CSE 326: Data Structures Shortest Path Problems

Ben Lerner

Spring 2008

Announcements (5/28/08)

- HW7 due Friday.
- Laura will not be holding office hours Friday 5/30 or Tuesday 6/3 but will be available by email.
- Reading for this lecture: Chapter 9.

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 $p = \Sigma_{i=0..k-1}\ c_{i,i+1}$ (a.k.a *cost*)

Path length equals path cost when ?

Single Source Shortest Paths (SSSP)

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

- Is this harder or easier than the previous problem?

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

Ideas?

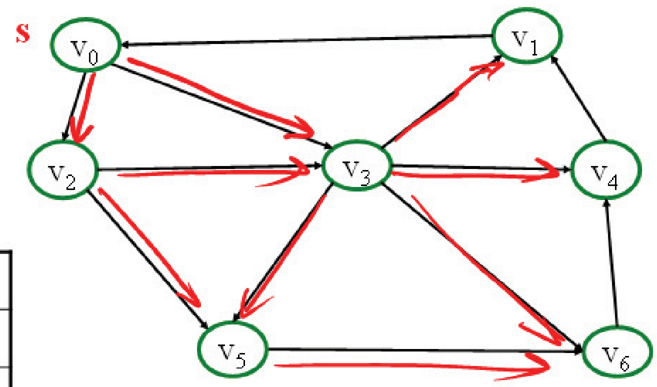

```
void Graph::unweighted (Vertex s){
  Queue q(NUM_VERTICES);
  Vertex v, w;
  q.enqueue(s);
  s.dist = 0;

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

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

each vertex enqueued at most once

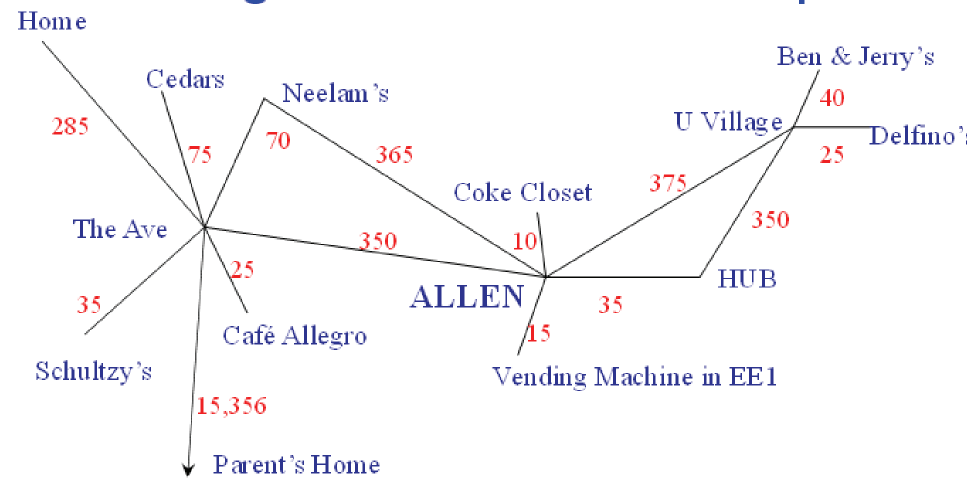
total running time: $O(|E| + |V|)$



V	Dist	path
v0	0	-
v1	2	v3
v2	1	v0
v3	1	v0
v4	2	v3
v5	2	v2
v6	2	v3

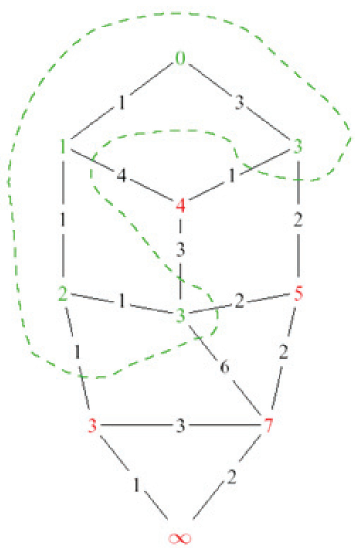
~~v0~~, ~~v2~~, ~~v3~~, ~~v5~~, ~~v1~~, ~~v4~~,
~~v6~~

Weighted SSSP: All edges are not created equal



Can we calculate shortest distance to all nodes from Allen Center?

Dijkstra's Algorithm: Idea

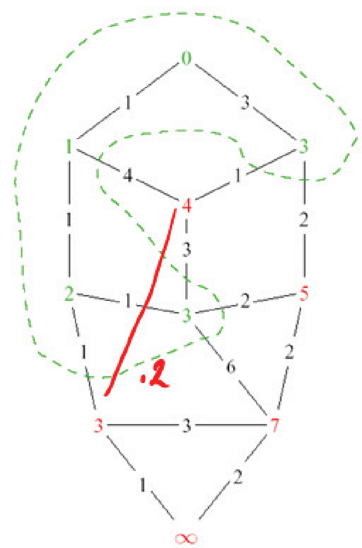


Adapt BFS to handle weighted graphs

