# CSE 322: Shortest Paths

Richard Anderson

Spring 2016

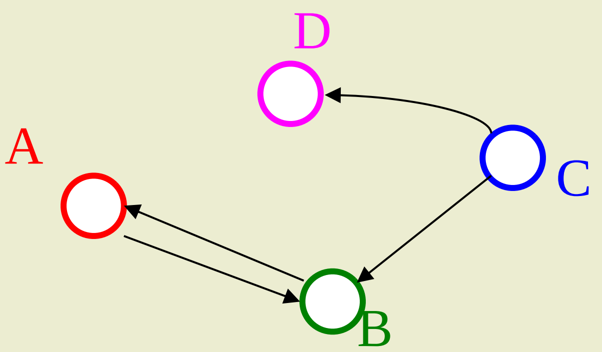
# Announcements

- 

# Graphs

- A formalism for representing relationships between objects

  - Graph $\mathbf{G = (V,E)}$
  - *Set of vertices*: $\mathbf{V = \{v_1, v_2, \ldots, v_n\}}$
  - *Set of edges*: $\mathbf{E = \{e_1, e_2, \ldots, e_m\}}$ where each $\mathbf{e_i}$ connects one
  - vertex to another $\mathbf{(v_j, v_k)}$




$\mathbf{V = \{A, B, C, D\}}$

$\mathbf{E = \{(C, B), (A, B), (B, A), (C, D)\}}$

- For *directed edges*, $\mathbf{(v_j, v_k)}$ and $\mathbf{(v_k, v_j)}$ are distinct. (More on this later…)

# Paths and connectivity

# The Shortest Path Problem

Given a graph $G$, and vertices $s$ and $t$ in $G$, find the shortest path from $s$ to $t$.

Two cases: weighted and unweighted.

For a path $p = v_0\ v_1\ v_2 \ldots v_k$

- *unweighted length* of path $p = k$ (a.k.a. *length*)
- *weighted length* of path $p = \sum_{i=0..k-1} c_{i,i+1}$ (a.k.a. *cost*)

# Single Source Shortest Paths (SSSP)

Given a graph *G* and vertex *s*, find the shortest paths from *s* to all vertices in G.

– How much harder is this than finding single shortest path from s to t?

# Variations of SSSP

- – Weighted vs. unweighted
- – Directed vs undirected
- – Cyclic vs. acyclic
- – Positive weights only vs. negative weights allowed
- – Shortest path vs. longest path
- – …

# Applications

- Network routing
- Driving directions
- Cheap flight tickets
- Critical paths in project management (see textbook)
- …

