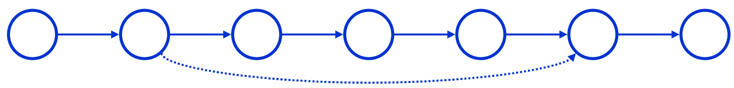
Single-Source Shortest Path

Single-Source Shortest Path

- Problem: given a weighted directed graph G, find the minimum-weight path from a given source vertex s to another vertex v
  - "Shortest-path" = minimum weight
  - Weight of path is sum of edges
  - E.g., a road map: what is the shortest path from Chapel Hill to Charlottesville?

Shortest Path Properties

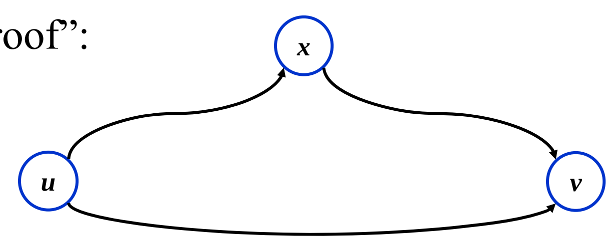
- Again, we have *optimal substructure*: the shortest path consists of shortest subpaths:



  - Proof: suppose some subpath is not a shortest path
    - There must then exist a shorter subpath
    - Could substitute the shorter subpath for a shorter path
    - But then overall path is not shortest path. Contradiction

Shortest Path Properties

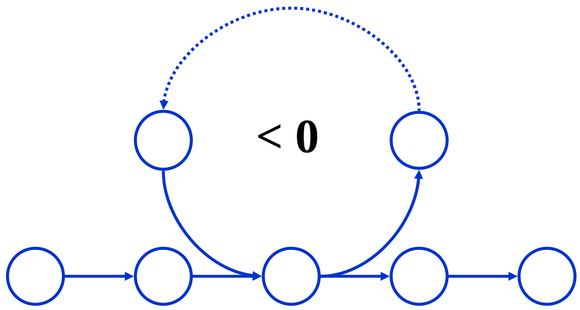
- Define $\delta(u,v)$ to be the weight of the shortest path from u to v
- Shortest paths satisfy the *triangle inequality*: $\delta(u,v) \le \delta(u,x) + \delta(x,v)$
- "Proof":



This path is no longer than any other path

Shortest Path Properties

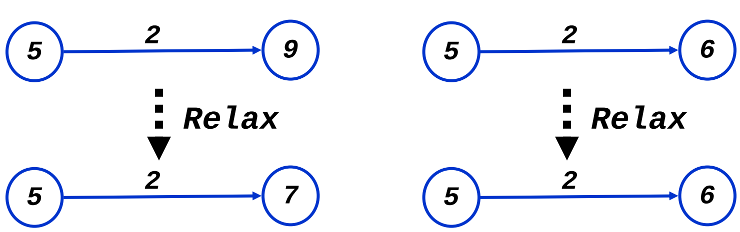
- In graphs with negative weight cycles, some shortest paths will not exist (*Why?*):



Relaxation

- A key technique in shortest path algorithms is *relaxation*
  - Idea: for all *v*, maintain upper bound d[*v*] on δ(s,v)

```
Relax(u,v,w) {
    if (d[v] > d[u]+w) then d[v]=d[u]+w;
}
```



Bellman-Ford Algorithm

```
BellmanFord()
   for each v ∈ V
      d[v] = ∞;
   d[s] = 0;
   for i=1 to |V|-1
      for each edge (u,v) ∈ E
         Relax(u,v, w(u,v));
   for each edge (u,v) ∈ E
      if (d[v] > d[u] + w(u,v))
           return "no solution";
```

Initialize d[], which will converge to shortest-path value δ

Relaxation: Make |V|-1 passes, relaxing each edge

Test for solution
Under what condition do we get a solution?

```
Relax(u,v,w): if (d[v] > d[u]+w) then d[v]=d[u]+w
```

Bellman-Ford Algorithm

```
BellmanFord()
   for each v ∈ V
      d[v] = ∞;
   d[s] = 0;
   for i=1 to |V|-1
      for each edge (u,v) ∈ E
         Relax(u,v, w(u,v));
   for each edge (u,v) ∈ E
      if (d[v] > d[u] + w(u,v))
           return "no solution";
```

