Lecture 16: April 12, 2001

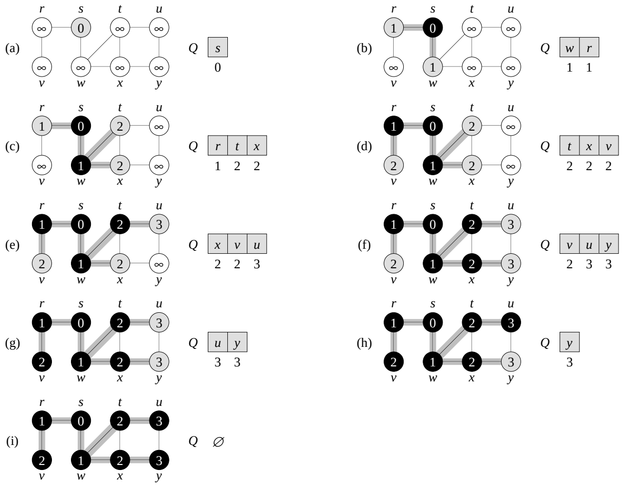
Today's Lecture: Graph Algorithms II

- Breadth-first search
- Single-source shortest paths
- Bellman-Ford Algorithm
- Dijkstra's Algorithm

Breadth-First Search

```
BFS($G, s$)
for each vertex $u \in V[G] - \{s\}$
        do $color[u] \leftarrow$ WHITE
           $d[u] \leftarrow \infty$
$color[s] \leftarrow$ GRAY
$d[s] \leftarrow 0$
$Q \leftarrow \{s\}$
while $Q \neq \emptyset$
        do $u \leftarrow head[Q]$
           for each $v \in Adj[u]$
                do if $color[v] =$ WHITE
                      then $color[v] \leftarrow$ GRAY
                           $d[v] \leftarrow d[u] + 1$
                           ENQUEUE($Q, v$)
           DEQUEUE($Q$)
           $color[u] \leftarrow$ BLACK
```

BFS Example



Shortest Paths

Digraph $G = (V, E)$ with weight function $W : E \rightarrow \Re$

Weight of path $p = v_1 \rightarrow v_2 \rightarrow \cdots \rightarrow v_k$ is:

$$w(p) = \sum_{i=1}^{k-1} w(v_i, v_{i+1})$$

"Shortest" path = path of minimum weight.

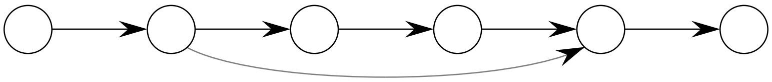
Applications

- Static/dynamic network routing
- Robot motion planning

Optimal Substructure

Theorem: Subpaths of shortest paths are shortest paths

Proof: Cut and paste:



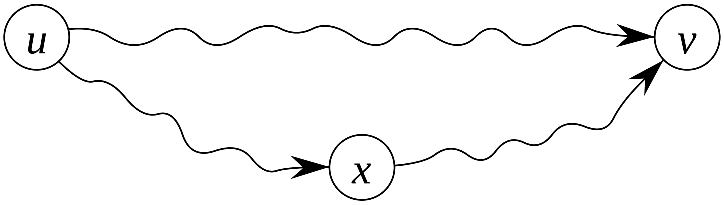
If some subpath were **not** a shortest path, could substitute the shorter subpath and create a shorter total path. □

Triangle Inequality

Definition: $\delta(u, v) \equiv$ weight of a shortest path from u to v.

