Kruskal's Algorithm
for
Minimum Spanning Tree

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(ElgE^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)  (for adj list representation)
   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?

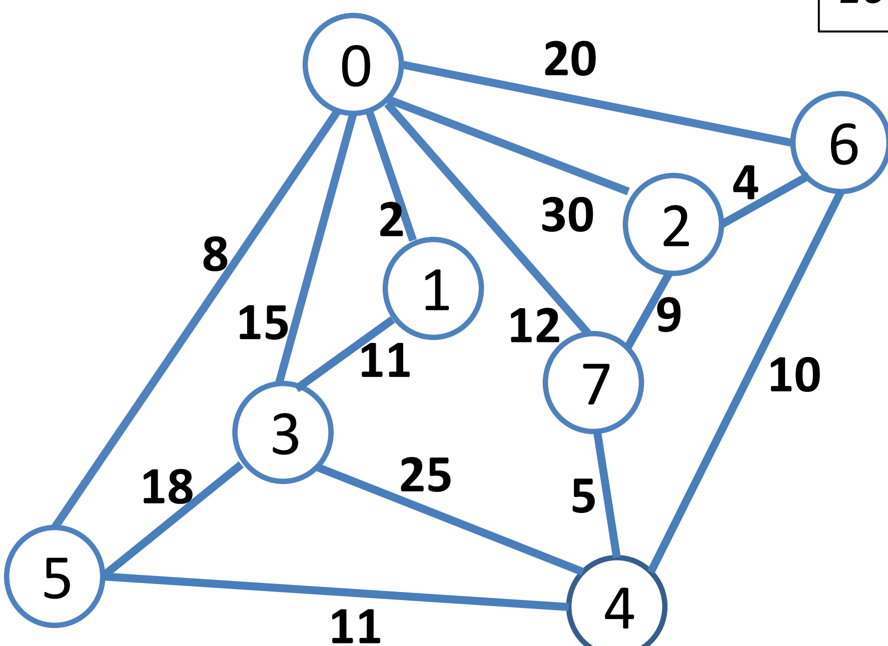
Kruskal's Algorithm

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).

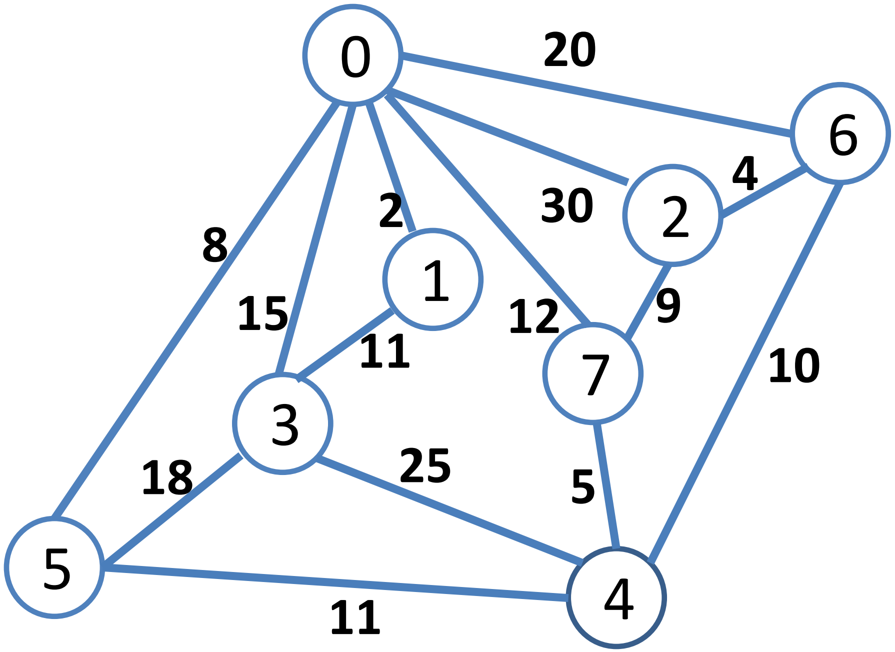
See Union-Find slides as well.

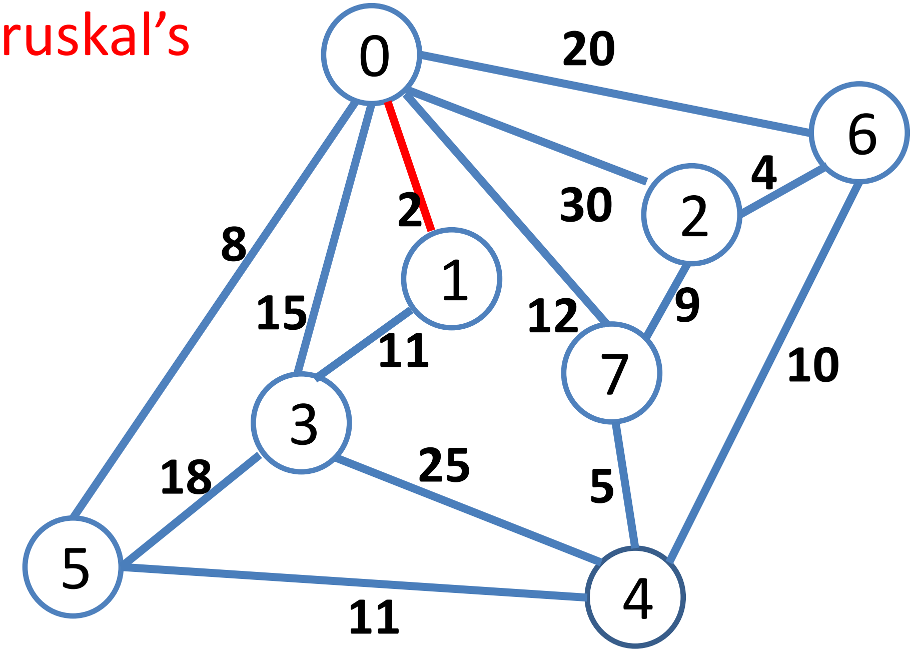
