CSE 326: Data Structures Dijkstra's Algorithm

James Fogarty

Autumn 2007

Dijkstra, Edsger Wybe

Legendary figure in computer science; was a professor at University of Texas.

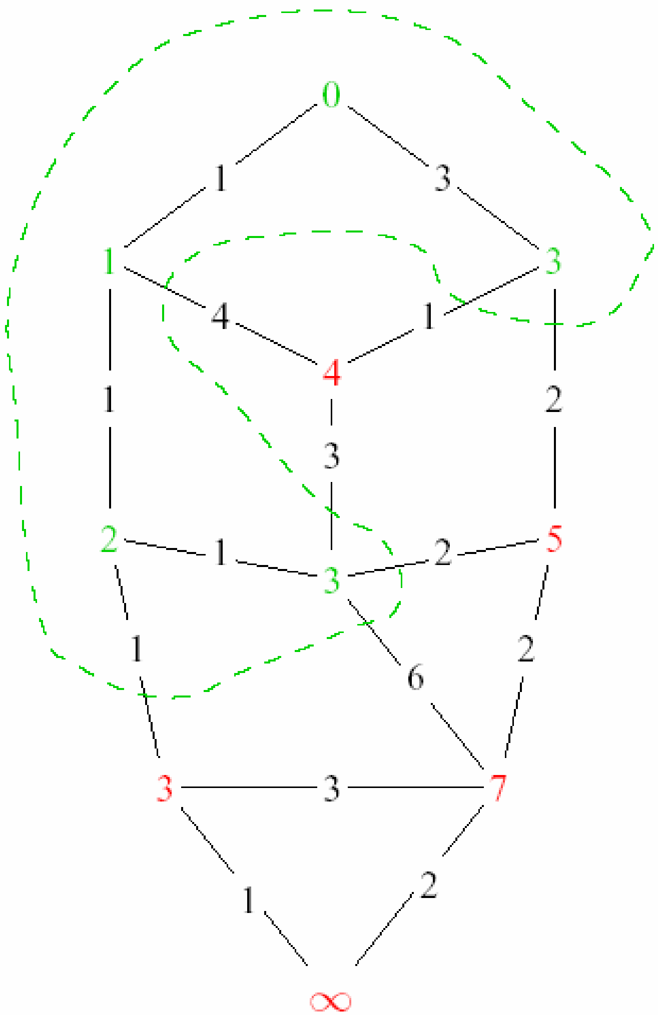
Supported teaching introductory computer courses without computers (pencil and paper programming)

Supposedly wouldn’t (until very late in life) read his e-mail; so, his staff had to print out messages and put them in his box.

E.W. Dijkstra (1930-2002)

1972 Turning Award Winner,
Programming Languages, semaphores, and …

Dijkstra's Algorithm: Idea

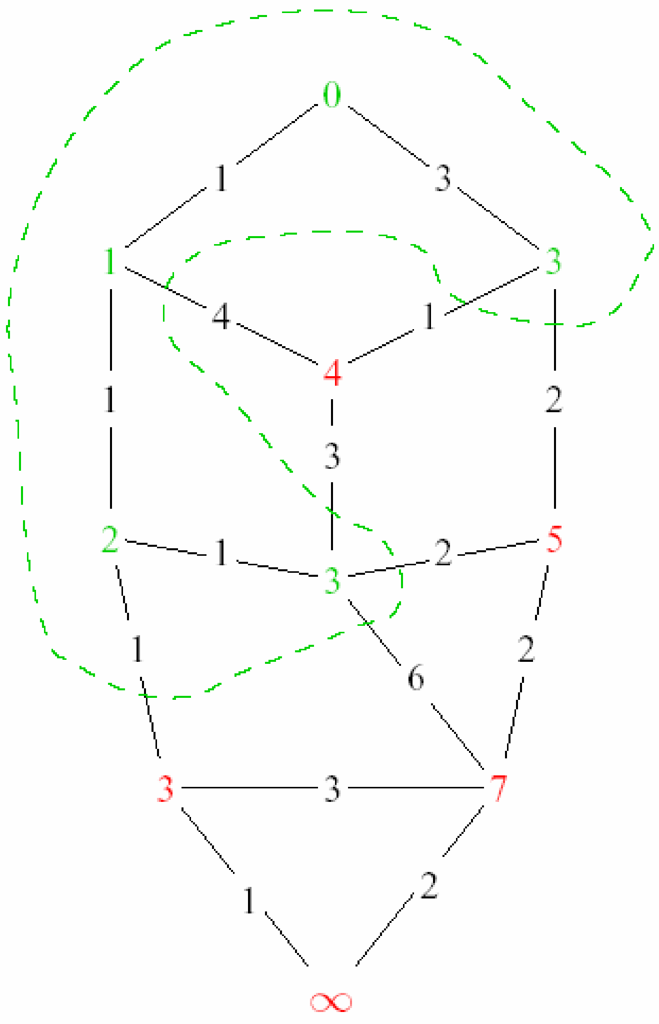


Adapt BFS to handle weighted graphs

Two kinds of vertices:

