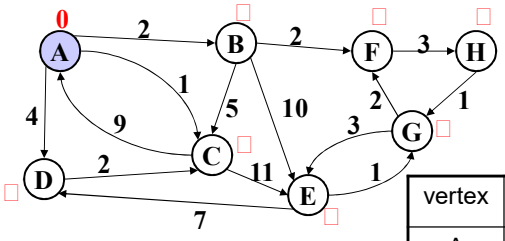
Efficiency, first approach

Use pseudocode to determine asymptotic run-time

- Notice each edge is processed only once

```
dijkstra(Graph G, Node start) {
  for each node: x.cost=infinity, x.known=false
  start.cost = 0
  while(not all nodes are known) {
    b = find unknown node with smallest cost
    b.known = true
    for each edge (b,a) in G
     if(!a.known)
       if(b.cost + weight((b,a)) < a.cost){
         a.cost = b.cost + weight((b,a))
         a.path = b
       }
}
```


Example #1



Order Added to Known Set:

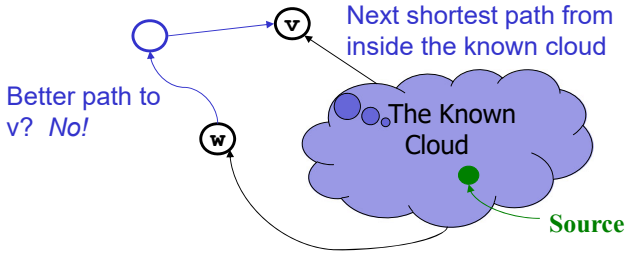
vertex	known?	cost	path
A			
B			
C			
D			
E			
F			
G			
H			

A Greedy Algorithm

- Dijkstra's algorithm
  - For single-source shortest paths in a weighted graph (directed or undirected) with no negative-weight edges

- An example of a *greedy algorithm*:
  - At each step, irrevocably does what seems best at that step
    - A locally optimal step, not necessarily globally optimal
  - Once a vertex is known, it is not revisited
    - Turns out to be globally optimal

Correctness: The Cloud (Rough Idea)



Suppose **v** is the next node to be marked known ("added to the cloud")

- The best-known path to **v** must have only nodes "in the cloud"
  - Since we've selected it, and we only know about paths through the cloud to a node right outside the cloud
- Assume the actual shortest path to **v** is different
  - It won't use only cloud nodes, (or we would know about it), so it must use non-cloud nodes
  - Let **w** be the *first* non-cloud node on this path.
  - The part of the path up to **w** is already known and must be shorter than the best-known path to **v**. So **v** would not have been picked.

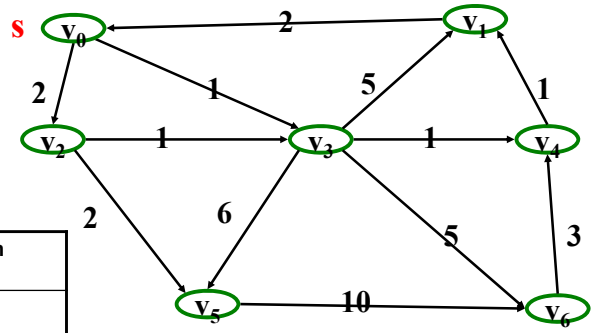
Contradiction!

Efficiency, second approach

Use pseudocode to determine asymptotic run-time

```
dijkstra(Graph G, Node start) {
  for each node: x.cost=infinity, x.known=false
  start.cost = 0
  build-heap with all nodes
  while(heap is not empty) {
    b = deleteMin()
    b.known = true
    for each edge (b,a) in G
     if(!a.known)
      if(b.cost + weight((b,a)) < a.cost){
        decreaseKey(a,"new cost - old cost")
        a.path = b
      }
}
```


Find the shortest path to each vertex from v_0



V	Known	Dist from s	Path
v0			
v1			
v2			
v3			
v4			
v5			
v6			

Order declared Known: