 7 | 7 | 2 |



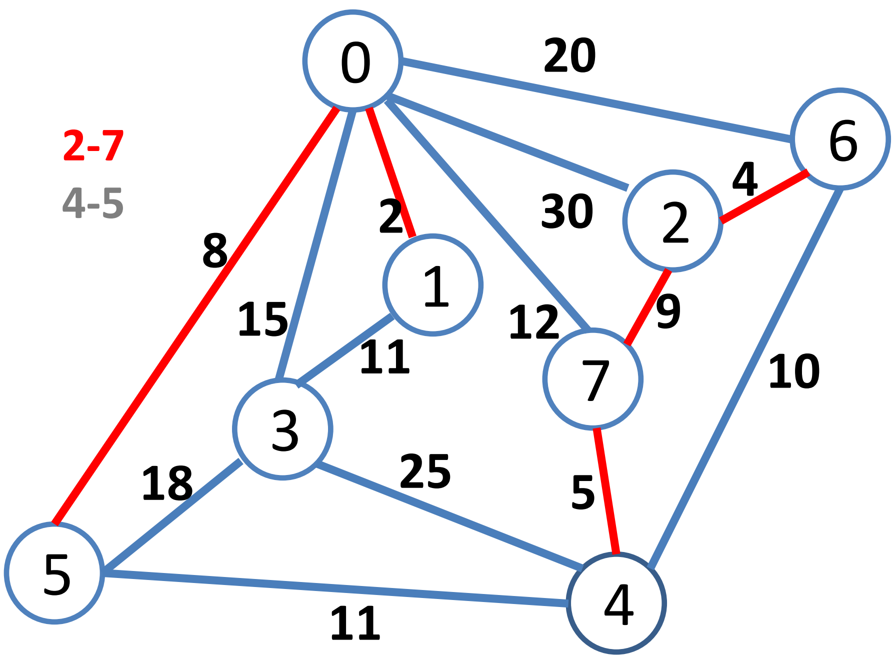
0-5



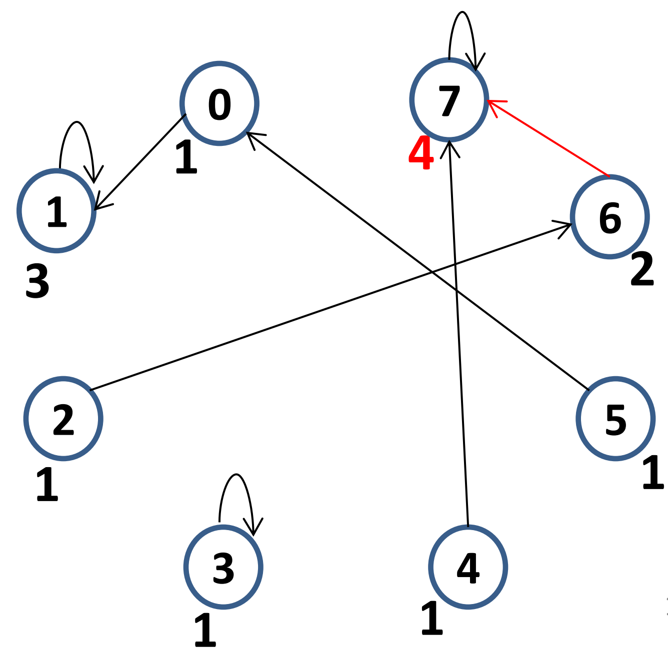

| idx | Id | Sz |
|---|---|---|
| 0 | 1 | 1 |
| 1 | 1 | 3 |
| 2 | 6 | 1 |
| 3 | 3 | 1 |
| 4 | 7 | 1 |
| 5 | 1 | 1 |
| 6 | 6 | 2 |
| 7 | 7 | 2 |



