Minimum Spanning Trees

CSE 3318 – Algorithms and Data Structures
Alexandra Stefan
University of Texas at Arlington

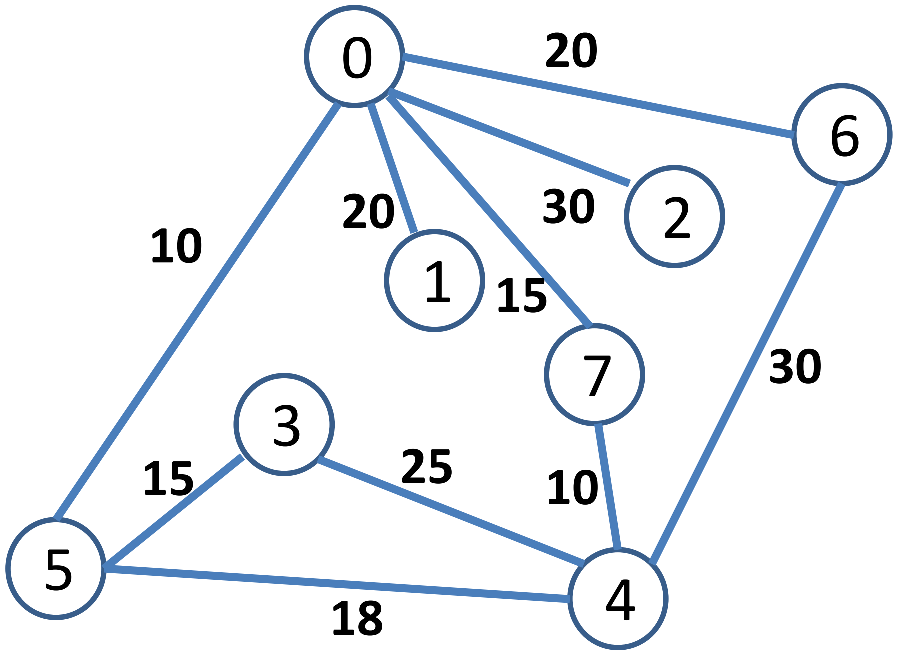
These slides are based on CLRS and "Algorithms in C" by R. Sedgewick

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 travel.
  - 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)

