CS 225

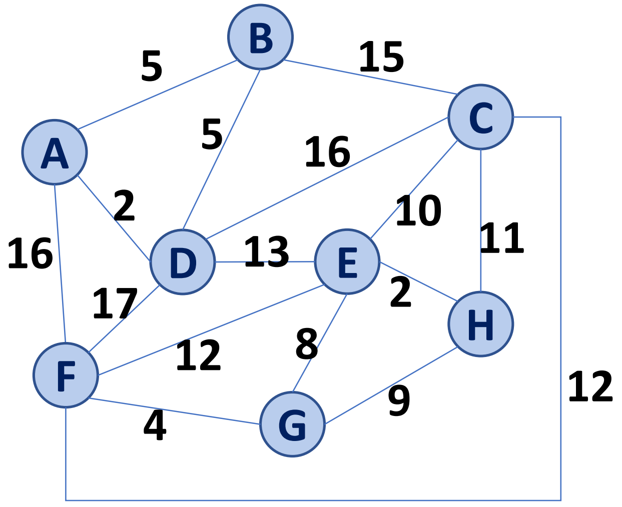
Data Structures

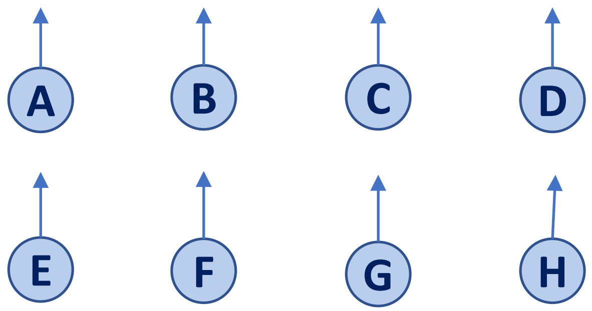
April 20 – Kruskal + Prim's Algorithm

Wade Fagen-Ulmschneider

Kruskal's Algorithm

(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)






```
KruskalMST(G):
  DisjointSets forest
  foreach (Vertex v : G):
    forest.makeSet(v)

  PriorityQueue Q    // min edge weight
  foreach (Edge e : G):
    Q.insert(e)

  Graph T = (V, {})

  while |T.edges()| < n-1:
    Vertex (u, v) = Q.removeMin()
    if forest.find(u) == forest.find(v):
       T.addEdge(u, v)
       forest.union( forest.find(u),
                     forest.find(v) )

  return T
```

Kruskal's Algorithm

Priority Queue:	Heap	Sorted Array
Building :6-8		
Each removeMin :13		

```
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)
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
19   return T
```

Kruskal's Algorithm

Priority Queue:	Total Running Time
Heap	
Sorted Array	

```
KruskalMST(G):
  DisjointSets forest
  foreach (Vertex v : G):
    forest.makeSet(v)

  PriorityQueue Q    // min edge weight
  foreach (Edge e : G):
    Q.insert(e)

  Graph T = (V, {})

  while |T.edges()| < n-1:
    Vertex (u, v) = Q.removeMin()
    if forest.find(u) == forest.find(v):
       T.addEdge(u, v)
       forest.union( forest.find(u),
                     forest.find(v) )

  return T
```

Kruskal’s Algorithm

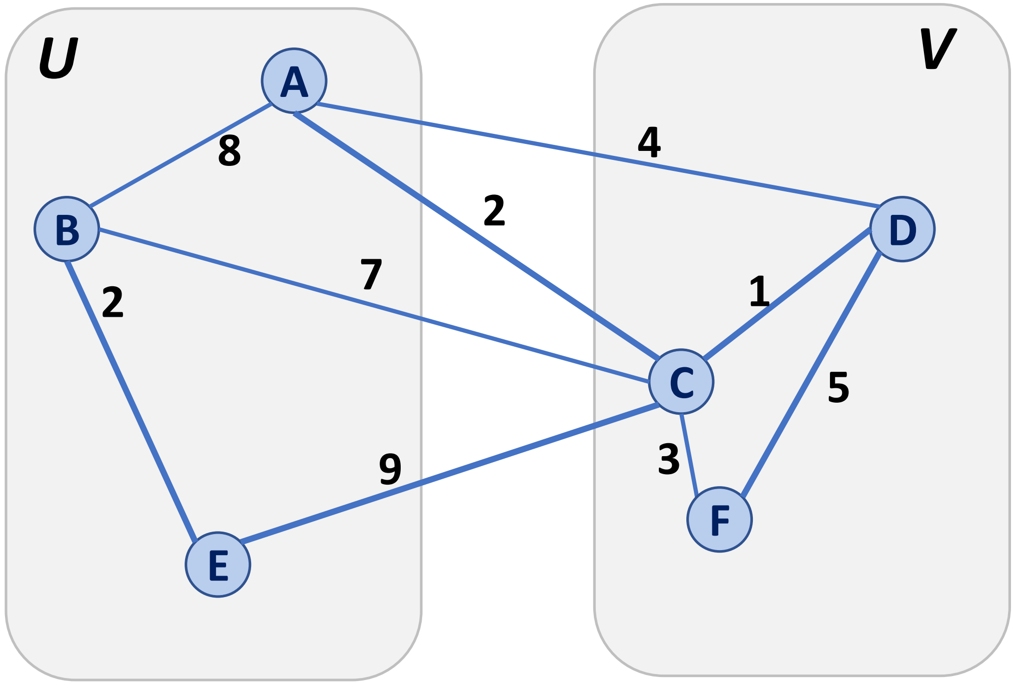
Which Priority Queue Implementation is better for running Kruskal’s Algorithm?

