

CSE 373

NOVEMBER 22ND – DIJKSTRAS ALGORITHM

ADMINISTRIVIA

- **P3P1 compiler problems fixed for most**
 - Check Generic typing
 - Eclipse is smarter than we are
 - Please only zip “src”
- **Thanksgiving!**
 - Please take the day off
 - Friday – 1 late day, Saturday – 2 late days
- **Next Tuesday**
 - Topo, Dijkstras, MST “section”
- **Style**
 - Exempt test files, 100% earn back

GRAPHS REVIEW

- **What is some of the terminology for graphs and what do those terms mean?**
 - Vertices and Edges
 - Directed v. Undirected
 - In-degree and out-degree
 - Connected
 - Weighted v. unweighted
 - Cyclic v. acyclic
 - DAG: Directed Acyclic Graph

TRAVERSALS

- **For an arbitrary graph and starting node v , find all nodes *reachable* from v .**
 - There exists a path from v
 - Doing something or “processing” each node
 - Determines if an undirected graph is connected?
If a traversal goes through all vertices, then it is connected
- **Basic idea**
 - Traverse through the nodes like a tree
 - Mark the nodes as visited to prevent cycles and from processing the same node twice

COMPARISON

Breadth-first always finds shortest length paths, i.e., “optimal solutions”

- Better for “what is the shortest path from x to y ”

But depth-first can use less space in finding a path

- If *longest path* in the graph is p and highest out-degree is d then DFS stack never has more than $d \cdot p$ elements
- But a queue for BFS may hold $O(|V|)$ nodes

A third approach (useful in Artificial Intelligence)

- *Iterative deepening (IDFS)*:
 - Try DFS but disallow recursion more than κ levels deep
 - If that fails, increment κ and start the entire search over
- Like BFS, finds shortest paths. Like DFS, less space.

SINGLE SOURCE SHORTEST PATHS

Done: BFS to find the minimum path length from v to u in $O(|E|+|V|)$

Actually, can find the minimum path length from v to *every node*

- Still $O(|E|+|V|)$
- No faster way for a “distinguished” destination in the worst-case

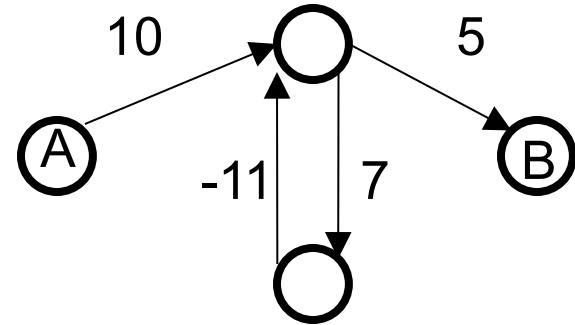
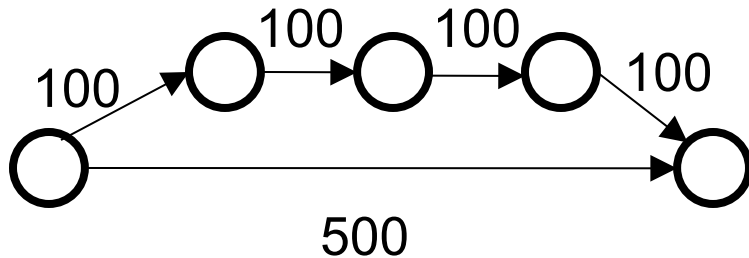
Now: **Weighted graphs**

Given a weighted graph and node v ,
find the minimum-cost path from v to every node

As before, asymptotically no harder than for one destination

Unlike before, BFS will not work -> only looks at path length.

NOT AS EASY



Why BFS won't work: Shortest path may not have the fewest edges

- Annoying when this happens with costs of flights

We will assume there are no negative weights

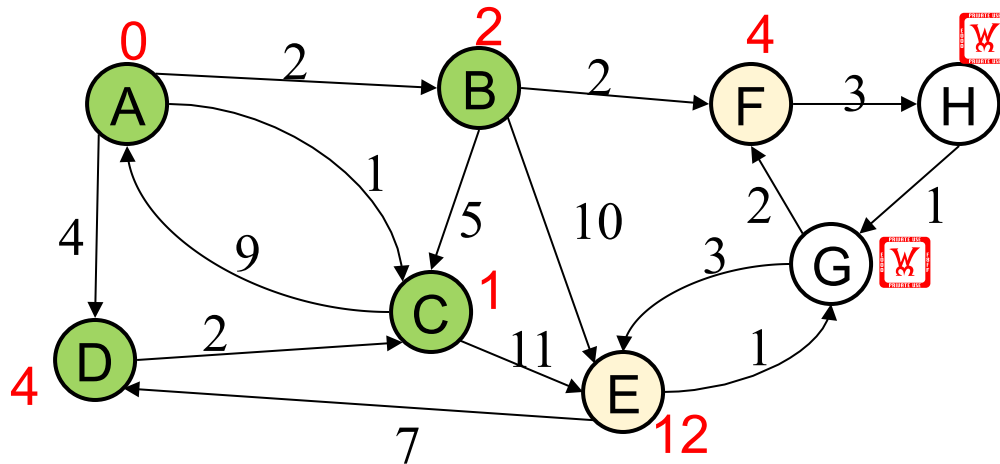
- *Problem is ill-defined* if there are negative-cost cycles
- Today's *algorithm is wrong* if edges can be negative
 - There are other, slower (but not terrible) algorithms