# SSSP: Unweighted Version

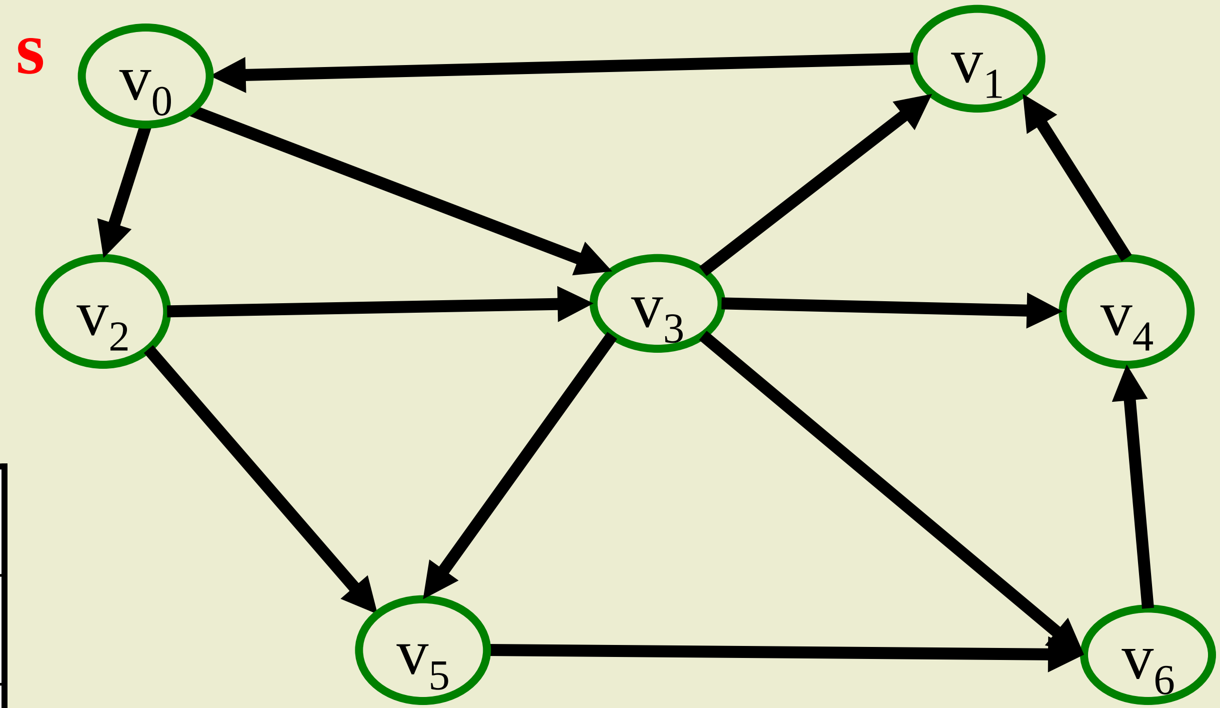
```
void Graph::unweighted (Vertex s){
  Queue q(NUM_VERTICES);
  Vertex v, w;
  q.enqueue(s);
  s.dist = 0;

  while (!q.isEmpty()){
    v = q.dequeue();
    for each w adjacent to v
      if (w.dist == INFINITY){
        w.dist = v.dist + 1;
        w.prev = v;
        q.enqueue(w);
      }
  }
}
```