```
MST_Kruskal(G,w)  // N = |V|
A = empty set of edges
int id[N], sz[N]
For v = 0 -> N-1
    id(v) = v;  sz(v)=1
Sort edges of G in increasing order of weight
For each edge (u,v) in increasing order of weight
    if Find_Set(u,id) == Find_Set(v,id)
        add edge (u,v) to A
        union(u,v,id,sz)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

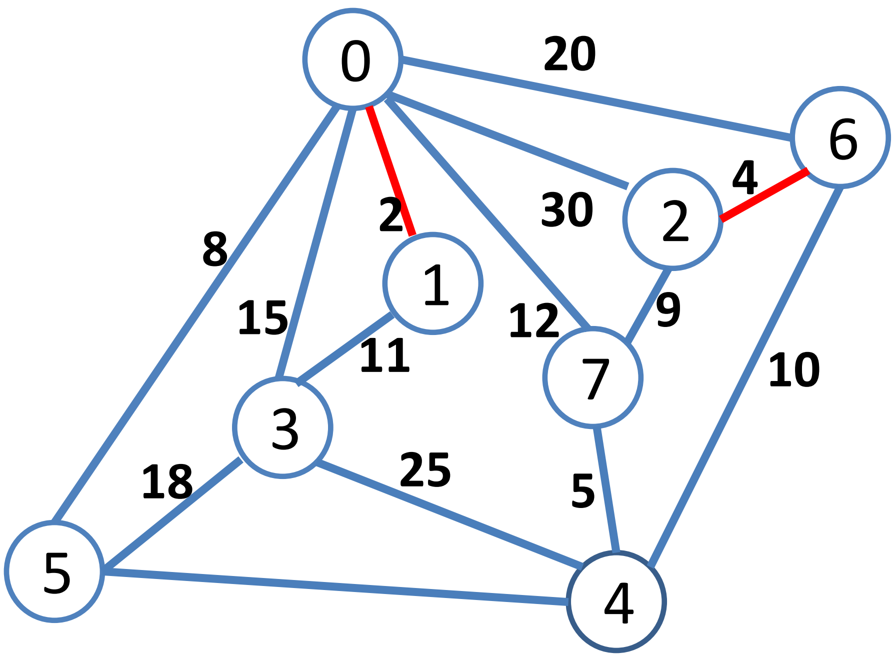


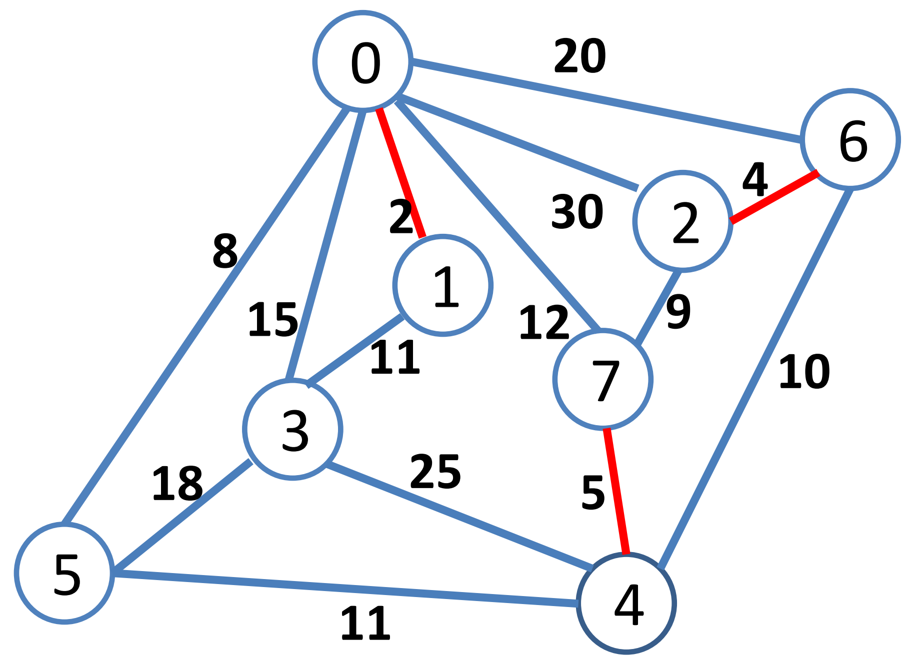
u, v, weight

Kruskal's