- Finished or known vertices
  - Shortest distance has been computed
- Unknown vertices
  - 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 ∞

Initialize the cost of the source to 0

While there are unknown nodes left in the graph

    Select an unknown node *b* with the lowest cost

    Mark *b* as known

    For each node *a* adjacent to *b*

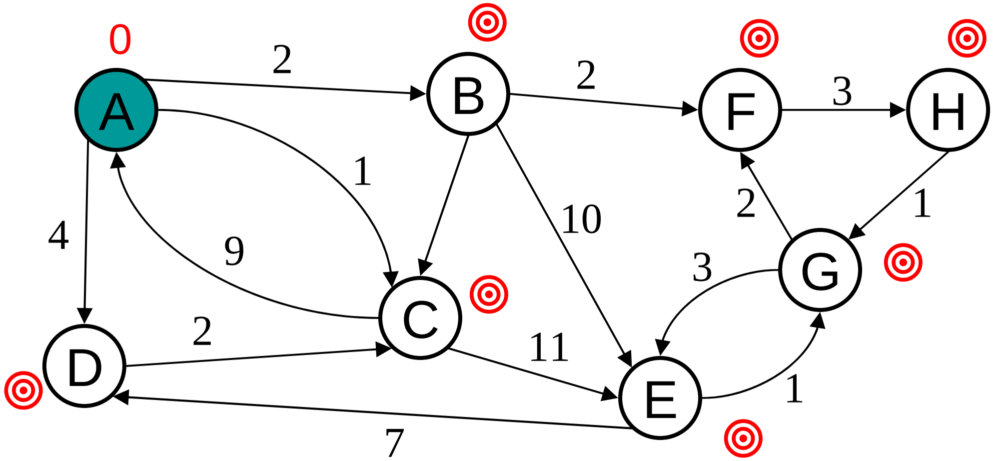
        a’s cost = min(*a*’s old cost, *b*’s cost + cost of (*b*, *a*))

        a’s prev path node = *b*

Important Features

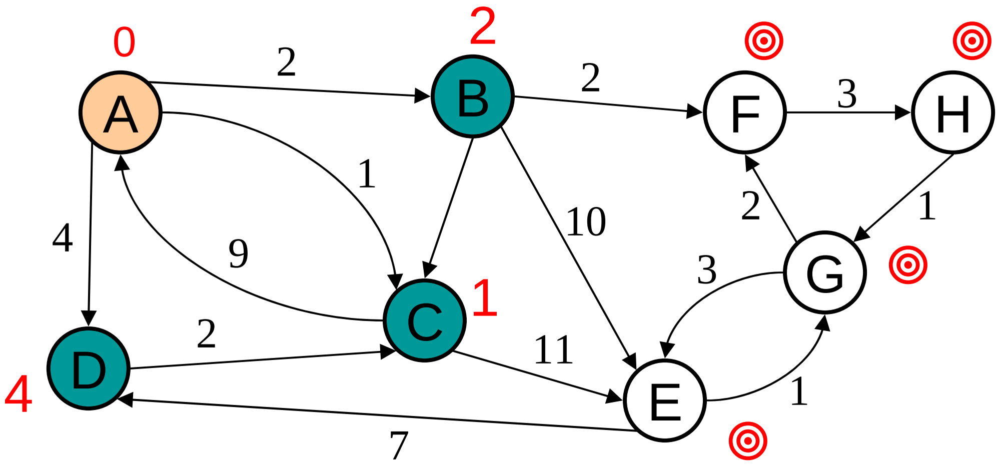
- Once a vertex is made **known**, the cost of the shortest path to that node is known
- While a vertex is still not **known**, another shorter path to it might still be found
- The shortest path itself can found by following the backward pointers stored in **`node.path`**