each edge examined at most once – if adjacency lists are used

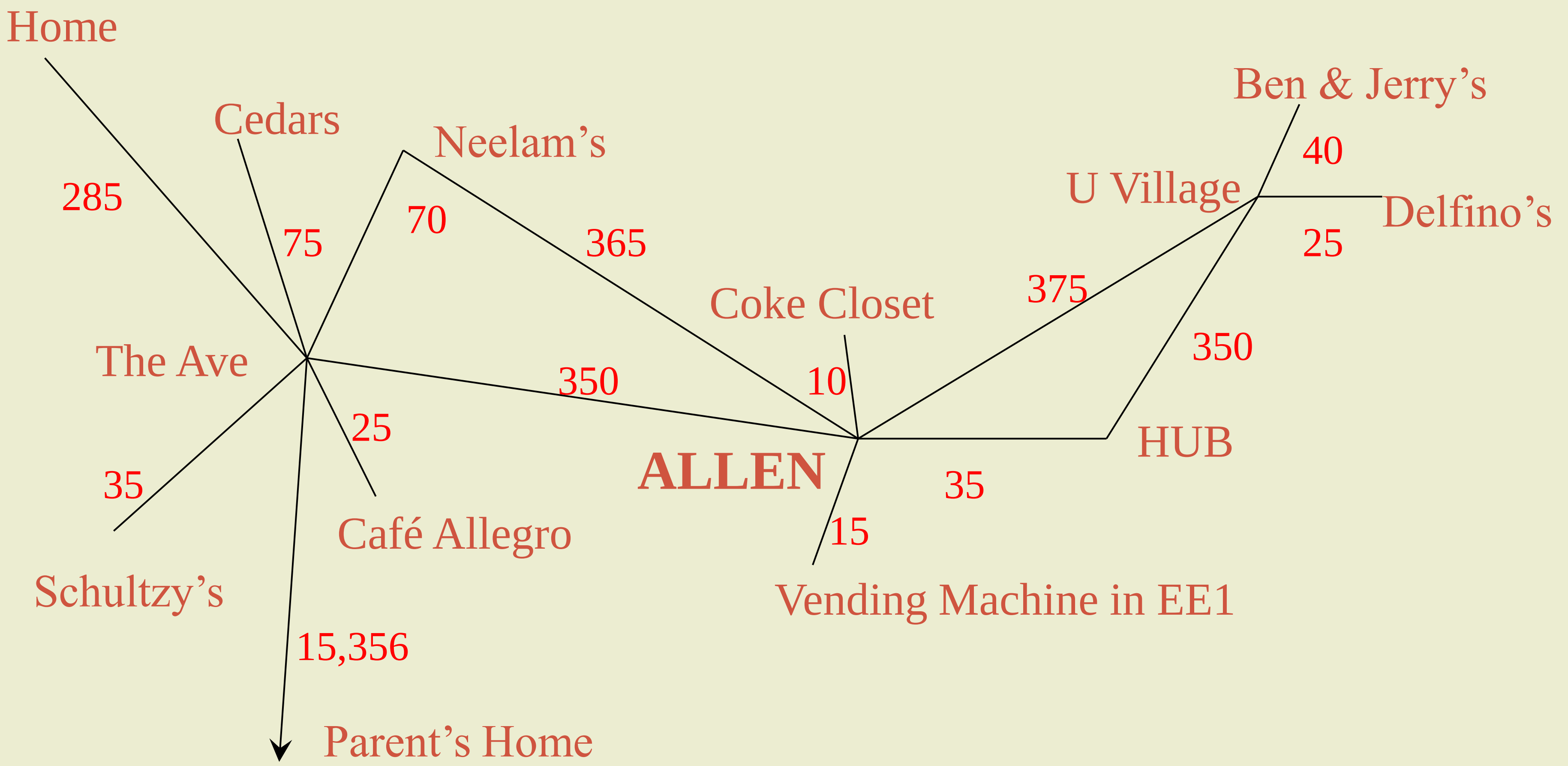
each vertex enqueued at most once

total running time: O(   )



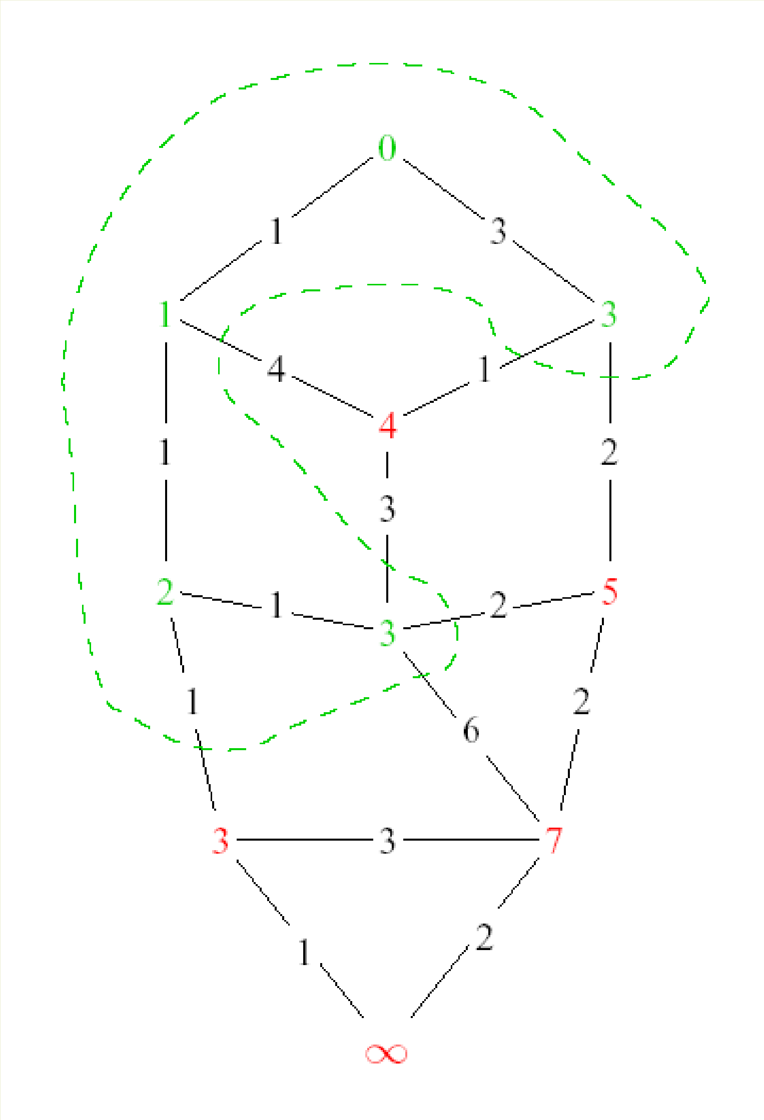

| V | Dist | prev |
|---|---|---|
| v0 | | |
| v1 | | |
| v2 | | |
| v3 | | |
| v4 | | |
| v5 | | |
| v6 | | |

# Weighted SSSP:
# All edges are not created equal




*Can we calculate shortest distance to all vertices from Allen Center?*

# Dijkstra's Algorithm: Idea

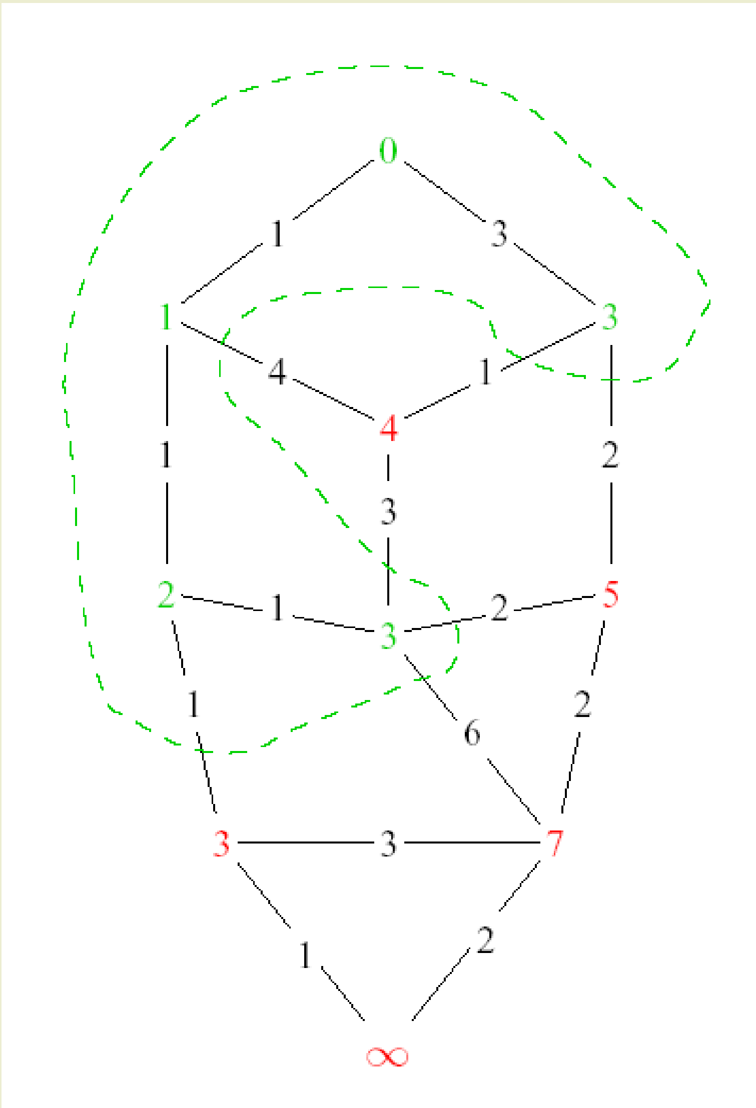



Adapt BFS to handle weighted graphs

Two kinds of vertices:

- Known
  - shortest distance is already known
- Unknown
  - Have tentative distance

# Dijkstra’s Algorithm: Idea




At each step:

1) Pick closest unknown vertex
2) Add it to known vertices
3) Update distances

# Dijkstra's Algorithm: Pseudocode

Initialize the cost of each node to $\infty$
Initialize the cost of the source to 0

