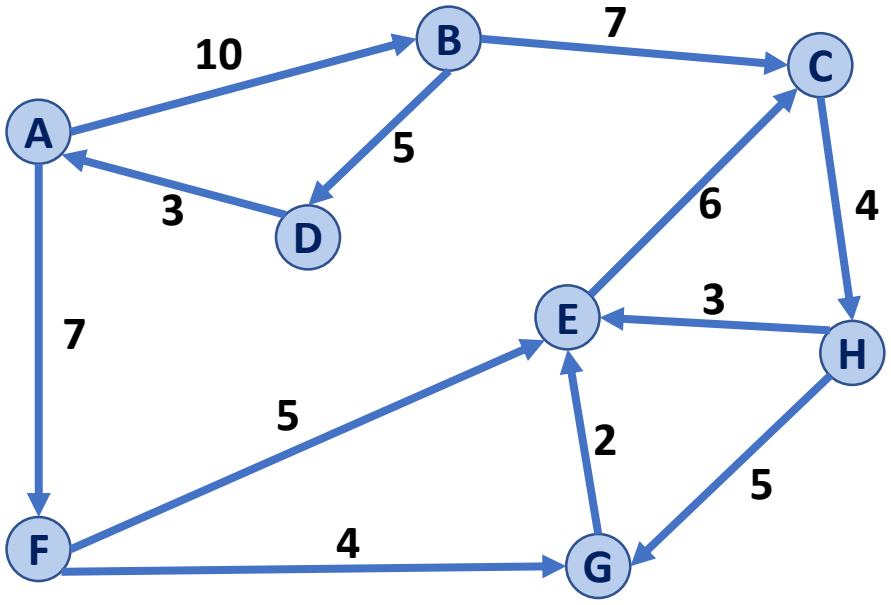
CS 225

Data Structures

April 27 – Graph Problems + End of Semester

Wade Fagen-Ulmschneider

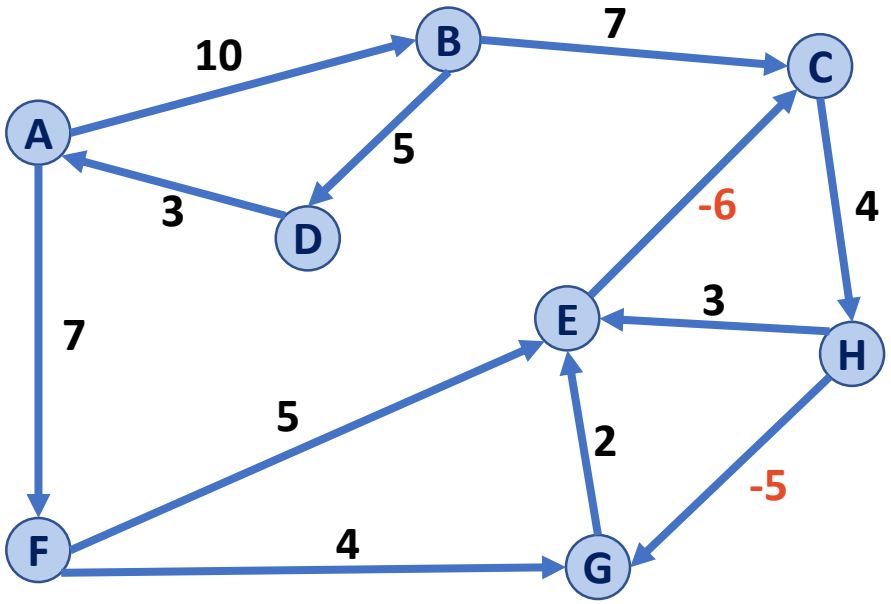
Dijkstra's Algorithm (SSSP)




```
    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 cost(u, v) + d[u] < d[v]:
20          d[v] = cost(u, v) + d[u]
21          p[v] = m
```

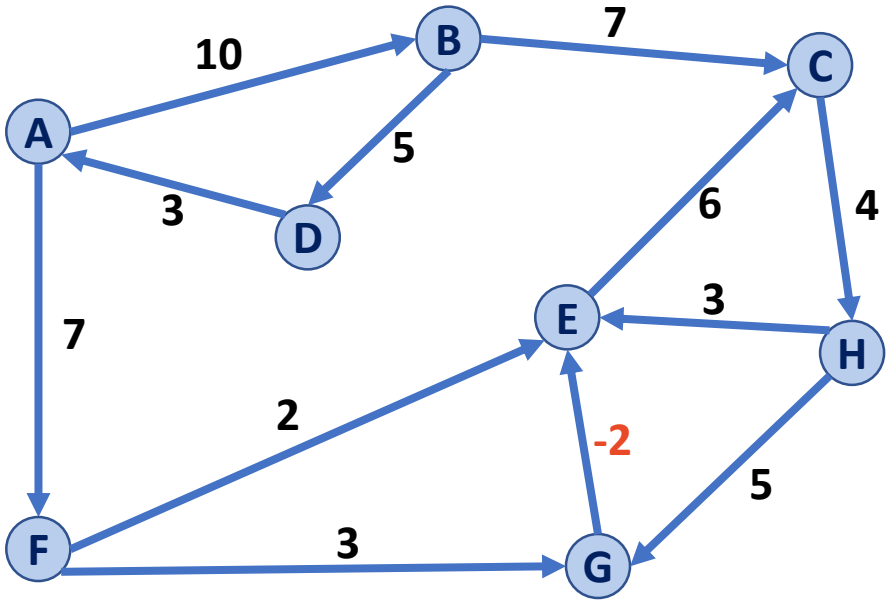
Dijkstra's Algorithm (SSSP)