Dijkstra’s Algorithm in action



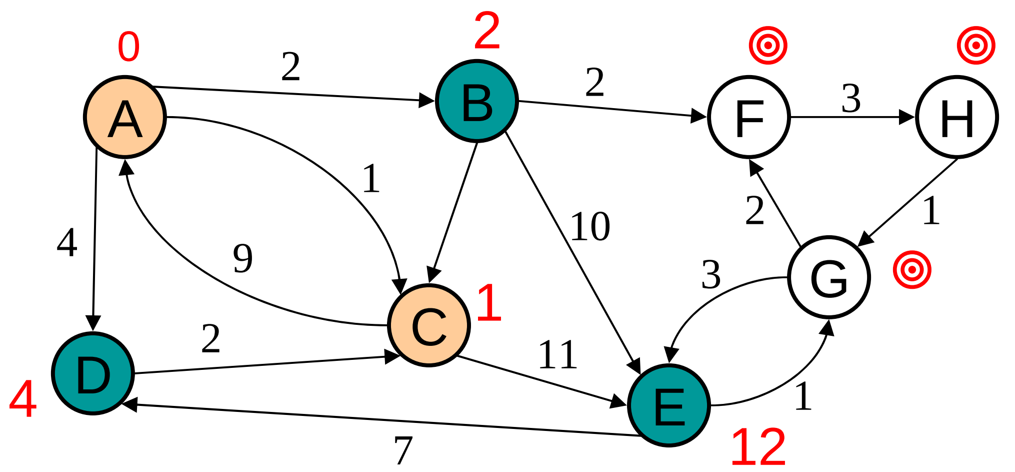
Vertex	Visited?	Cost	Found by
A		0	
B		??	
C		??	
D		??	
E		??	
F		??	
G		??	
H		??	

Dijkstra’s Algorithm in action



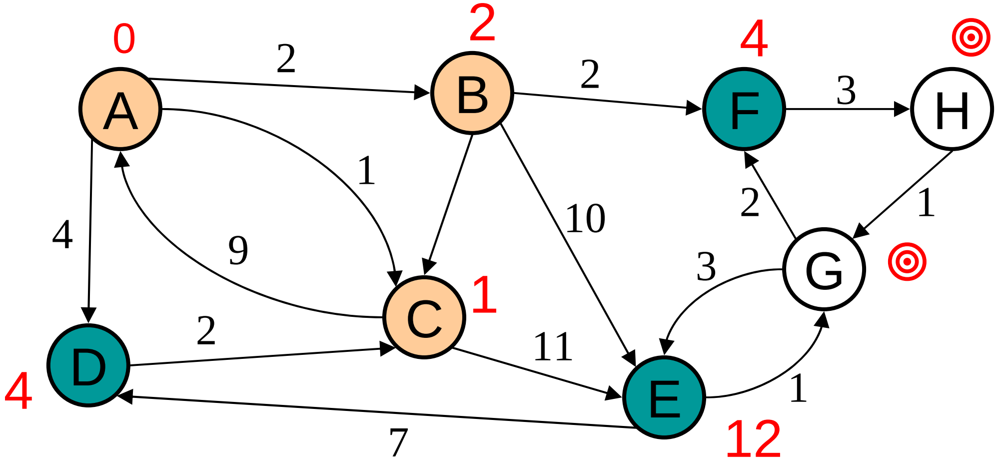
Vertex	Visited?	Cost	Found by
A	Y	0	
B		<=2	A
C		<=1	A
D		<=4	A
E		??	
F		??	
G		??	
H		??	

Dijkstra's Algorithm in action



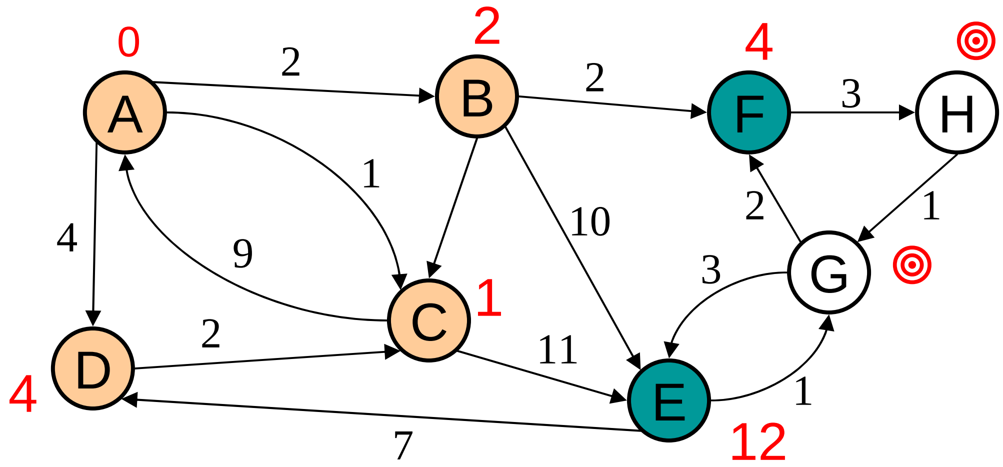
Vertex	Visited?	Cost	Found by
A	Y	0	
B		<=2	A
C	Y	1	A
D		<=4	A
E		<=12	C
F		??	
G		??	
H		??	

