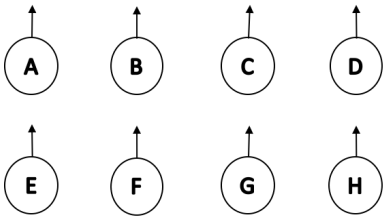
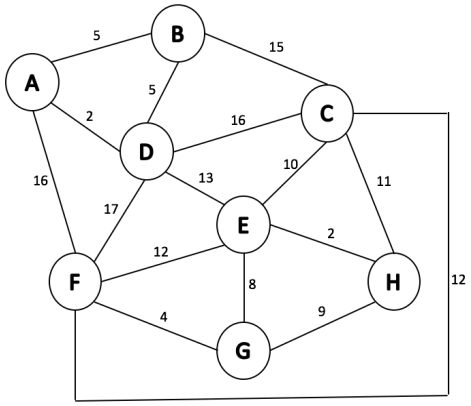
CS 225 Spring 2019 :: TA Lecture Notes 4/19 MST

By Wenjie

Kruskal's Algorithm

- Setup
  - Maintain a list of edges sorted by weight in increasing order → min heap
  - Initialize a disjoint set for every vertex
    - If two vertices are in the same upTree, they are connected





(A,D)
(E,H)
(F,G)
(A,B)
(B,D)
(G,E)
(G,H)
(E,C)
(C,H)
(E,F)
(F,C)
(D,E)
(B,C)
(C,D)
(A,F)
(D,F)

- The code

```
1  KruskalMST(G):
2       DisjointSets forest
3       foreach (Vertex v : G):
4            forest.makeSet(v)
5
6       PriorityQueue Q     // min edge weight
7       foreach (Edge e : G):
8            Q.insert(e)
```

CS 225 Spring 2019 :: TA Lecture Notes 4/19 MST

By Wenjie

```
9
10      Graph T = (V, {})
11
12      while |T.edges()| < n-1:
13           Vertex (u, v) = Q.removeMin()
14           if forest.find(u) != forest.find(v):
15                T.addEdge(u, v)
16                forest.union( forest.find(u),
17                                        forest.find(v) )
18      return T
```

- The algorithm logic:
  - Take the edge with the smallest weight from the heap
  - If the two endpoints are not already connected
    - add the edge into our spanning tree
    - union the two vertices
  - If they are already connected: skip the edge and do nothing, otherwise we create a cycle

- **Running time**
  - Depending on implementation

Priority Queue	Heap	Sorted Array
Build DSet (line 2-4)	$O(n)$	$O(n)$
Build Heap (line 6-8)	$O(m)$	$O(m\ log(m))$
While Loop (line 12)	$O(m)$	$O(m)$
removeMin (line 13)	$O(log(m))$	$O(1)$
DSet operations (14-17)	$O(1)$	$O(1)$
Total	$O(n + m + m\ log(m))$ $= O(n + m\ log(m))$	$O(n + m\ log(m) + m)$ $= O(n + m\ log(m))$

  - So they are the same!

CS 225 Spring 2019 :: TA Lecture Notes 4/19 MST

By Wenjie

    - However, we would like to have everything in terms of data (n) and not edges (m).
        - $m = O(n^2) \leq c * n^2$ → we can log both sides because it is an increasing function.
        - $log(m) \leq log(c * n^2) \Rightarrow log(m) \leq log(c) + log(n^2) \Rightarrow log(m) \leq log(c) + 2 * log(n)$
        - Therefore, we can conclude that $O(log(m))$ is $O(log(n))$ and we can sub log(n) for log(m) everywhere in the previously done analysis.
        - For the worst case, $m = n^2$
        - We cannot substitute n^2 for m because that will fundamentally change the running time (relationship between n and m depends on the sparsity of the graph).
    - Final running time
        - $O(n + m\ log(n))$

