CS 225

Data Structures

December 3 – Prim's Algorihtm

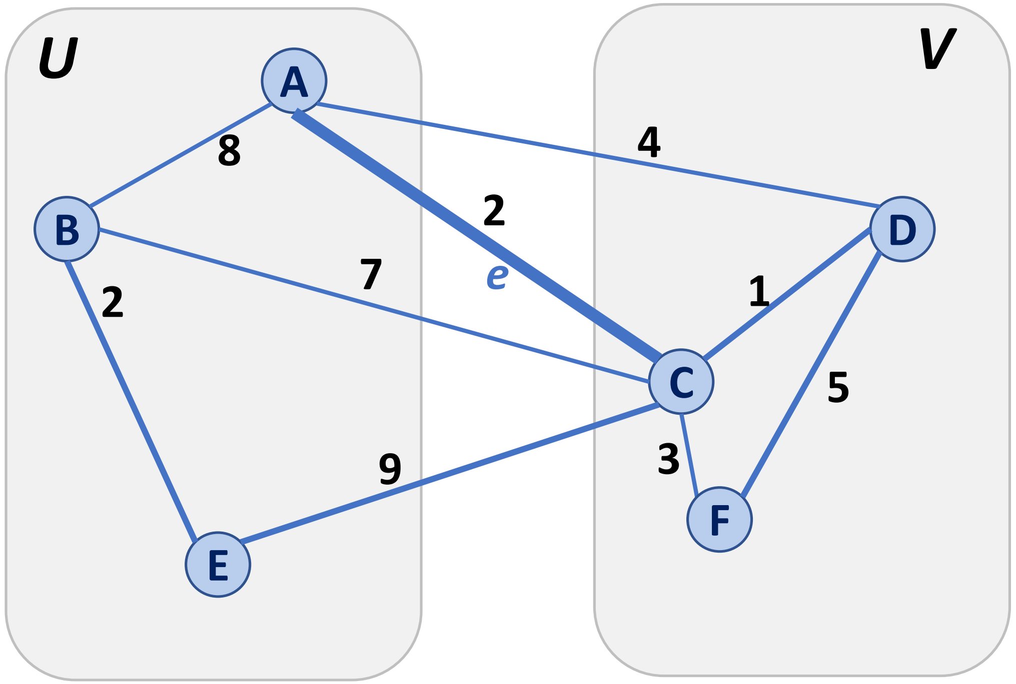
Wade Fagen-Ulmschneider

Partition Property

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

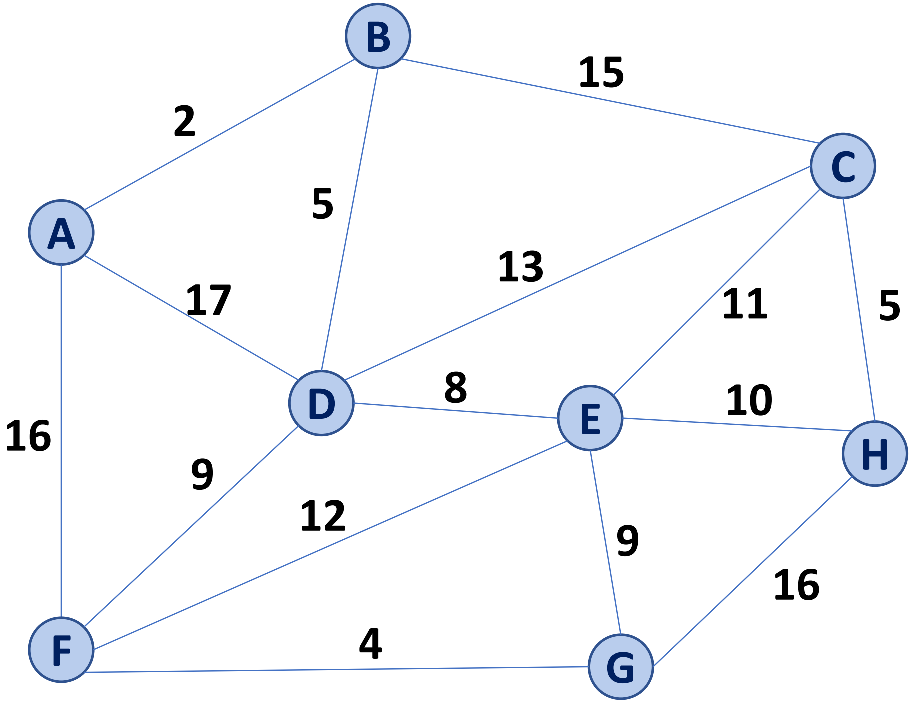
Let **e** be an edge of minimum weight across the partition.