Dijkstra's Algorithm in action



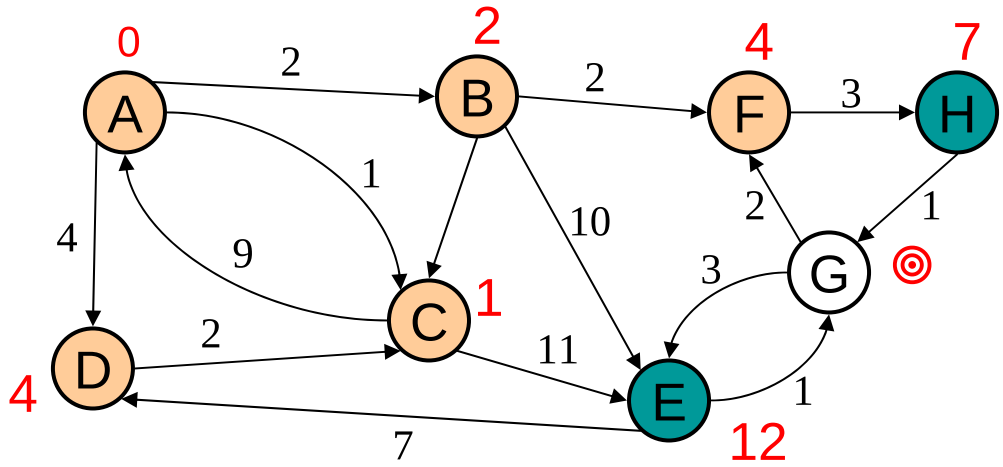
Vertex	Visited?	Cost	Found by
A	Y	0	
B	Y	2	A
C	Y	1	A
D		<=4	A
E		<=12	C
F		<=4	B
G		??	
H		??	

Dijkstra’s Algorithm in action



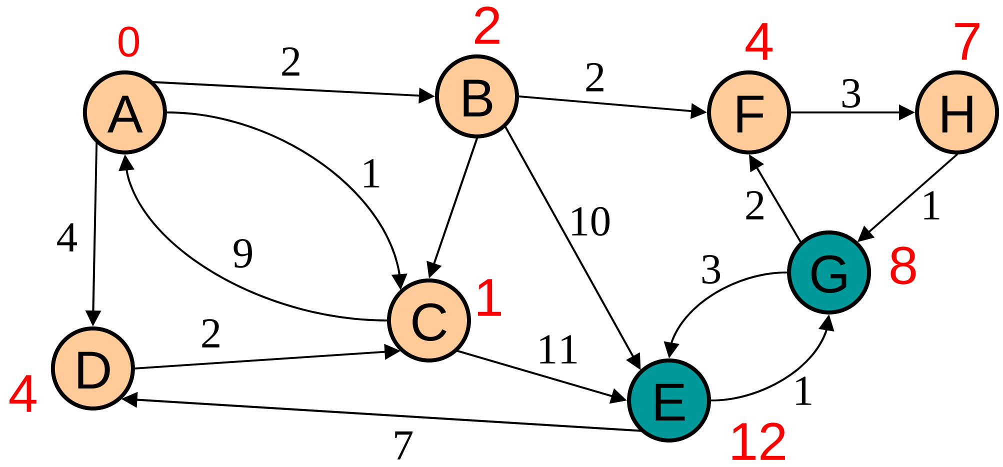
Vertex	Visited?	Cost	Found by
A	Y	0	
B	Y	2	A
C	Y	1	A
D	Y	4	A
E		<=12	C
F		<=4	B
G		??	
H		??	