- Heap:

- Sorted Array:

Partition Property

Consider an arbitrary partition of the vertices on **G** into two subsets **U** and **V**.

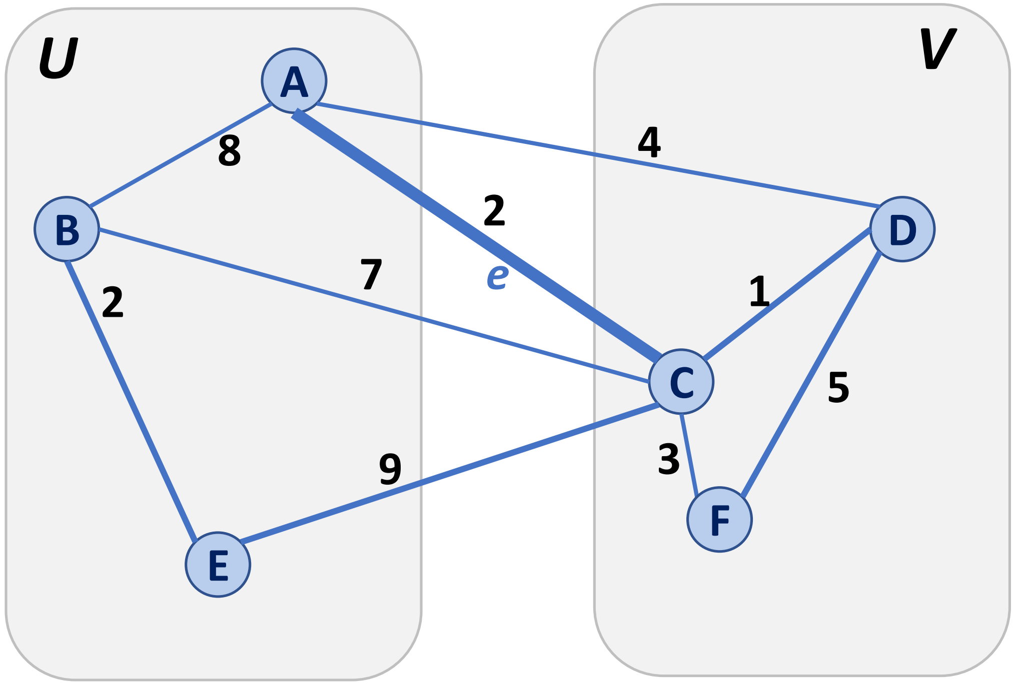


Partition Property

Consider an arbitrary partition of the vertices on **G** into two subsets **U** and **V**.

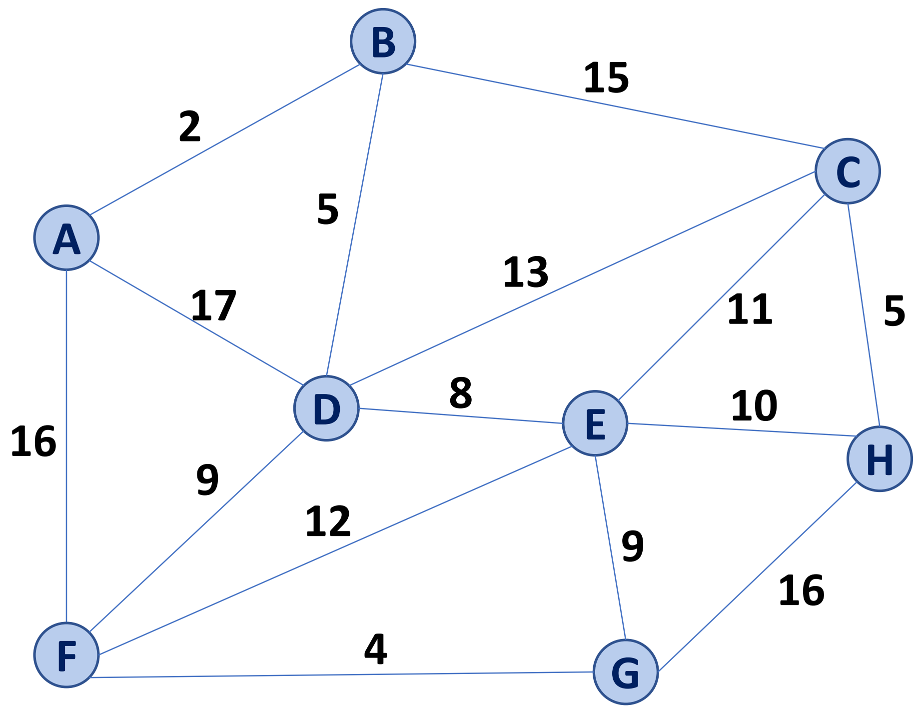
Let **e** be an edge of minimum weight across the partition.

