CS 225

Data Structures

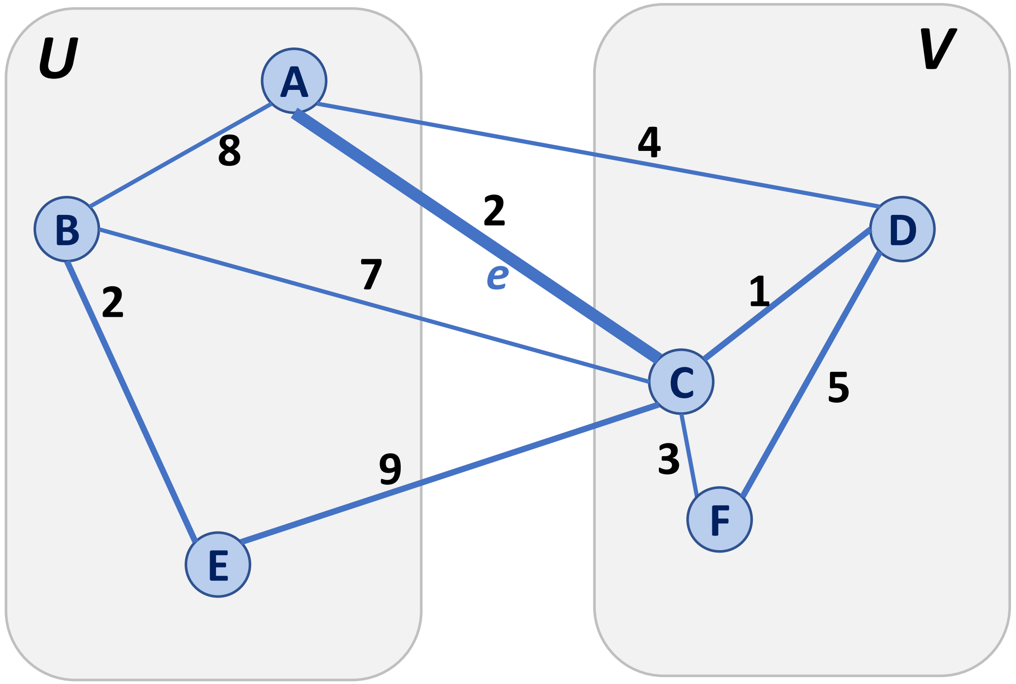
April 23 – Dijkstra's Algorithm

Wade Fagen-Ulmschneider

Partition Property

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

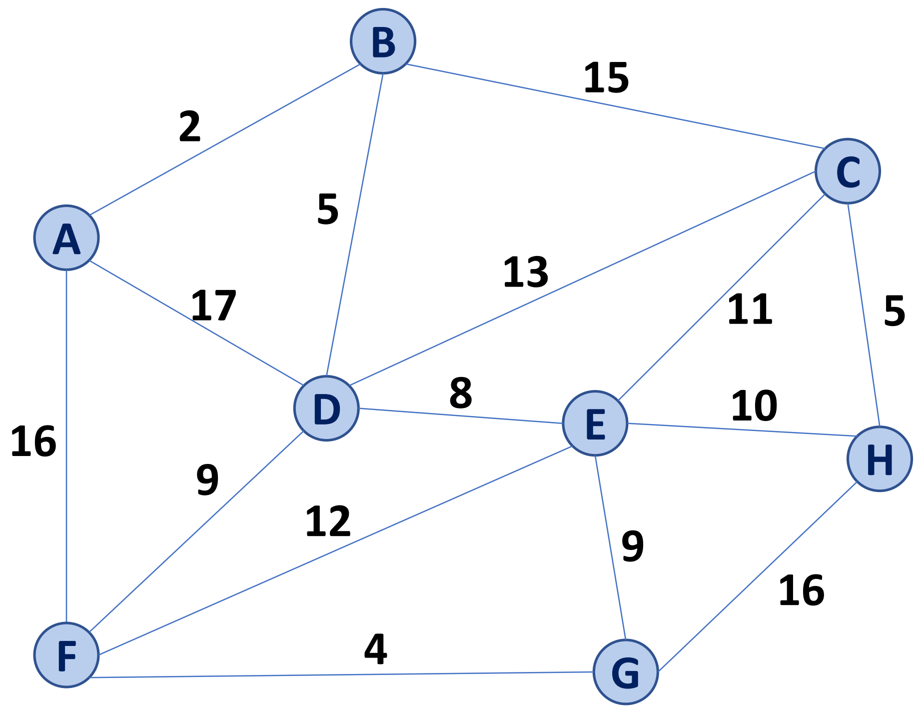
Let **e** be an edge of minimum weight across the partition.