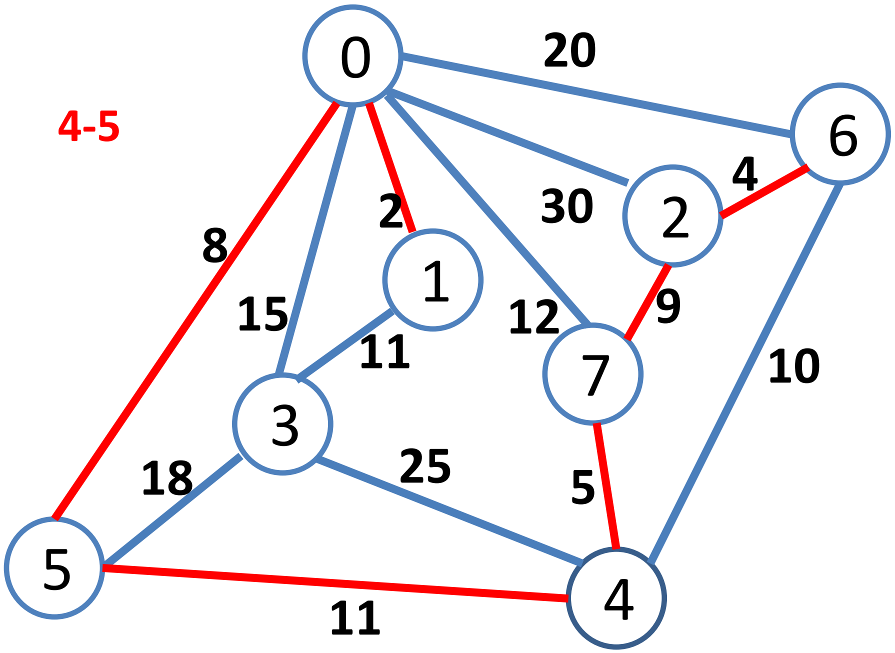

2-7
4-5



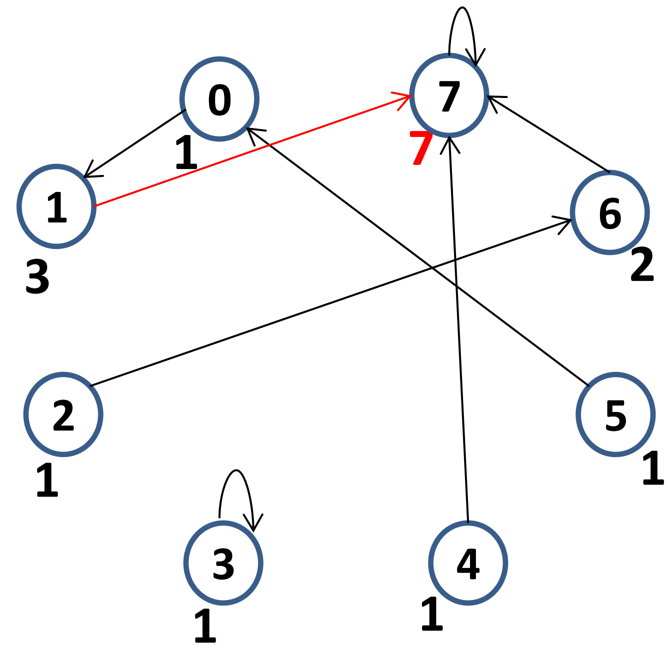

| idx | Id | Sz |
|---|---|---|
| 0 | 1 | 1 |
| 1 | 1 | 3 |
| 2 | 6 | 1 |
| 3 | 3 | 1 |
| 4 | 7 | 1 |
| 5 | 0 | 1 |
| 6 | 7 | 2 |
| 7 | 7 | 4 |



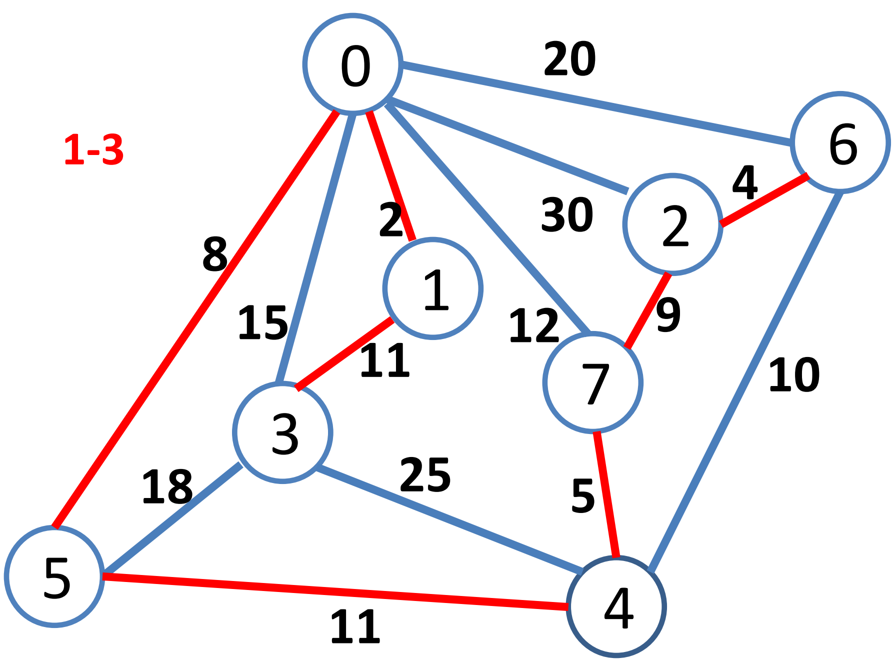
**4-5**



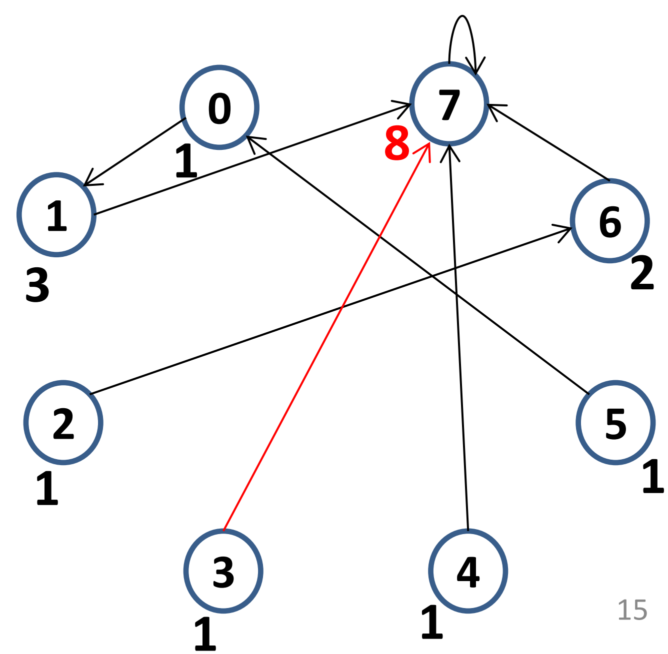

| idx | Id | Sz |
|---|---|---|
| 0 | 1 | 1 |
| 1 | 7 | 3 |
| 2 | 6 | 1 |
| 3 | 3 | 1 |
| 4 | 7 | 1 |
| 5 | 0 | 1 |
| 6 | 7 | 2 |
| 7 | 7 | 7 |




**1-3**




| idx | Id | Sz |
|---|---|---|
| 0 | 1 | 1 |
| 1 | 7 | 3 |
| 2 | 6 | 1 |
| 3 | 7 | 1 |
| 4 | 7 | 1 |
| 5 | 0 | 1 |
| 6 | 7 | 2 |
| 7 | 7 | 8 |



# Prim's Algorithm

```
MST_Prim(G,w,s) // N = |V|
int d[N], p[N]
For v = 0 -> N-1
    d[v]=inf   //min weight of edge connecting v to MST
    p[v]=-1    //(p[v],v) in MST and  w(p[v],v) =d[v]
d[s]=0
Q = PriorityQueue(G.V,w)
While notEmpty(Q)
    u = removeMin(Q,w)   //u is picked
    for each v adjacent to u
        if v in Q and w(u,v)<d[v]
            p[v]=u
            d[v] = w(u,v)
            decreasedKeyFix(Q,v,d)
```

# MST-Prim (G,w,7) Worksheet

Start from ANY vertex, s. (this is an MST).

Repeat until all vertices are added to the MST:

- Add to the MST the edge (and the non-MST vertex of that edge) that is the smallest of all edges connecting vertices from the MST to vertices outside of the MST.

CLRS pseudocode.

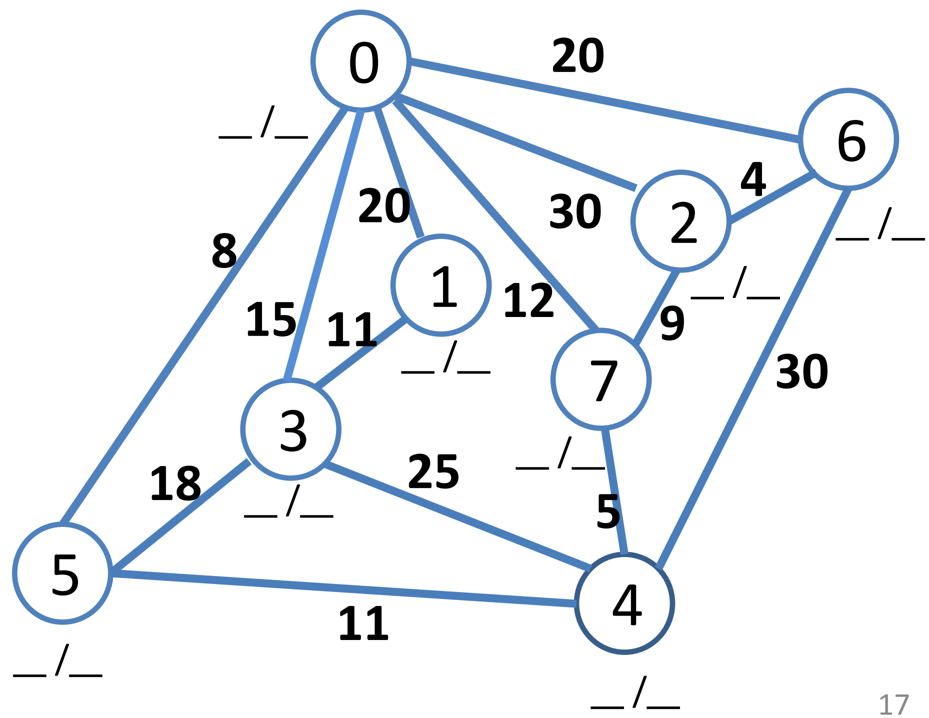
Run Prims algorithm starting at vertex 7.

__ / __ = d[v] / p[v]

(p = predecessor or parent)

(Edge weight changes: (0,1,**20**) and (4,6,**30**). )

| u,v,w |
|---|
| |
| |
| |
| |
| |
| |
| |



# MST-Prim (G,w,7) Answer

```
MST_Prim(G,w,s) // N = |V|
    int d[N], p[N]
    For v = 0 -> N-1
        d[v]=inf   //min weight of edge connecting u to MST
        p[v]=-1    //(p[v],v) in MST and  w(p[v],v) =d[v]
    d[s]=0
    Q = PriorityQueue(G.V,w)
    While notEmpty(Q)
        u = removeMin(Q,w)   //u is picked
        for each v adjacent to u
            if v in Q and w(u,v)<d[v]
                p[v]=u
                d[v] = w(u,v)
                decreasedKeyFix(Q,v,d)
```

Start from ANY vertex, s. (this is an MST).

Repeat until all vertices are added to the MST:

- Add to the MST the edge (and the non-MST tree vertex of that edge) that is the smallest of all edges connecting vertices from the MST to vertices outside of the MST.

CLRS pseudocode.