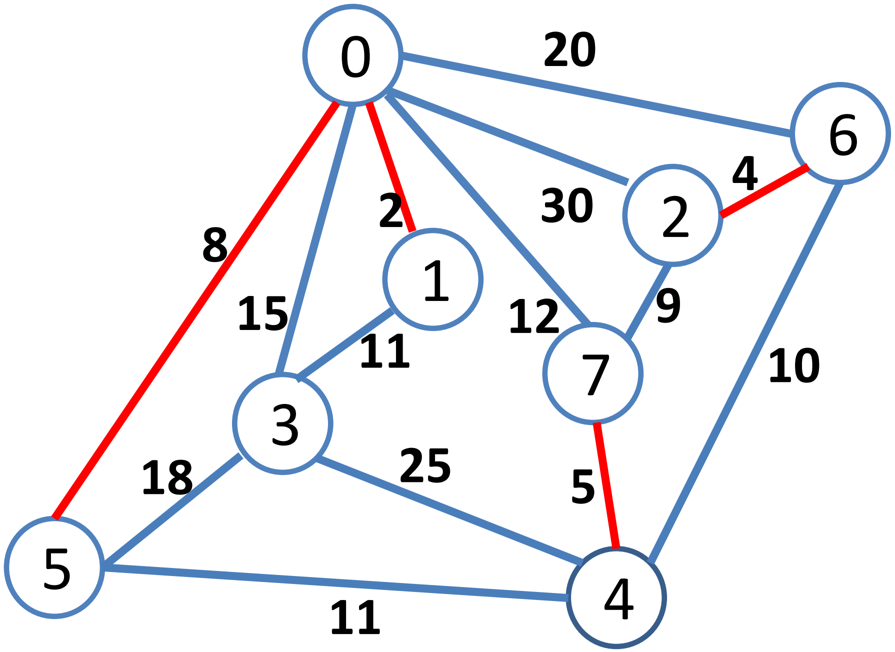

0
20
6
4
30
2
2
8
1
15
12
9
11
7
10
3
18
25
5
5
4
11

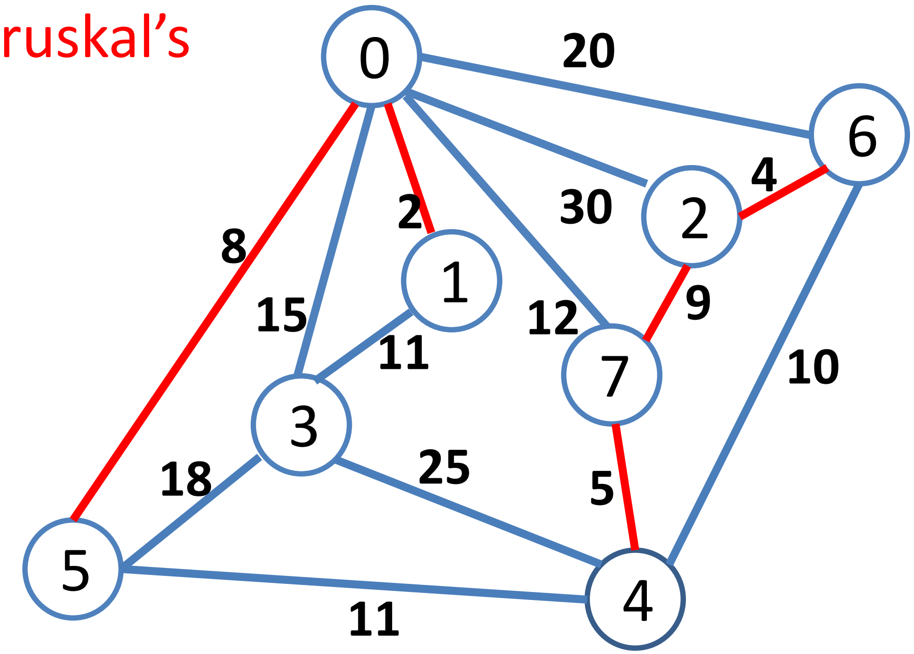

0
20
6
4
30
2
2
8
1
15
12
9
11
7
10
3
18
25
5
5
4
11

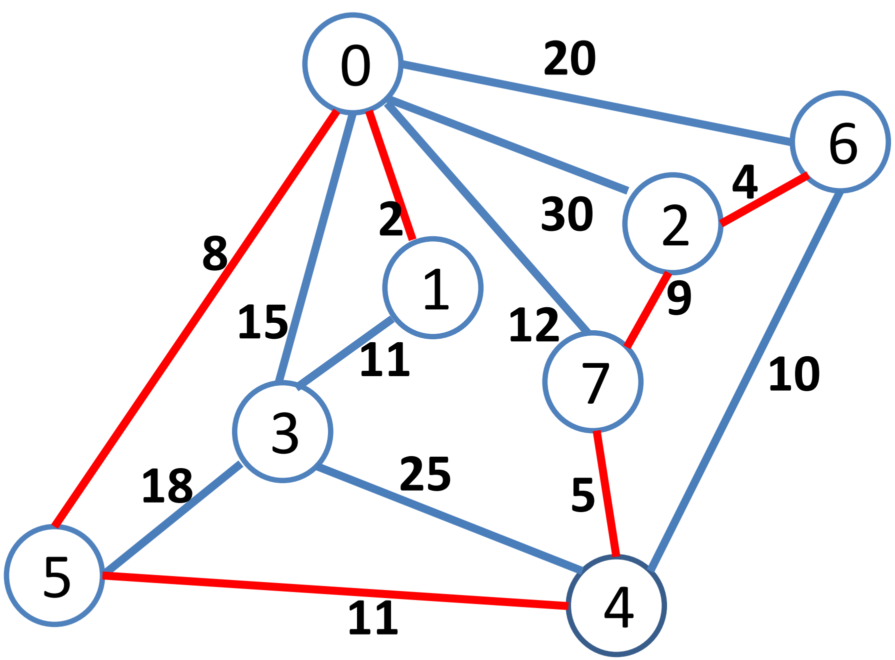

0
20
6
4
30
2
2
8
1
15
12
9
11
7
10
3
18
25
5
5
4

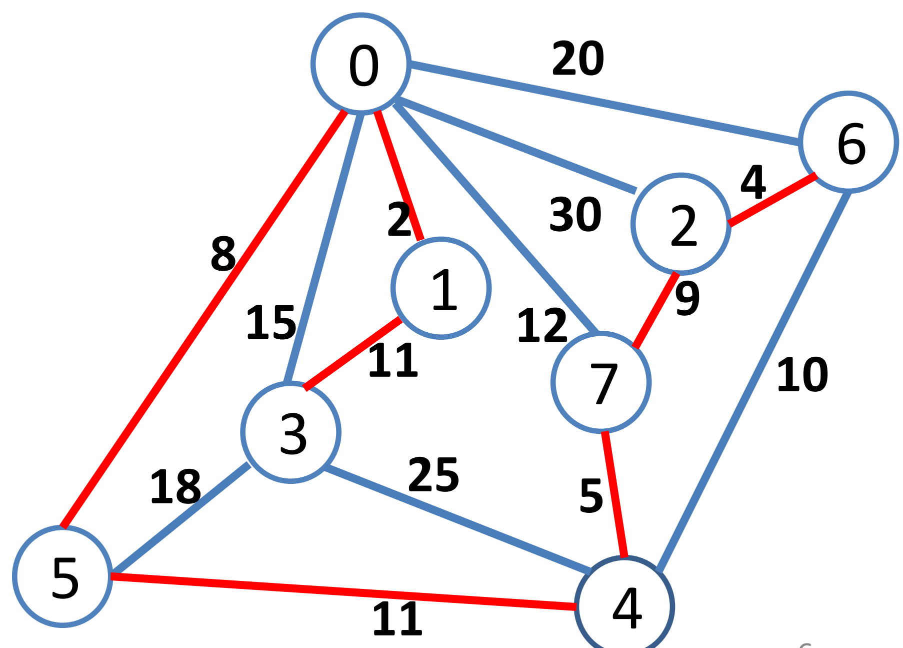

0
20
6
4
30
2
2
8
1
15
12
9
11
7
10
3
18
25
5
5
4
11

Kruskal's







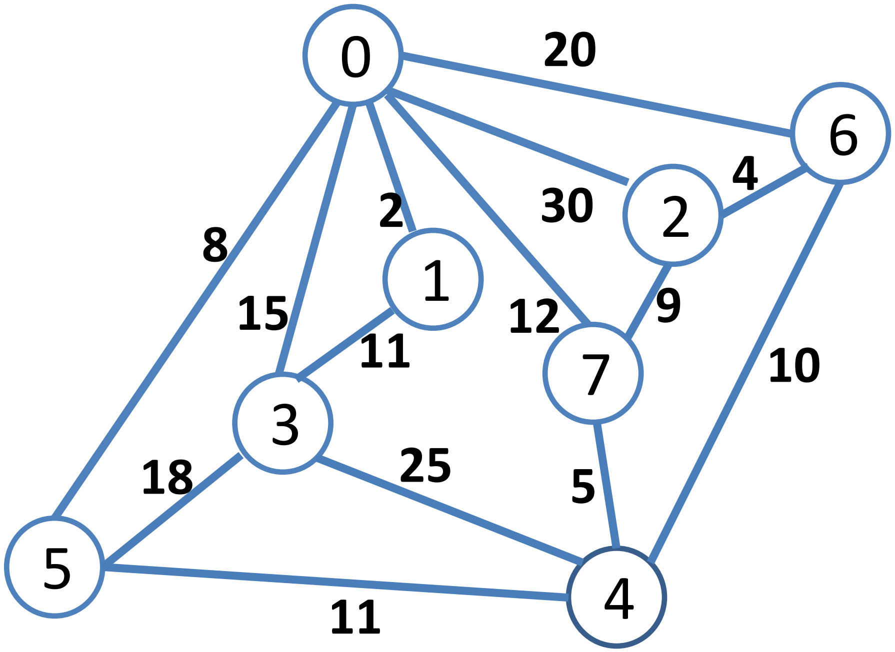


Edge (4,6,10) was not picked b.c. it makes a cycle.