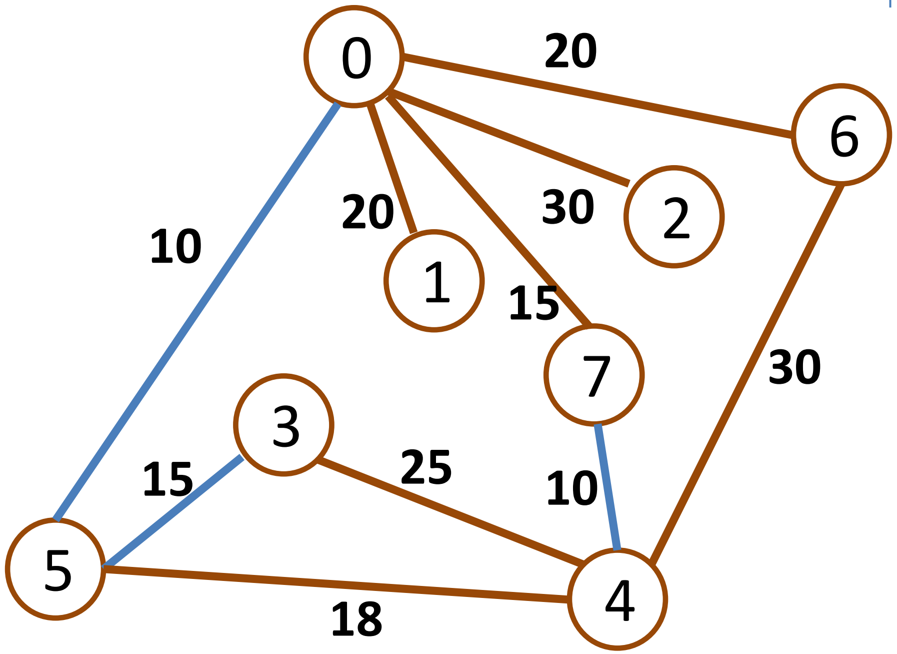
Spanning Tree

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

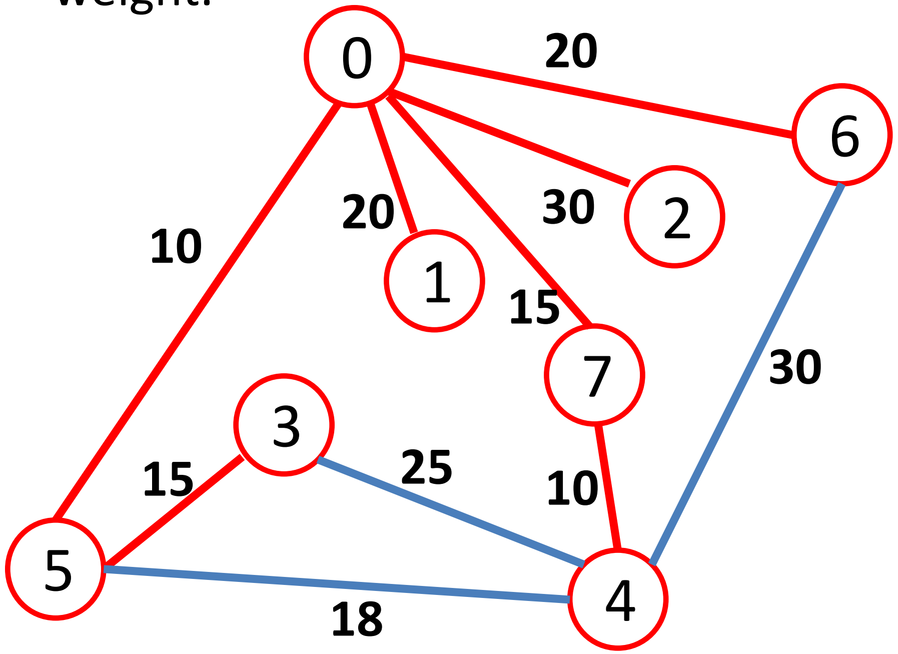
- The weight/cost of a spanning tree is the sum of weights of its edges.

- 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.