What will be the running time?

```
Relax(u,v,w): if (d[v] > d[u]+w) then d[v]=d[u]+w
```

Bellman-Ford Algorithm

```
BellmanFord()
   for each v ∈ V
      d[v] = ∞;
   d[s] = 0;
   for i=1 to |V|-1
      for each edge (u,v) ∈ E
         Relax(u,v, w(u,v));
   for each edge (u,v) ∈ E
      if (d[v] > d[u] + w(u,v))
          return "no solution";
```

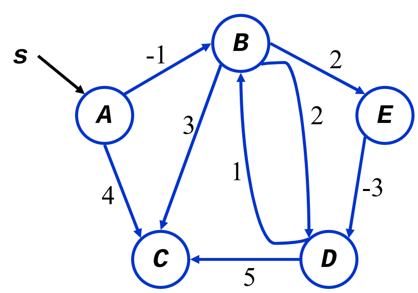
What will be the running time?

A: O(VE)

`Relax(u,v,w): if (d[v] > d[u]+w) then d[v]=d[u]+w`

Bellman-Ford Algorithm

```
BellmanFord()
   for each v ∈ V
      d[v] = ∞;
   d[s] = 0;
   for i=1 to |V|-1
      for each edge (u,v) ∈ E
         Relax(u,v, w(u,v));
   for each edge (u,v) ∈ E
      if (d[v] > d[u] + w(u,v))
          return "no solution";
```



Ex: work on board

`Relax(u,v,w): if (d[v] > d[u]+w) then d[v]=d[u]+w`

Bellman-Ford

- Note that order in which edges are processed affects how quickly it converges
- Correctness: show $d[v] = \delta(s,v)$ after $|V|$-1 passes
  - Lemma: $d[v] \geq \delta(s,v)$ always
    - Initially true
    - Let v be first vertex for which $d[v] < \delta(s,v)$
    - Let u be the vertex that caused d[v] to change: $d[v] = d[u] + w(u,v)$
    - Then $d[v] < \delta(s,v)$
      $\delta(s,v) \leq \delta(s,u) + w(u,v)$ (*Why?*)
      $\delta(s,u) + w(u,v) \leq d[u] + w(u,v)$ (*Why?*)
    - So $d[v] < d[u] + w(u,v)$. Contradiction.

Bellman-Ford

- Prove: after $|V|$-1 passes, all *d* values correct
  - Consider shortest path from s to v:
    $s \to v_1 \to v_2 \to v_3 \to v_4 \to v$
    - Initially, $d[s] = 0$ is correct, and doesn't change (*Why?*)
    - After 1 pass through edges, $d[v_1]$ is correct (*Why?*) and doesn't change
    - After 2 passes, $d[v_2]$ is correct and doesn't change
    - …
    - Terminates in $|V|$ - 1 passes: (*Why?*)
    - *What if it doesn't?*

DAG Shortest Paths

- Problem: finding shortest paths in DAG
  - Bellman-Ford takes O(VE) time.
  - *How can we do better?*
  - Idea: use topological sort
    - If were lucky and processes vertices on each shortest path from left to right, would be done in one pass
    - Every path in a dag is subsequence of topologically sorted vertex order, so processing verts in that order, we will do each path in forward order (will never relax edges out of vert before doing all edges into vert).
    - Thus: just one pass. *What will be the running time?*

Dijkstra's Algorithm

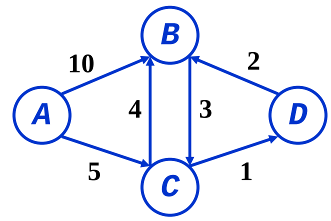
- If no negative edge weights, we can beat BF
- Similar to breadth-first search
  - Grow a tree gradually, advancing from vertices taken from a queue
- Also similar to Prim's algorithm for MST
  - Use a priority queue keyed on d[v]

Dijkstra's Algorithm

```
Dijkstra(G)
   for each v ∈ V
      d[v] = ∞;
   d[s] = 0; S = ∅; Q = V;
   while (Q ≠ ∅)
      u = ExtractMin(Q);
      S = S U {u};
      for each v ∈ u->Adj[]
         if (d[v] > d[u]+w(u,v))
            d[v] = d[u]+w(u,v);
```

Relaxation Step

Note: this is really a call to Q->DecreaseKey()



Ex: run the algorithm

Dijkstra's Algorithm

```
Dijkstra(G)
   for each v ∈ V
      d[v] = ∞;
   d[s] = 0; S = ∅; Q = V;
   while (Q ≠ ∅)
      u = ExtractMin(Q);
      S = S U {u};
      for each v ∈ u->Adj[]
         if (d[v] > d[u]+w(u,v))
            d[v] = d[u]+w(u,v);
```

How many times is ExtractMin() called?