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

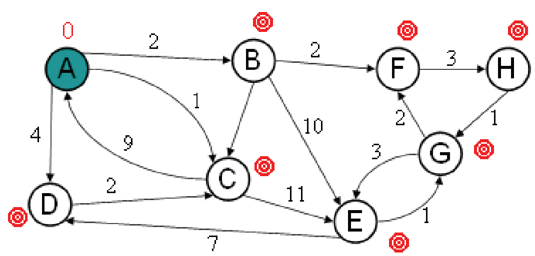
Initialize the cost of each node to ∞
Initialize the cost of the source to 0

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

Important Features

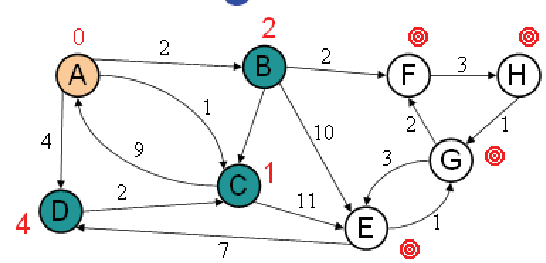
- Once a vertex is made **known**, the cost of the shortest path to that node is known
- While a vertex is still not **known**, another shorter path to it might still be found
- The shortest path itself can found by following the backward pointers stored at each node

Dijkstra's Algorithm in action



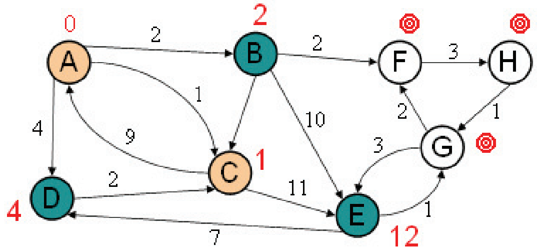
Vertex	Visited?	Cost	Found by
A		0	
B		??	
C		??	
D		??	
E		??	
F		??	
G		??	
H		??	

Dijkstra's Algorithm in action



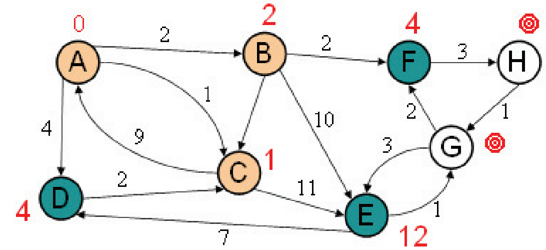
Vertex	Visited?	Cost	Found by
A	Y	0	
B		<=2	A
C		<=1	A
D		<=4	A
E		??	
F		??	
G		??	
H		??	

Dijkstra's Algorithm in action



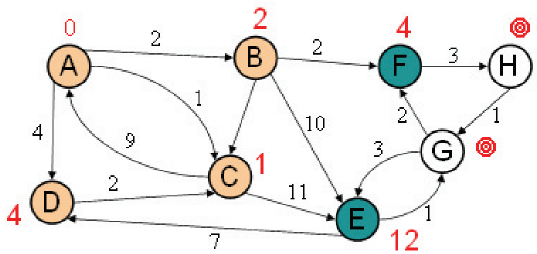
Vertex	Visited?	Cost	Found by
A	Y	0	
B		<=2	A
C	Y	1	A
D		<=4	A
E		<=12	C
F		??	
G		??	
H		??	

Dijkstra's Algorithm in action



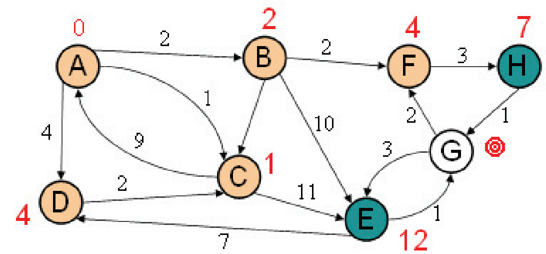
Vertex	Visited?	Cost	Found by
A	Y	0	
B	Y	2	A
C	Y	1	A
D		<=4	A
E		<=12	C
F		<=4	B
G		??	
H		??	

Dijkstra's Algorithm in action



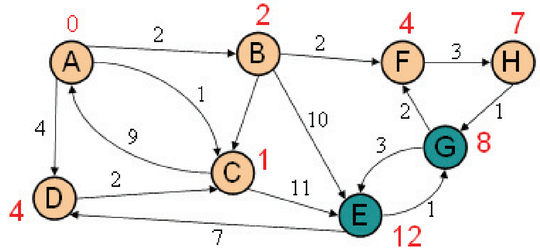
Vertex	Visited?	Cost	Found by
A	Y	0	
B	Y	2	A