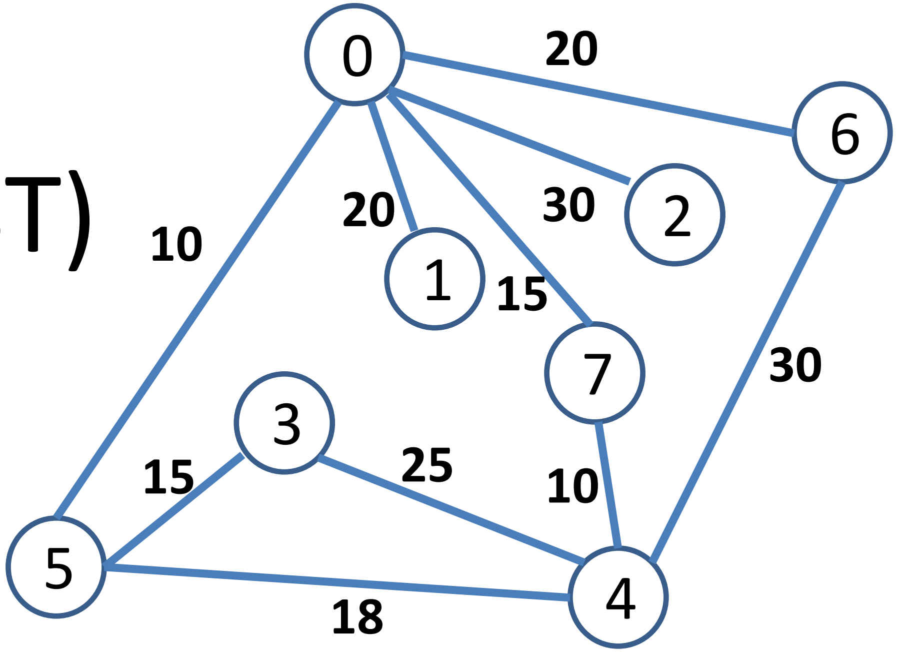


Weight:20+15+30+20+30+25+18 = 158

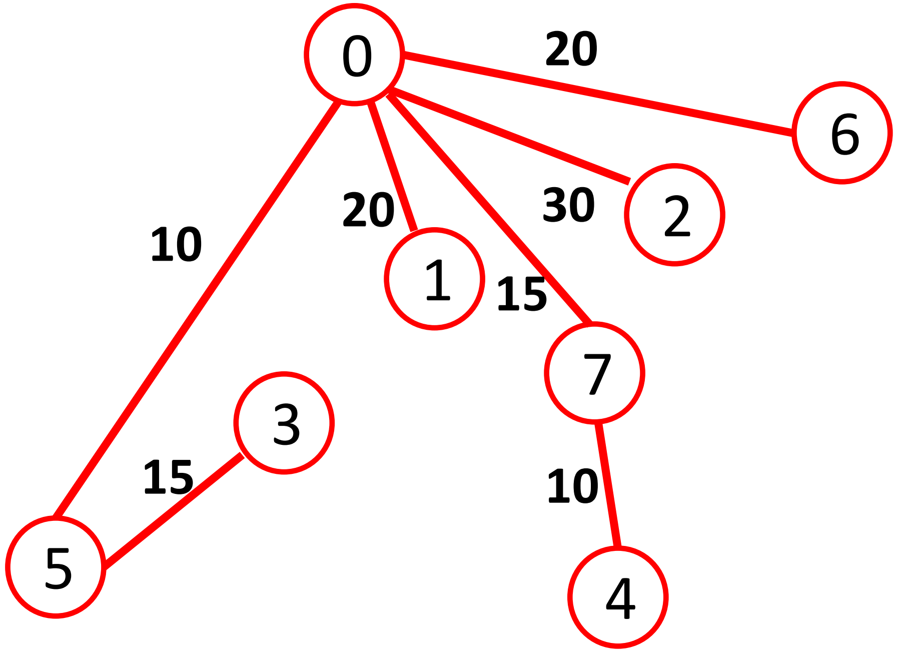


Weight:10+20+15+30+20+10+15 = 120

Minimum-Cost Spanning Tree (MST)



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



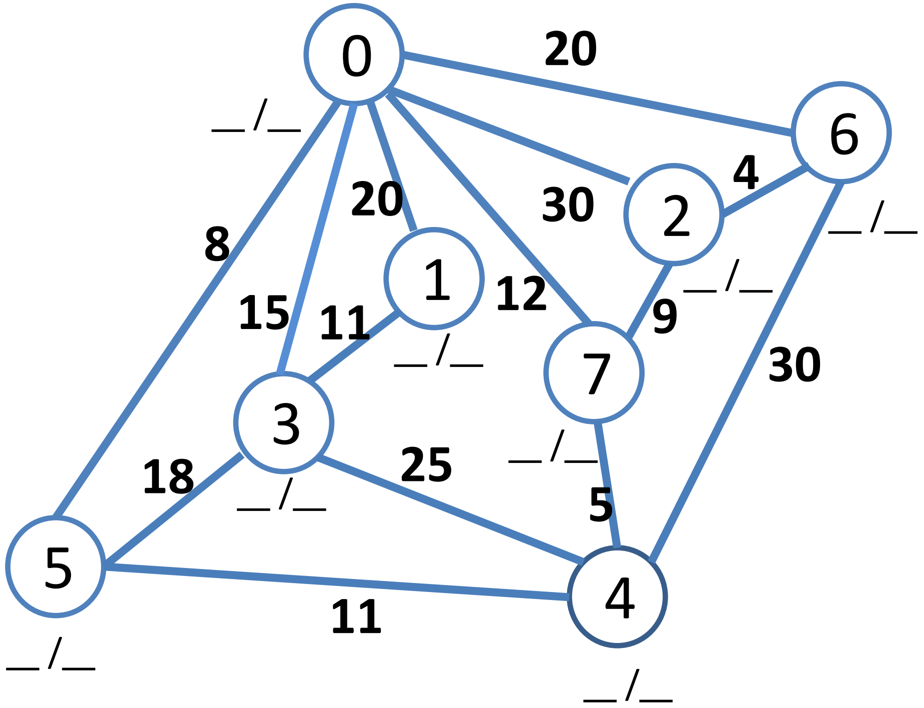
- Warning: later in the course (when we discuss Dijkstra's algorithm) we will make the opposite assumptions:
  - Allow directed graphs.
  - Not allow negative weights.