Theorem: $\delta(u, v) \leq \delta(u, x) + \delta(x, v)$

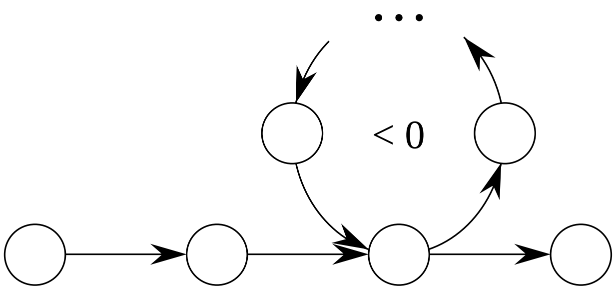
Proof: Shortest path $u \rightsquigarrow v$ is no longer than any other path $u \rightsquigarrow v$ — in particular, the path concatenating the shortest path $u \rightsquigarrow x$ with the shortest path $x \rightsquigarrow v$. □



Is shortest-path well-defined?

Negative weight cycle ⇒ no shortest path.

Argument: Can shorten path by traversing cycle. □



Bellman-Ford Algorithm

Most basic “single-source” shortest-**paths** algorithm

- Finds shortest path weights from specified **source** s to **all** $v \in V$
- Maintains estimate $d[v]$ of path length from s to v, which is updated iteratively
- Actual paths easily reconstructed (CLR §25.3)

Bellman-Ford Algorithm

```
Bellman-Ford($G, w, s$)
 1 for each $v \in V$
 2       do $d[v] \leftarrow \infty$
 3 $d[s] \leftarrow 0$    ▷ Initialize-Single-Source($G, s$

 4 for $i \leftarrow 1$ to $|V| - 1$
 5       do for each edge $(u, v) \in E$ ▷ Relax
 6                do if $d[v] > d[u] + w(u, v)$
 7                      then $d[v] \leftarrow d[u] + w(u, v)$

 8 for each edge $(u, v) \in E$
 9       do if $d[v] > d[u] + w(u, v)$
10             then no solution
```

Bellman-Ford Algorithm

Three code sections:

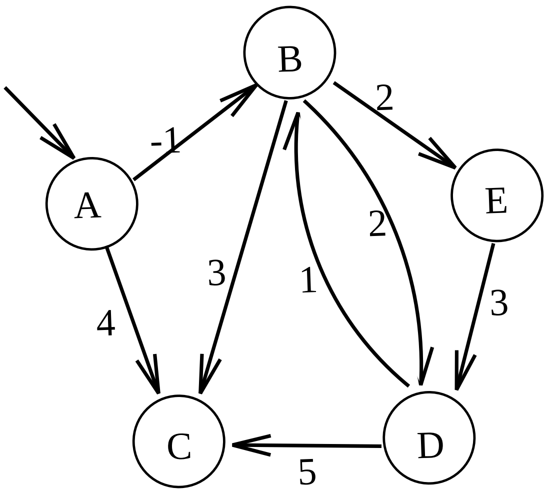
- Lines 1 – 3:
  Initialize: $d[v]$, which will converge to shortest-path values δ.

- Lines 4 – 7:
  Relax: $|V| - 1$ times, do **relaxation step** for each edge.

- Lines 8 – 10:
  Test: was a solution achieved (iff no negative-weight cycles)?

Bellman-Ford Algorithm

Example:



Bellman-Ford Algorithm: Running time

Running Time: $O(V \cdot E)$

- constants are good
- it is simple
- short code

very practical.

Bellman-Ford Algorithm

Example:

- **Initialization.** Put initial d values in nodes: $A \leftarrow 0$, rest $\leftarrow \infty$.
- **1st relaxation pass.** Process edges in order (A,B), (A,C), (B,C), (B,D), (D,B), (D,C), (E,D), (B,E).
- **2nd relaxation pass.** Process edges in same order. Only D changes.

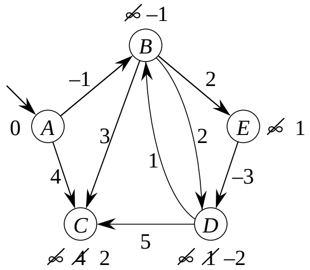
Bellman-Ford Algorithm

Example:

- Can stop when no change is detected

d	A	B	C	D	E
init	0	∞	∞	∞	∞
pass 1	0	-1	2	1	1
pass 2	0	-1	2	-2	1
pass 3	0	-1	2	-2	1

- Distances each pass, and convergence speed, depend on edge processing order.