Dijkstra's Algorithm in action



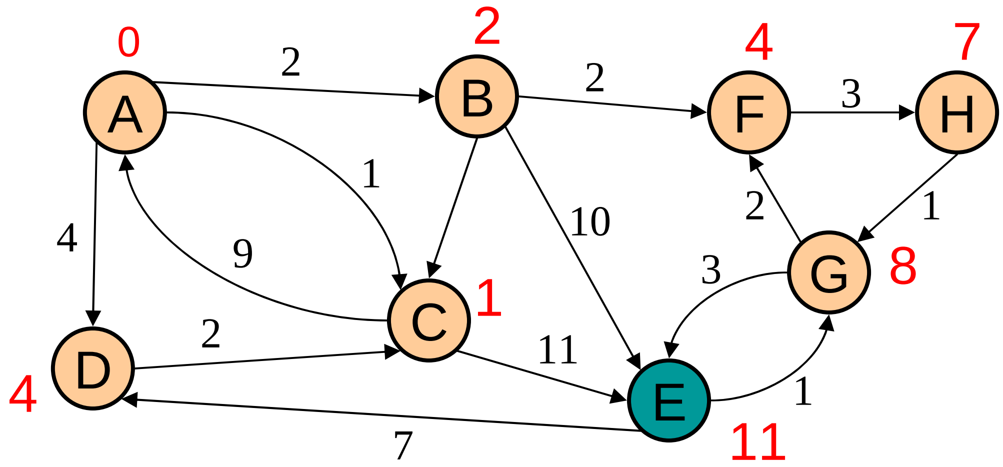
Vertex	Visited?	Cost	Found by
A	Y	0	
B	Y	2	A
C	Y	1	A
D	Y	4	A
E		<=12	C
F	Y	4	B
G		??	
H		<=7	F

Dijkstra's Algorithm in action



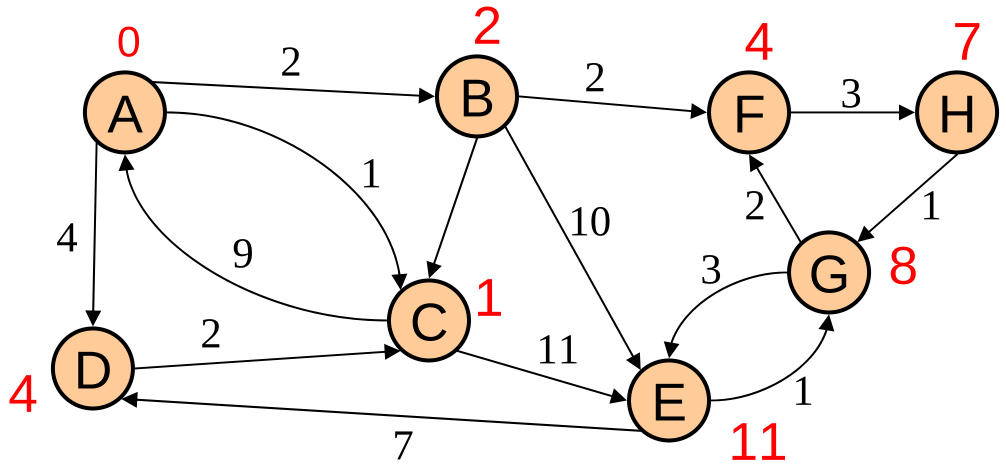
Vertex	Visited?	Cost	Found by
A	Y	0	
B	Y	2	A
C	Y	1	A
D	Y	4	A
E		<=12	C
F	Y	4	B
G		<=8	H
H	Y	7	F

Dijkstra’s Algorithm in action



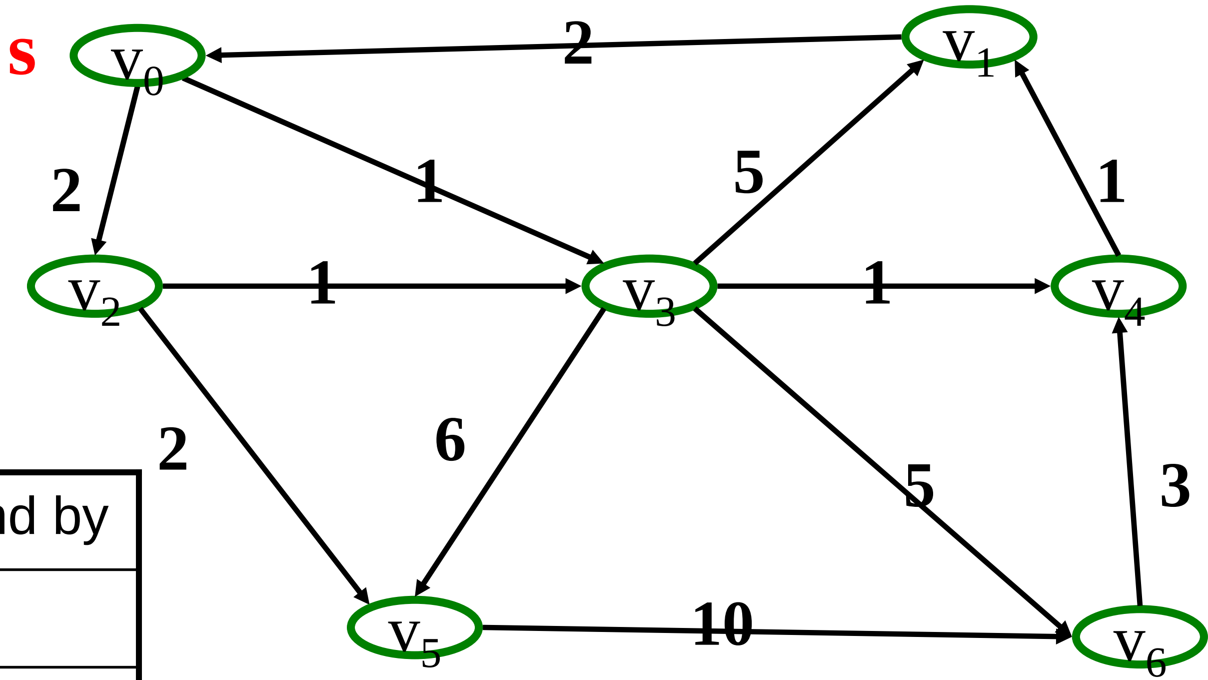
Vertex	Visited?	Cost	Found by
A	Y	0	
B	Y	2	A
C	Y	1	A
D	Y	4	A
E		<=11	G
F	Y	4	B
G	Y	8	H
H	Y	7	F