DIJKSTRA'S ALGORITHM

The idea: reminiscent of BFS, but adapted to handle weights

- Grow the set of nodes whose shortest distance has been computed
- Nodes not in the set will have a “best distance so far”
- A priority queue will turn out to be useful for efficiency

DIJKSTRA'S ALGORITHM



Initially, start node has cost 0 and all other nodes have cost ∞

At each step:

- Pick closest unknown vertex v
- Add it to the “cloud” of known vertices
- Update distances for nodes with edges from v

That's it! (But we need to prove it produces correct answers)

THE ALGORITHM

1. For each node v , set $v.cost = \infty$ **and** $v.known = false$
2. Set $source.cost = 0$
3. While there are unknown nodes in the graph
 - a) Select the unknown node v with lowest cost
 - b) Mark v as known
 - c) For each edge (v, u) with weight w ,

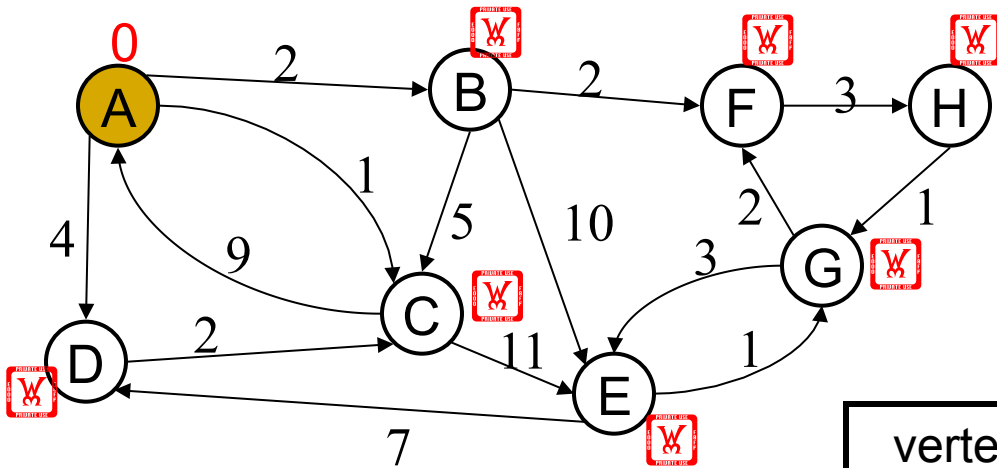
```
c1 = v.cost + w // cost of best path through v to u
c2 = u.cost // cost of best path to u previously known
if (c1 < c2) { // if the path through v is better
    u.cost = c1
    u.path = v // for computing actual paths
}
```

IMPORTANT FEATURES

When a vertex is marked known, the cost of the shortest path to that node is known

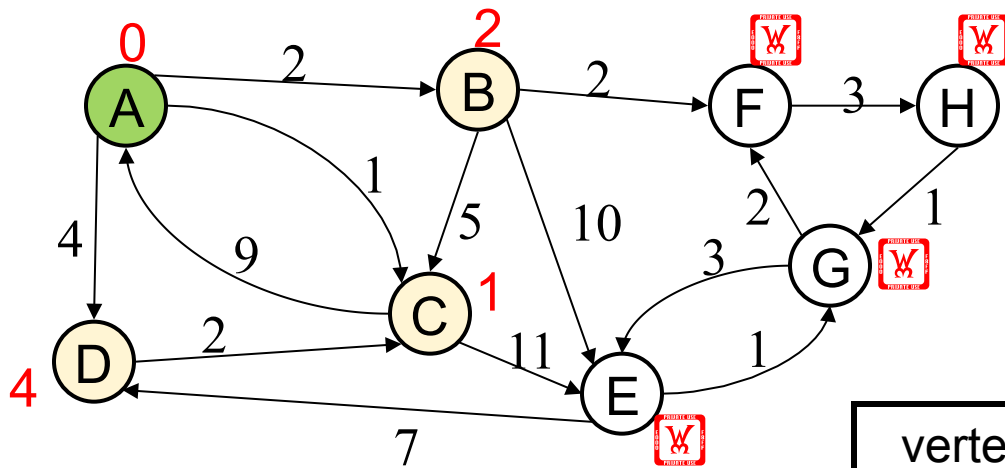
- The path is also known by following back-pointers

While a vertex is still not known, another shorter path to it *might* still be found



vertex	known?	cost	path
A		0	
B		??	
C		??	
D		??	
E		??	
F		??	
G		??	
H		??	

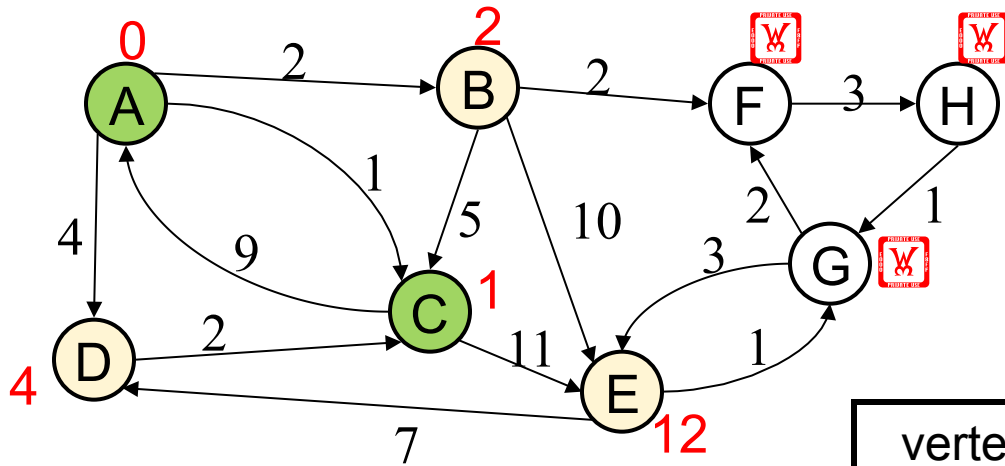
Order Added to Known Set:



vertex	known?	cost	path
A	Y	0	
B		≤ 2	A
C		≤ 1	A
D		≤ 4	A
E		??	
F		??	
G		??	
H		??	

Order Added to Known Set:

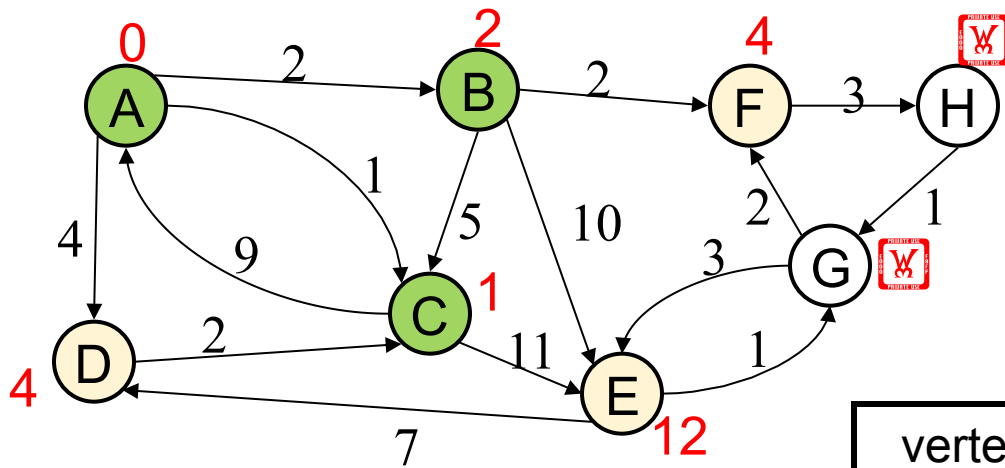
A



vertex	known?	cost	path
A	Y	0	
B		≤ 2	A
C	Y	1	A
D		≤ 4	A
E		≤ 12	C
F		??	
G		??	
H		??	

Order Added to Known Set:

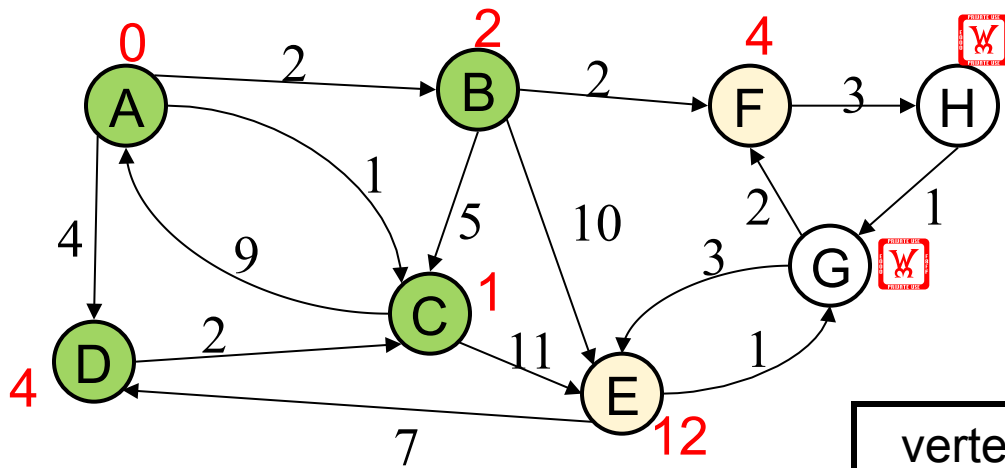
A, C



vertex	known?	cost	path
A	Y	0	
B	Y	2	A
C	Y	1	A
D		≤ 4	A
E		≤ 12	C
F		≤ 4	B
G		??	
H		??	

Order Added to Known Set:

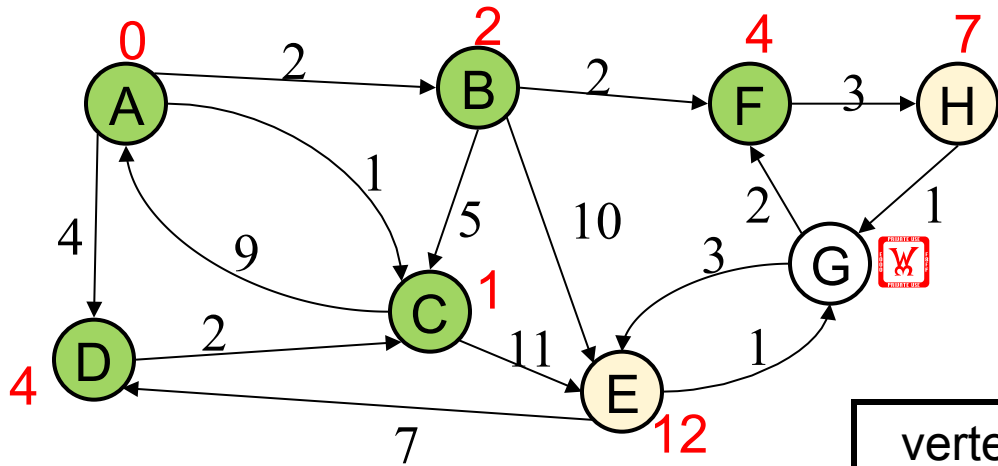
A, C, B



vertex	known?	cost	path
A	Y	0	
B	Y	2	A
C	Y	1	A
D	Y	4	A
E		≤ 12	C
F		≤ 4	B
G		??	
H		??	

Order Added to Known Set:

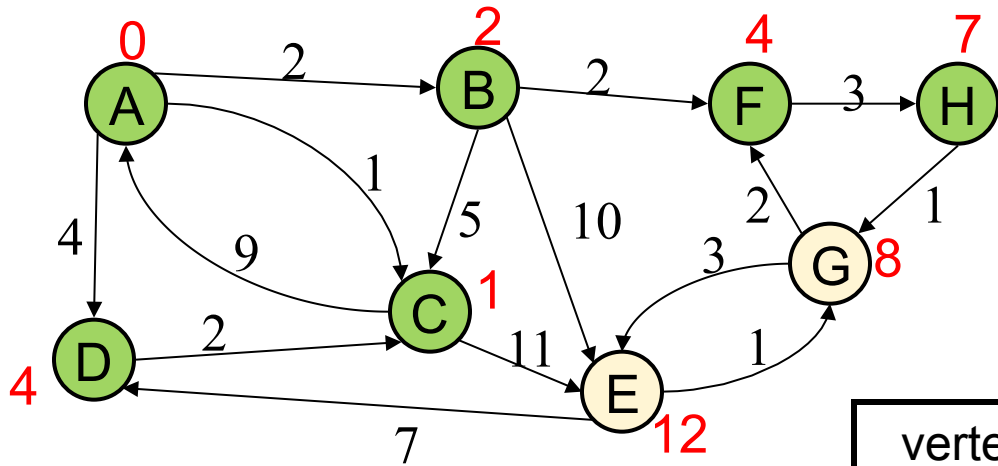
A, C, B, D



vertex	known?	cost	path
A	Y	0	
B	Y	2	A
C	Y	1	A
D	Y	4	A
E		≤ 12	C
F	Y	4	B
G		??	
H		≤ 7	F

Order Added to Known Set:

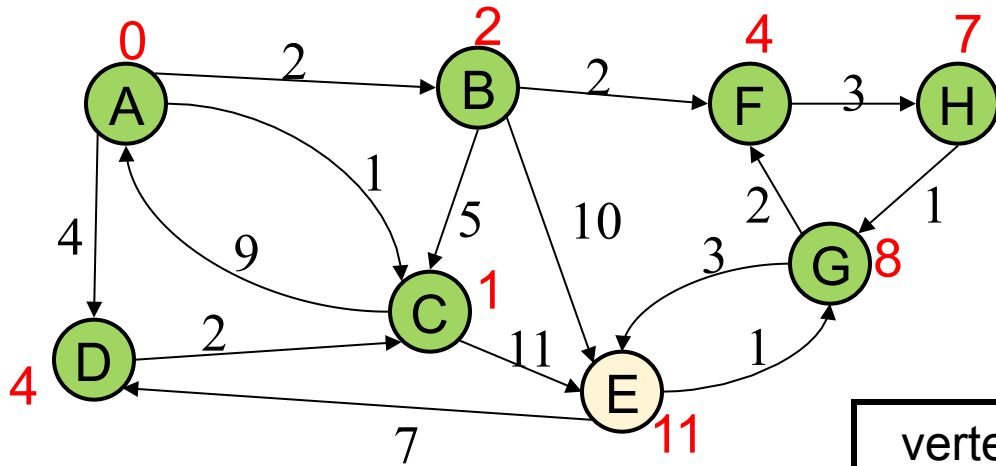
A, C, B, D, F



vertex	known?	cost	path
A	Y	0	
B	Y	2	A
C	Y	1	A
D	Y	4	A
E		≤ 12	C
F	Y	4	B
G		≤ 8	H
H	Y	7	F

Order Added to Known Set:

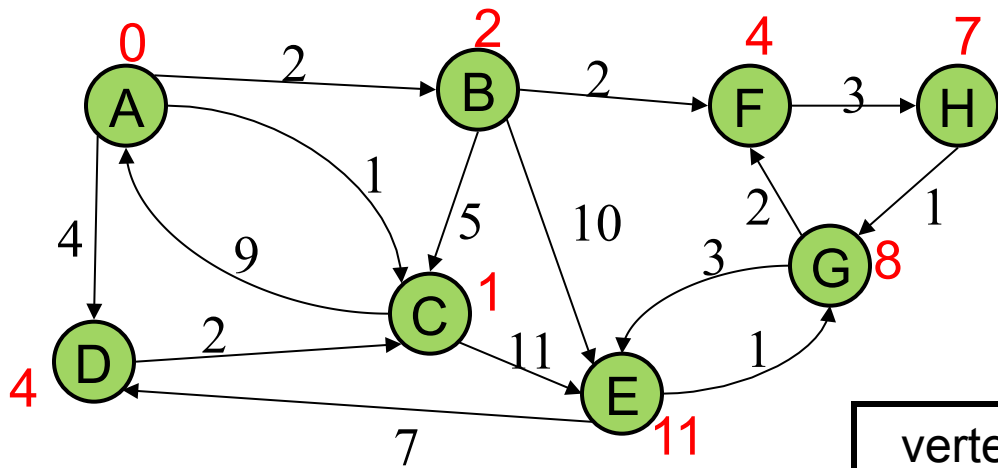
A, C, B, D, F, H



vertex	known?	cost	path
A	Y	0	
B	Y	2	A
C	Y	1	A
D	Y	4	A
E		≤ 11	G
F	Y	4	B
G	Y	8	H
H	Y	7	F