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



Partition Property

The partition property suggests an algorithm:



Mattox Monday

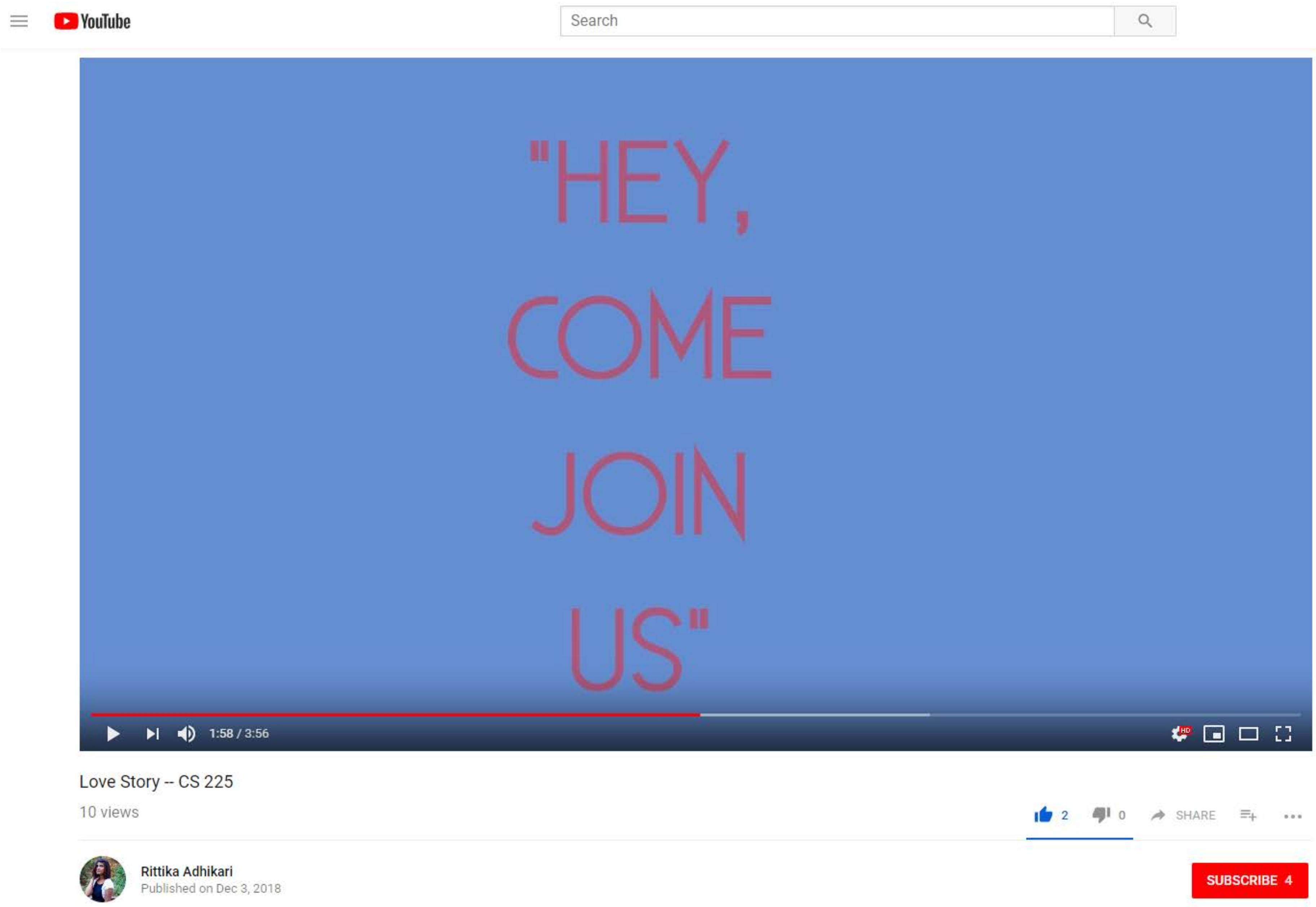
End of Semester Logistics

Lab: Your final CS 225 lab is this week.