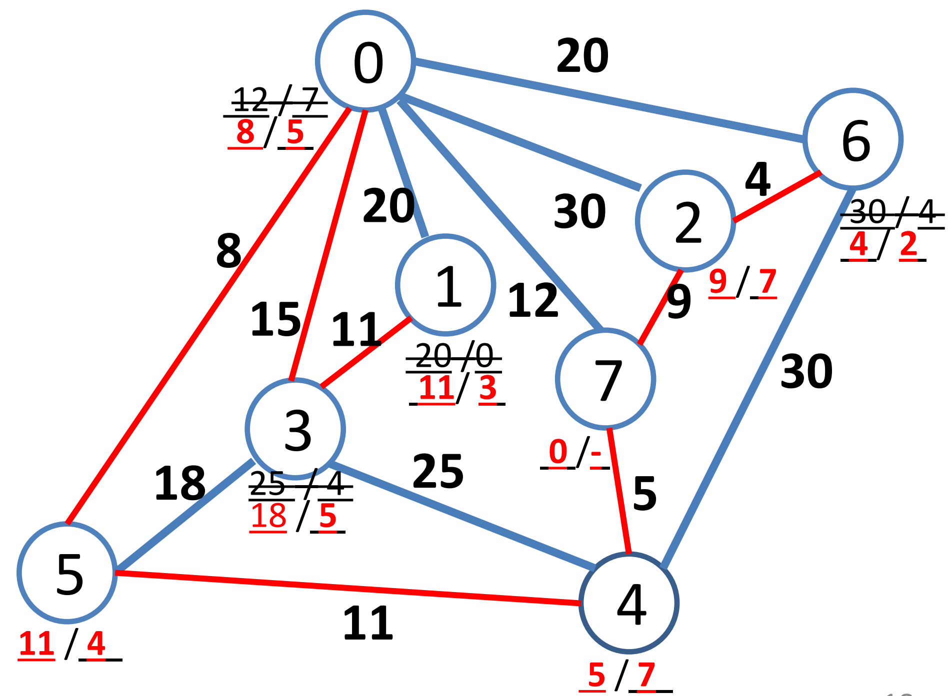
Run Prims algorithm starting at vertex 7.

__ / __ = d[v] / p[v]

(p = predecessor or parent)

(Edge weight changes: (0,1,**20**) and (4,6,**30**). )

| u,v,w |
|---|
| 7,4,5 |
| 7,2,9 |
| 2,6,4 |
| 4,5,11 |
| 5,0,8 |
| 0,3,15 |
| 3,1,11 |






The p array stores the tree. Branches: ( p(i),i )

# Prim's Algorithm
# Time Complexity

- Q – is a priority queue

Time complexity:

```
MST_Prim(G,w,s) // N = |V|
int d[N], p[N]
For v =0 -> N-1
    d[v]=inf    //min weight of edge connecting v to MST
    p[v]=-1     //MST vertex, s.t. w(p[v],v) =d[v]
d[s]=0
Q = PriorityQueue(d)
While notEmpty(Q)
    u = removeMin(Q,w)
    for each v adjacent to u
        if v in Q and w(u,v)<d[v]
            p[v]=u
            d[v] = w(u,v);
            decreasedKeyFix(Q,v,d)  //v is neither index nor key
```

# Prim's Algorithm Time Complexity

- Q – is a priority queue

Time complexity: O(ElgV)

O(V + VlgV + E lgV) =

O(ElgV)

connected graph => |E| ≥ (|V|-1)

```
MST_Prim(G,w,s) // N = |V|
1   int d[N], p[N]
2   For v=0 -> N-1            -------------->      O(V)
3       d[v]=inf    //min weight of edge connecting v to MST
4       p[v]=-1     //MST vertex, s.t. w(p[v],v) =d[v]
5   d[s]=0
6   Q = PriorityQueue(d)      ------------>   O(V)   (build heap)
7   While notEmpty(Q)         ------------>   O(V)
8       u = removeMin(Q,w)        ------------>    O(lgV)
9       for each v adjacent to u          //lines 7 & 9 together:  ---->  O(E)
10          if v in Q and w(u,v)<d[v]     //(touch each edge twice)
11              p[v]=u
12              d[v] = w(u,v);
13              decreasedKeyFix(Q,v,d)  //v is neither index nor key  -------->  O(lgV)
```

O(V*lgV)

O(E*lgV) from lines: 7,9,13

# Prim's Algorithm Implementation Details

```
MST_Prim(G,w,s) // N = |V|
int d[N], p[N]
For v =0 -> N-1
    d[v]=inf
    p[v]=-1
d[s]=0
Q = PriorityQueue(d)
While notEmpty(Q)
    u = removeMin(Q,w)
    for each v adjacent to u
        if v in Q and w(u,v)<d[v]
            p[v]=u
            d[v] = w(u,v);
            decreasedKeyFix(Q,v,d)
                //v is neither index nor key
```

- See if v is in Q.
  - Θ(1) if we have the Array->Heap mapping.
  - Else, O(V).
- Find heap node corresponding to v.
  - Needed to update the heap for according to smaller *d[v]*.
  - Note the difference between v and node in heap corresponding to v.
  - See heap slides : Index Heap Example

# Other

- Variations
  - start with an empty priority queue
  - For dense graphs, keep and array (instead of a priority queue => $O(V^2)$ – optimal for dense graphs ) – see Sedgewick if interested.

- Make sure you understand what happens with the data in an implementation:
  - How do you know if a vertex is still in the priority queue?
  - Going from a vertex to its place in the priority queue.
  - The updates to the priority queue.

# Proof of Correctness

- Is the MST a specific type of problem?

- What type of method is:
  - Kruskal’s
  - Prim’s

- Can we prove that they give the MST?

# Definitions

(CLRS, pg 625)

- A ***cut*** (S, V-S) of an graph is a partition of its vertices, V.
- An edge (u,v) ***crosses*** the cut (S, V-S) if one of its endpoints is in S and the other in V-S.
- Let A be a subset of a minimum spanning tree over G. An edge ***(u,v) is safe for A*** if $A \cup \{(u,v)\}$ is still a subset of a minimum spanning tree.
- A cut respects a set of edges, A, if no edge in A crosses the cut.
- An edge is a ***light edge*** crossing a cut if its weight is the minimum weight of any edge crossing the cut.