Dijkstra's Algorithm in action



Vertex	Visited?	Cost	Found by
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

Your turn



V	Visited?	Cost	Found by
v0			
v1			
v2			
v3			
v4			
v5			
v6			

Dijkstra's Alg: Implementation

Initialize the cost of each node to ∞

Initialize the cost of the source to 0

While there are unknown nodes left in the graph

- Select the unknown node *b* with the lowest cost
- Mark *b* as known
- For each node *a* adjacent to *b*
  - *a*'s cost = min(*a*'s old cost, *b*'s cost + cost of (*b*, *a*))
  - *a*'s prev path node = *b* *(if we updated a's cost)*

What data structures should we use?

Running time?

```
void Graph::dijkstra(Vertex s){
  Vertex v,w;

  Initialize s.dist = 0 and set dist of all other
  vertices to infinity

  while (there exist unknown vertices, find the
  one b with the smallest distance)
    b.known = true;

    for each a adjacent to b
      if (!a.known)
        if (b.dist + weight(b,a) < a.dist){
          a.dist = (b.dist + weight(b,a));
          a.path = b;
        }
  }
}
```

Sounds like deleteMin on a heap…

Sounds like adjacency lists

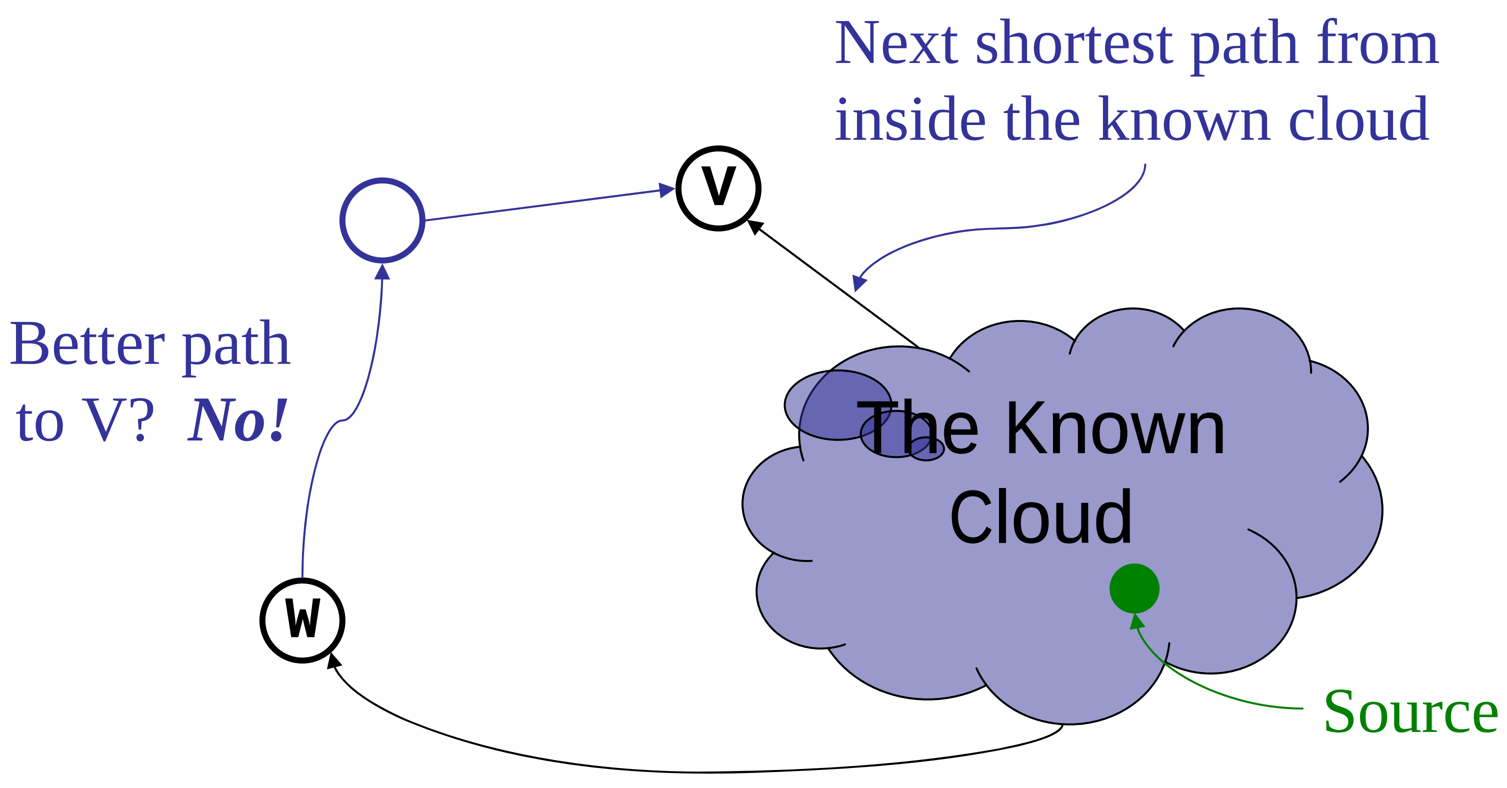
Sounds like decreaseKey

Running time: O(|E| log |V|) – there are |E| edges to examine, and each one causes a heap operation of time O(log |V|)

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)
- Intuition for correctness:
  - shortest path from source vertex to itself is 0
  - cost of going to adjacent nodes is at most edge weights
  - cheapest of these must be shortest path to that node