Q: How does Dijkstra handle negative weight cycles?



Dijkstra's Algorithm (SSSP)

Q: How does Dijkstra handle negative weight edges, without a negative weight cycle?



Dijkstra's Algorithm (SSSP)

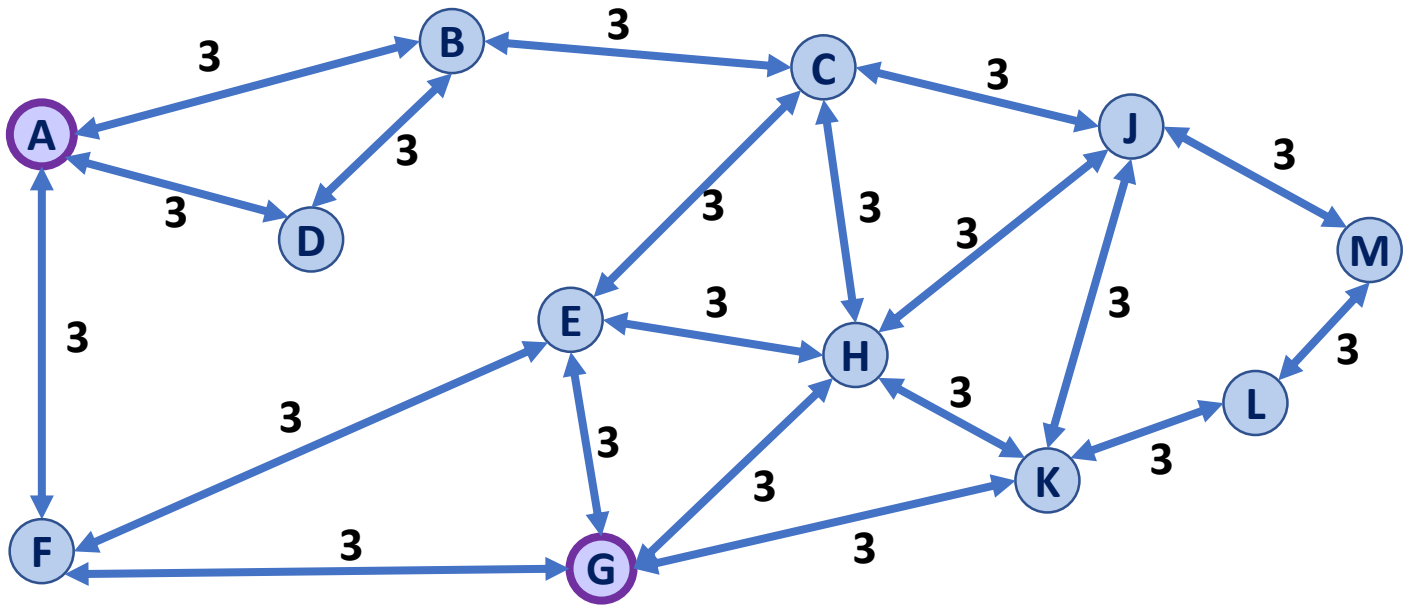
What is Dijkstra's running time?

```
    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 cost(u, v) + d[u] < d[v]:
20          d[v] = cost(u, v) + d[u]
21          p[v] = m
22
23    return T
```

Landmark Path Problem

Suppose you want to travel from **A** to **G**.

