CMSC 451 - Algorithm Design
Lecture 4 - Shortest Paths - Dijkstra + Bellman-Ford

Problem: Given a digraph with numeric edge weights, compute shortest paths (s.p.'s)

Notation: $G = (V, E)$ - the digraph $n = |V|$, $m = |E|$

$w(u,v)$ - weight of edge $(u,v) \in E$

cost of a path = sum of edge weights

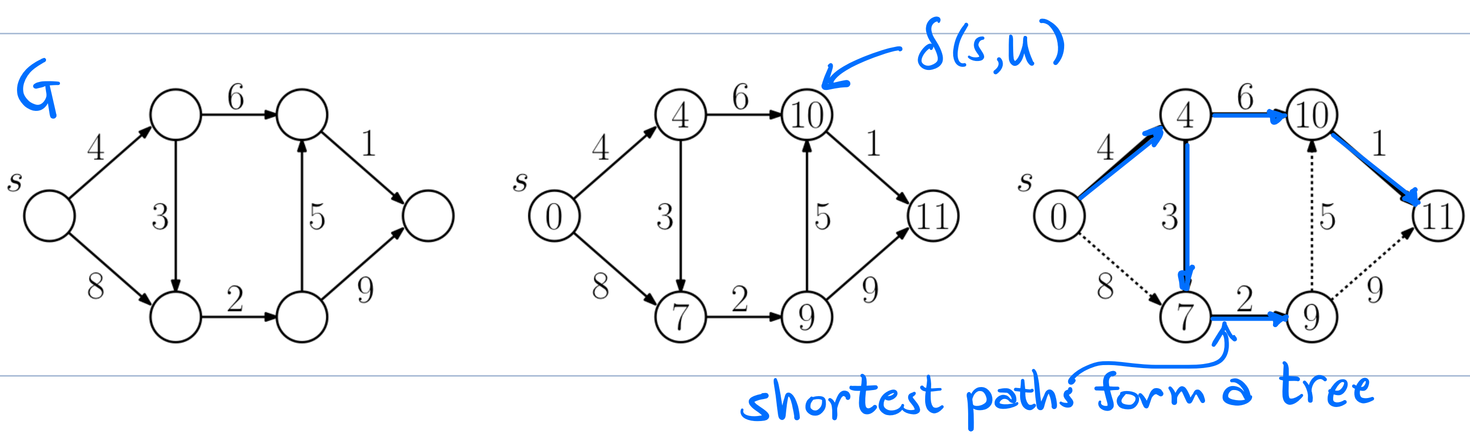
$u \xrightarrow{2} \circ \xrightarrow{-1} \circ \xrightarrow{5} v$ cost $= 2 - 1 + 5 = 6$

distance from u to v is minimum cost of any path from u to v (∞ if no path)

Denoted $\delta(u,v)$. $\delta(u,u) = 0$

single-source: for a given source $s \in V$, compute $\delta(s,u)$ for all $u \in V$

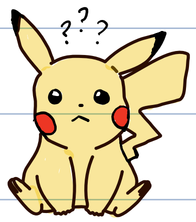
(Also, encode shortest path info.)



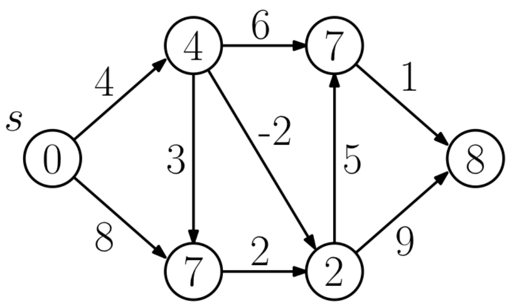
Preliminaries:

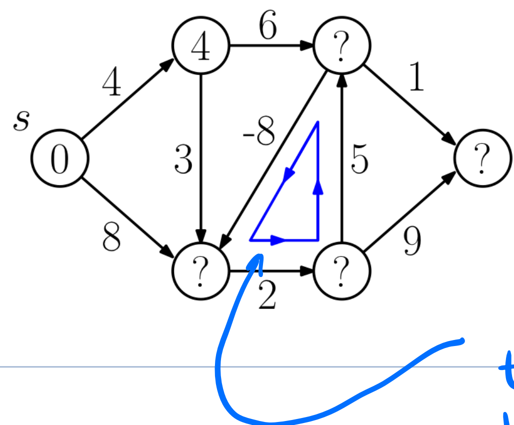
If edge weights are uniform (e.g. $w(u,v)=1, \forall (u,v)\in E$) fastest algorithm is breadth first search (BFS) $O(n+m)$ time.