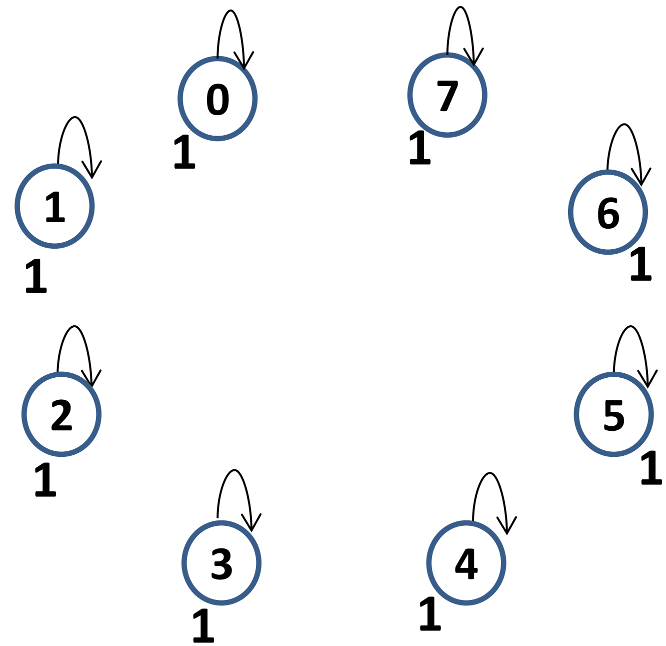
Kruskal’s Algorithm and the Union-Find Structure

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

Kruskal & Union Find

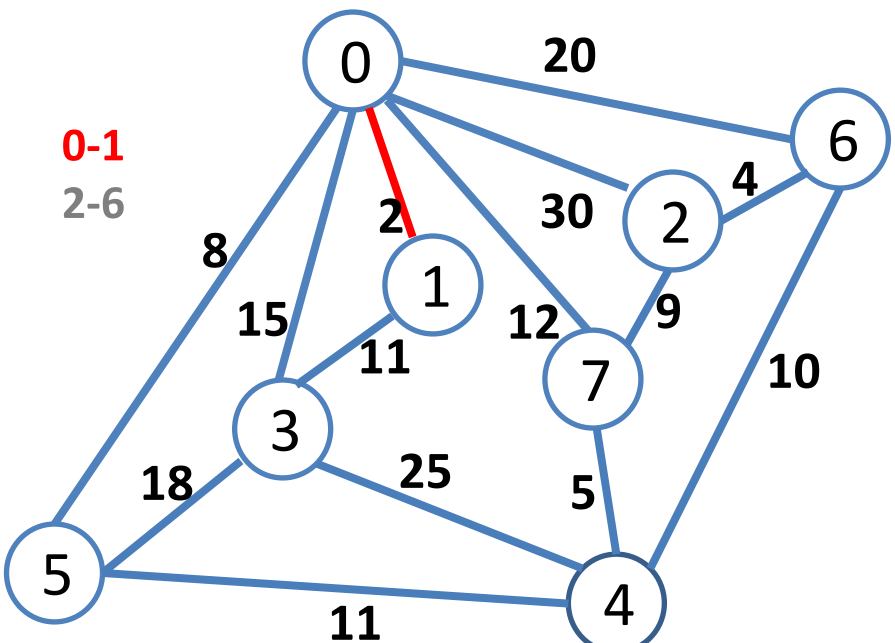


idx	Id	Sz
0	0	1
1	1	1
2	2	1
3	3	1
4	4	1
5	5	1
6	6	1
7	7	1