C	Y	1	A
D	Y	4	A
E		<=12	C
F		<=4	B
G		??	
H		??	

Dijkstra's Algorithm in action



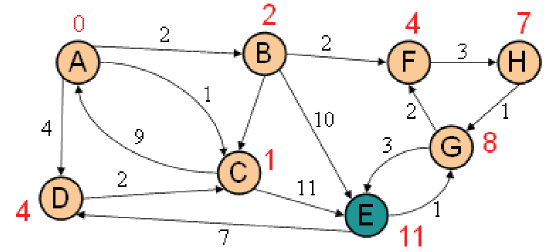
Vertex	Visited?	Cost	Found by
A	Y	0	
B	Y	2	A
C	Y	1	A
D	Y	4	A
E		<=12	C
F	Y	4	B
G		??	
H		<=7	F

Dijkstra's Algorithm in action



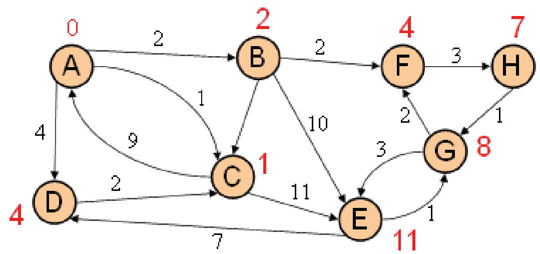
Vertex	Visited?	Cost	Found by
A	Y	0	
B	Y	2	A
C	Y	1	A
D	Y	4	A
E		<=12	C
F	Y	4	B
G		<=8	H
H	Y	7	F

Dijkstra's Algorithm in action



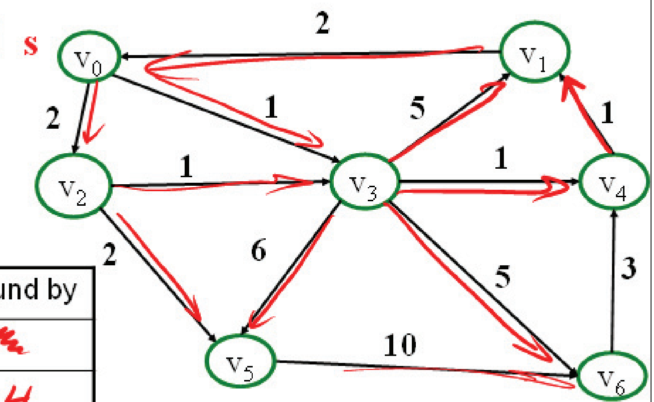
Vertex	Visited?	Cost	Found by
A	Y	0	
B	Y	2	A
C	Y	1	A
D	Y	4	A
E		<=11	G
F	Y	4	B
G	Y	8	H
H	Y	7	F

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	Y	0	[illegible]
v1	Y	3	v4
v2	Y	2	v0
v3	Y	1	v0
v4	Y	2	v3
v5	Y	4	v2
v6	Y	6	v3

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 *a* with the lowest cost
- Mark *a* as known
- For each node *b* adjacent to *a*
  - cost(*b*) = min(cost(*b*), cost(*a*) + cost(*a*, *b*))
  - previous(*b*) = ~~*b*~~ a (*if we updated b's cost*)

What data structures should we use?

priority q. + dictionary

Running time?

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

better for $|E| = \Theta(|V|^2)$ (dense) → No heap: $O(|V|^2+|E|)$ vs. Heap: $O(|V|\log|V| + |E| \log |V|)$

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

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

No heap: look at all vertices $|V|^2$ heap: $|V| \log |V|$