Negative edge weights?



- Used, e.g., when edges model financial transactions - buy + sell
- Shortest paths are well defined ($\delta(s,u)$ may be negative) provided there are no negative-cost cycles.



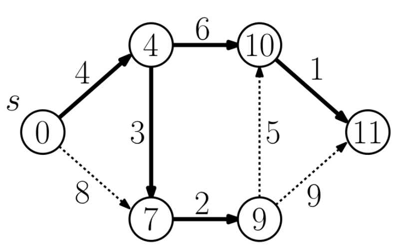


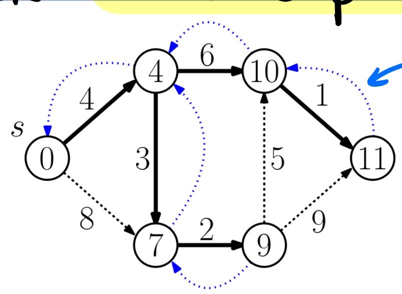
The more times you go around this loop, the lower the cost

$\delta(s,u) \to -\infty$

Paths?

pred[u] points to prior vertex on path from s to u. (pred[s] = null)
Follow these back = reverse path.





pred links

Dijkstra's Algorithm:

- Simple greedy algorithm for single-source s.p.
- Discovered in 1956 by Dutch comp. sci.
  Edsger Dijkstra
- The best from a worst-case perspective.
  More practical → A^*-search

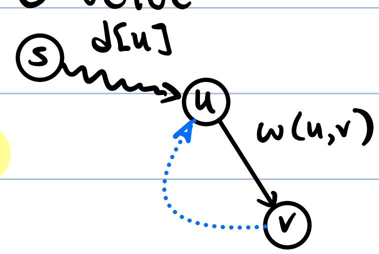
Basics: Maintains, for all $v \in V$

$d[v]$ = current distance estimate

There is a path $s \rightsquigarrow v$ of this cost,
but might not be shortest $d[v] \geq \delta(s,v)$

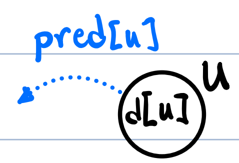
$pred[v]$ = predecessor based on d-value

If $u = pred[v]$ then
$d[v] = d[u] + w(u,v)$



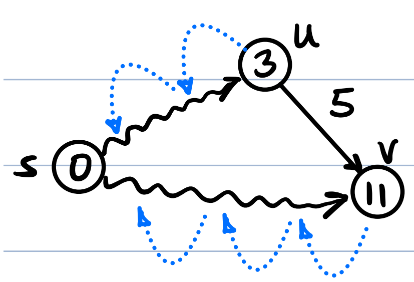
Relaxation:

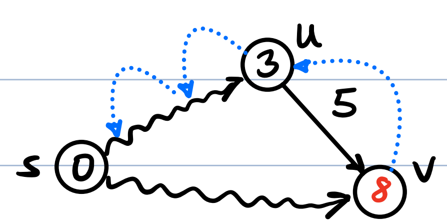
- Propagates shortest paths forward
  one edge at a time



relax(u,v):

if
$d[u] + w(u,v) < d[v]$
then
$d[v] \leftarrow d[u] + w(u,v)$
$pred[v] \leftarrow u$





Overview:

- **Init:** $d[s] \leftarrow 0$; $d[v] \leftarrow \infty$, o.w.
- **Repeat:**
  - Select unprocessed vertex with min d-value
  - Apply relax on all outgoing edges

Details:

- Store unprocessed vertices in a priority queue, sorted by d-values
- Priority queue operations:
  - Build initial – $O(n)$
  - Extract min – $O(\log n)$
  - Decrease key – $O(\log n)$