# Correctness of Prim and Kruskall

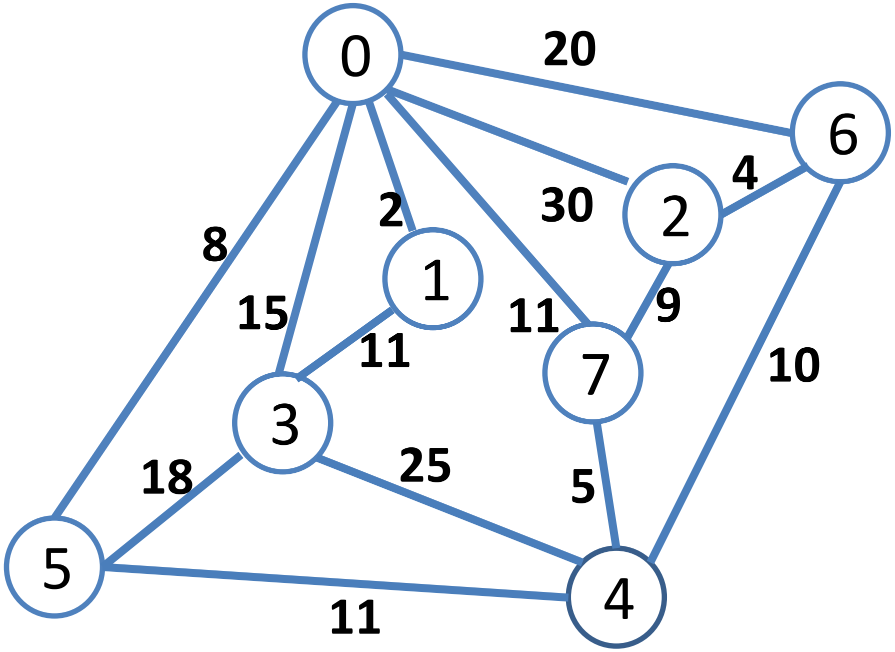
(CLRS, pg 625)

- Invariant for both Prim and Kruskal: At every step of the algorithm, the set, A, of edges is a subset of a MST.
- Let G = (V,E) be a connected, undirected, weighted graph. Let A be a subset of some minimum spanning tree, T, for G, let (S, V-S) be some cut of G that respects A, and let (u,v) be a light edge crossing (S, V-S). Then, edge (u,v) is safe for A.
- Proof:

If (u,v) is part of T, done

Else, in T, u and v must be connected through another path, p. One of the edges of p, must connect a vertex x from A and a vertex, y, from V-A. Adding edge(u,v) to T will create a cycle with the path p. (x,y) also crosses (A, V-A) and (u,v) is light => weight(u,v) ≤ weight(x,y) => weight(T') ≤weight(T) , but T is MST => T' also MST (where T' is T with (u,v) added and (x,y) removed) and A U {(u,v)} is a subset of T'.