```
  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;
        }
  }
}
```

— No heap: $|E|$ Heap: $|E| \log |V|$

Sounds like deleteMin on a heap…

Sounds like adjacency lists

Sounds like decreaseKey

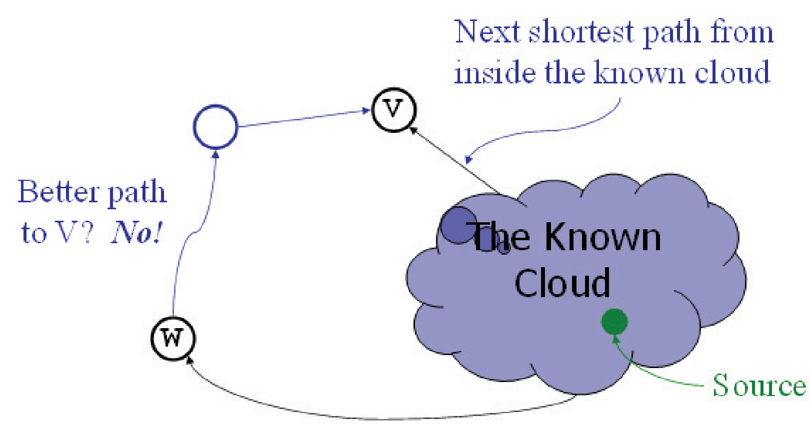
if graph is connected

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

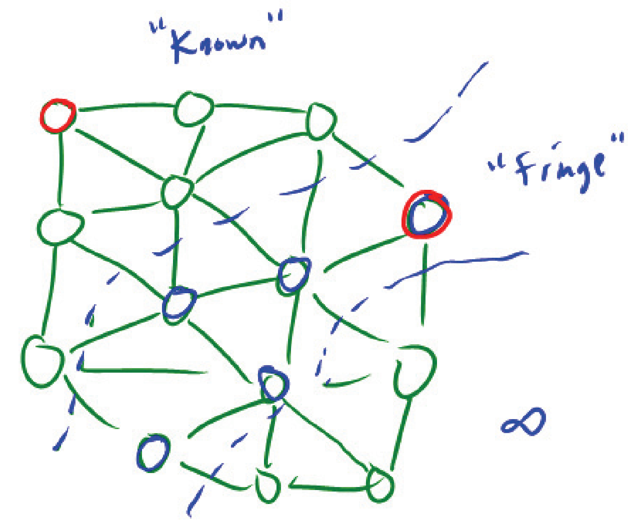
Prove by induction on # of nodes in the cloud:

Initial cloud is just the source with shortest path 0

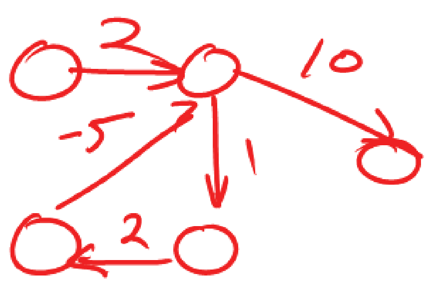
Assume: Everything inside the cloud has the correct shortest path

Inductive step: Only when we prove the shortest path to some node ***v*** (which is *not* 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

fringe = cost < ∞

At each step:

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

Breadth-first Search

fringe = queue

Some Similarities:

- known nodes
- new nodes "fringe"