Order Added to Known Set:

A, C, B, D, F, H, G



vertex	known?	cost	path
A	Y	0	
B	Y	2	A
C	Y	1	A
D	Y	4	A
E	Y	11	G
F	Y	4	B
G	Y	8	H
H	Y	7	F

Order Added to Known Set:

A, C, B, D, F, H, G, E

FEATURES

**When a vertex is marked known,
the cost of the shortest path to that node is known**

- The path is also known by following back-pointers

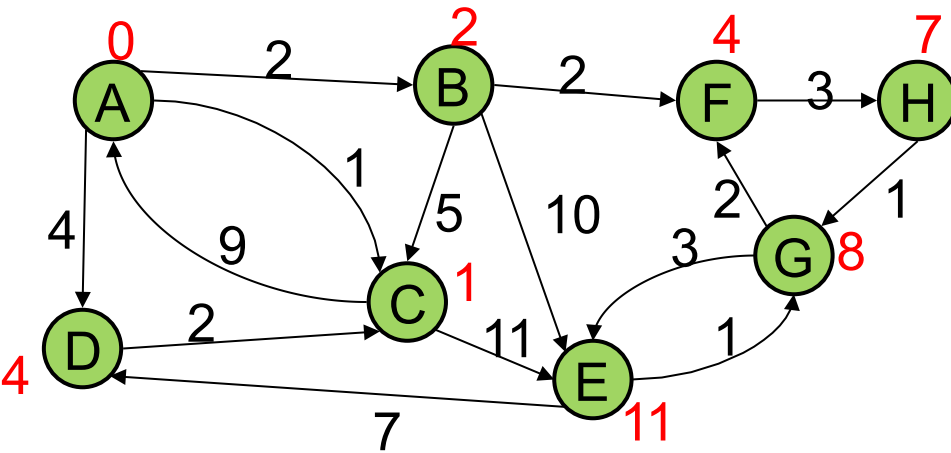
**While a vertex is still not known,
another shorter path to it might still be found**

Note: The “Order Added to Known Set” is not important

- A detail about how the algorithm works (client doesn't care)
- Not used by the algorithm (implementation doesn't care)
- It is sorted by path-cost, resolving ties in some way
 - Helps give intuition of why the algorithm works

INTERPRETING THE RESULTS

Now that we're done, how do we get the path from, say, A to E?



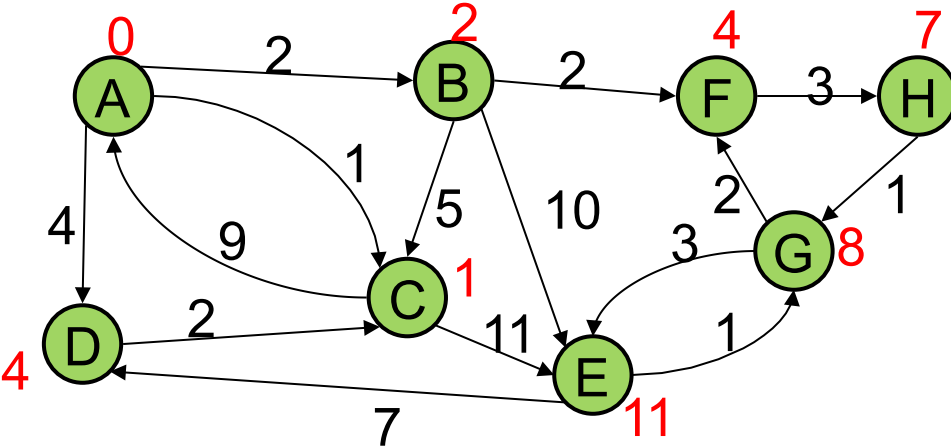
Order Added to Known Set:

A, C, B, D, F, H, G, E

vertex	known?	cost	path
A	Y	0	
B	Y	2	A
C	Y	1	A
D	Y	4	A
E	Y	11	G
F	Y	4	B
G	Y	8	H
H	Y	7	F

How would this have worked differently if we were only interested in:

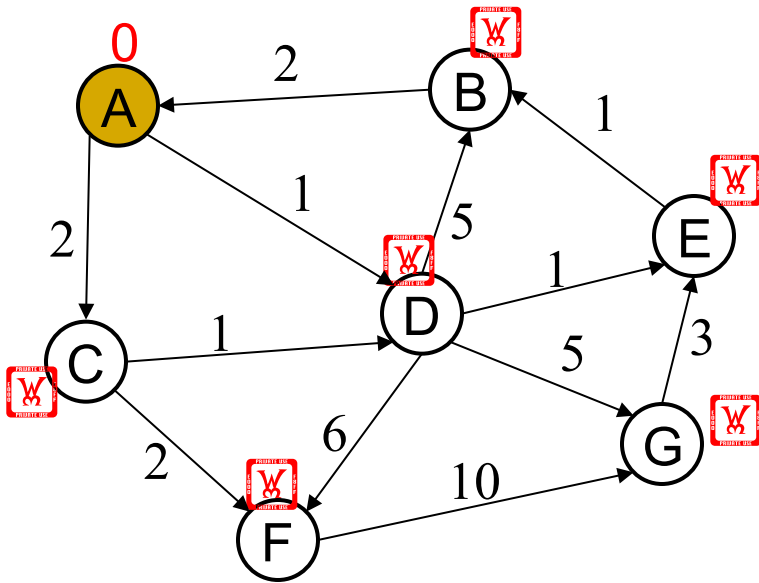
- The path from A to G?
- The path from A to E?