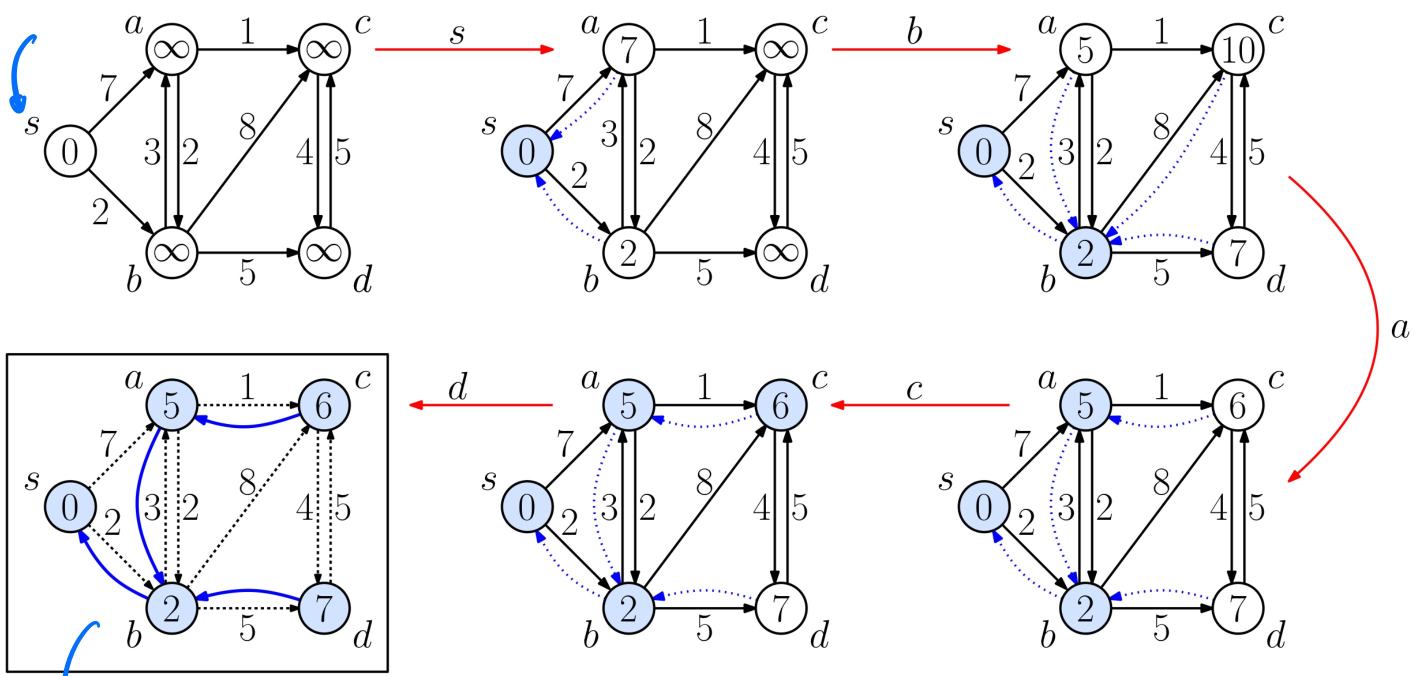
```
dijkstra ( G=(V,E), w, s )
    for each (v ∈ V)
        d[u] ← ∞; pred[u] ← null
    d[s] ← 0
    Q ← priority queue sorted by d-values
    while (Q is not empty)
        u ← Q.extractMin()
        for each (v ∈ Adj[u])
            relax(u,v) + update Q if necessary
    [pred links define an inverted s.p. tree]
```

weights

source

since d[v] may have changed

Example: source = s



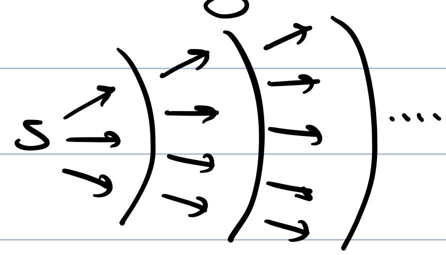
Pred links yield reverse of shortest path

E.g. c ⇢ a ⇢ b ⇢ s

⇒ path to c is s → b → a → c

Correctness:

Intuitively, Dijkstra's algorithm propagates costs forward in increasing distance order from s.

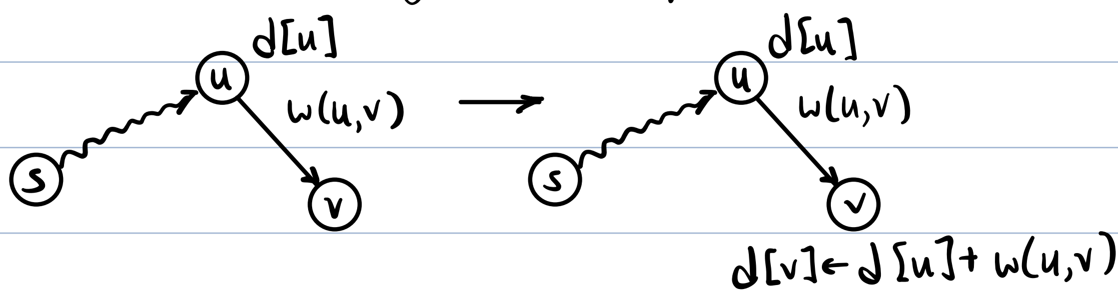


Safe because later vertices cannot affect distances to closer ones. (No neg. weight edges)

Correctness follows from the next lemma.
[for $u \in V$, $d[u]$ = estimate $\delta(s,u)$ = true dist.]

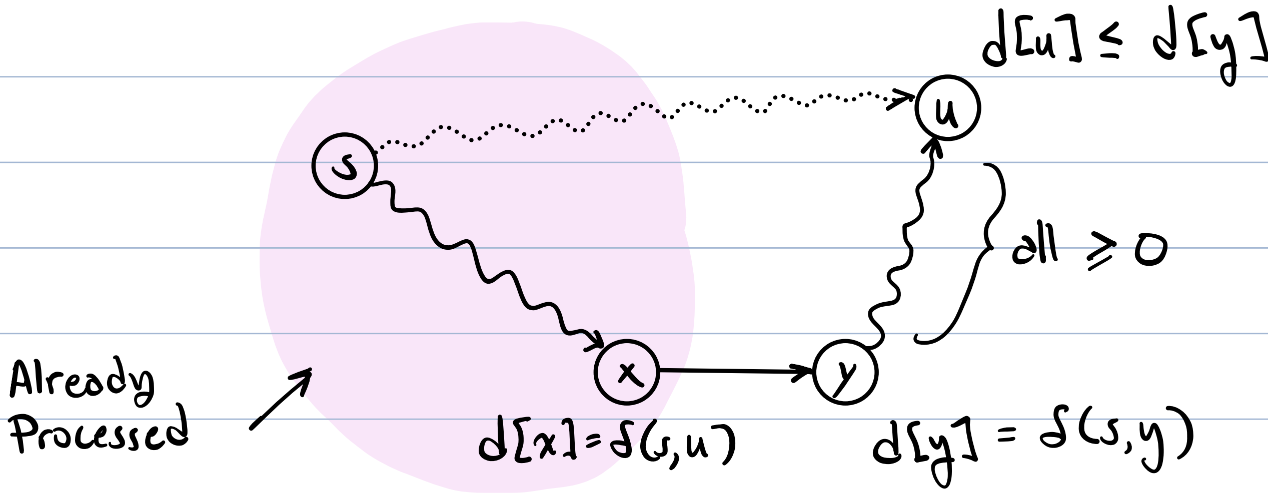
Lemma: For all $u \in V$,
(1) If $d[u] \neq \infty$, there exists a path of this cost
(2) After u is processed, $d[u] = \delta(s,u)$

Proof:
(1) Follows by induction + fact that d-values are defined by relax op.



(2) Suppose not. Consider the first vertex u, where $d[u] \neq \delta(s,u)$ after processing u. By (1), $d[u] > \delta(s,u)$ ⓐ

- Let S be the set of processed vertices prior to u.
- The true shortest path $s \rightsquigarrow u$ must first jump outside of S - let (x,y) be this edge. (possibly $x=s$, $y=u$)



- Since no errors up to now

  $$d[x] = \delta(s,x) \quad (b)$$

- Since x was processed, relax(x,y) sets

  $$\begin{aligned} d[y] &\leftarrow d[x] + w(x,y) \\ &\overset{(b)}{=} \delta(s,x) + w(x,y) \\ &= \delta(s,y) \quad (c) \end{aligned}$$

  [since this is the true shortest path]

- Since u is processed next, we know

  $$d[u] \leq d[y] \quad (d)$$

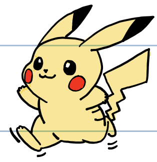
- Since edge weights ≥ 0, $\delta(s,u) \geq \delta(s,y)$ (e)
- Putting this together:

$$\delta(s,u) \overset{(a)}{<} d[u] \overset{(d)}{\leq} d[y] \overset{(c)}{=} \delta(s,y) \overset{(e)}{\leq} \delta(s,u)$$

contradiction!

QED.

Running time: $n = |V|$ $m = |E|$



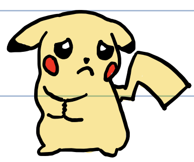
- Outer loop – n times
  - extract min – $\log n$ time
- relax – once for each edge – m times
  - update key value – $\log n$ time
  - Fibonacci heap – $O(1)$ amortized

Total: $O(n \log n + m \log n)$

Fibonacci heap: $O(n \log n + m)$

Dijkstra assumes edge weights are ≥ 0.

What if not? (But no neg. cost cycles)



Give an example that show that Dijkstra's algorithm fails (incorrect final d-value) if even one edge cost < 0.

Bellman-Ford Algorithm -

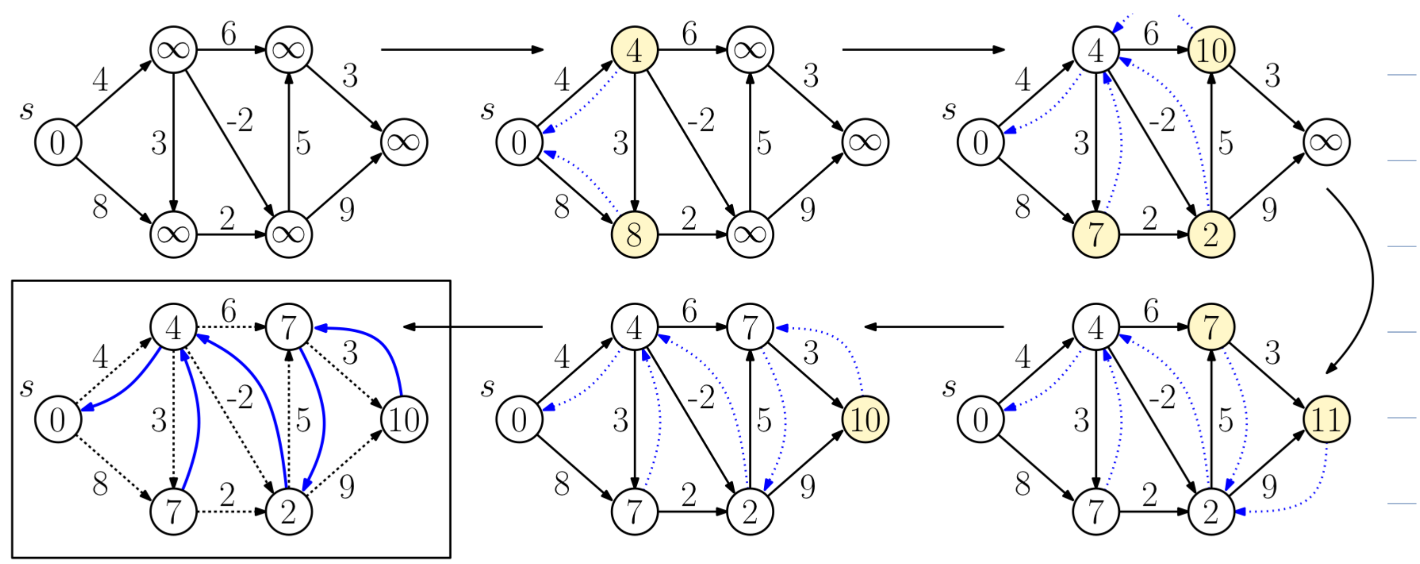
- Solves the single source shortest path prob. for arbitrary edge weights - no neg. cost cycles.
- Invented 1955 (pre-dates Dijkstra!)
- Super simple, but slower than Dijkstra - $O(n \cdot m)$ vs. $O(m + n\log n)$