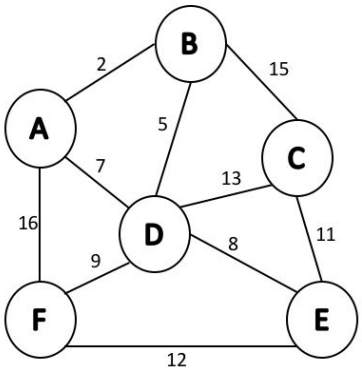
- Tradeoffs between using heap and sorted array:
    - Heap:
        - The build time for heap is faster.
            - This is important because build time is unavoidable
        - We might short circuit in the middle of our run → may run less than $m * log(n)$ operations
        - It is faster to update if we have a dynamically changing graph → heaps take $log(n)$ to update while sorted array takes $O(n)$
    - Sorted Array:
        - It is easier to implement
        - Sorted array stays intact after our algorithm
            - When we remove stuff from the heap, we slowly destroy our heap
        - Sorted arrays are more useful (we can get more out of them than just minimum)

CS 225 Spring 2019 :: TA Lecture Notes 4/19 MST

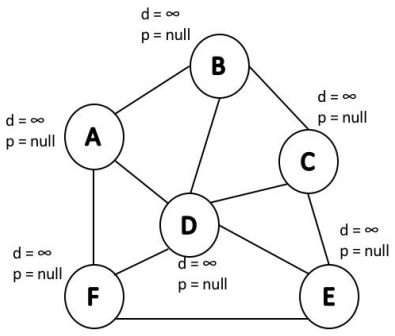
By Wenjie

- ## Prim's Algorithm

Prim's algorithm constructs an MST by using the idea of partitions.
We will work with the following graph:



- Algorithm setup:
  - Label all vertices with:
    - Distance → initialized to infinity.
    - Predecessor → initialized to null.
  - Set up a min heap.



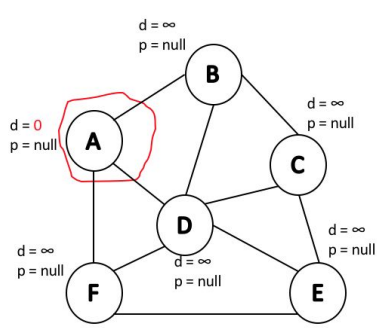
A	∞
B	∞
C	∞
D	∞
E	∞
F	∞

- Algorithm logic:
  - Choose an arbitrary starting point and set its distance to 0.

CS 225 Spring 2019 :: TA Lecture Notes 4/19 MST

By Wenjie

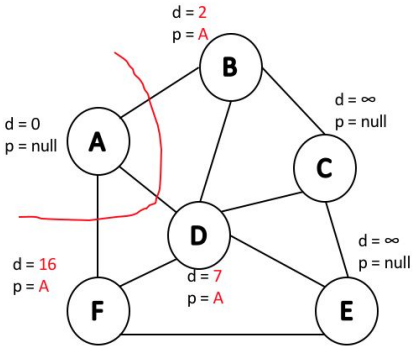
In our example, we will choose vertex A as a starting point.



A	0
B	∞
C	∞
D	∞
E	∞
F	∞

- Pop the starting vertex from the heap and update the distance/predecessor of adjacent vertices.

We pop A and update adjacent vertices B, D, and F.



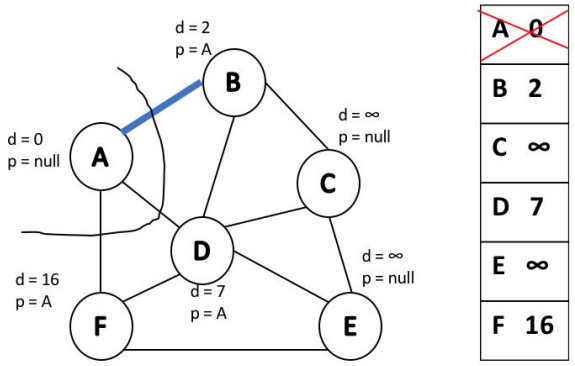
~~A~~	~~0~~
B	2
C	∞
D	7
E	∞
F	16

- We want to add an edge to the node with the smallest distance.

In the example, that is an edge from A to B because it has the smallest distance.