Order Added to Known Set:

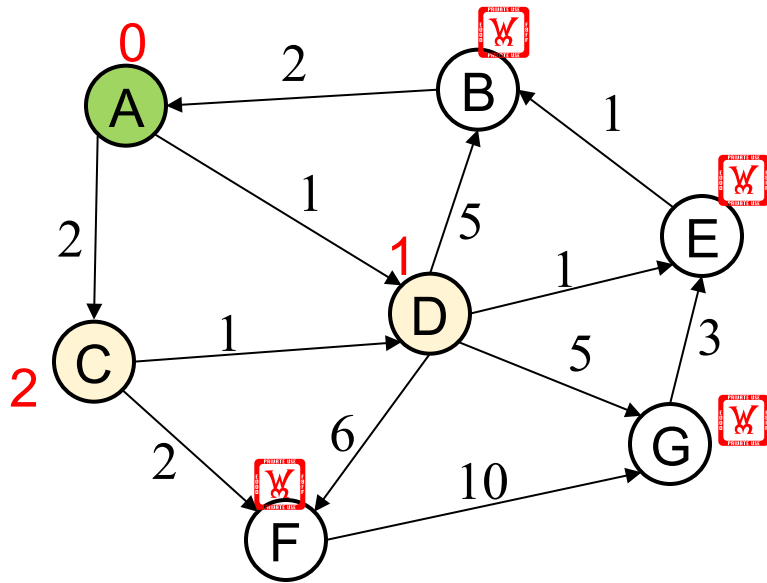
A, C, B, D, F, H, G, E

vertex	known?	cost	path
A	Y	0	
B	Y	2	A
C	Y	1	A
D	Y	4	A
E	Y	11	G
F	Y	4	B
G	Y	8	H
H	Y	7	F



Order Added to Known Set:

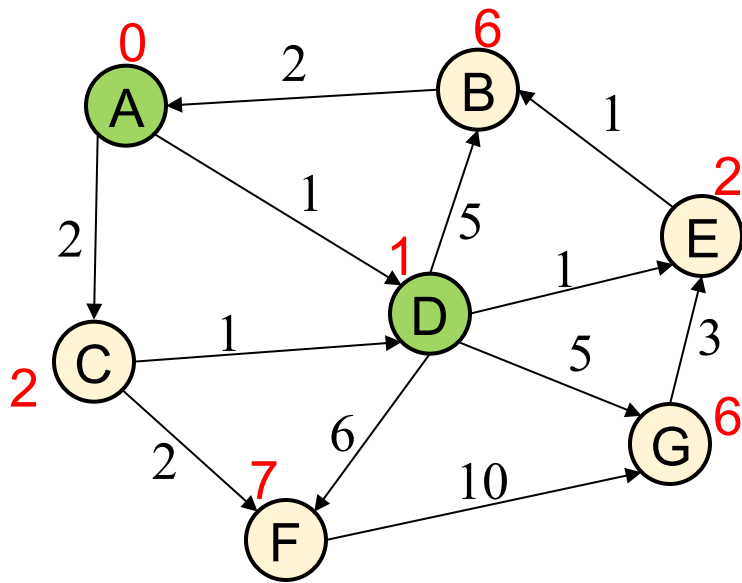
vertex	known?	cost	path
A		0	
B		??	
C		??	
D		??	
E		??	
F		??	
G		??	



Order Added to Known Set:

A

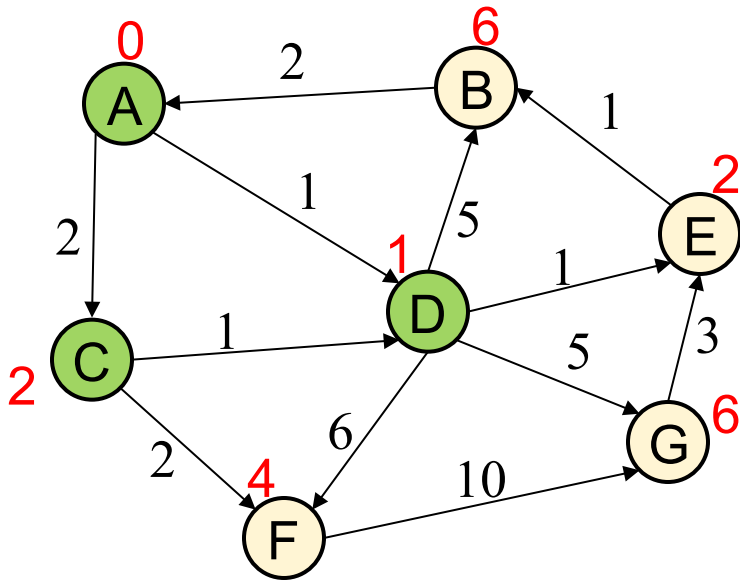
vertex	known?	cost	path
A	Y	0	
B		??	
C		≤ 2	A
D		≤ 1	A
E		??	
F		??	
G		??	



Order Added to Known Set:

A, D

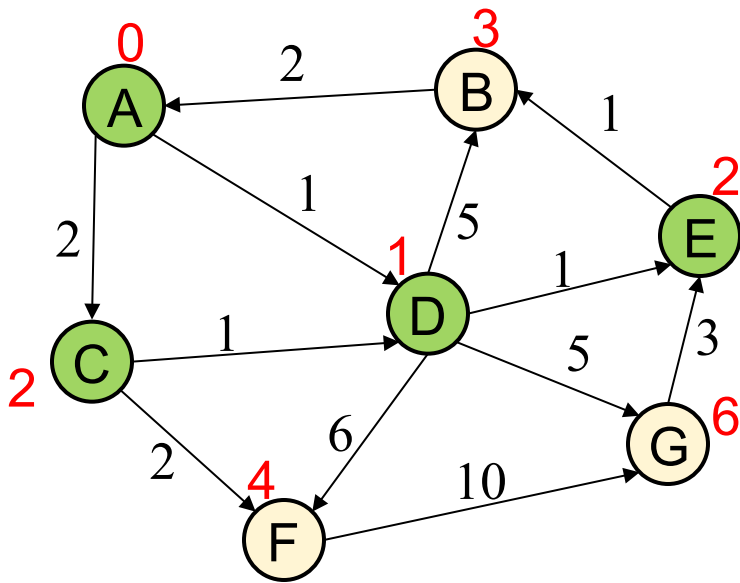
vertex	known?	cost	path
A	Y	0	
B		≤ 6	D
C		≤ 2	A
D	Y	1	A
E		≤ 2	D
F		≤ 7	D
G		≤ 6	D



Order Added to Known Set:

A, D, C

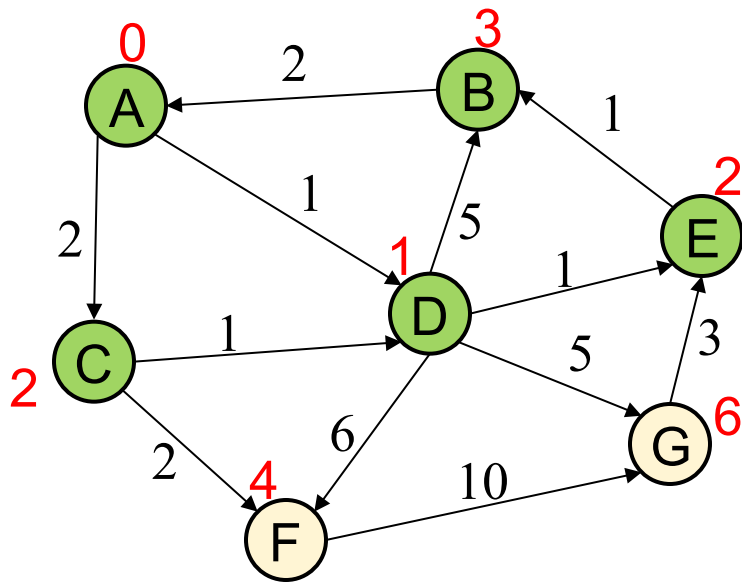
vertex	known?	cost	path
A	Y	0	
B		≤ 6	D
C	Y	2	A
D	Y	1	A
E		≤ 2	D
F		≤ 4	C
G		≤ 6	D



Order Added to Known Set:

A, D, C, E

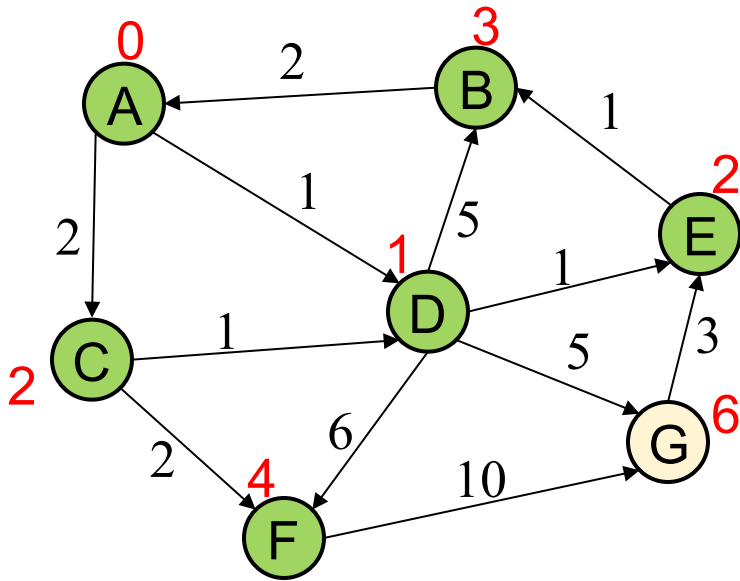
vertex	known?	cost	path
A	Y	0	
B		≤ 3	E
C	Y	2	A
D	Y	1	A
E	Y	2	D
F		≤ 4	C
G		≤ 6	D



Order Added to Known Set:

A, D, C, E, B

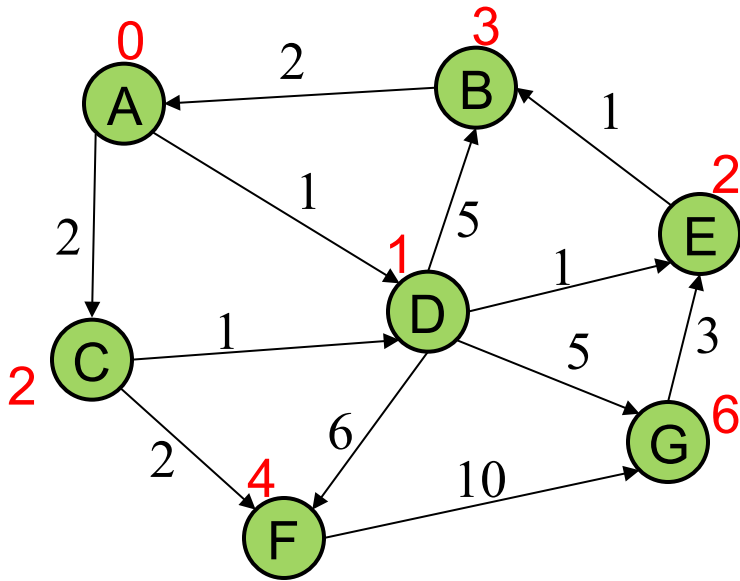
vertex	known?	cost	path
A	Y	0	
B	Y	3	E
C	Y	2	A
D	Y	1	A
E	Y	2	D
F		≤ 4	C
G		≤ 6	D



Order Added to Known Set:

A, D, C, E, B, F

vertex	known?	cost	path
A	Y	0	
B	Y	3	E
C	Y	2	A
D	Y	1	A
E	Y	2	D
F	Y	4	C
G		≤ 6	D



Order Added to Known Set:

A, D, C, E, B, F, G

vertex	known?	cost	path
A	Y	0	
B	Y	3	E
C	Y	2	A
D	Y	1	A
E	Y	2	D
F	Y	4	C
G	Y	6	D

RUNTIME AND IMPLEMENTATION

- **To keep track of which vertex should be added next, we use a priority queue.**
 - Each of the $|V|$ vertices will need to be added into the queue

RUNTIME AND IMPLEMENTATION

- **To keep track of which vertex should be added next, we use a priority queue.**
 - Each of the $|V|$ vertices will need to be added into the queue
 - Together this is $O(|V| \log |V|)$

RUNTIME AND IMPLEMENTATION

- **To keep track of which vertex should be added next, we use a priority queue.**
 - Each of the $|V|$ vertices will need to be added into the queue
 - Together this is $O(|V| \log |V|)$
- **Each edge has an opportunity to change the value in the heap (notice this means we need the change priority function)**
 - For each edge, change priority is a $\log|V|$ operation, so this is total $O(|E| \log |V|)$

RUNTIME AND IMPLEMENTATION

- Together then, we have that Dijkstra's algorithm, if smartly implemented using a priority queue is $O(|V| \log |V| + |E| \log |V|)$
 - If the graph is connected, however (which is reasonable to assume since we're trying to find a path from a single source to all other nodes, then there must be at least $|V|-1$ edges.

RUNTIME AND IMPLEMENTATION

- Together then, we have that Dijkstra's algorithm, if smartly implemented using a priority queue is $O(|V| \log |V| + |E| \log |V|)$
 - If the graph is connected, however (which is reasonable to assume since we're trying to find a path from a single source to all other nodes, then there must be at least $|V|-1$ edges.
 - This algorithm is $O(|E| \log |V|)$ time

RUNTIME AND IMPLEMENTATION

- **Together then, we have that Dijkstra's algorithm, if smartly implemented using a priority queue is $O(|V| \log |V| + |E| \log |V|)$**
 - If the graph is connected, however (which is reasonable to assume since we're trying to find a path from a single source to all other nodes, then there must be at least $|V|-1$ edges.
 - This algorithm is $O(|E| \log |V|)$ time
 - Without the priority queue, it runs in $O(|E||V|)$ time

CORRECTNESS

- **Dijkstra's algorithm is an example of a greedy-first approach**

CORRECTNESS

- **Dijkstra's algorithm is an example of a greedy-first approach**
 - Take the closest next available vertex and add it to the known cloud

CORRECTNESS

- **Dijkstra's algorithm is an example of a greedy-first approach**
 - Take the closest next available vertex and add it to the known cloud
 - Since we do not allow negative weights, we know that there cannot be a way from A to v that is shorter if it is currently the shortest available path

CORRECTNESS

- **Dijkstra's algorithm is an example of a greedy-first approach**
 - Take the closest next available vertex and add it to the known cloud
 - Since we do not allow negative weights, we know that there cannot be a way from A to v that is shorter if it is currently the shortest available path
 - Recursively path-finds, the last element only knows what vertex came before us, and how to optimally reach that—single source to ALL other vertices

NEW GRAPH PROBLEM

- **Minimum spanning trees**

NEW GRAPH PROBLEM

- **Minimum spanning trees**
 - Given an undirected graph $G(V,E)$, find a subgraph $G_2(V,E_{\min})$ such that:
 - G_2 is connected
 - E_{\min} is a subset of E
 - The sum of the weights of E_{\min} is minimal

PROBLEM STATEMENT

Given a *connected* undirected graph $G=(V,E)$, find a minimal subset of edges such that G is still connected

- A graph $G_2=(V,E_2)$ such that G_2 is connected and removing any edge from E_2 makes G_2 disconnected