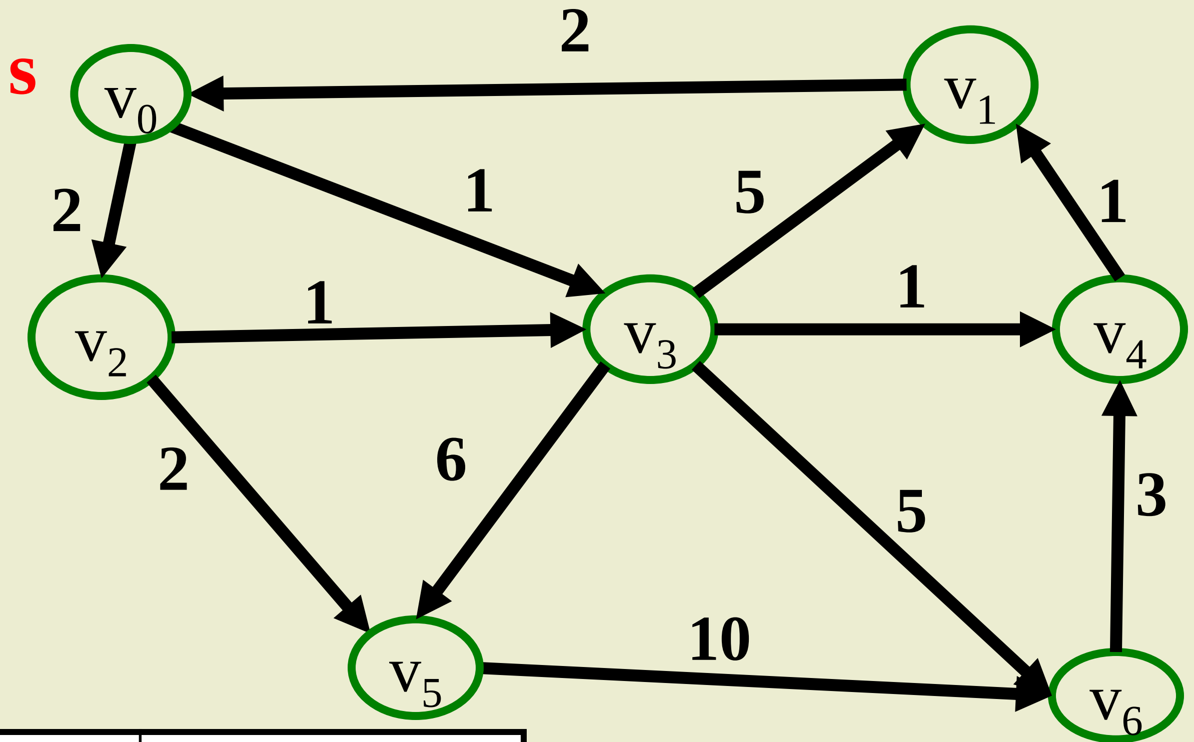
While there are unknown vertices left in the graph
    Select an unknown vertex ***a*** with the lowest cost
    Mark ***a*** as known
    For each vertex ***b*** adjacent to ***a***
        newcost = cost(**a**) + cost(**a**,**b**)
        if (newcost < cost(**b**))
            cost(**b**) = newcost
            previous(**b**) = **a**

# Important Features

- Once a vertex is known, the cost of the shortest path to that vertex is known

- While a vertex is still unknown, another shorter path to it might still be found

- The shortest path can found by following the previous pointers stored at each vertex




| V | Known? | Cost | Previous |
|---|---|---|---|
| v0 | | | |
| v1 | | | |
| v2 | | | |
| v3 | | | |
| v4 | | | |
| v5 | | | |
| v6 | | | |

# Dijkstra's Alg: Implementation

Initialize the cost of each vertex to $\infty$

Initialize the cost of the source to 0

While there are unknown vertices left in the graph

  Select the unknown vertex ***a*** with the lowest cost

  Mark ***a*** as known

  For each vertex ***b*** adjacent to ***a***

    newcost = min(cost(***b***), cost(***a***) + cost(***a***, ***b***))

    if newcost < cost(**b**)