Final Exam: Final exams start on Reading Day (Dec. 13)

- **Final is [One Theory Exam] + [One Programming Exam] together in a single exam.**
- **Time: 3 hours**

Grades: There will be a "December" grade update posted today with all grades up until now.



https://www.youtube.com/watch?v=7Ug1fr_ID_s

CAs

CS 225 | Course Info ▾ | Calendar | Lectures | Labs ▾ | MPs ▾ | Exams ▾ | Resources ▾ | Honors Section ▾

Staff

Instructors

Wade Fagen-Ulmschneider

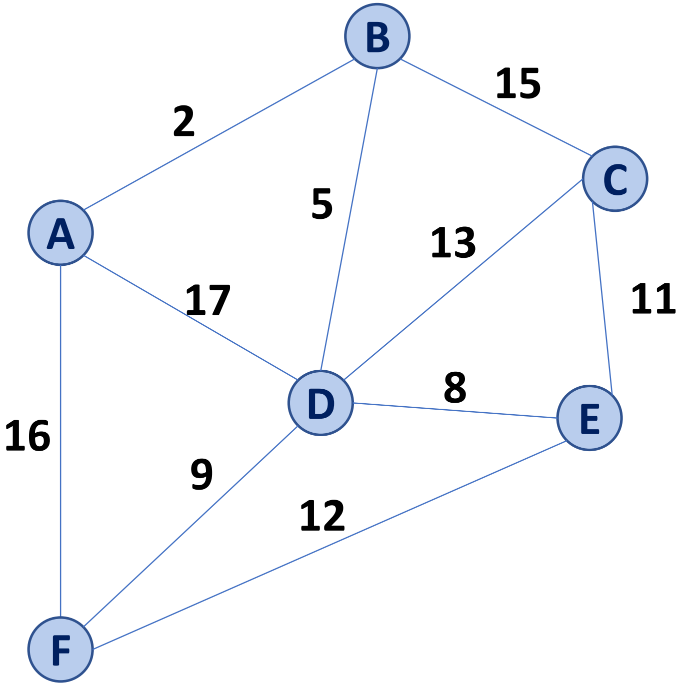
Office
2215 Siebel (2nd floor, south-west corner of Siebel)
Debug Your Brain
Every Wednesday, 4pm-5:30pm, 2036 ECEB

Mattox Beckman

Office
2212 Siebel (2nd floor, south-west corner of Siebel)
Debug Your Brain
Every Wednesday, 12:15pm - 1:45pm, 3036 ECEB

Thierry Ramais
Head of Course Logistics

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

```
  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
```


```
  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
```

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:

  $O(n \lg(n) + m \lg(n))$

Suppose I have a new heap:

	Binary Heap	Fibonacci Heap
Remove Min	O(lg(n))	O(lg(n))
Decrease Key	O(lg(n))	O(1)*

What's the updated running time?

```
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
```

Final Big-O MST Algorithm Runtimes:

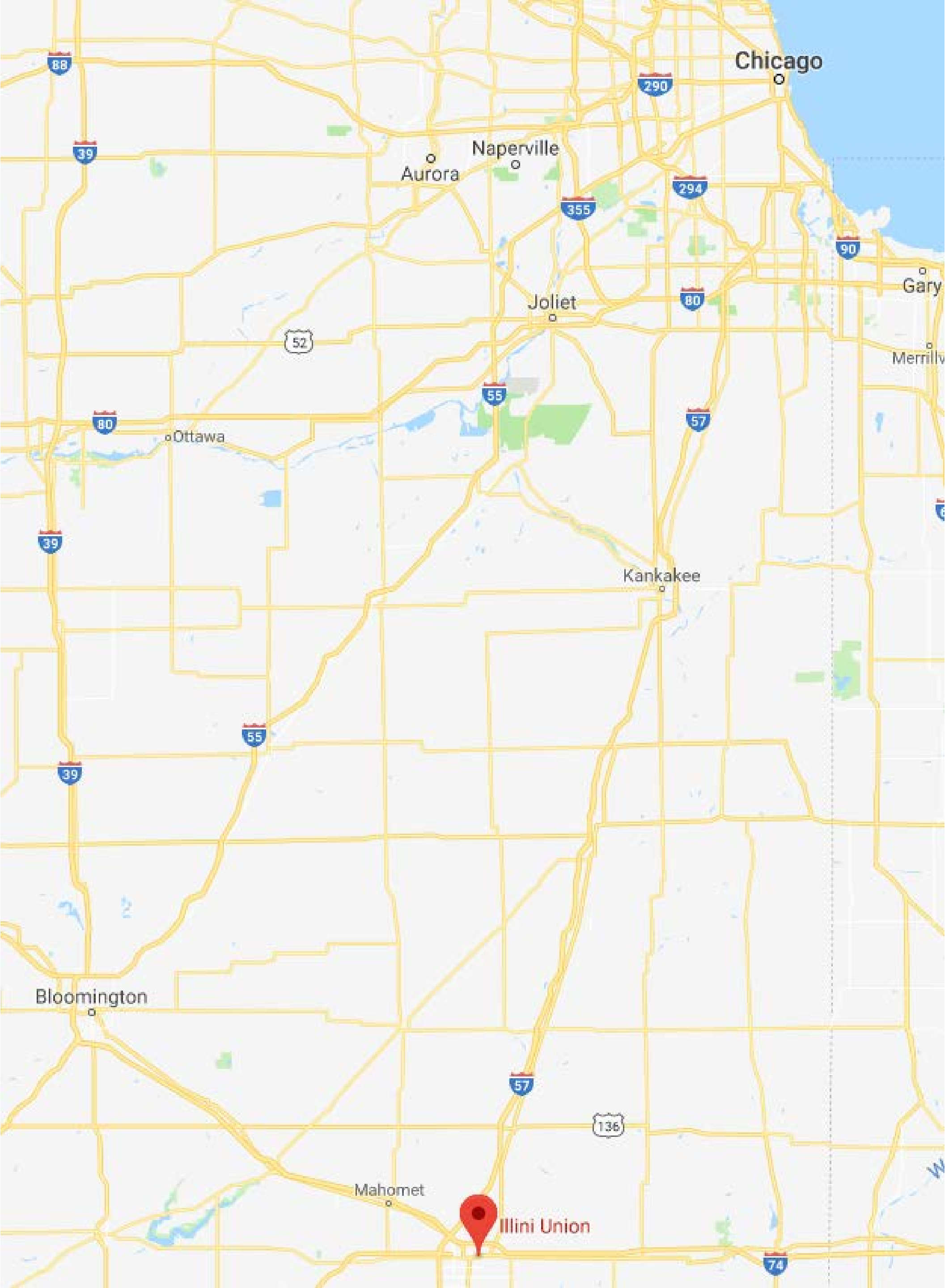
- Kruskal's Algorithm:

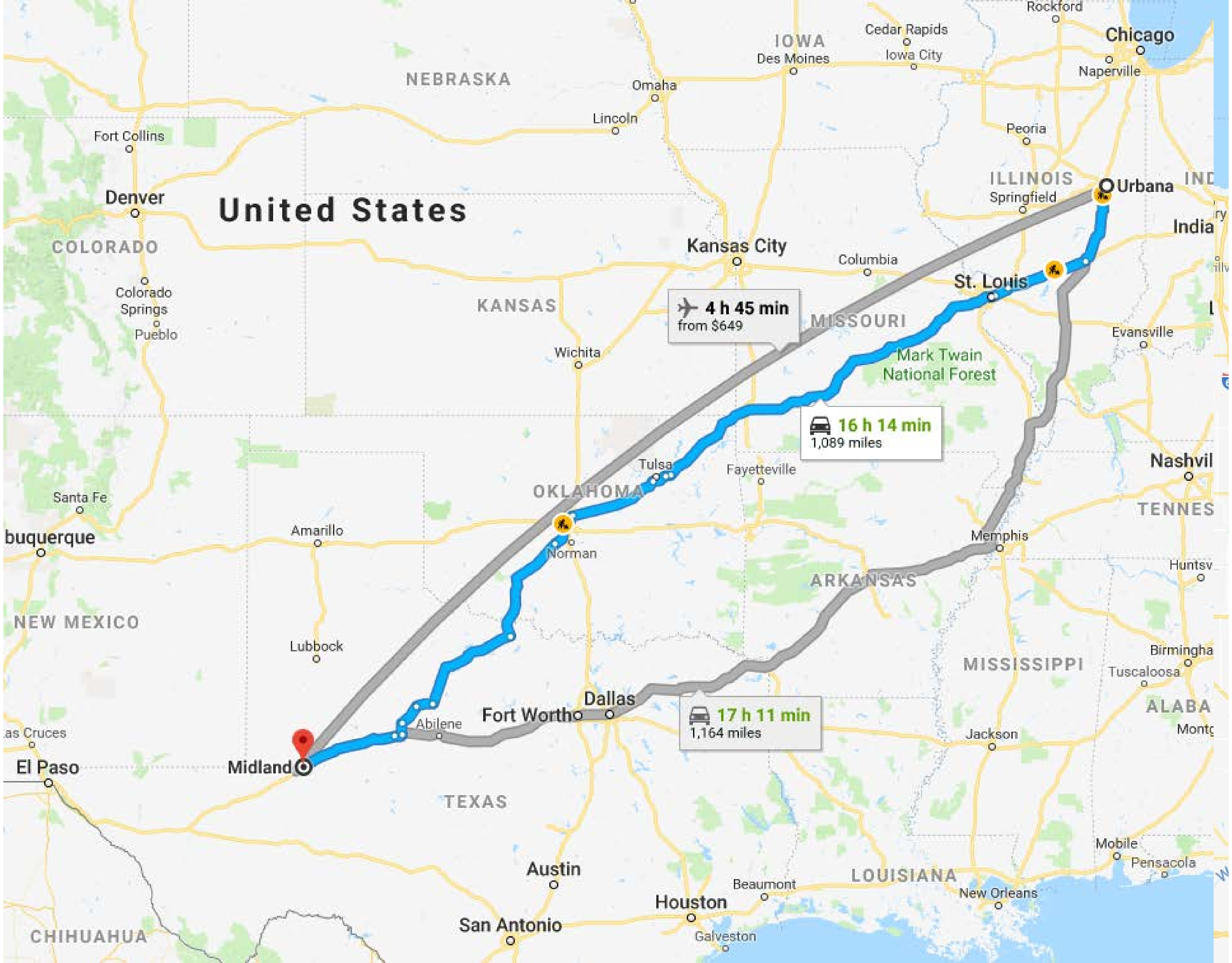
  O(m lg(n))