MST using

Prim's Algorithm

MST-Prim (G,w,7) Worksheet

```
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,d)
    While notEmpty(Q)
        u = removeMin(Q,d)  //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)
```



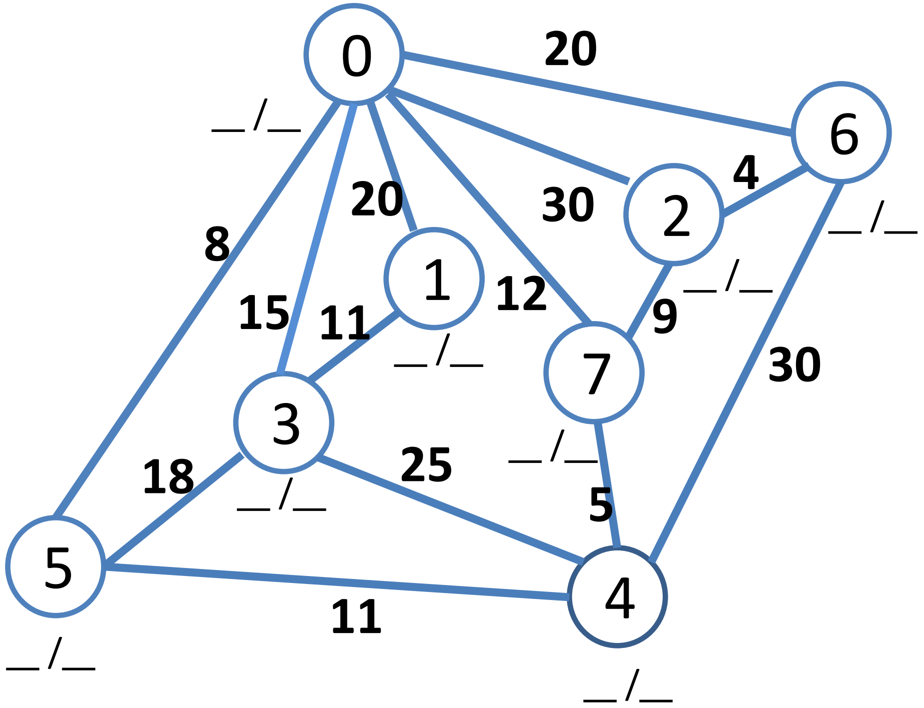
Vertex	0	1	2	3	4	5	6	7
d/p Work (dist and parent updates for nodes)	d[0]/p[0]	d[1]/p[1]	d[2]/p[2]	d[3]/p[3]	d[4]/p[4]	d[5]/p[5]	d[6]/p[6]	d[7]/p[7]

u,v,w

```
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,d)
    While notEmpty(Q)
        u = removeMin(Q,d)  //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)

Solution



Vertex	0	1	2	3	4	5	6	7
d/p Work (dist and parent updates for nodes)	d[0]/p[0]	d[1]/p[1]	d[2]/p[2]	d[3]/p[3]	d[4]/p[4]	d[5]/p[5]	d[6]/p[6]	d[7]/p[7]

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,d)
    While notEmpty(Q)
        u = removeMin(Q,d)  //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.

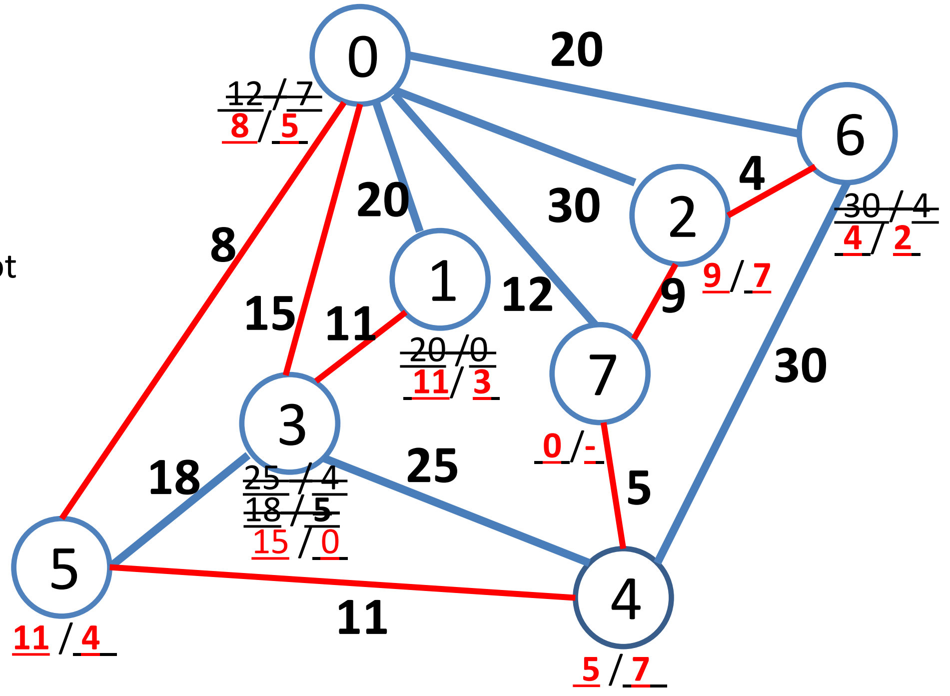
Run Prims algorithm starting at vertex 7.

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

(p = predecessor or parent)

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

Note: picked edges are not in increasing order of w



The p array stores the tree. Edges: (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(G.V, d)
While notEmpty(Q)
    u = removeMin(Q,d)
    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 Alg for Adjacency Matrix Graphs (without PriorityQueue)

```
int MST_Prim(int ** E, int ** weights, int V, int startVertex){
   int i, k, v, minVal, minVertex, total_cost = 0;
   int d[V], p[V];
   int mst[V]; // records what vertices are part of the MST so far. If mst[i]==1 then i is in the MST, else it is not.
   for(i=0;i<V;i++){
      d[i] = INT_MAX;        p[i] = -1;       mst[i]=0;   //mst[i]=0 => i is not in the mstf
   }
   d[startVertex] = 0;
   minVertex = startVertex;
   for(k=0; k<V; k++){   // (V-1) iterations to add the remaining V-1 vertices. Assume graph is connected.
      mst[minVertex] = 1; // mark that minVertex is part of the MST now
      total_cost += d[minVertex];
      for(v=0; v<V; v++){     // check neighbours of minVertex and update their min distances d[v] if needed
           //edge (minVertex,v) exists && v NOT in MST && weight of (minVertex,v) is less than the best seen so far
           if ( (E[minVertex][v]==1) && (mst[v]==0) && (d[v]>weights[minVertex][v]) ) {
                  d[v] = weights[minVertex][v];
                  p[v] = minVertex;
            }
       }
      // find a not colored vertex of min dist
      minVal = INT_MAX;
      minVertex = -1; // no vertex of min distance found so far
      for(v = 0; v<V; v++){
          if ( (mst[v]==0) && (d[v]<minVal) ) {
              minVal = d[v];
              minVertex = v;
          }
      }
      if ( minVertex==-1 && k<(V-1) ) {
           printf("Graph was not connected.");
           break;
       }
   }
   return total_cost;
}
```

Time: $O(V^2)$

Space: $O(V)$
for d, p, mst

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

Prim's Algorithm - Time Complexity – Adj Matrix

This TC analysis assumes :
- "v in Q" is O(1)
- "find v in Q" is O(1)
- Q – is a Heap

Space complexity: Θ(V) (for d,p, and Q)

Time complexity: $O(V^2 lgV)$ (for adj Matrix)
b.c.: $O(V + VlgV + V^2lgV)$
connected graph => $|E| \geq (|V|-1)$

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

```
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(G.V, d) ------------>   O(V)   (build heap)
7   While notEmpty(Q)        ------------>   O(V)
8       u = removeMin(Q,d)     ------------>   O(lgV)
9       for each v adjacent to u          //lines 7 & 9 together:  ---->  O(V^2)
10          if v in Q and w(u,v)<d[v]
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)

$O(V^2lgV)$
from lines:
7,9,13

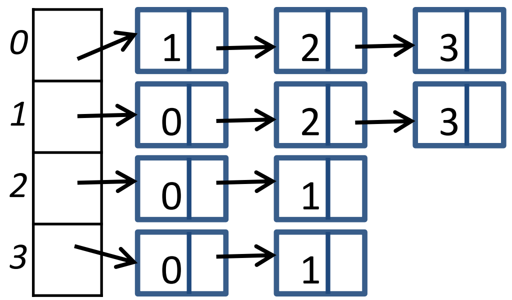
Prim's Algorithm - Time Complexity – Adj List