How many times is DecraseKey() called?

What will be the total running time?

Dijkstra's Algorithm

```
Dijkstra(G)
   for each v ∈ V
      d[v] = ∞;
   d[s] = 0; S = ∅; Q = V;
   while (Q ≠ ∅)
      u = ExtractMin(Q);
      S = S U {u};
      for each v ∈ u->Adj[]
         if (d[v] > d[u]+w(u,v))
            d[v] = d[u]+w(u,v);
```

How many times is ExtractMin() called?

How many times is DecraseKey() called?

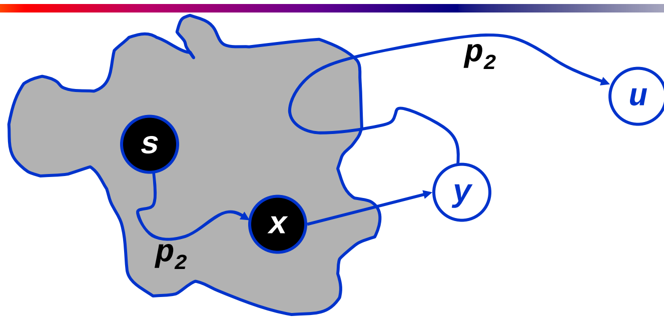
A: O(E lg V) using binary heap for Q
Can acheive O(V lg V + E) with Fibonacci heaps

Dijkstra's Algorithm

```
Dijkstra(G)
   for each v ∈ V
      d[v] = ∞;
   d[s] = 0; S = ∅; Q = V;
   while (Q ≠ ∅)
      u = ExtractMin(Q);
      S = S U{u};
      for each v ∈ u->Adj[]
         if (d[v] > d[u]+w(u,v))
            d[v] = d[u]+w(u,v);
```

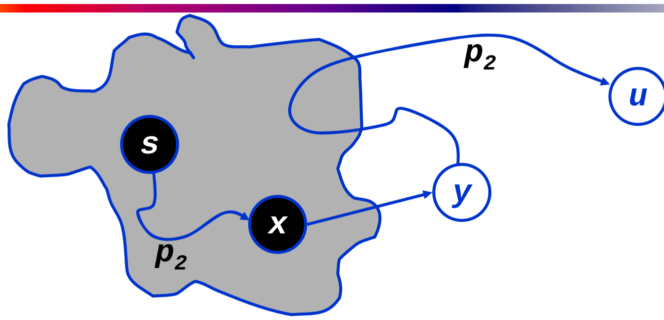
Correctness: we must show that when u is removed from Q, it has already converged

Correctness Of Dijkstra's Algorithm



- Note that d[v] ≥ δ(s,v) ∀v
- Let u be first vertex picked s.t. ∃ shorter path than d[u] ⇒d[u] > δ(s,u)
- Let y be first vertex ∈V-S on actual shortest path from s→u ⇒ d[y] = δ(s,y)
  - Because d[x] is set correctly for y's predecessor x ∈ S on the shortest path, and
  - When we put x into S, we relaxed (x,y), giving d[y] the correct value

Correctness Of Dijkstra's Algorithm



- Note that d[v] ≥ δ(s,v) ∀v
- Let u be first vertex picked s.t. ∃ shorter path than d[u] ⇒d[u] > δ(s,u)
- Let y be first vertex ∈V-S on actual shortest path from s→u ⇒ d[y] = δ(s,y)
- d[u] > δ(s,u)
  = δ(s,y) + δ(y,u) (*Why?*)
  = d[y] + δ(y,u)
  ≥ d[y] But if d[u] > d[y], wouldn't have chosen u. Contradiction.

The End