Bellman-Ford Algorithm: Lemma

Lemma: $d[v] \geq \delta(s,v)$ always.

Proof:

- Initially true
- Let v be first vertex for which $d[v] < \delta(s,v)$, and let u be vertex that caused $d[v]$ to change:

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

- Then

$$\begin{aligned} d[v] &< \delta(s,v) \\ &\leq \delta(s,u) + \delta(u,v) && \text{(Triangle inequality)} \\ &\leq \delta(s,u) + w(u,v) && \text{(shortest path} \leq \text{specifi} \\ &\leq d[u] + w(u,v) && (v \text{ is first violation)} \end{aligned}$$

contradicts $d[v] = d[u] + w(u,v)$ (above).

Once $d[v]$ reaches $\delta(s,v)$, can't change. Why?

Bellman-Ford Algorithm: Correctness

Claim: Bellman-Ford correct (i.e., after $|V| - 1$ passes, all the d values are correct)

Proof: Let v be a vertex, and consider shortest path from s to v (assuming no neg-weight cycles):

$$s \rightarrow v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow v_4 \rightarrow v$$

- Initially, $d[s] = 0$ is correct (and doesn't change thereafter — code never increases d)

Bellman-Ford Algorithm: Correctness

Proof: *(continued)*

- After 1 pass thru edges, $d[v_1]$ is correct
  (and doesn't change...)

  ($d[s]$ is correct, and by optimal substructure, shortest distance is $w(s, v_1)$.
  1st pass sets $d[v_1] = d[s] + w(s, v_1)$, which is right answer.)

- After 2 passes through edges, $d[v_2]$ is correct (and doesn't change...)

⋮

⋮

- Terminates in $|V| - 1$ passes. Why?
  If no negative-weight cycles:
  – every shortest path is **simple** (no cycles)
  – longest simple path has $|V| - 1$ edges

Bellman-Ford Algorithm: Correctness

Proof: *(continued)*

- Thus if no neg-weight cycles, all the $d[v]$ converge in $|V| - 1$ passes.
  Equivalently, if a value $d[v]$ fails to converge after $|V| - 1$ passes, $\exists$ neg-weight cycle.
- Last part of algorithm tests for success by seeing if another pass would change anything.

The converse is also true:
If $\exists$ neg-weight cycle reachable from s, a value $d[v]$ fails to converge after $|V| - 1$ passes.

(Proof left as exercise.) *(CLR Theorem 25.14.)*

So... Bellman-Ford can be used to check for negative-weight cycles.

Dijkstra's Algorithm

Dijkstra's Algorithm:

- Non-negative edge weights ⇒ shortest paths always exist
  (If no weights negative, can beat Bellman-Ford)
- Like breadth-first-search
  (If all weights = 1, use BFS, otherwise Dijkstra.)
- Use Q, priority queue keyed by $d[v]$.
  Greedy, like Prim's algorithm for MST
  BFS used FIFO queue

Dijkstra's Algorithm: Pseudocode

```
DIJKSTRA$(G, w, s)$
 1  for each $v \in V$
 2        do $d[v] \leftarrow \infty$
 3  $d[s] \leftarrow 0$
 4  $S \leftarrow \emptyset$
 5  $Q \leftarrow V$
 6  while $Q \neq \emptyset$
 7        do $u \leftarrow$ EXTRACT-MIN$(Q)$
 8           $S \leftarrow S \cup \{u\}$
 9           for each $v \in Adj[u]$
10                do if $d[v] > d[u] + w(u, v)$
11                      then $d[v] \leftarrow d[u] + w(u, v)$
```

What is line 7 doing?
What is line 11 doing?