Then **e** is part of some minimum spanning tree.

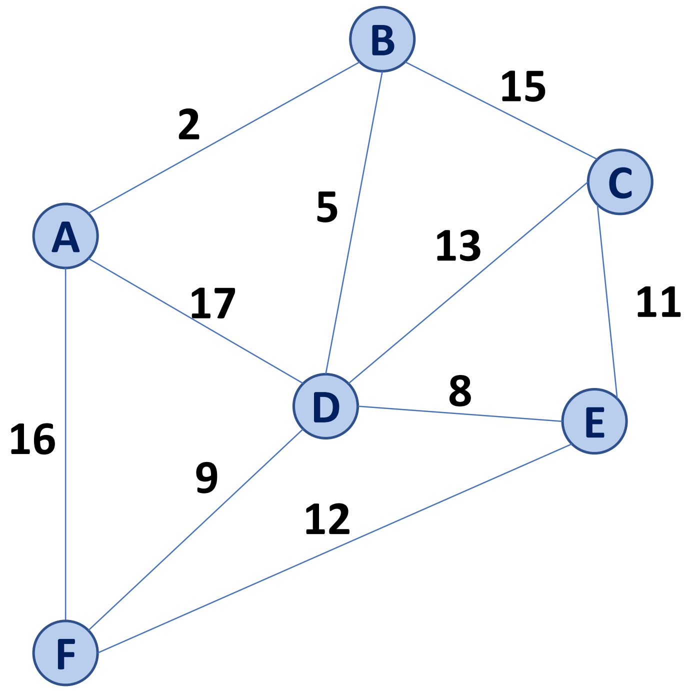


Partition Property

The partition property suggests an algorithm:



Prim’s Algorithm




```
PrimMST(G, s):
  Input: G, Graph;
         s, vertex in G, starting vertex
  Output: T, a minimum spanning tree (MST) of G

  foreach (Vertex v : G):
    d[v] = +inf
    p[v] = NULL
  d[s] = 0

  PriorityQueue Q   // min distance, defined by d[v]
  Q.buildHeap(G.vertices())
  Graph T           // "labeled set"

  repeat n times:
    Vertex m = Q.removeMin()
    T.add(m)
    foreach (Vertex v : neighbors of m not in T):
      if cost(v, m) < d[v]:
        d[v] = cost(v, m)
        p[v] = m

  return T
```

Prim's Algorithm

```
PrimMST(G, s):
  foreach (Vertex v : G):
    d[v] = +inf
    p[v] = NULL
  d[s] = 0

  PriorityQueue Q // min distance, defined by d[v]
  Q.buildHeap(G.vertices())
  Graph T         // "labeled set"

  repeat n times:
    Vertex m = Q.removeMin()
    T.add(m)
    foreach (Vertex v : neighbors of m not in T):
      if cost(v, m) < d[v]:
        d[v] = cost(v, m)
        p[v] = m
```

	Adj. Matrix	Adj. List
Heap		
Unsorted Array		

Prim's Algorithm

Sparse Graph:

Dense Graph:

```
PrimMST(G, s):
  foreach (Vertex v : G):
    d[v] = +inf
    p[v] = NULL
  d[s] = 0

  PriorityQueue Q // min distance, defined by d[v]
  Q.buildHeap(G.vertices())
  Graph T          // "labeled set"

  repeat n times:
    Vertex m = Q.removeMin()
    T.add(m)
    foreach (Vertex v : neighbors of m not in T):
      if cost(v, m) < d[v]:
        d[v] = cost(v, m)
        p[v] = m
```

	Adj. Matrix	Adj. List
Heap	$O(n^2 + m \lg(n))$	$O(n \lg(n) + m \lg(n))$
Unsorted Array	$O(n^2)$	$O(n^2)$

MST Algorithm Runtime:

- Kruskal's Algorithm:

  O(n + m lg(n))

- Prim's Algorithm:

  O(n lg(n) + m lg(n))

- What must be true about the connectivity of a graph when running an MST algorithm?

  - How does n and m relate?

MST Algorithm Runtime:

- Kruskal's Algorithm:

  O(n + m lg(n))

- Prim's Algorithm:

  O(n lg(n) + m lg(n))