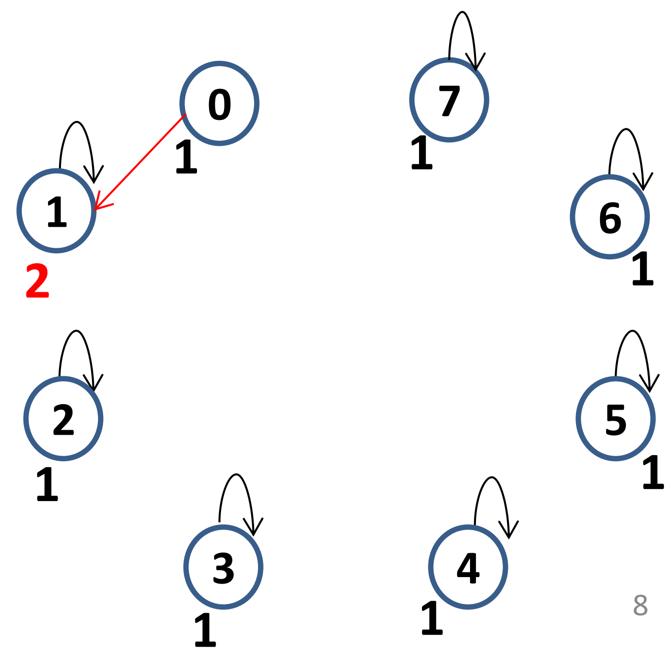


0-1

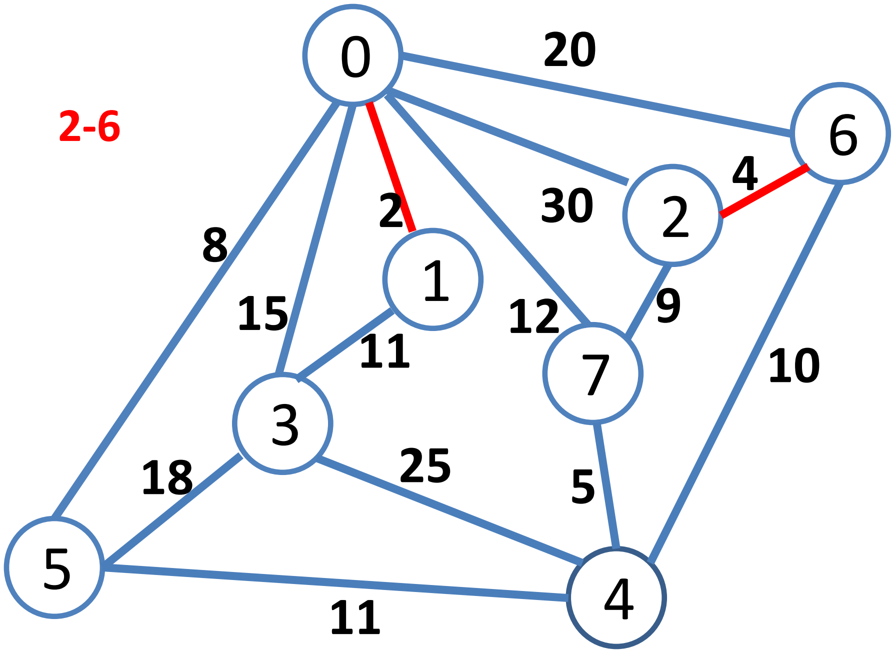
2-6



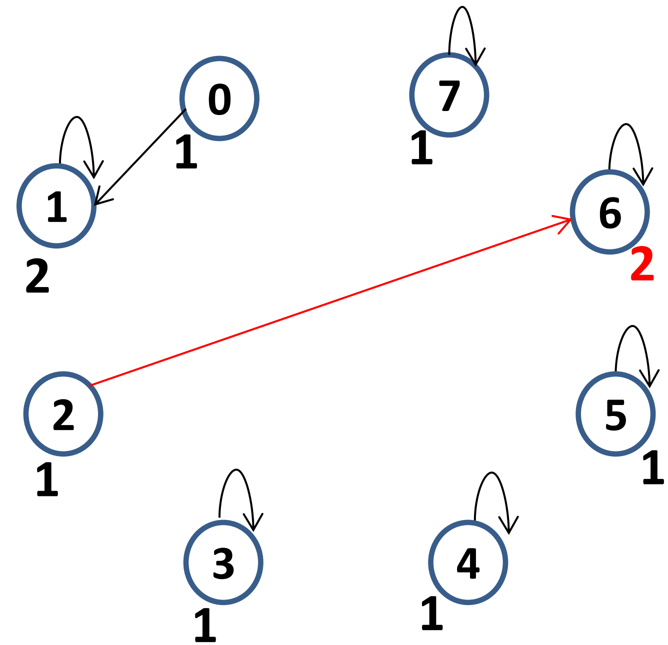
idx	Id	Sz
0	1	1
1	1	2
2	2	1
3	3	1
4	4	1
5	5	1
6	6	1
7	7	1



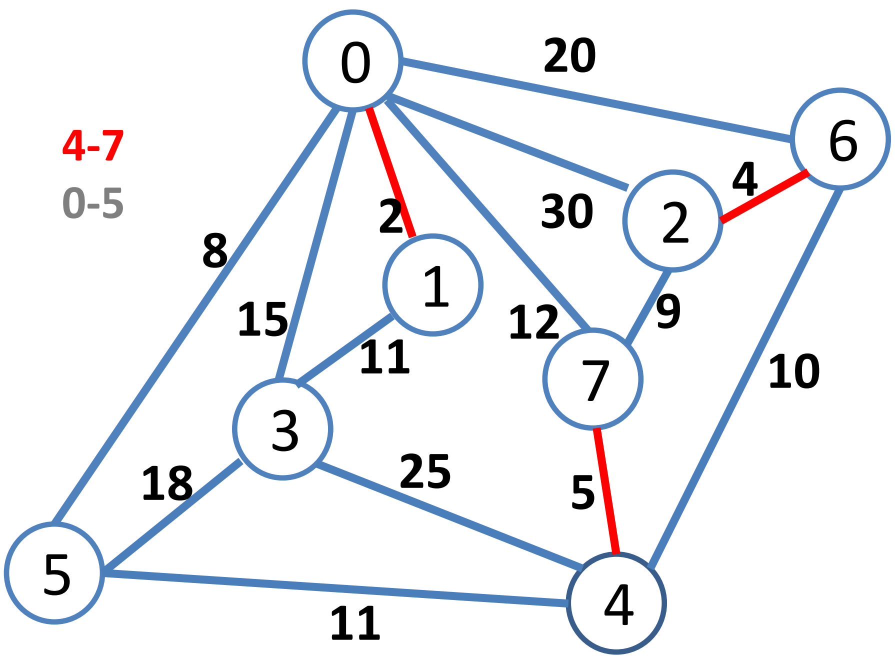
2-6



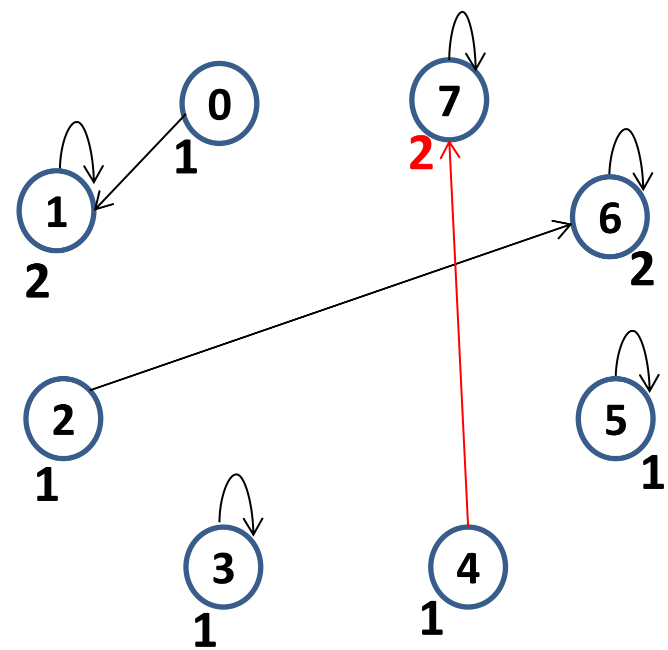
idx	Id	Sz
0	1	1
1	1	2