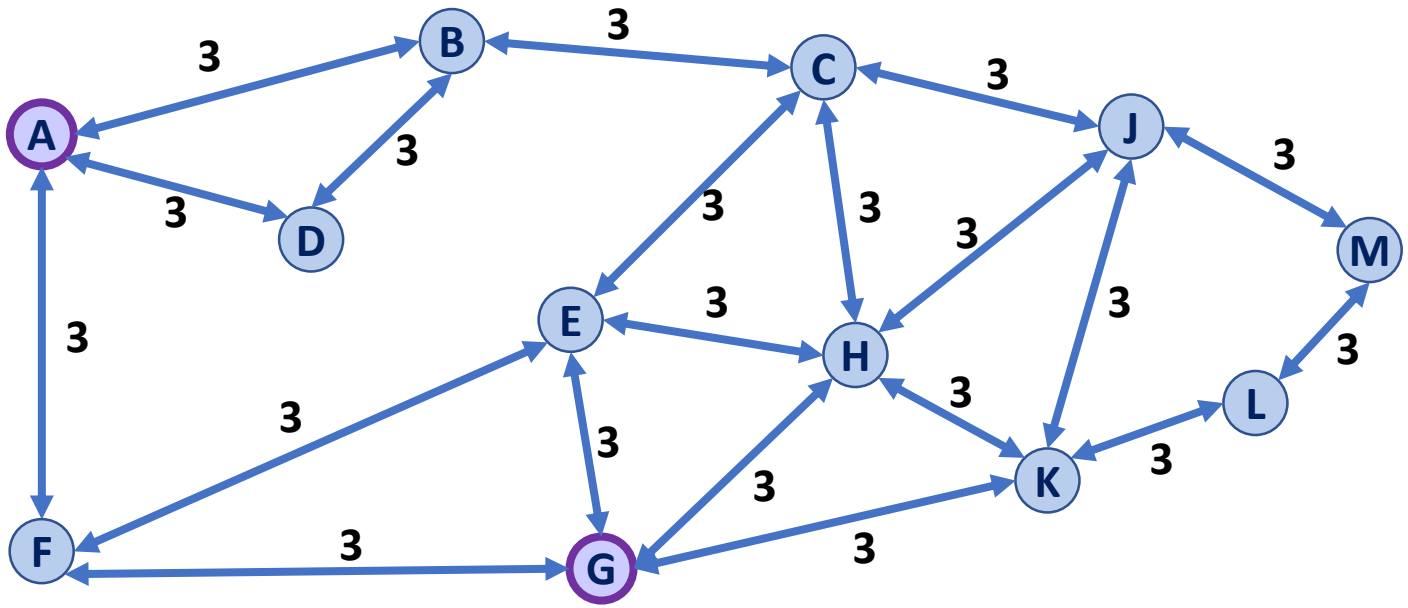
Q1: What is the shortest path from **A** to **G**?



Landmark Path Problem

Suppose you want to travel from **A** to **G**.

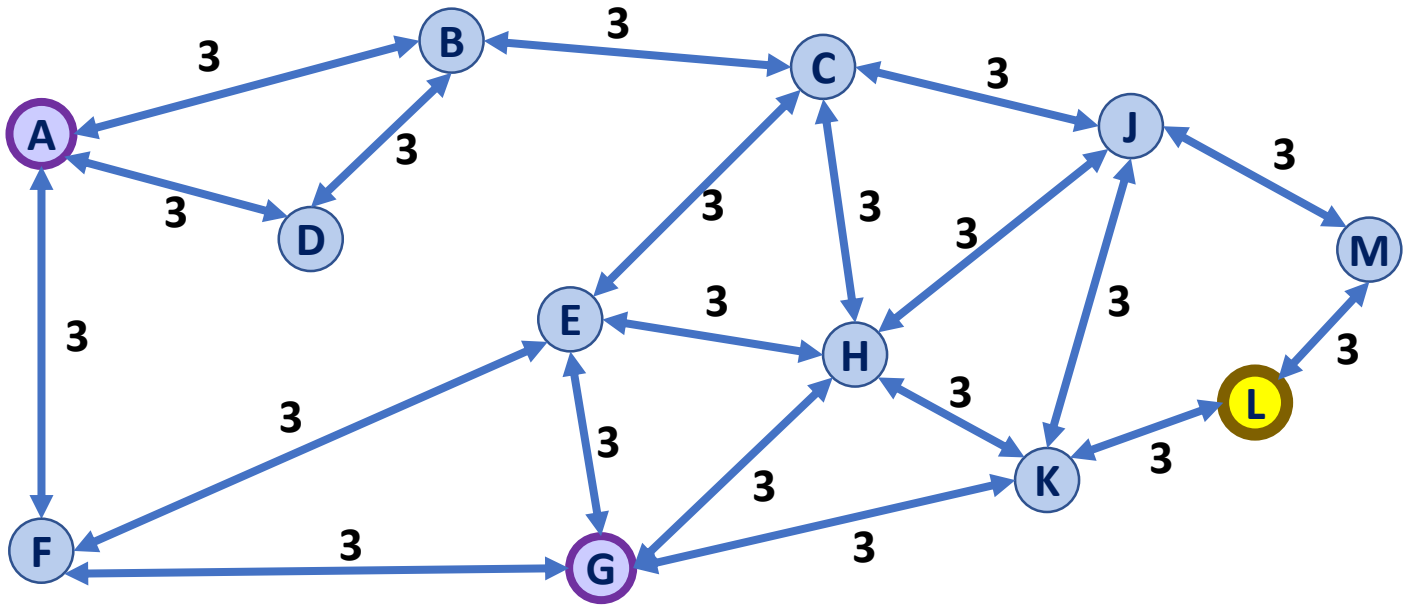
Q2: What is the fastest algorithm to use to find the shortest path?