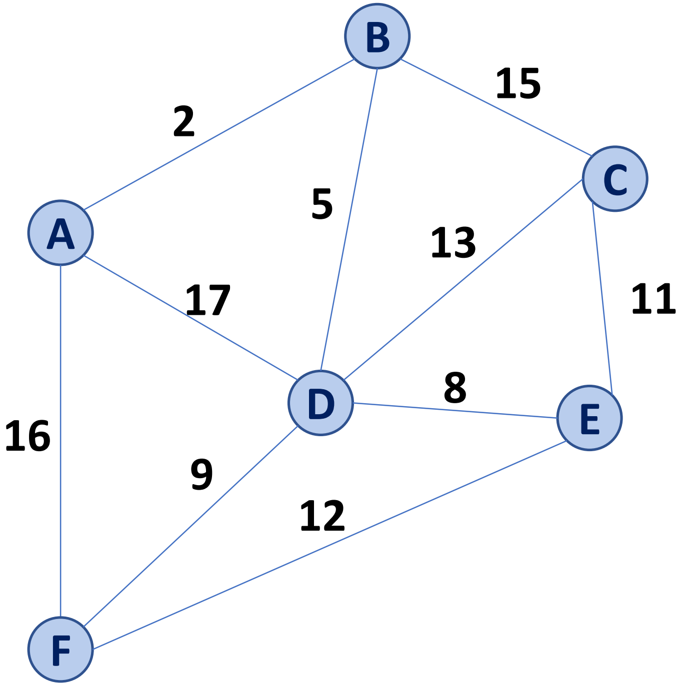
Then **e** is part of some minimum spanning tree.

Partition Property

The partition property suggests an algorithm:



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))

MST Algorithm Runtime:

- Upper bound on MST Algorithm Runtime:
  O(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
```

End of Semester Logistics

Lab: Your final CS 225 lab is this week.

- **No lab sections next week (partial week).**

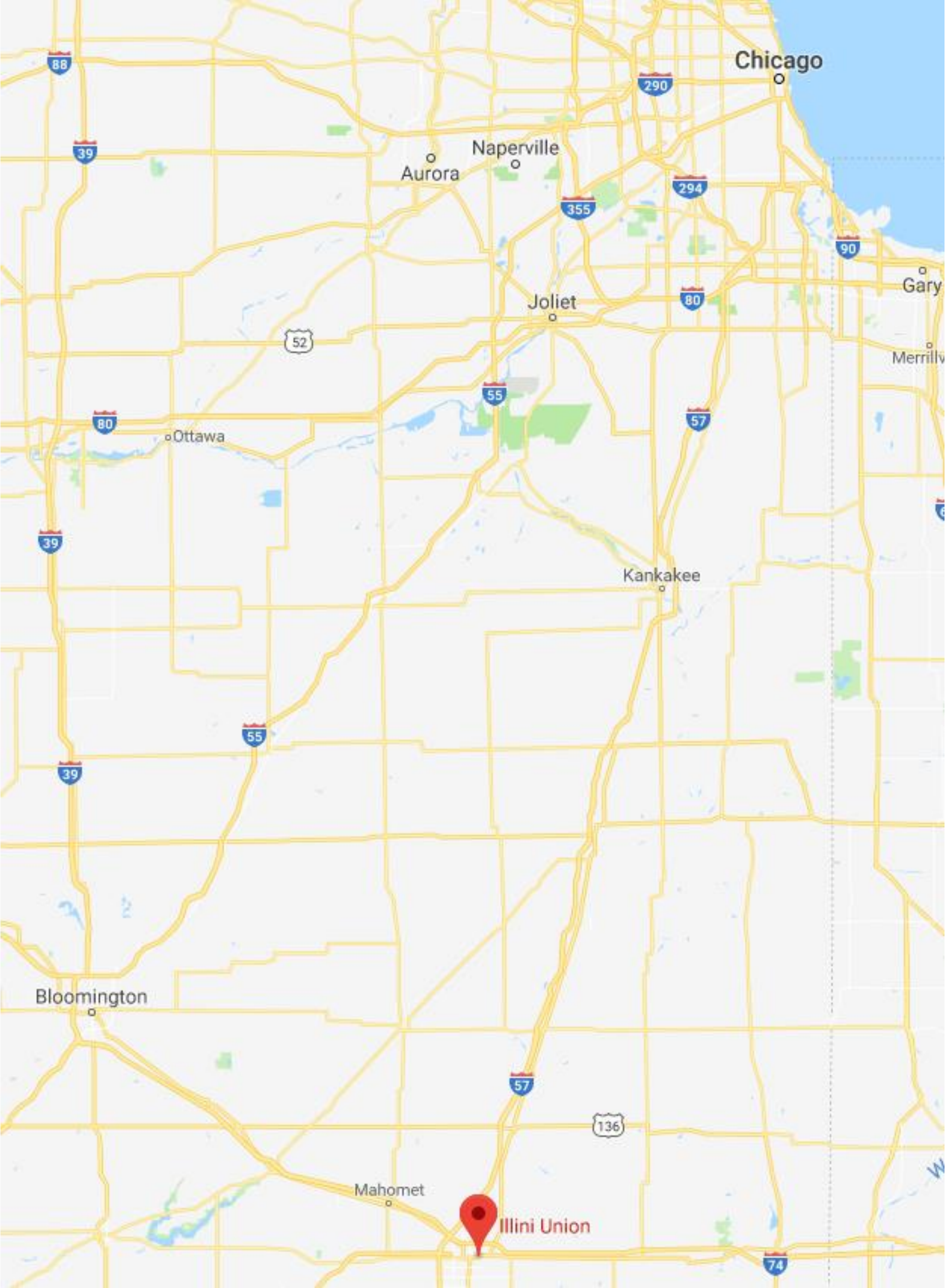
Final Exam: Final exams start on Reading Day (May 3)

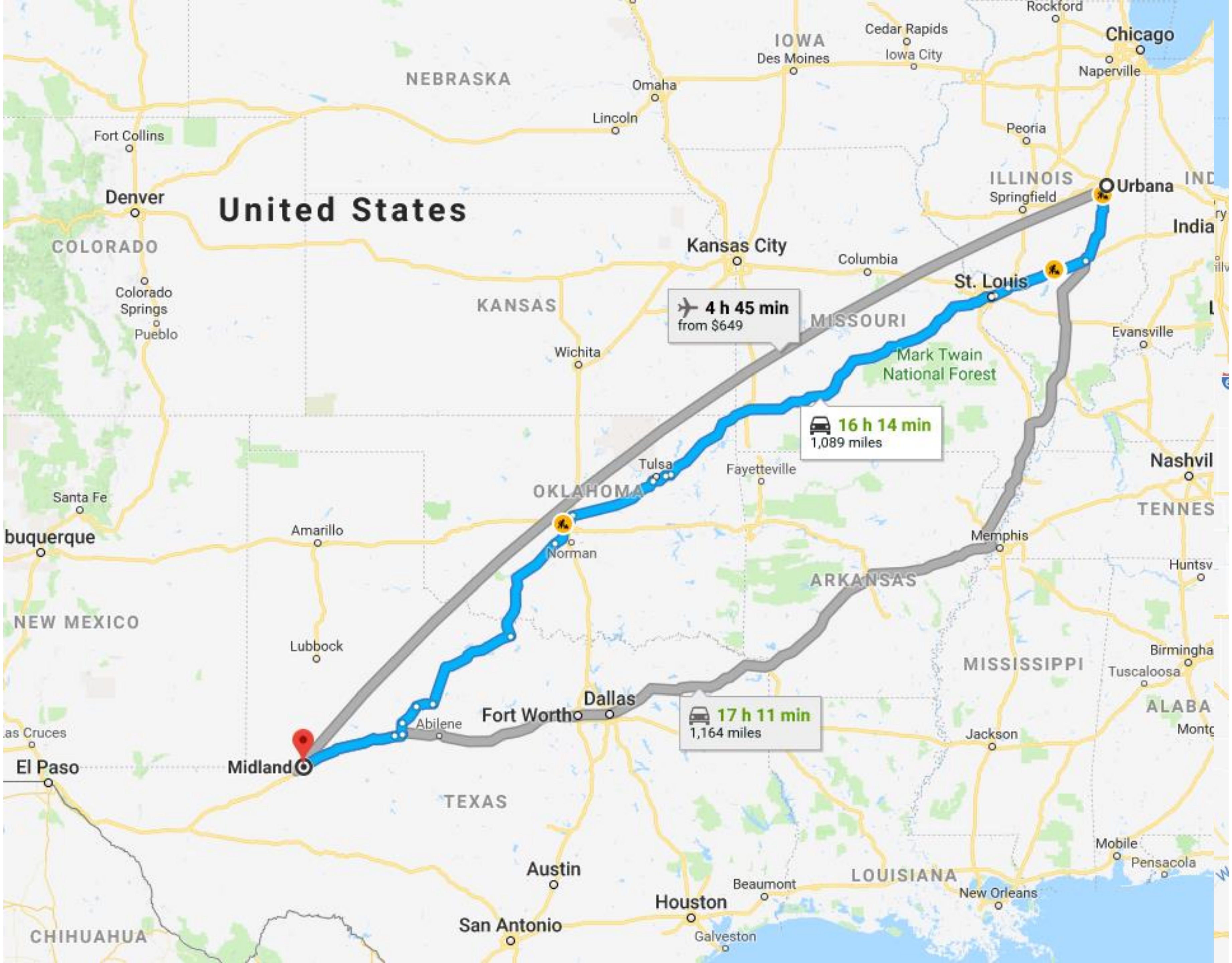
- **Last day of office hours is Wednesday, May 2.**
- **No office/lab hours once the first final exam is given.**

Grades: There will be a "Pre-Final" grade update posted next week with all grades except your final.