2	6	1
3	3	1
4	4	1
5	5	1
6	6	2
7	7	1



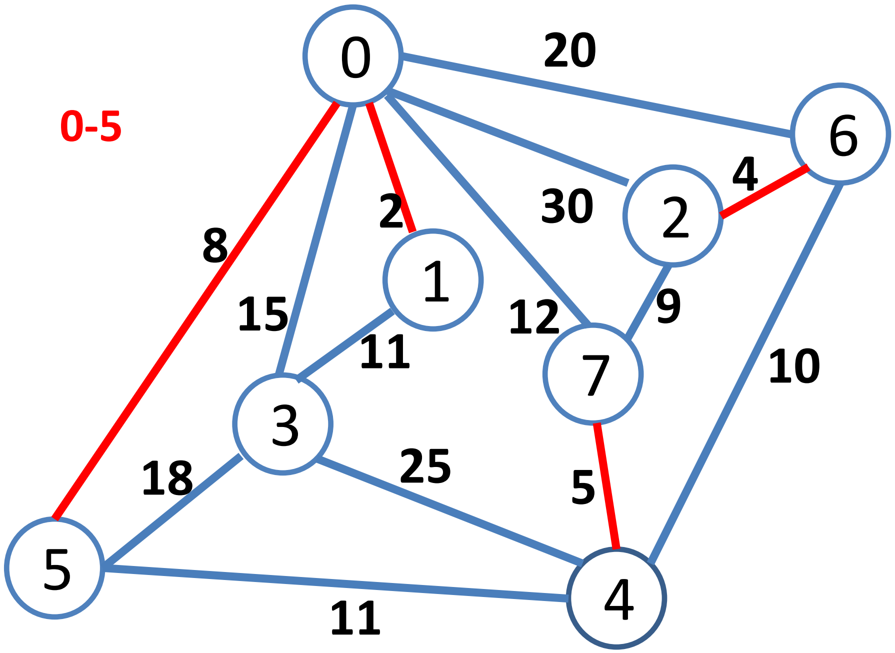
4-7
0-5



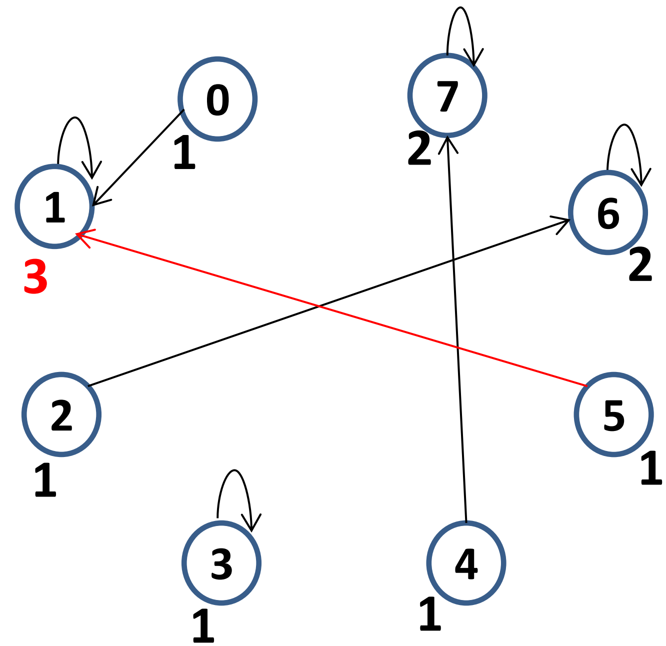
idx	Id	Sz
0	1	1
1	1	2
2	6	1
3	3	1
4	7	1
5	5	1
6	6	2
7	7	2



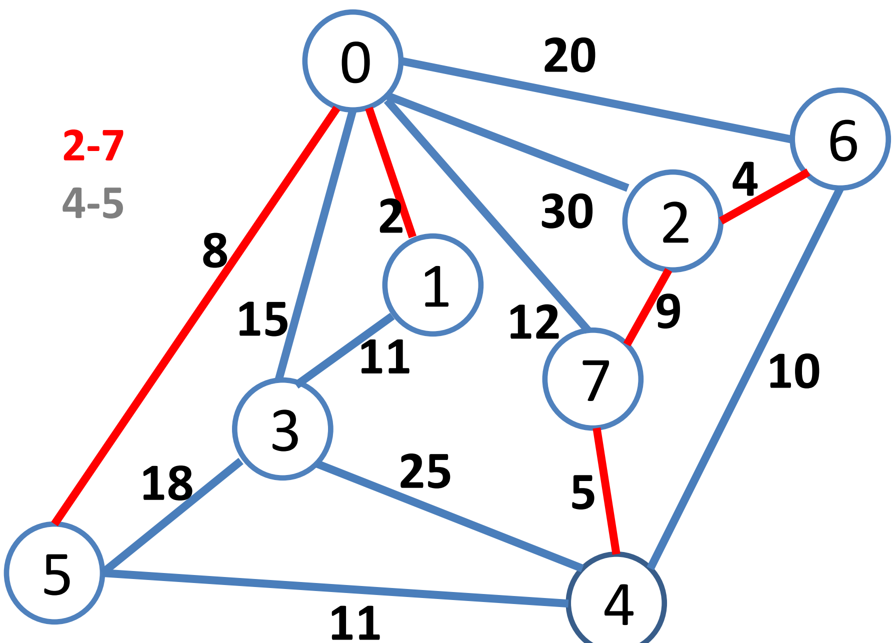
0-5



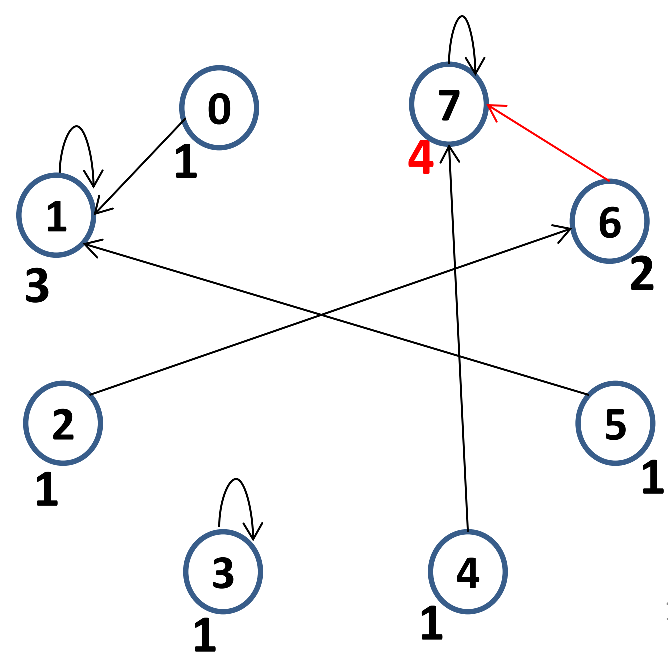
idx	Id	Sz