Landmark Path Problem

In your journey between **A** and **G**, you also want to visit the landmark **L**.

Q3: What is the shortest path from **A** to **G** that visits **L**?

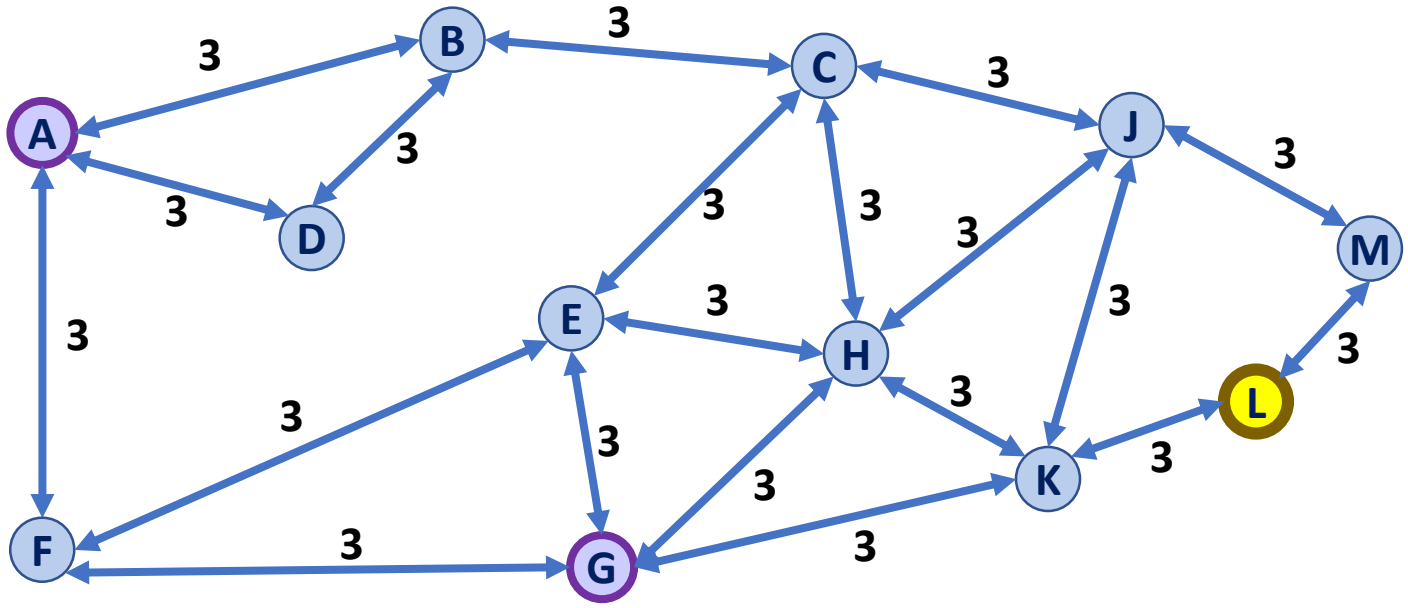


Landmark Path Problem

In your journey between **A** and **G**, you also want to visit the landmark **L**.

Q4: What is the fastest algorithm to find this path?

Q5: What are the specific call(s) to this algorithm?



End of Semester Logistics

CS 225 Final Exam

- The final exam begins on Thursday, May 3rd
- The final exam is a **3 hour CBTF exam**, is a **cumulative** exam, and has the format of a **combined theory + programming exam**
- The last office hours is Wednesday, May 2nd
- *We'll use lecture on Wednesday, May 2nd as a final exam review!*

End of Semester Logistics

"Pre-Final" Grade Dump

- I believe there's only a few remaining issues left with grading; I'll be starting to wrap these up myself over the weekend:
  - +EC from creative components
  - Working on recovering repos that were force deleted
- As soon as possible after MP7's deadline, we'll provide a "Pre-Final" grade in Compass that incorporates everything except the final exam into your CS 225 grade.

End of Semester Logistics

End of Semester Grade Review

- Excel sheet will be provided once final grades are posted.
- Must submit an Excel sheet for this review.

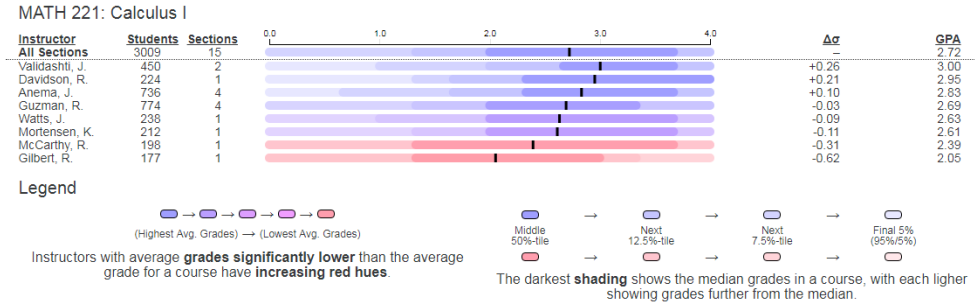
End of Semester Logistics

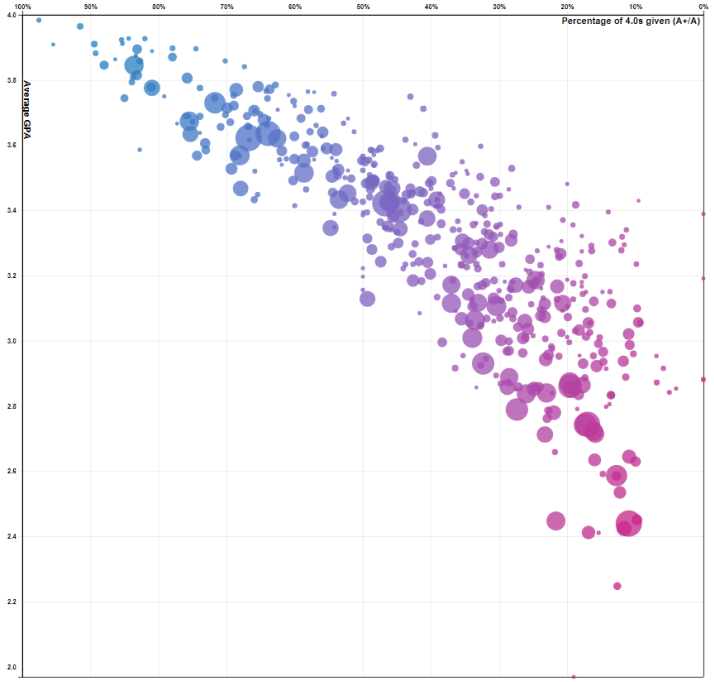
You’re Awesome -- +1 To Your Skills!

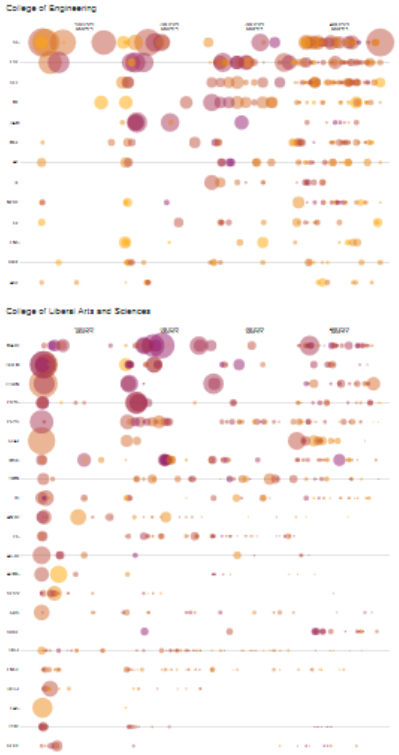
My Passion: Data Discovery

http://waf.cs.illinois.edu/discovery/

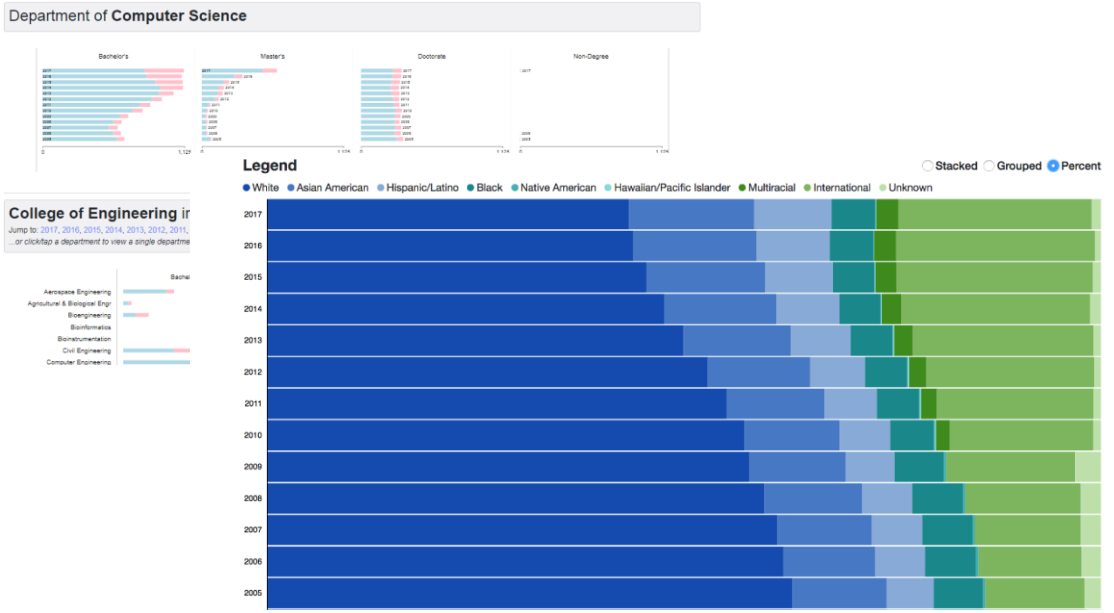
GPAs at Illinois:



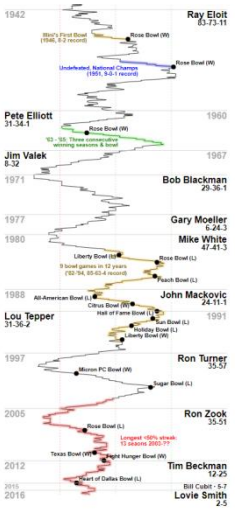




Diversity at Illinois:



And others:



Floyd-Warshall Algorithm

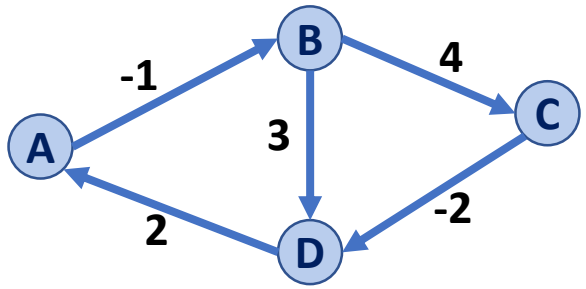
Floyd-Warshall's Algorithm is an alterative to Dijkstra in the presence of negative-weight edges (not negative weight cycles).

```
    FloydWarshall(G):
 6    Let d be a adj. matrix initialized to +inf
 7    foreach (Vertex v : G):
 8      d[v][v] = 0
 9    foreach (Edge (u, v) : G):
10      d[u][v] = cost(u, v)
11
12    foreach (Vertex u : G):
13      foreach (Vertex v : G):
14        foreach (Vertex w : G):
15          if d[u, v] > d[u, w] + d[w, v]:
16            d[u, v] = d[u, w] + d[w, v]
```

Floyd-Warshall Algorithm

```
    FloydWarshall(G):
 6    Let d be a adj. matrix initialized to +inf
 7    foreach (Vertex v : G):
 8      d[v][v] = 0
 9    foreach (Edge (u, v) : G):
10      d[u][v] = cost(u, v)
11
12    foreach (Vertex u : G):
13      foreach (Vertex v : G):
14        foreach (Vertex w : G):
15          if d[u, v] > d[u, w] + d[w, v]:
16            d[u, v] = d[u, w] + d[w, v]
```

	A	B	C	D
A				
B				
C				
D				

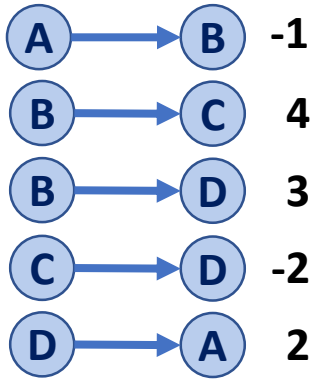


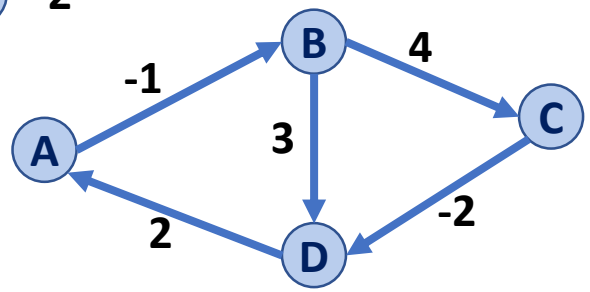
Floyd-Warshall Algorithm

```
12    foreach (Vertex u : G):
13      foreach (Vertex v : G):
14        foreach (Vertex w : G):
15          if d[u, v] > d[u, w] + d[w, v]:
16            d[u, v] = d[u, w] + d[w, v]
```

	A	B	C	D
A	0	-1		
B		0	4	3
C			0	-2
D	2			0

Initially:



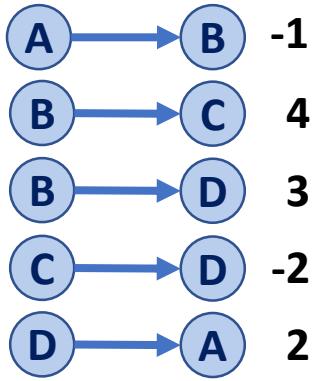


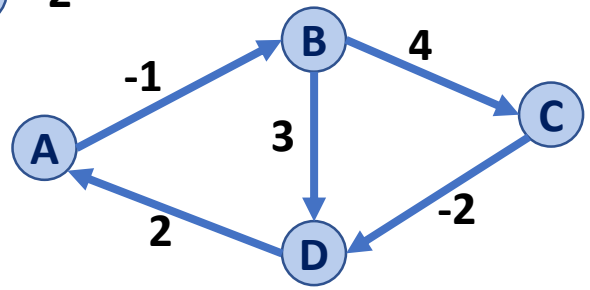
Floyd-Warshall Algorithm

```
12    foreach (Vertex u : G):
13      foreach (Vertex v : G):
14        foreach (Vertex w : G):
15          if d[u, v] > d[u, w] + d[w, v]:
16            d[u, v] = d[u, w] + d[w, v]
```