CS 225 Spring 2019 :: TA Lecture Notes 4/19 MST

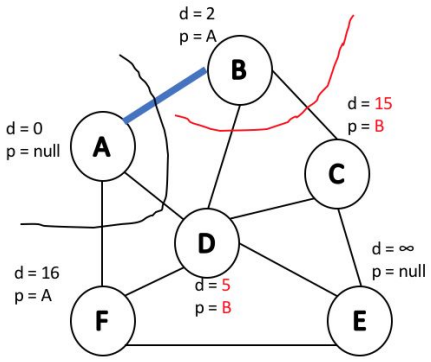
By Wenjie



A	0
B	2
C	∞
D	7
E	∞
F	16

- Next, we pop a vertex with the smallest distance and update adjacent vertices. However, we update vertices only if the distance is smaller than the current.

We update adjacent vertices D and C.



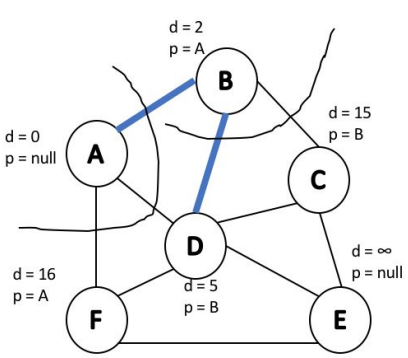
A	0
B	2
C	15
D	5
E	∞
F	16

- We add an edge to the node with the smallest distance.

The node with the smallest distance is D, so we add edge BD.

CS 225 Spring 2019 :: TA Lecture Notes 4/19 MST

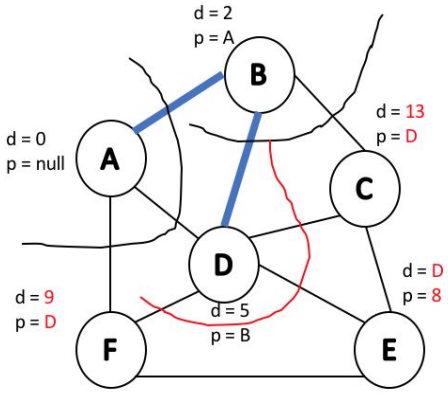
By Wenjie



A	0
B	2
C	15
D	5
E	∞
F	16

- Again, we pop a vertex with the smallest distance, we update adjacent vertices if needed, and we add the edge with the smallest distance. These steps are repeated until the heap is empty.

Continuing through the example graph → we pop D and we update all its adjacent vertices F, E, and C; because for all three, current distance is higher than from D to each of them.

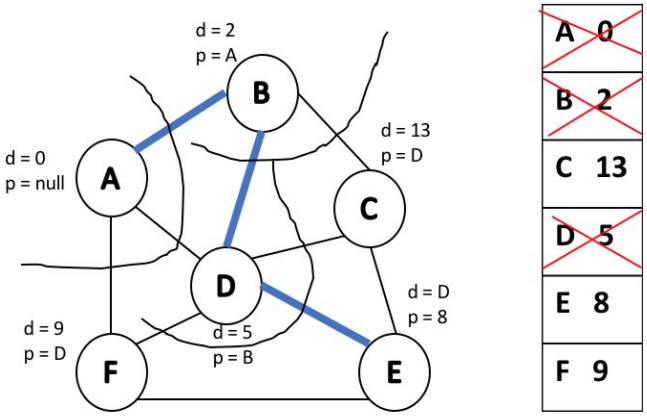


A	0
B	2
C	13
D	5
E	8
F	9

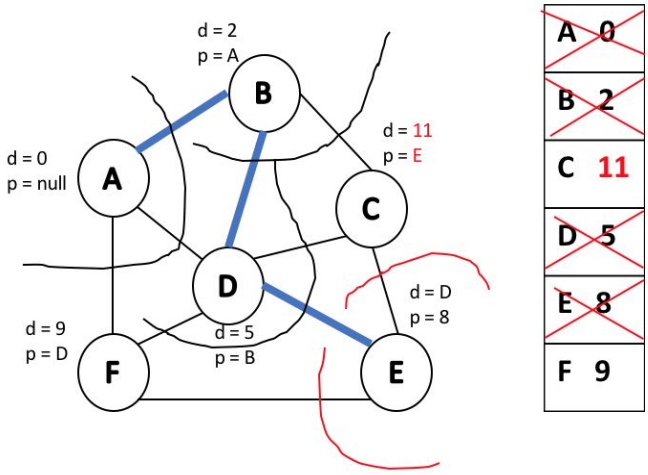
The next vertex with smallest distance is E. Thereby, we add an edge from D to E.