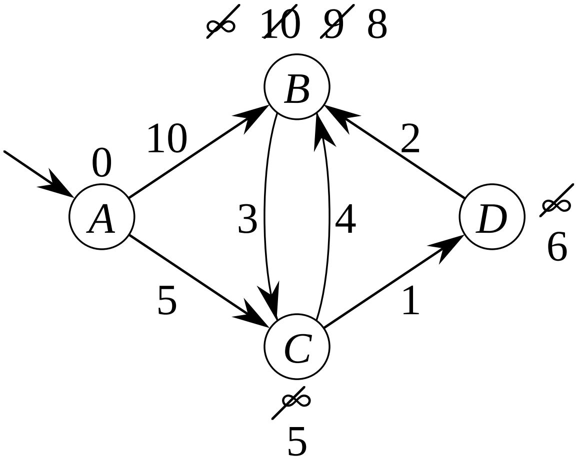
Dijkstra's Algorithm: Notes

Observe:

- relaxation step
- setting $d[v]$ updates Q (DECREASE-KEY)
- similar to Prim's minimum-spanning-tree algorithm

Dijkstra's Algorithm

Example:



Dijkstra's Algorithm: Analysis

Run Time Analysis:

- EXTRACT-MIN executed $|V|$ times
- DECREASE-KEY executed $|E|$ times

$$\text{Time} = |V| \cdot T_{\text{EXTRACT-MIN}} + |E| \cdot T_{\text{DECREASE-KEY}}$$

Dijkstra's Algorithm: Analysis

Time = $|V| \cdot T_{\textsc{Extract-Min}} + |E| \cdot T_{\textsc{Decrease-Key}}$

Analysis: Look at different Q implementations.

Q	$T_{\textsc{Extract-Min}}$	$T_{\textsc{Decrease-Key}}$	Total
array	$O(V)$	$O(1)$	$O(V^2)$
binary heap	$O(\lg V)$	$O(\lg V)$	$O(E \lg V)$
Fibonacci heap	$O(\lg V)$	$O(1)$	$O(V \lg V + E)$
	amortized	amortized	worst case

Dijkstra's Algorithm: Analysis

- Q = unsorted array:

  scan to find minimum
  just index and update to change key

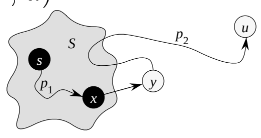
- Q = Fibonacci heap

  Note advantage of amortized analysis:
  Can use amortized Fibonacci heap bounds in analysis, as if they were worst-case bounds, and get (real) worst-case bounds on aggregate running time.

Dijkstra's Algorithm: Correctness

Correctness: Prove that whenever u is added to S, $d[u] = \delta(s, u)$

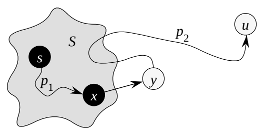
Proof:



- Note that $\forall\ v\ \ d[v] \geq \delta(s, v)$
- Let u be first vertex picked such that $\exists$ shorter path than $d[u]$
  $\Rightarrow d[u] > \delta(s, u)$

Dijkstra's Algorithm: Correctness

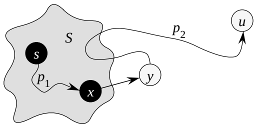
(Proof continued)



- Let y be first vertex $\in V - S$ on actual shortest path from s to u
  $\Rightarrow d[y] = \delta(s, y)$
  Because:
  - $d[x]$ is set correctly for y's predecessor $x \in S$ on the shortest path (by choice of u as first choice for which that's not true)
  - when put x into S, relaxed (x, y), giving $d[y]$ correct value

Dijkstra's Algorithm: Correctness

(Proof continued)



$$\begin{aligned} d[u] &> \delta(s,u) \\ &= \delta(s,y) + \delta(y,u) && \text{(optimal substructure)} \\ &= d[y] + \delta(y,u) \\ &\geq d[y] && \text{(no negative weights)} \end{aligned}$$

- But $d[u] > d[y] \Rightarrow$ algorithm would have chosen y to process next, not u. *Contradiction.*