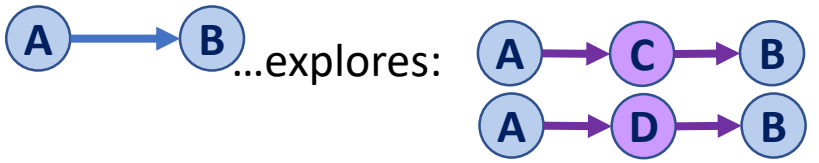
	A	B	C	D
A	0	-1		
B		0	4	3
C			0	-2
D	2			0

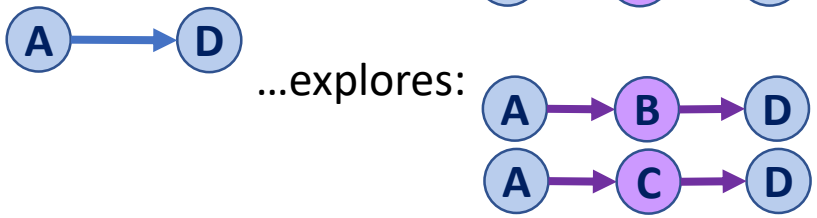
Initially:





Let u = A; v and w explores for better paths:



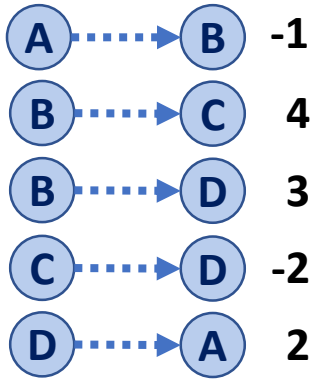


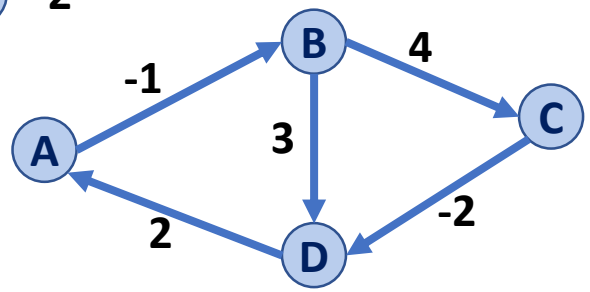
Floyd-Warshall Algorithm

```
12    foreach (Vertex u : G):
13      foreach (Vertex v : G):
14        foreach (Vertex w : G):
15          if d[u, v] > d[u, w] + d[w, v]:
16            d[u, v] = d[u, w] + d[w, v]
```

	A	B	C	D
A	0	-1	2	1
B		0	4	3
C			0	-2
D	2			0

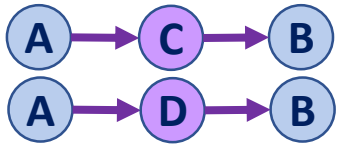
Initially:



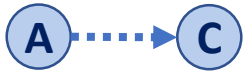


Let u = A; v and w explores for better paths:

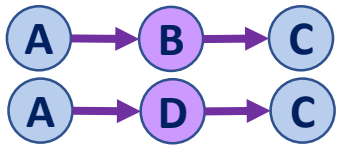
...explores:



+∞
+∞

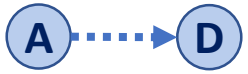


...explores:

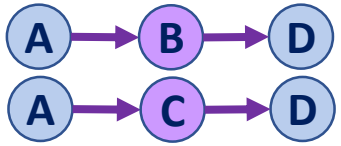


-1 + 3 = **2** UPDATE!

+∞



...explores:



-1 + 3 = **2** UPDATE!

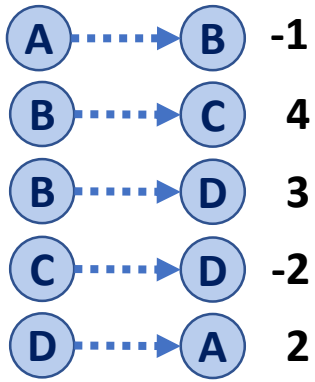
3 + (-2) = **1** UPDATE!