CS 225 Spring 2019 :: TA Lecture Notes 4/19 MST

By Wenjie



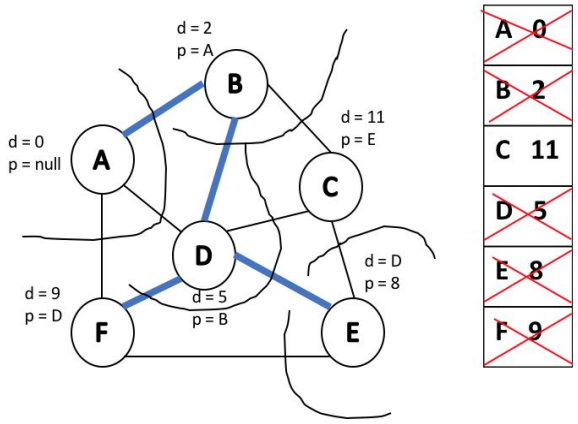
We further pop E and we only update C, because F's current distance is smaller than the one from E to F.



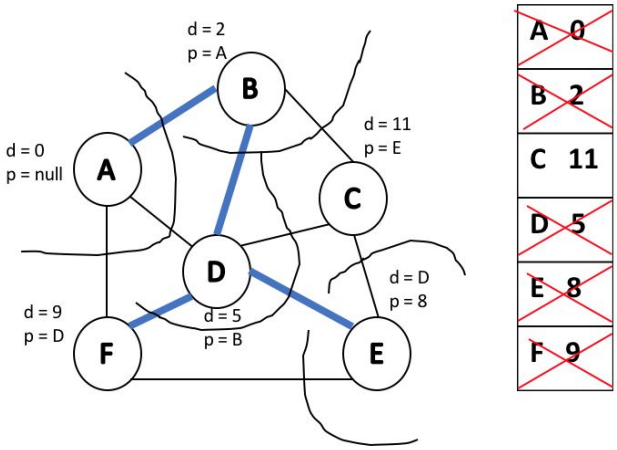
The shortest distance is from D to F, so we add that edge to the graph.

CS 225 Spring 2019 :: TA Lecture Notes 4/19 MST

By Wenjie



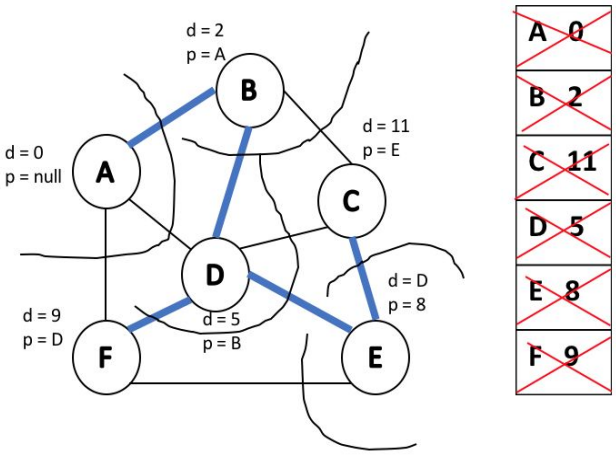
We pop 9 and we don't have anything to update because all neighbouring edges have been added to the graph.



Finally, we pop C and add an edge from E to C. After this step the heap is empty and we are done.

CS 225 Spring 2019 :: TA Lecture Notes 4/19 MST

By Wenjie



- **Pseudocode for Prim's algorithm**

```
1   PrimMST(G, s):
2     Input: G, Graph;
3            s, vertex in G, starting vertex
4     Output: T, a minimum spanning tree (MST) of G
5
6     foreach (Vertex v : G):
7       d[v] = +inf
8       p[v] = NULL
9     d[s] = 0
10
11    PriorityQueue Q   // min distance, defined by d[v]
12    Q.buildHeap(G.vertices())
13    Graph T           // "labeled set"
14
15    repeat n times:
16      Vertex m = Q.removeMin()
17      T.add(m)
18      foreach (Vertex v : neighbors of m not in T):
19        if cost(v, m) < d[v]:
20          d[v] = cost(v, m)
21          p[v] = m
22
23    return T
```