      cost(***b***) = newcost

      previous(***b***) = *a*

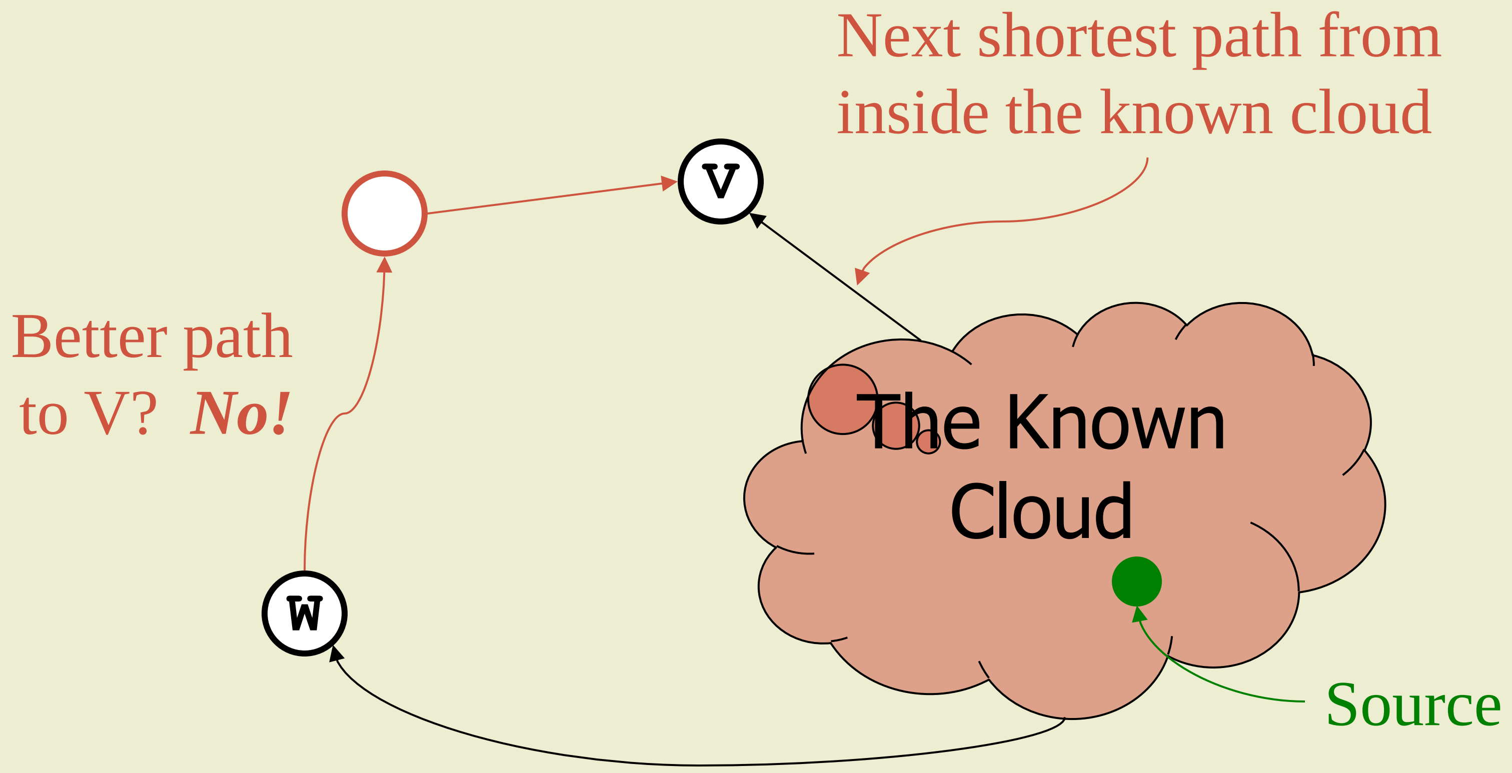
What data structures should we use?

Running time?

# Dijkstra's Algorithm: Summary

- Classic algorithm for solving SSSP in weighted graphs *without negative weights*
- A *greedy* algorithm (irrevocably makes decisions without considering future consequences)
- Why does it work?

# Correctness: The Cloud Proof




How does Dijkstra's decide which vertex to add to the Known set next?

- If path to **V** is shortest, path to **W** must be *at least as long*
  *(or else we would have picked* **W** *as the next vertex)*
- So the path through **W** to **V** cannot be any shorter!