- Prim's Algorithm:

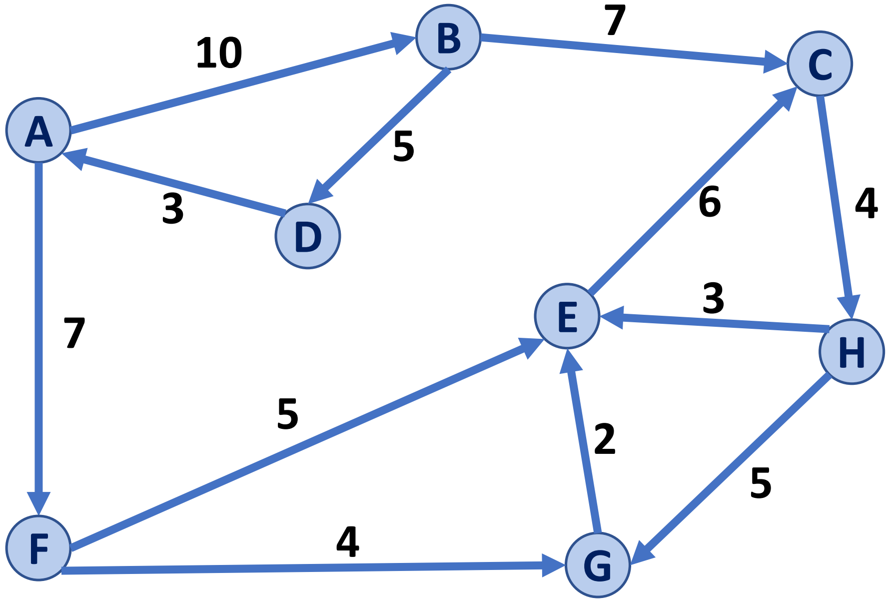
  O(n lg(n) + m)

Shortest Path




United States
4 h 45 min
from $649
16 h 14 min
1,089 miles
17 h 11 min
1,164 miles
Urbana
Midland
Chicago
Rockford
Cedar Rapids
IOWA
Des Moines
Iowa City
Naperville
NEBRASKA
Omaha
Lincoln
Fort Collins
Denver
COLORADO
Colorado Springs
Pueblo
KANSAS
Wichita
Kansas City
Columbia
ILLINOIS
Peoria
Springfield
St. Louis
MISSOURI
Mark Twain National Forest
Evansville
Tulsa
Fayetteville
OKLAHOMA
Norman
Nashvil
TENNES
Memphis
ARKANSAS
Huntsv
Santa Fe
buquerque
Amarillo
Lubbock
NEW MEXICO
MISSISSIPPI
Birmingha
Tuscaloosa
ALABA
Montg
Jackson
Dallas
Fort Worth
Abilene
as Cruces
El Paso
TEXAS
Austin
San Antonio
Houston
Galveston
Beaumont
LOUISIANA
New Orleans
Mobile
Pensacola
CHIHUAHUA
India