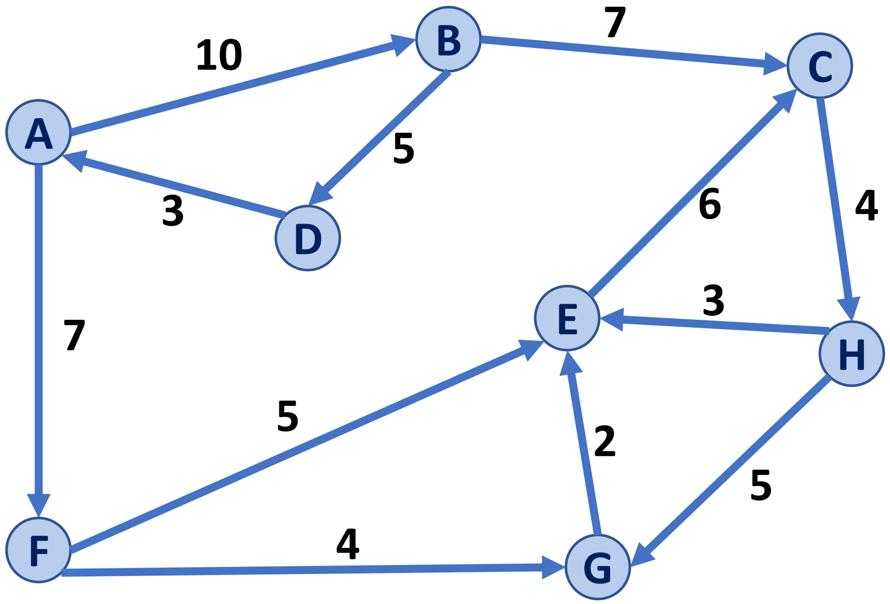
- **MP7's grace period extends until Tuesday, May 1**
- **Goal: Have "Pre-Final" grade on Wednesday/Thursday**

Shortest Path




4 h 45 min
from $649
16 h 14 min
1,089 miles
17 h 11 min
1,164 miles
United States
Urbana
Midland
Chicago
St. Louis
Kansas City
Tulsa
Norman
Abilene
Fort Worth
Dallas
Memphis
Denver
Amarillo
Lubbock
El Paso
Austin
San Antonio
Houston
ILLINOIS
MISSOURI
KANSAS
OKLAHOMA
TEXAS
ARKANSAS
NEBRASKA
IOWA
COLORADO
NEW MEXICO
MISSISSIPPI
LOUISIANA
Mark Twain National Forest

Dijkstra's Algorithm (SSSP)



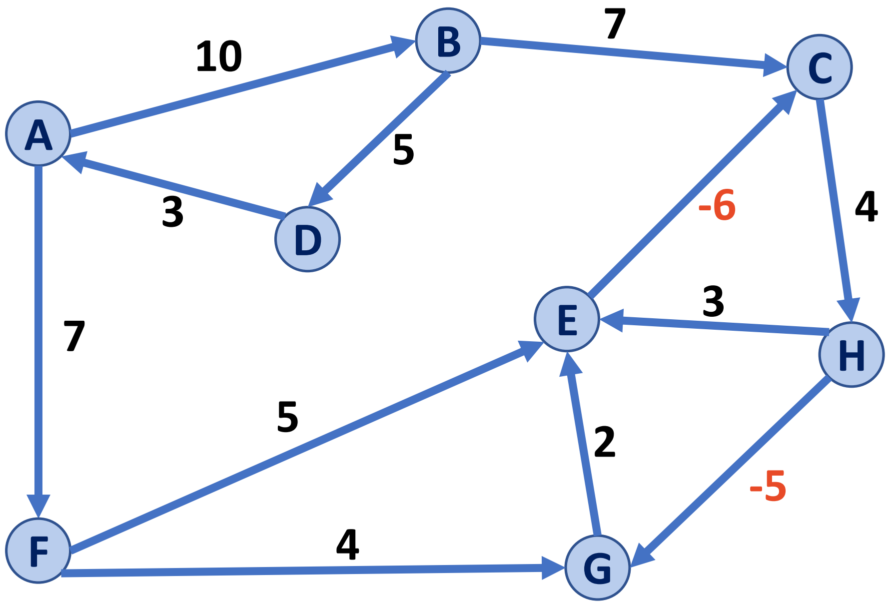