Basic idea:

- Relax op. propagates distances forward
- Rather than being clever, just repeatedly apply to all edges

Example:

- In each phase apply relax(u,v), $\forall$ edges (u,v)



```
bellman-ford (G=(V,E), w, s)      -> source
                                  -> weights
    for each (v ∈ V)
        d[u] ← ∞ ; pred[u] ← null      -> init
    d[s] ← 0
    repeat          // relax all edges until converge
        converged ← true
        for each ((u,v) ∈ E)
            relax(u,v)
            if d[v] changed
                converged ← false
    until (converged)
    [pred links define an inverted s.p. tree]
```

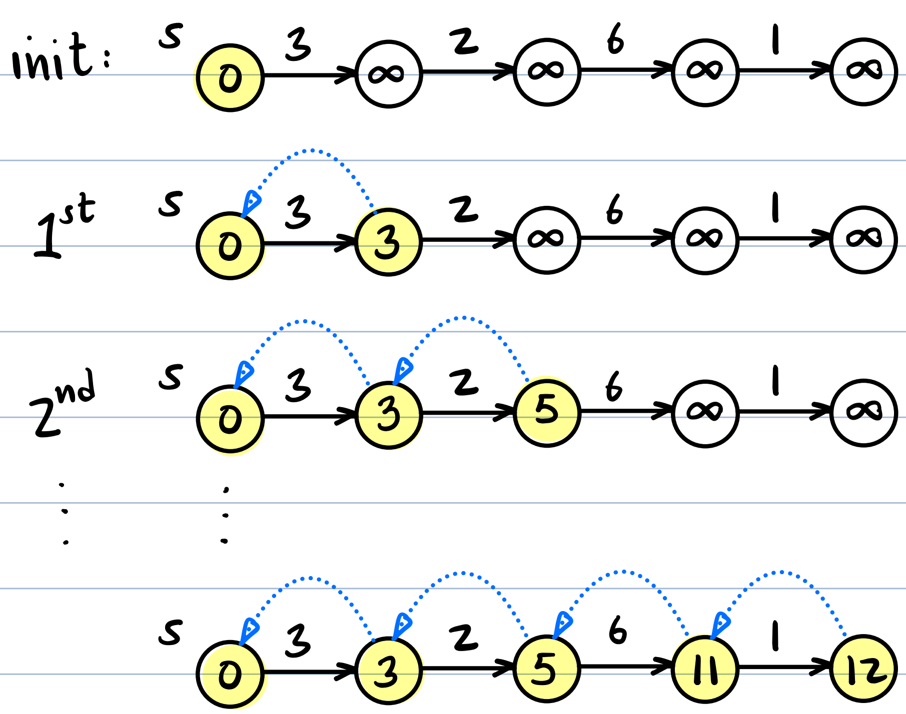
Running time: $n = |V|$ $m = |E|$

- Each repeat loop takes $O(m)$ time
- Will show convergence within $n-1$ iterations
- Total time: $O(nm)$

Correctness:

Key - Consider any shortest path

- Each iteration of the repeat loop propagates distances one more edge



- By induction: After k^{th} iteration, all vertices whose shortest path has $\leq k$ edges have $d[u] = \delta(s,u)$

- If no neg. cost cycles, any shortest path has $\leq n-1$ edges (no repeats)
  $\Rightarrow$ Bellman-Ford converges with correct distances in $\leq n-1$ iterations.

Summary:

Single-Source Shortest Paths in Digraphs
- Nonnegative Weights
  Dijkstra – $O(n \log n + m)$
- Neg. weights (no neg. cycles)
  Bellman-Ford – $O(n \cdot m)$