Dijkstra's Algorithm (SSSP)



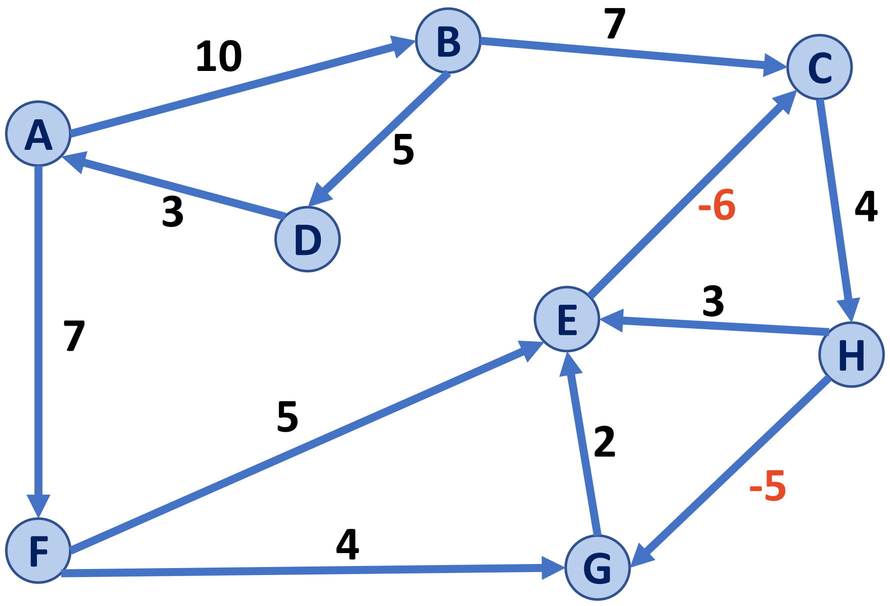

```
DijkstraSSSP(G, s):
  foreach (Vertex v : G):
    d[v] = +inf
    p[v] = NULL
  d[s] = 0

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

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

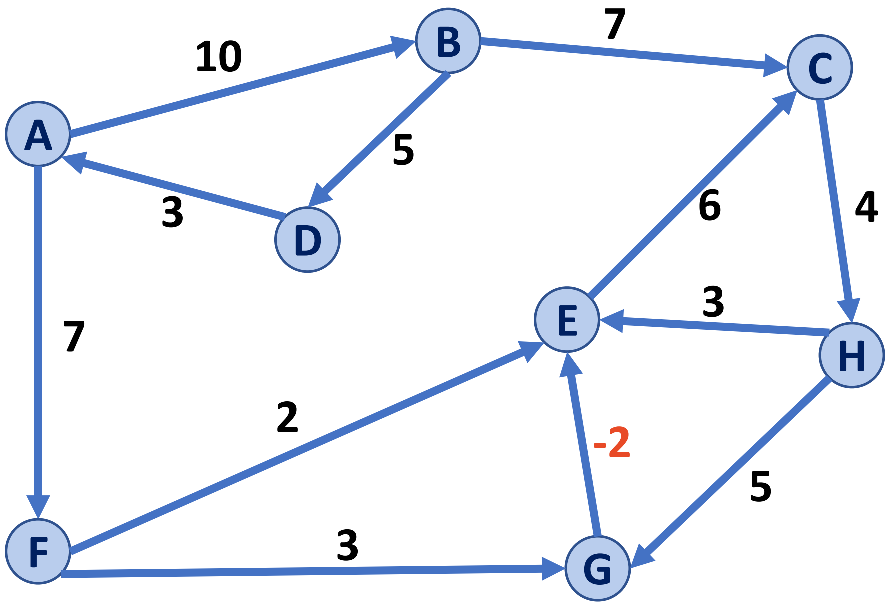
Dijkstra's Algorithm (SSSP)

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):
  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 u = Q.removeMin()
    T.add(u)
    foreach (Vertex v : neighbors of u not in T):
      if _______________ < d[v]:
        d[v] = __________________
        p[v] = m
```