This TC analysis assumes :

- "v in Q" is O(1)
- "find v in Q" is O(1)
- Q – is a Heap

Space complexity: Θ(V) (for d,p, and Q)

Time complexity: O(ElgV) (for adj List)
b.c.: O(V + VlgV + E lgV)
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(G.V, d) ------------>   O(V)   (build heap)
7   While notEmpty(Q)         ------------>   O(V)
8       u = removeMin(Q,d)     ------------>    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(G.V, d)
While notEmpty(Q)
    u = removeMin(Q,d)
    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).
  - Can you use PriorityQueue in Java?
    - how? What class method will you use?
    - what time complexity do you get?
- Find heap node corresponding to v.
  - Needed to update the heap according to smaller *d[v]*.
  - Note the difference between v and node in heap corresponding to v.
  - See heap slides : Index Heap Example
  - how will you "implement" this if you are using PriorityQueue in Java ?

Other

- Variations
  - start with an empty priority queue, add vertexes newly discovered.
    - Need to know when a vertex is in the tree/in frontier/undiscovered
  - For dense graphs, keep and array (instead of a priority queue => $O(V^2)$ – optimal for dense graphs) – see Sedgewick if interested.
  - Keep a priority queue of edges
- 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?
  - Optimization
- What type of method is:
  - Prim's – Greedy
  - If covered: Kruskal's - Greedy
- Can we prove that they give the MST? - Yes (see extra slides)