# Extra Materials:

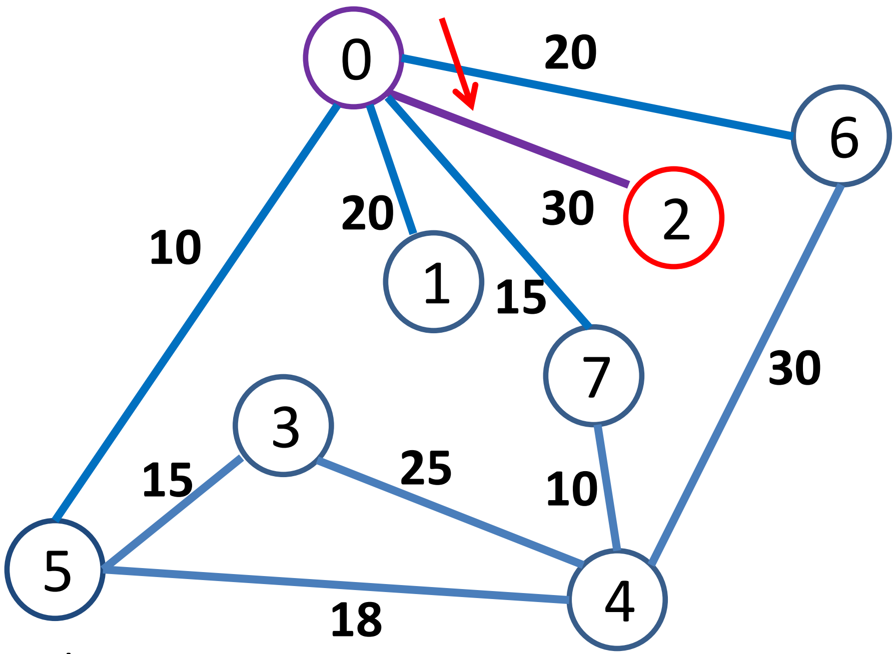


# Prim's Alg Example 2

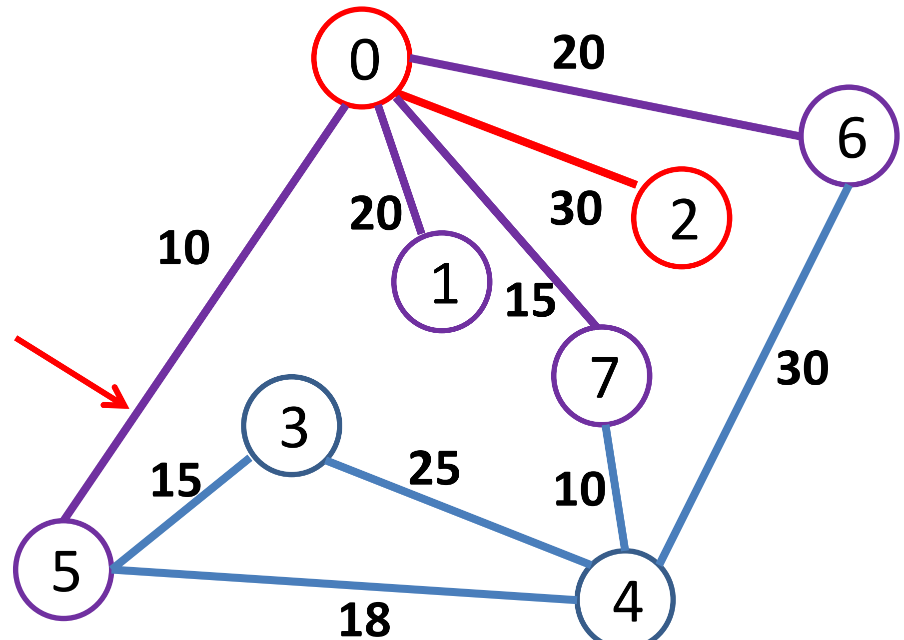
```
MST_Prim(G,w,s) // N = |V|
    int d[N], p[N]
    For v =0 -> N-1
        d[v]=inf
        p[v]=-1
    d[s]=0
    Q = PriorityQueue(G.V,w)
    While notEmpty(Q)
        u = removeMin(Q,w)  //u is picked
        for each v adjacent to u
            if v in Q and w(u,v)<d[v]
                p[v]=u
                d[v] = w(u,v)
                decreasedKeyFix(Q,v,d)
```

MST-Prim(G, w, **2**)    (here:  r = 2)

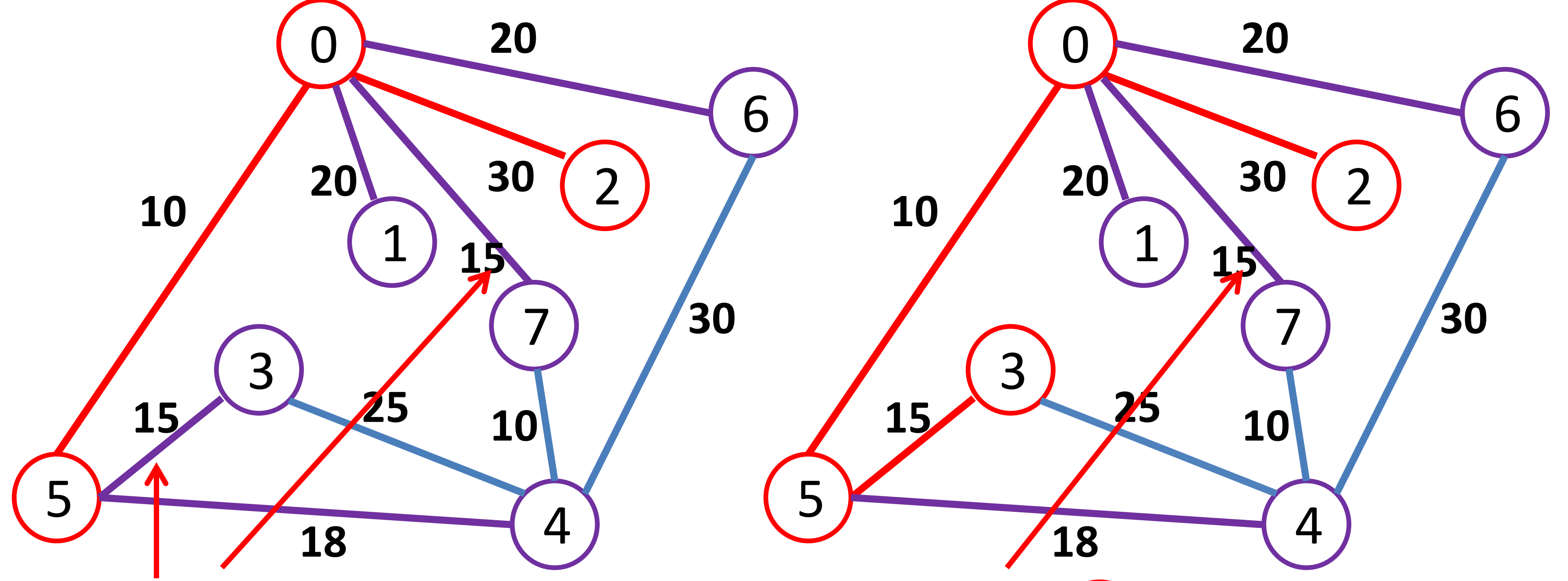
(This example shows the frontier (edges and vertices).