Floyd-Warshall Algorithm

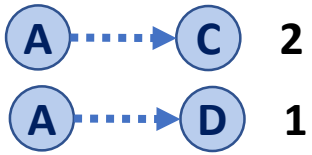
```
12    foreach (Vertex u : G):
13      foreach (Vertex v : G):
14        foreach (Vertex w : G):
15          if d[u, v] > d[u, w] + d[w, v]:
16            d[u, v] = d[u, w] + d[w, v]
```

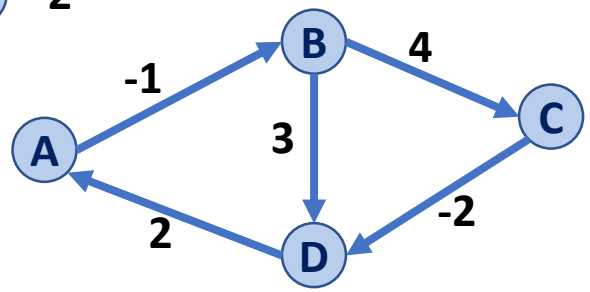
	A	B	C	D
A	0	-1	2	1
B		0	4	3
C			0	-2
D	2			0

Initially:



Let u = A; v and w explores for better paths:



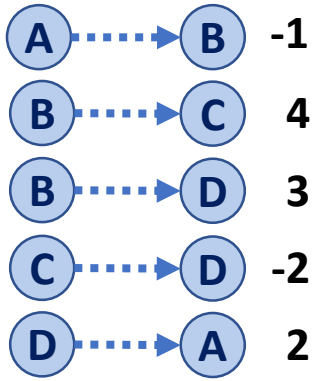


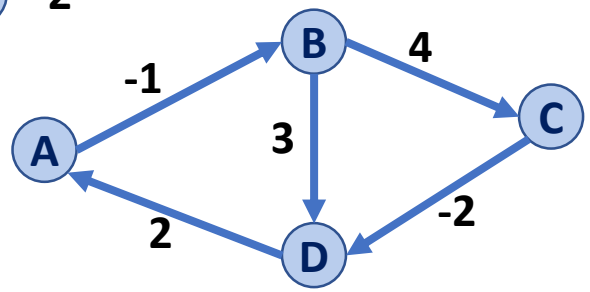
Floyd-Warshall Algorithm

```
12    foreach (Vertex u : G):
13      foreach (Vertex v : G):
14        foreach (Vertex w : G):
15          if d[u, v] > d[u, w] + d[w, v]:
16            d[u, v] = d[u, w] + d[w, v]
```

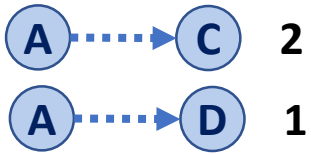
	A	B	C	D
A	0	-1	2	1
B	5	0	4	2
C			0	-2
D	2			0

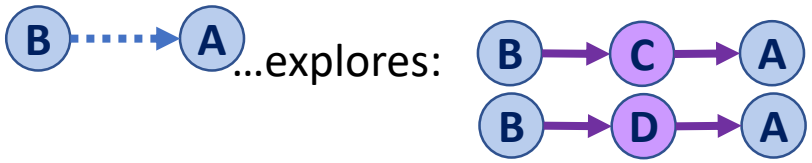
Initially:

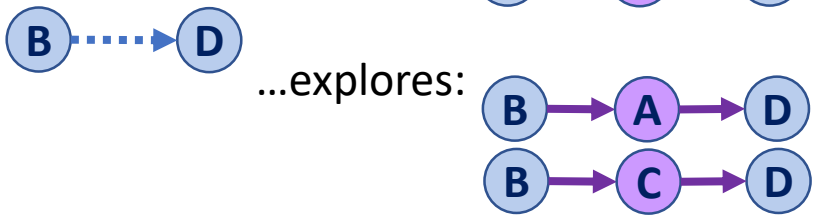




Let u = B; v and w explores for better paths:







Floyd-Warshall Algorithm

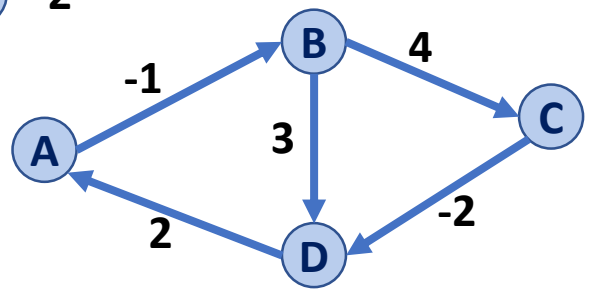
```
12    foreach (Vertex u : G):
13      foreach (Vertex v : G):
14        foreach (Vertex w : G):
15          if d[u, v] > d[u, w] + d[w, v]:
16            d[u, v] = d[u, w] + d[w, v]
```

	A	B	C	D
A	0	-1	2	1
B	5	0	4	2
C			0	-2
D	2			0

Initially:

A → B -1
B → C 4
B → D 3
C → D -2
D → A 2

A → C 2
A → D 1



Let u = B; v and w explores for better paths:

B → A …explores:
- B → C → A +∞
- B → D → A 3 + 2 = **5** **UPDATE!**

B → C …explores:
- B → A → C 5 + 2 = **7** **> 4, no update**
- B → D → C +∞

B → D …explores:
- B → A → D 5 + 1 = **6** **> 3, no update**
- B → C → D 4 + (-2) = **2** **UPDATE!**