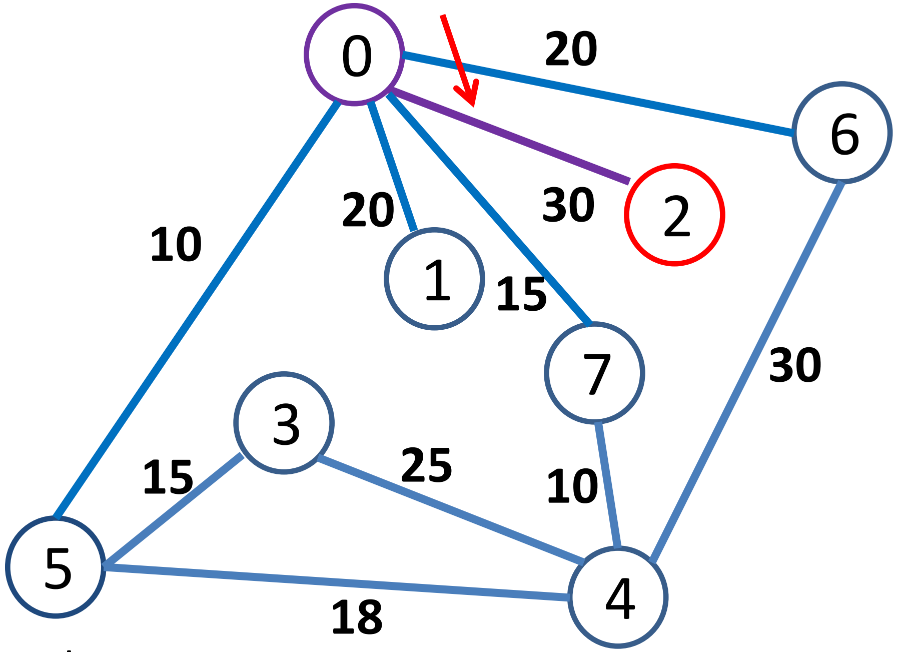
Prim – Example 2
Step-by-Step

Prim's Alg Example 2 step-by-step

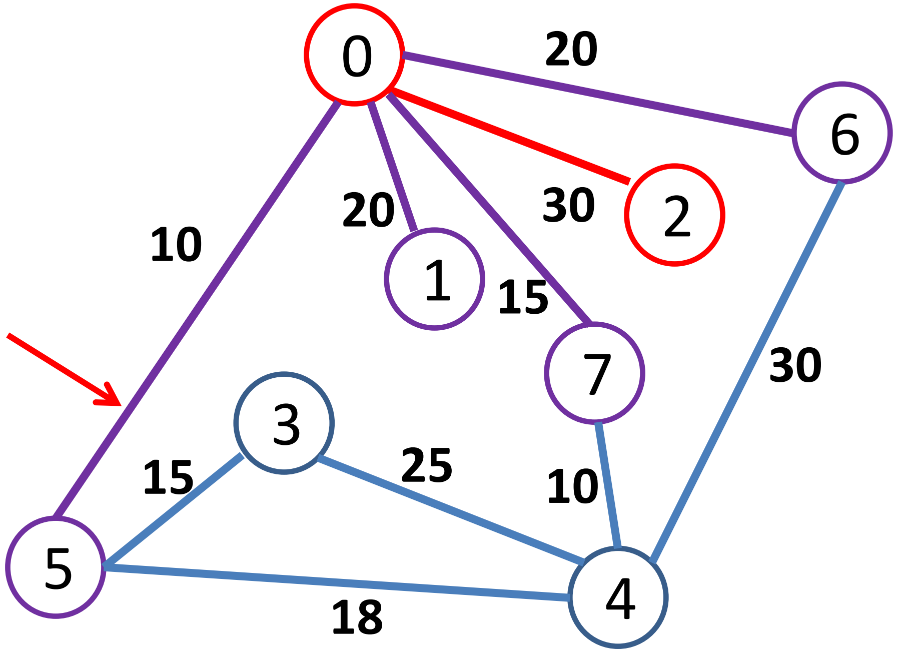
```
MST_Prim(G,w,s) // N = |V|
    int d[N], p[N]
    For v =0 -> N-1
        d[v]=inf
        p[v]=-1
    d[s]=0
    Q = PriorityQueue(G.V,w)
    While notEmpty(Q)
        u = removeMin(Q,w)  //u is picked
        for each v adjacent to u
            if v in Q and w(u,v)<d[v]
                p[v]=u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