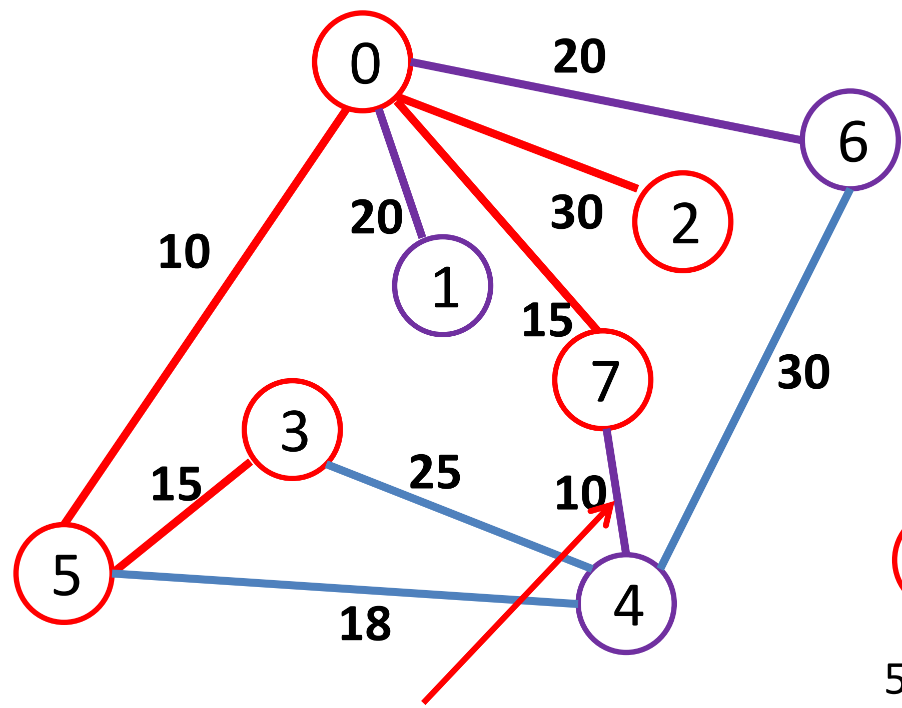


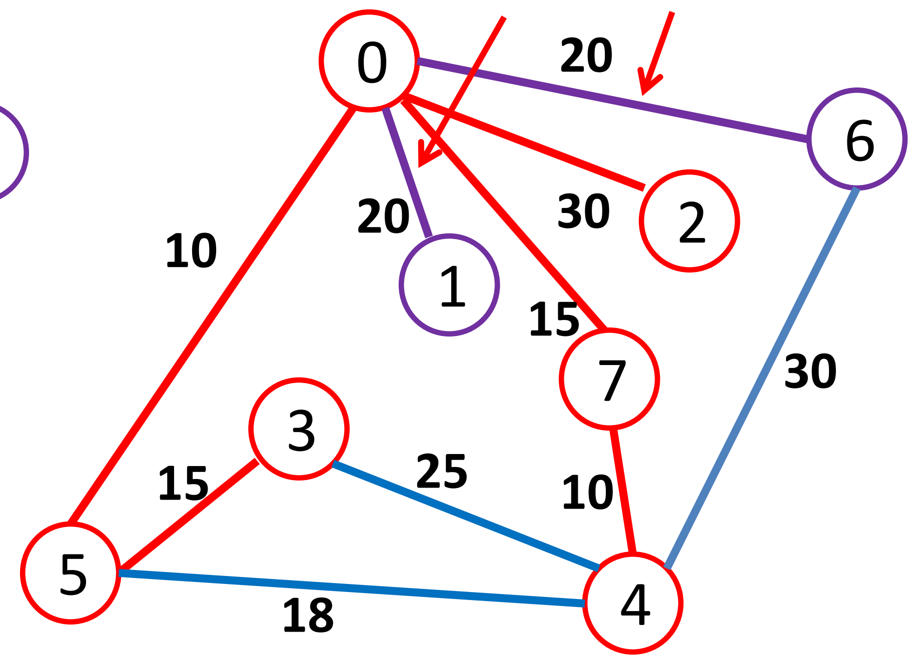

Red - current MST
Purple - potential edges and vertices
Blue – unprocessed edges and vertices.




The algorithm will keep a MIN-Priority Queue for the vertices.