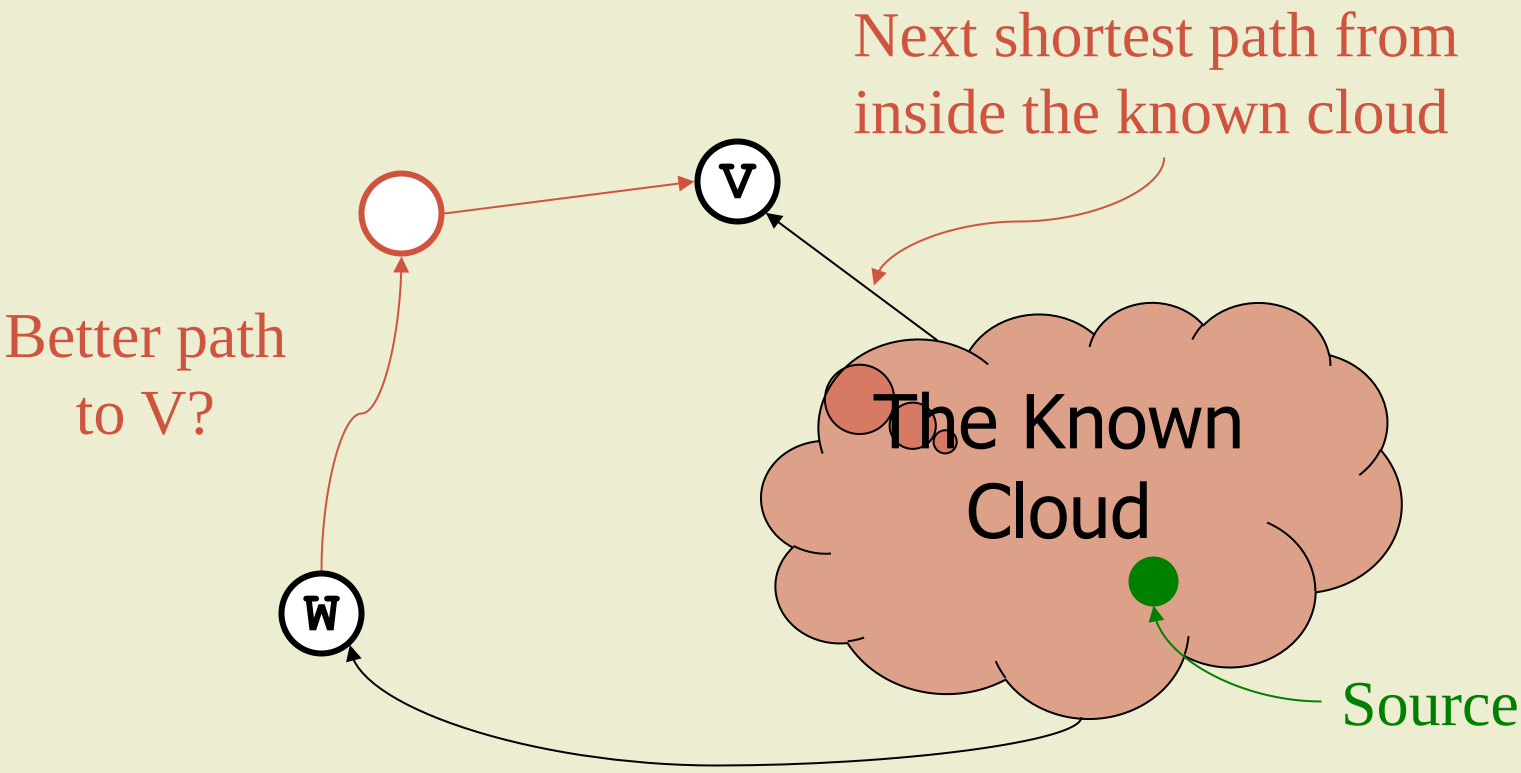
# Correctness: Inside the Cloud

Prove by induction on # of nodes in the cloud:

- Initial cloud is just the source with shortest path 0
- Assume: Everything inside the cloud has the correct shortest path
- Inductive step: by argument on previous slide, we can safely add min-cost vertex to cloud

**When does Dijkstra's algorithm not work?**

# Negative Weights?




How does Dijkstra's decide which vertex to add to the Known set next?

- If path to **v** is shortest, path to **w** must be *at least as long*
  *(or else we would have picked* **w** *as the next vertex)*
- So the path through **w** to **v** cannot be any shorter!

# Dijkstra for BFS

- You can use Dijkstra's algorithm for BFS
- Is this a good idea?