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

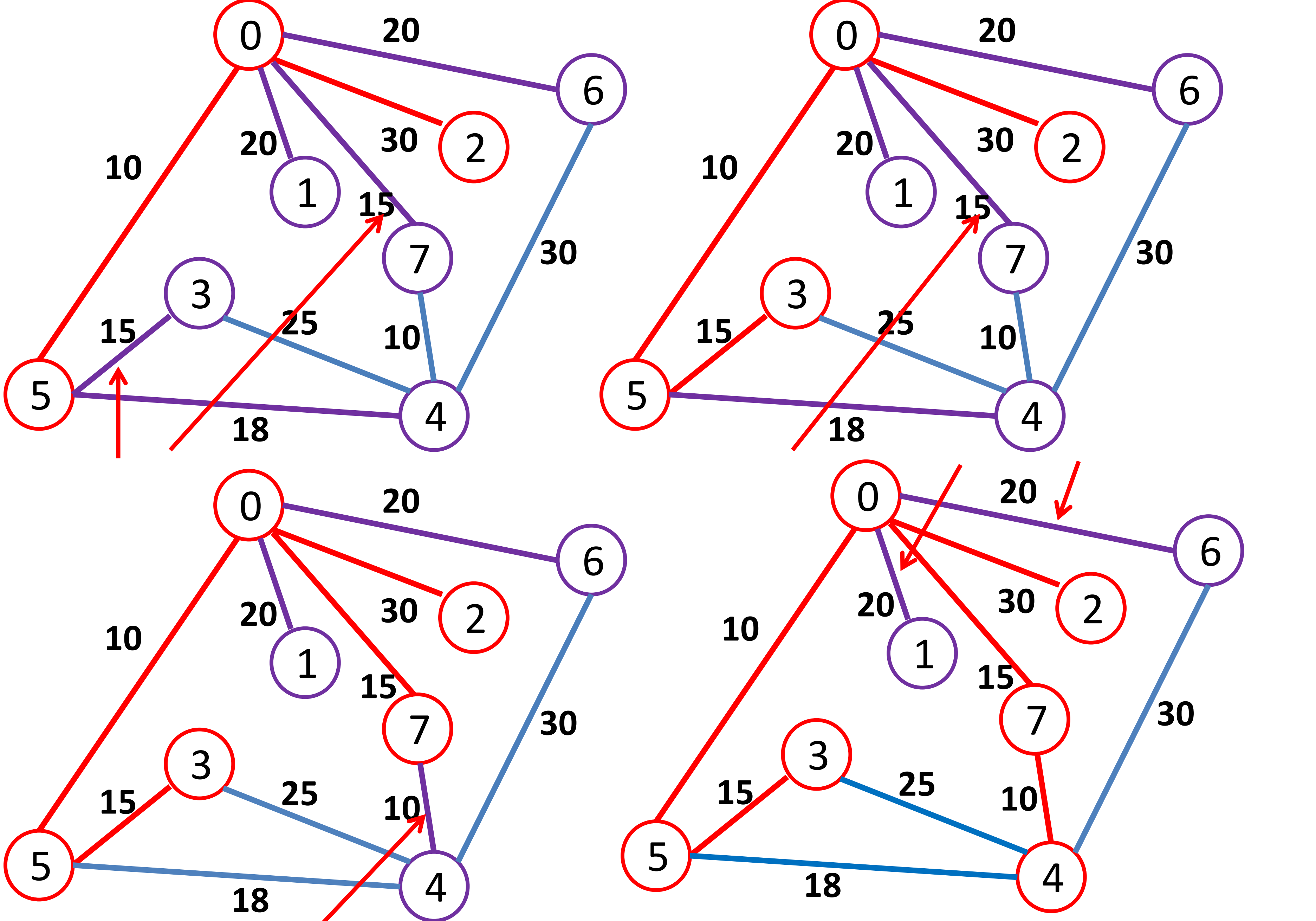
(This example shows the frontier (edges and vertices).



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

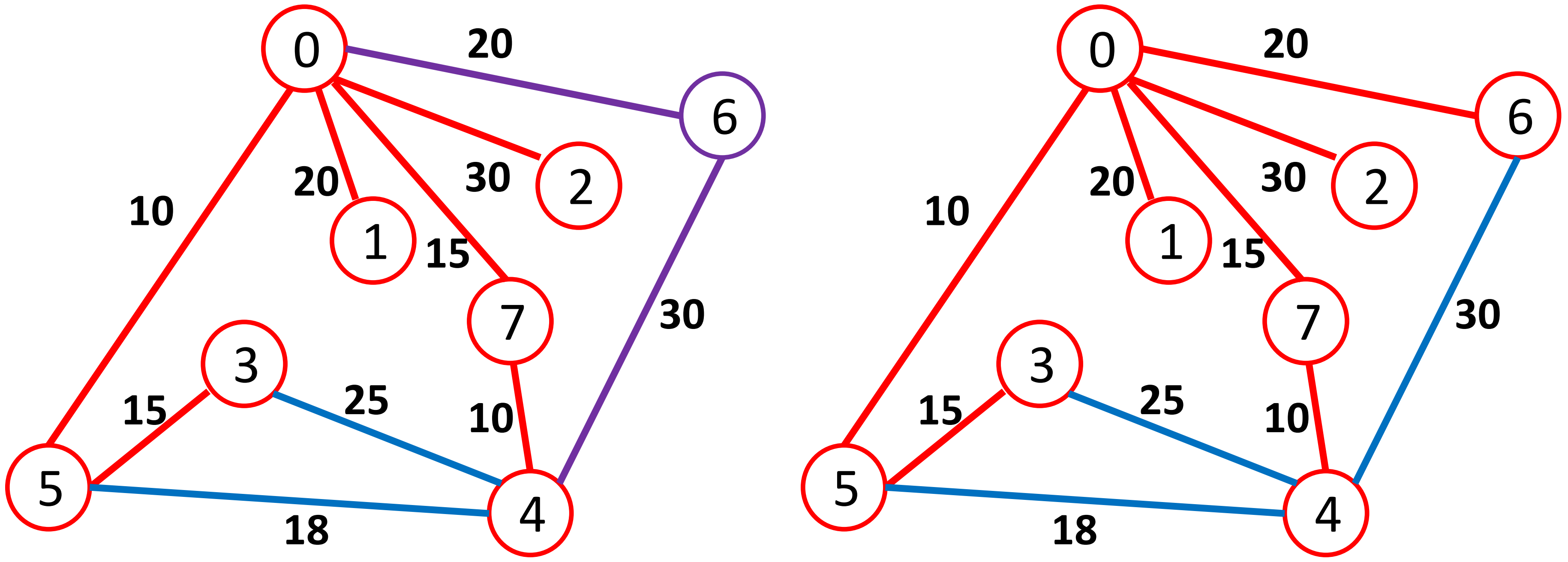


The algorithm will keep a MIN-Priority Queue for the vertices.



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



Correctness of Prim's Algorithm

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'.