0	1	1
1	1	3
2	6	1
3	3	1
4	7	1
5	1	1
6	6	2
7	7	2



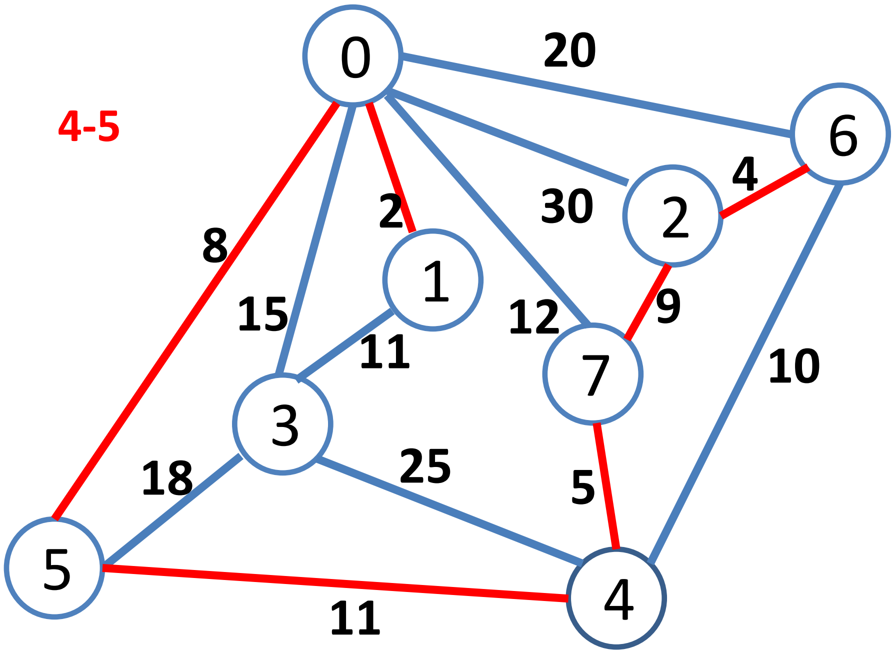
2-7
4-5



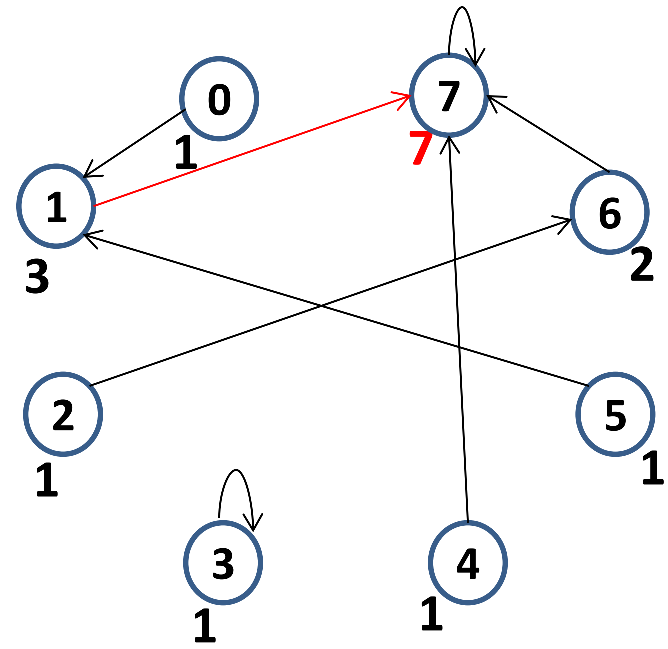
idx	Id	Sz
0	1	1
1	1	3
2	6	1
3	3	1
4	7	1
5	1	1
6	7	2
7	7	4



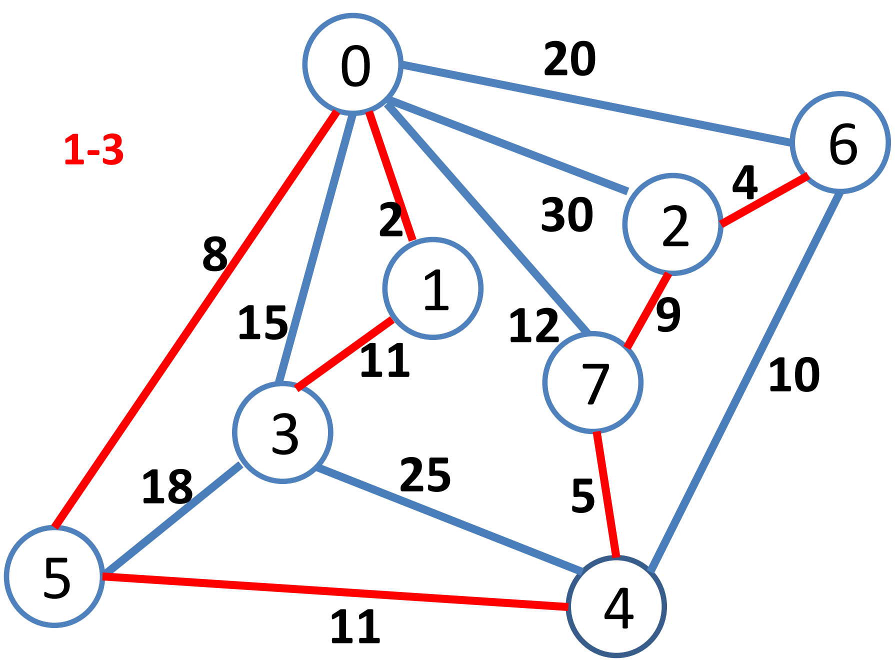
4-5



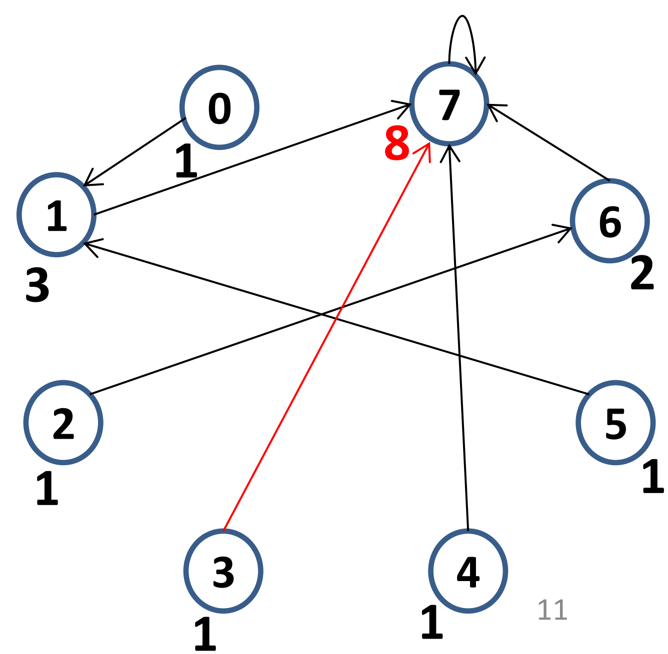
idx	Id	Sz
0	1	1
1	7	3
2	6	1
3	3	1
4	7	1
5	1	1
6	7	2
7	7	7



1-3

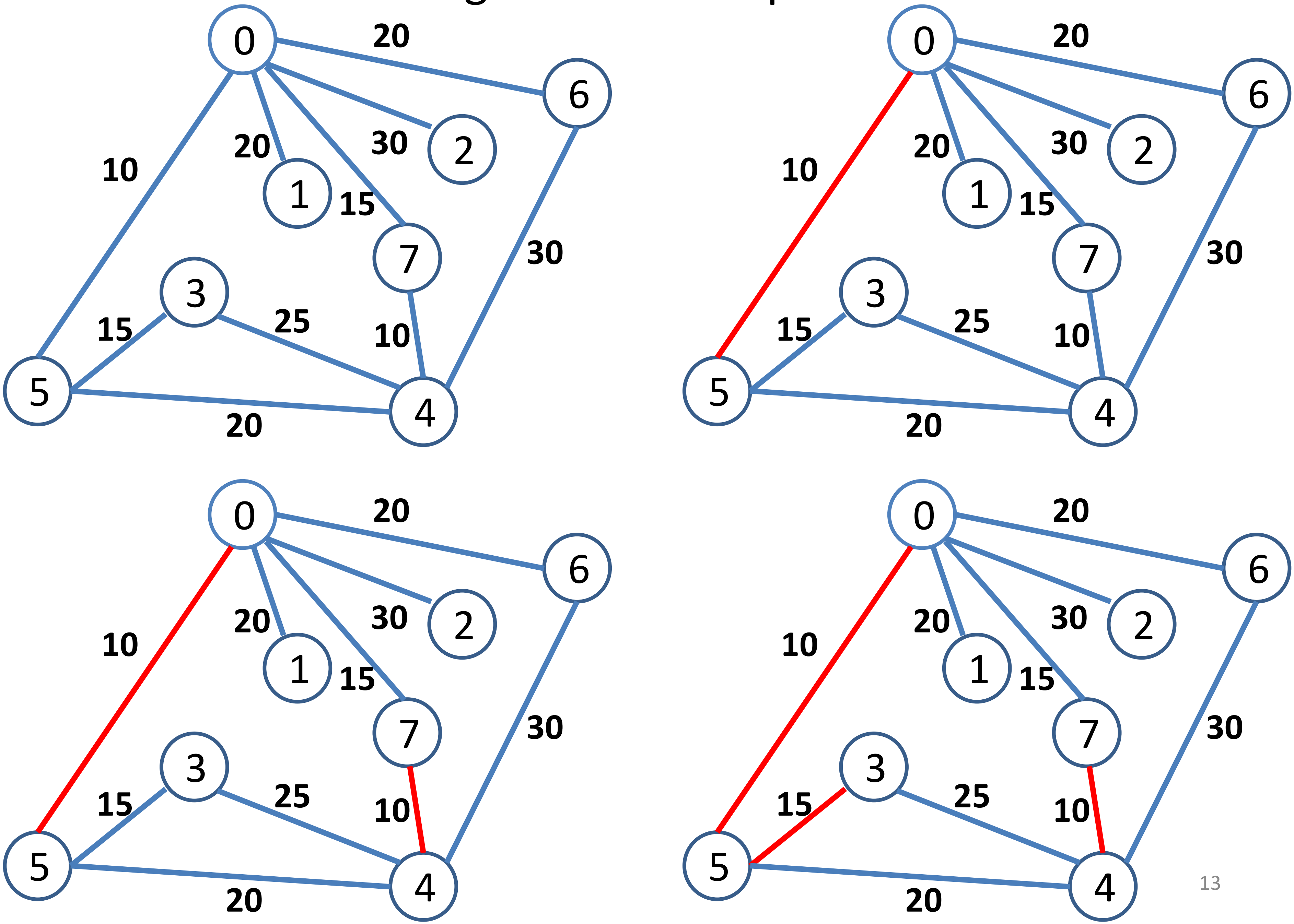


idx	Id	Sz
0	1	1
1	7	3
2	6	1
3	7	1
4	7	1
5	1	1
6	7	2
7	7	8



Kruskal’s Algorithm Example 2

Kruskal's Algorithm: Example 2 workout



Kruskal's Algorithm: Example 2 workout