```
DijkstraSSSP(G, s):
  foreach (Vertex v : G):
    d[v] = +inf
    p[v] = NULL
  d[s] = 0

  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
```

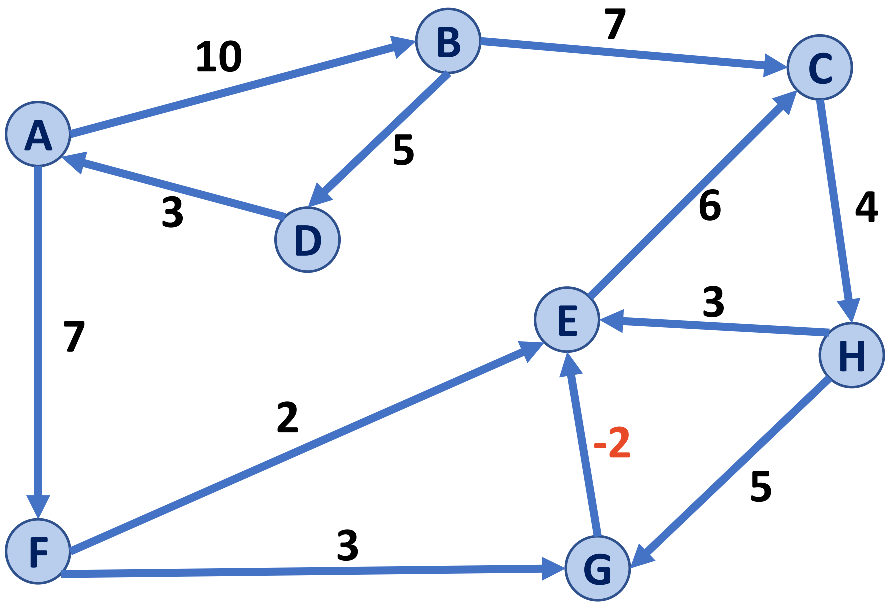
Dijkstra's Algorithm (SSSP)

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
```