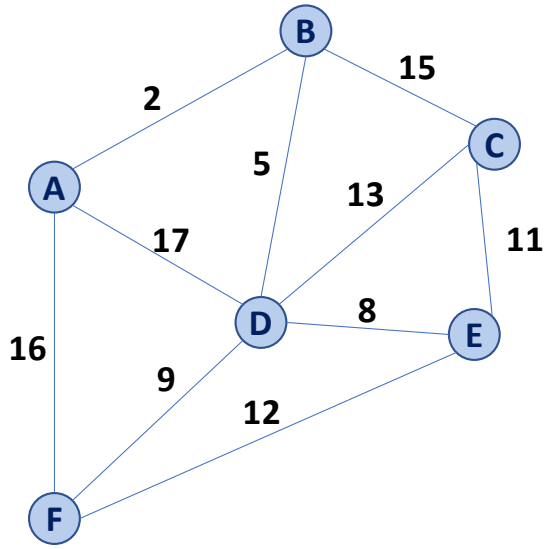
# CS 225

**Data Structures**

*December 2 – MST and Dijkstra's Algorihtm*

*G Carl Evans*

# 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
```

# Prim's Algorithm

```
 6  PrimMST(G, s):
 7    foreach (Vertex v : G):
 8      d[v] = +inf
 9      p[v] = NULL
10    d[s] = 0
11
12    PriorityQueue Q // min distance, defined by d[v]
13    Q.buildHeap(G.vertices())
14    Graph T          // "labeled set"
15
16    repeat n times:
17      Vertex m = Q.removeMin()
18      T.add(m)
19      foreach (Vertex v : neighbors of m not in T):
20        if cost(v, m) < d[v]:
21          d[v] = cost(v, m)
22          p[v] = m
```

| | Adj. Matrix | Adj. List |
|---|---|---|
| **Heap** | | |
| **Unsorted Array** | | |

# Prim's Algorithm

**Sparse Graph:**

**Dense Graph:**

```
 6  PrimMST(G, s):
 7    foreach (Vertex v : G):
 8      d[v] = +inf
 9      p[v] = NULL
10    d[s] = 0
11
12    PriorityQueue Q // min distance, defined by d[v]
13    Q.buildHeap(G.vertices())
14    Graph T          // "labeled set"
15
16    repeat n times:
17      Vertex m = Q.removeMin()
18      T.add(m)
19      foreach (Vertex v : neighbors of m not in T):
20        if cost(v, m) < d[v]:
21          d[v] = cost(v, m)
22          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:

We know that MSTs are always run on a minimally connected graph:

$$\mathbf{n-1 \leq m \leq n(n-1) / 2}$$

$$\mathbf{O(n) \leq O(m) \leq O(n^2)}$$

# MST Algorithm Runtime:

- Kruskal's Algorithm:

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

Sparse Graph:

Dense Graph:

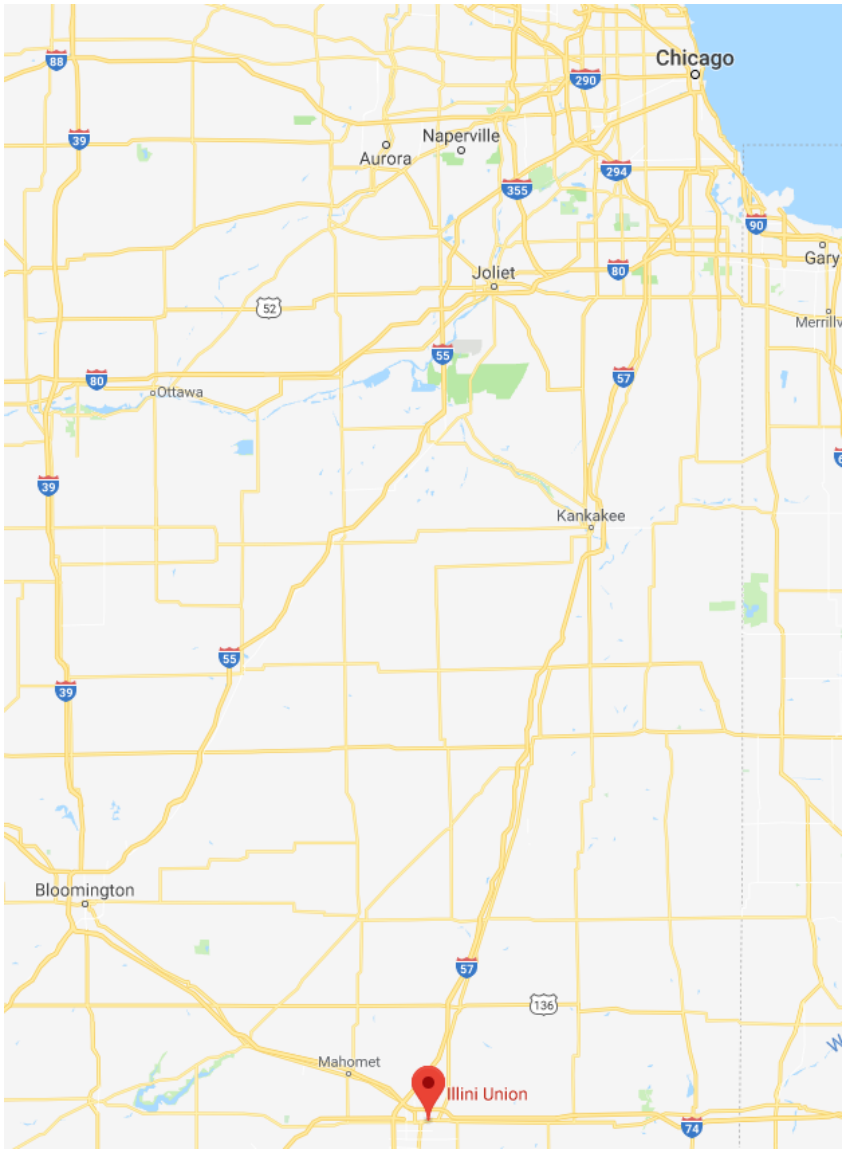
- Prim's Algorithm:

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

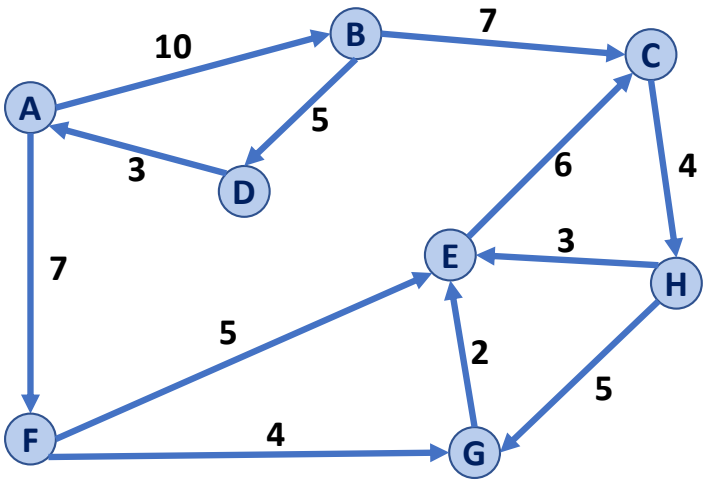
Sparse Graph:

Dense Graph:

# Shortest Path



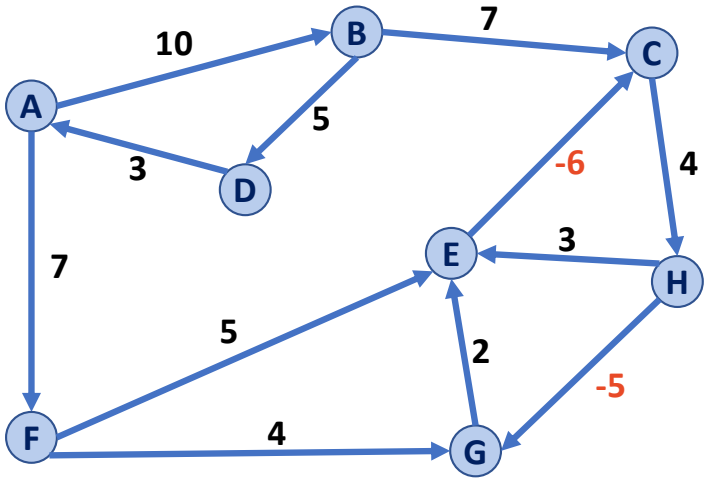
# Dijkstra's Algorithm (SSSP)




```
    DijkstraSSSP(G, s):
 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 u = Q.removeMin()
17      T.add(u)
18      foreach (Vertex v : neighbors of u not in T):
19        if _________________ < d[v]:
20          d[v] = ____________________
21          p[v] = m
```

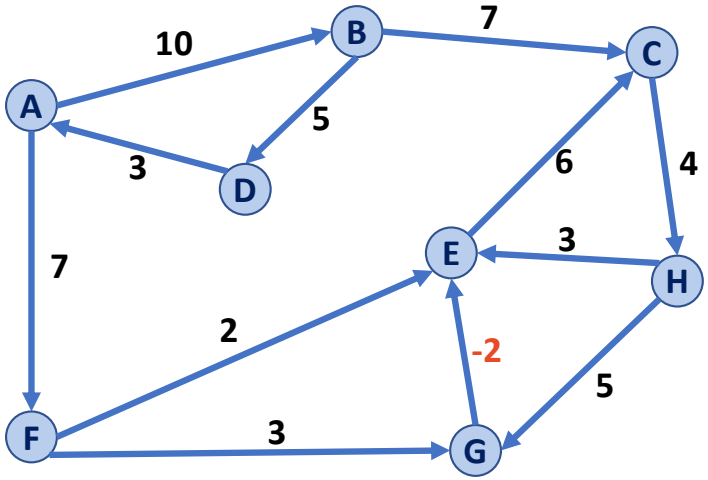
# Dijkstra's Algorithm (SSSP)

What about negative weight cycles?



# Dijkstra's Algorithm (SSSP)

What about negative weight edges, without negative weight cycles?



# Dijkstra's Algorithm (SSSP)

What is the running time?

```
    DijkstraSSSP(G, s):
 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 u = Q.removeMin()
17      T.add(u)
18      foreach (Vertex v : neighbors of u not in T):
19        if _________________ < d[v]:
20          d[v] = ____________________
21          p[v] = m
```