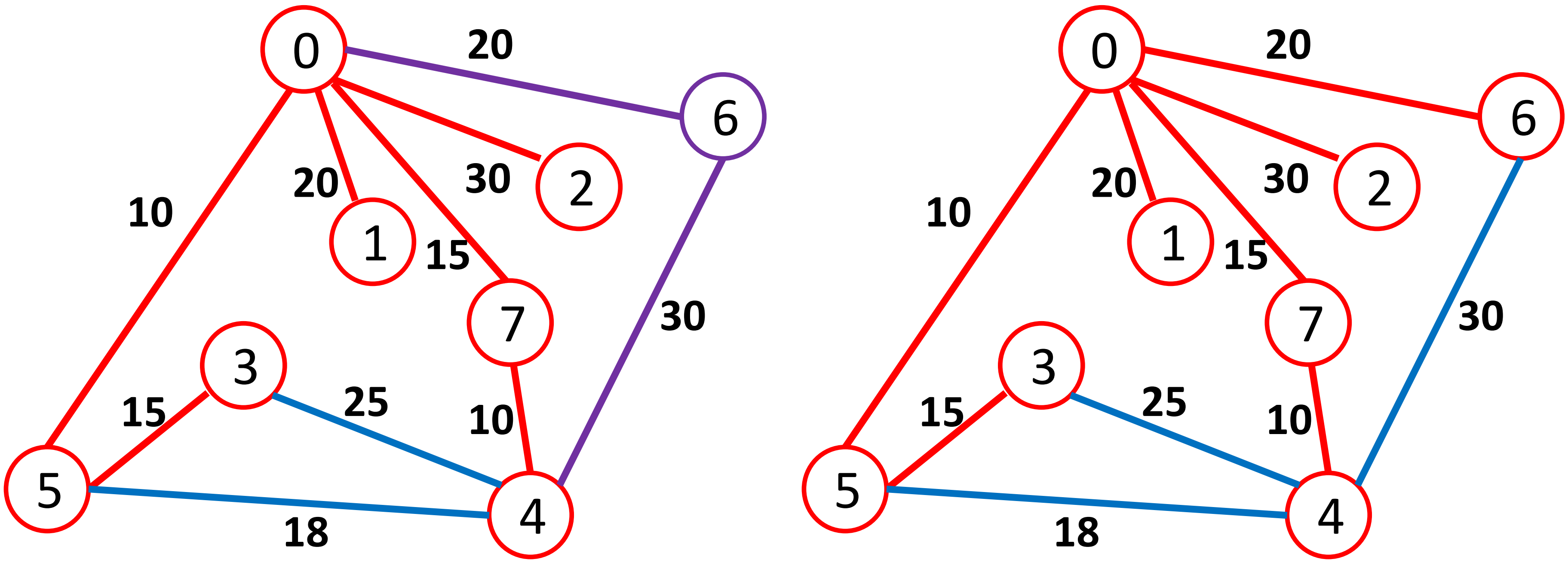






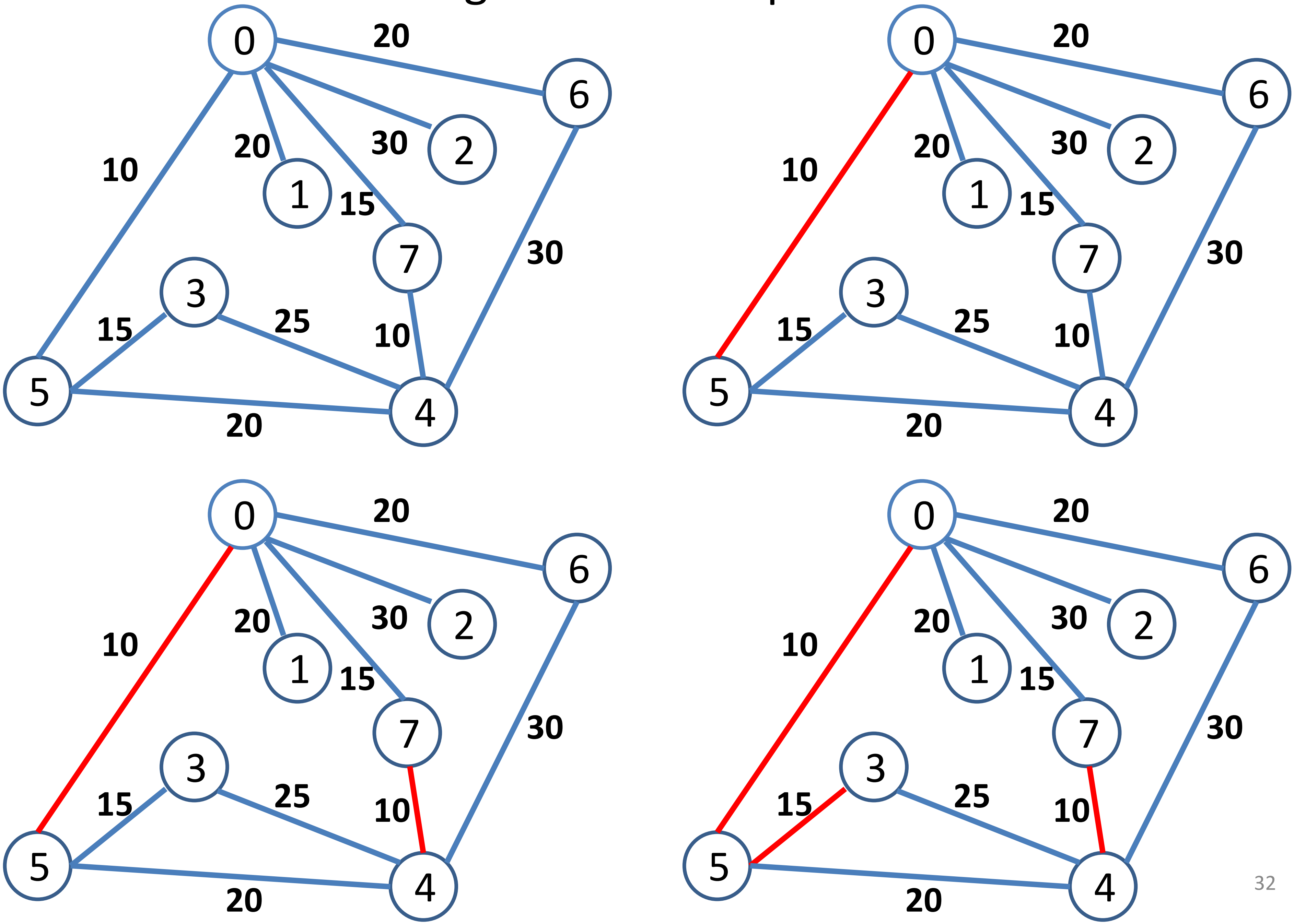
5, 4 already in MST => (5,4,  18) not picked
(4,6,30) not better than (0,6,20) => no 6 update

# Prim’s Alg Example 2 - cont



# Kruskal’s Algorithm Example 2

# Kruskal's Algorithm: Example 2 workout



# Kruskal's Algorithm: Example 2 workout