  - update paths for new node and continue picking cheapest path

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!

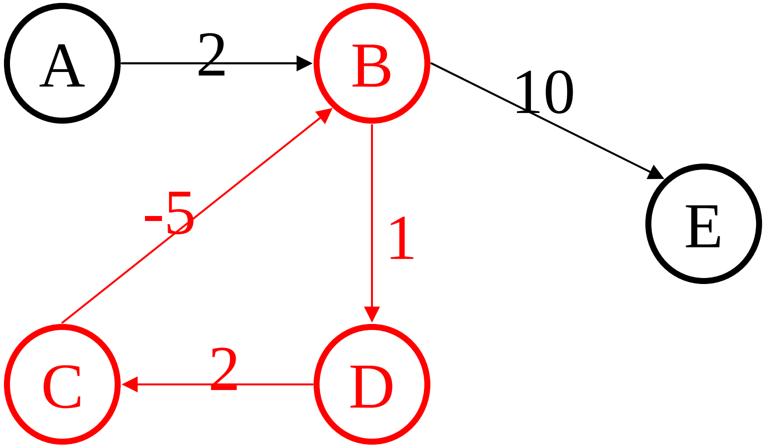
Correctness: Inside the Cloud

Prove by induction on # of nodes in the cloud:

- Initial cloud is just the source with shortest path 0
- <u>Assume</u>: Everything inside the cloud has the correct shortest path
- <u>Inductive step</u>: Only when we prove the shortest path to some node ***v*** (which is <u>*not*</u> in the cloud) is correct, we add it to the cloud

When does Dijkstra's algorithm not work?

The Trouble with Negative Weight Cycles



What’s the shortest path from A to E?

Problem?

Dijkstra's vs BFS

At each step:

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

Dijkstra's Algorithm

At each step:

1) Pick vertex from queue
2) Add it to visited vertices
3) Update queue with neighbors

Breadth-first Search

Some Similarities:

Single-Source Shortest Path

- Given a graph **G** = (V, E) and a single distinguished vertex **s**, find the shortest weighted path from **s** to every other vertex in **G**.

All-Pairs Shortest Path:

- Find the shortest paths between all pairs of vertices in the graph.
- How?

Analysis

- Total running time for Dijkstra's:

  $O(|V| \log |V| + |E| \log |V|)$ (heaps)

  What if we want to find the shortest path from each point to ALL other points?

Dynamic Programming

Algorithmic technique that systematically <u>records</u> the answers to sub-problems in a table and <u>re-uses</u> those recorded results (rather than re-computing them).

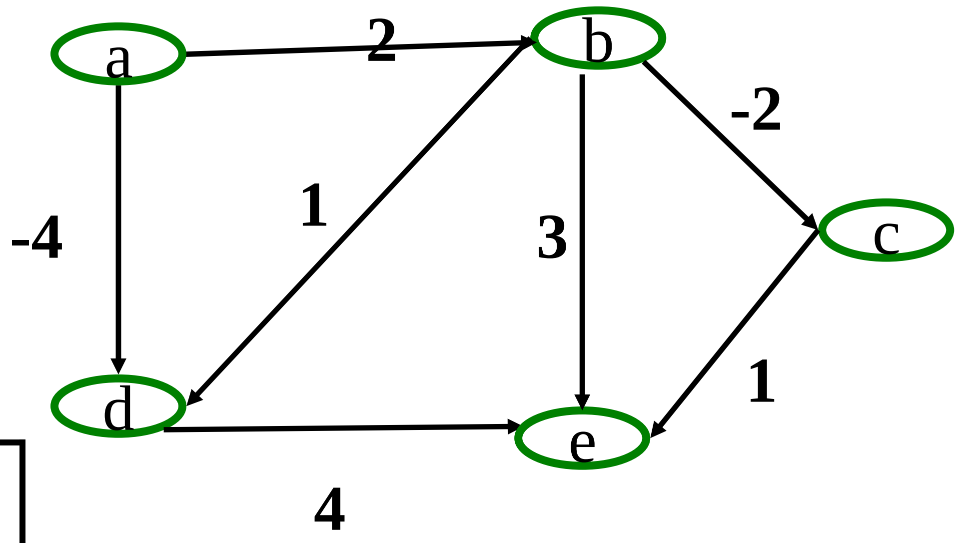
Simple Example: Calculating the Nth Fibonacci number.

$$Fib(N) = Fib(N-1) + Fib(N-2)$$

Floyd-Warshall

```
for (int k = 1; k =< V; k++)
 for (int i = 1; i =< V; i++)
  for (int j = 1; j =< V; j++)
   if ( ( M[i][k]+ M[k][j] ) < M[i][j] )
     M[i][j] =  M[i][k]+ M[k][j]
```

Invariant: After the kth iteration, the matrix includes the shortest paths for all pairs of vertices (i,j) containing only vertices 1..k as intermediate vertices

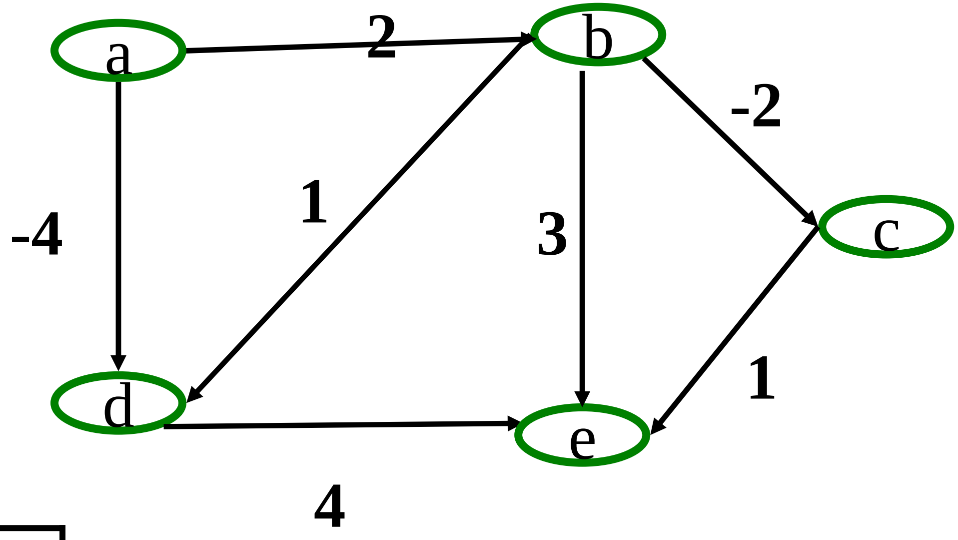


Initial state of the matrix:

	a	b	c	d	e
a	0	2	-	-4	-
b	-	0	-2	1	3
c	-	-	0	-	1
d	-	-	-	0	4
e	-	-	-	-	0

$$M[i][j] = \min(M[i][j], M[i][k] + M[k][j])$$

Floyd-Warshall - for All-pairs shortest path



	a	b	c	d	e
a	0	2	0	-4	0
b	-	0	-2	1	-1
c	-	-	0	-	1
d	-	-	-	0	4
e	-	-	-	-	0

Final Matrix Contents