CS4800: Algorithms & Data
Jonathan Ullman

Lecture 15:

- Bellman-Ford Shortest Paths
- Negative Cycle Detection
- All pairs shortest paths (Floyd-Warshall)

Mar 2, 2018

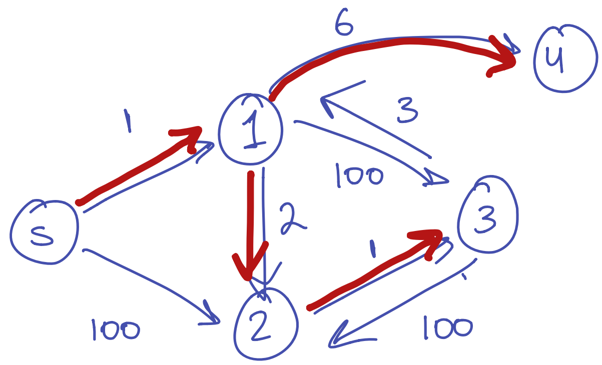
Shortest Paths with Negative Edges

Dijkstra Recap

- Input:
  - Directed, graph $G = (V, E, \{\ell_e\})$
    - **Non-negative edge lengths** $\ell_e \geq 0$
  - Source node s

- Output:
  - Arrays d, p
    - $d(v)$ is the length of the shortest $s - v$ path
    - $p(v)$ is the final hop on the shortest $s - v$ path

- Running time $O(m \log n)$ (Implement using heaps)



$d(1) = 1$ $\quad$ $p(1) = s$

$d(2) = 3$ $\quad$ $p(2) = 1$

$d(3) = 4$ $\quad$ $p(3) = 2$

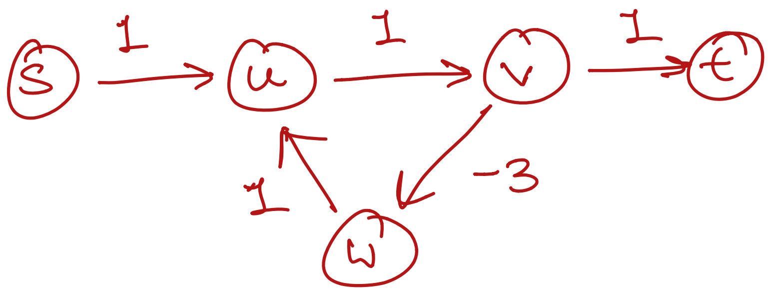
$d(4) = 7$ $\quad$ $p(4) = 1$

- Red edges are all edges $(p(v), v)$ $v \in V$
- Red edges form a tree
- The unique $s \to v$ path using red edges a shortest $s \to v$ path in G.

Ask the Audience

- Does Dijkstra's algorithm still solve shortest paths in graphs with negative edge lengths?

- Negative Cycle



go around

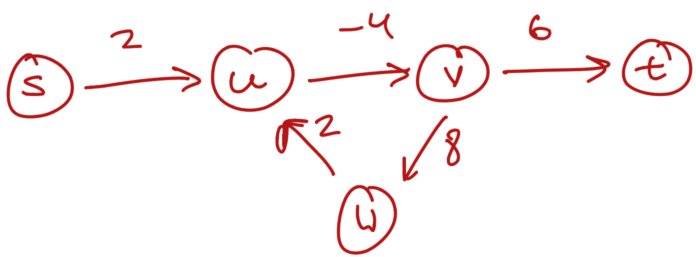
$u \to v \to w \to u$

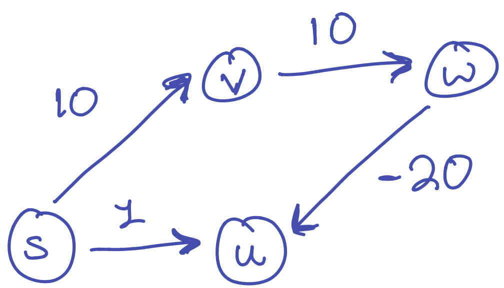
cycle ∞ times the length

of the path is $-\infty$

Ask the Audience

- Does Dijkstra's algorithm still solve shortest paths in graphs with negative edge lengths? ... but no negative-length cycles

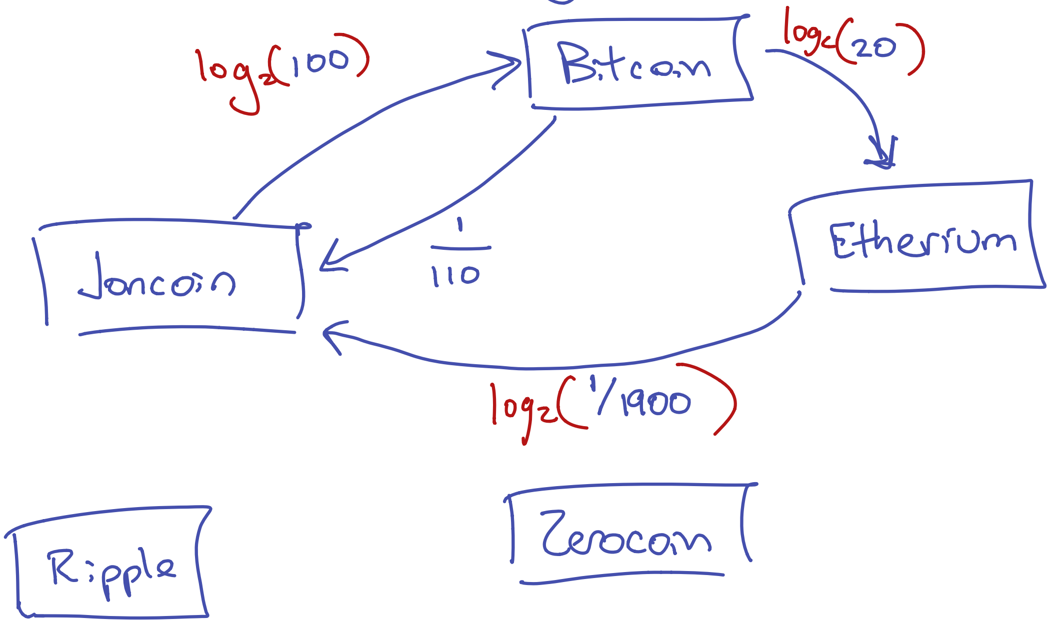




Dijkstra would explore u first but would not have found the shortest path from s to u.

Why Care About Negative Lengths?

Suppose you're arbitraging cryptocurrencies



Negative weight cycle represents an arbitrage opportunity

Algorithms for shortest paths w/ negative edge weights are "robust."

- Routing algorithms that have to handle changing graphs.

Shortest Paths with Negative Lengths

- Input:
  - Directed, graph $G = (V, E, \{\ell_e\})$
    - **Possibly negative edge lengths** ℓ_e $\in \mathbb{R}$
    - **No negative length cycles**
  - Source node s

- Output:
  - Arrays d, p
    - $d(v)$ is the length of the shortest $s - v$ path
    - $p(v)$ is the final hop on the shortest $s - v$ path

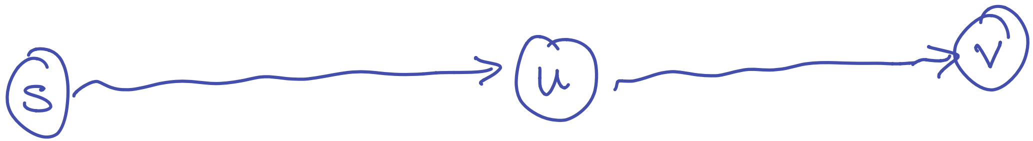
Minimum Cycle Detection

- Input:
  - Directed, graph $G = (V, E, \{\ell_e\})$
    - **Possibly negative edge lengths** ℓ_e
- Output:
  - A negative-length cycle C if one exists

Ask the Audience

- $G = (V, E, \{\ell_e\})$ is a graph with negative lengths
- G' is the same but we add $(\min \ell_e)$ to every length
- Why doesn't it work to run Dijkstra on G'?

Structure of Shortest Paths



<u>Fact:</u> If the shortest path from s to v passes through u, then $s \rightsquigarrow v = s \rightsquigarrow u \rightsquigarrow v$ where $s \rightsquigarrow u$ and $u \rightsquigarrow v$ are shortest paths

$$d(s,v) = d(s,u) + d(u,v)$$

If $(u,v) \in E$, then $d(s,v) \leq d(s,u) + \ell_{u,v}$

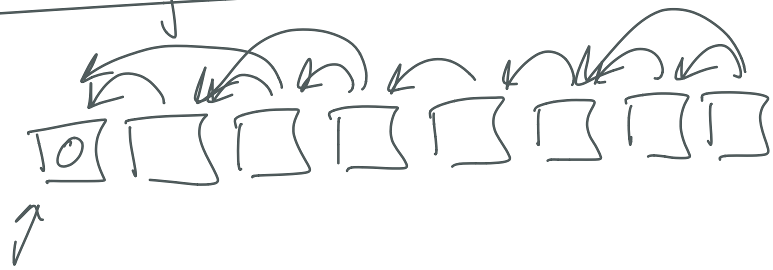
Dynamic Programming

- Consider the shortest $s \to v$ path
- The path must be $s \rightsquigarrow u \to v$ for some u
- If I knew u then $d(s,v) = d(s,u) + \ell_{u,v}$
- If $OPT(v)$ = the length of the shortest $s \to v$ path

$$OPT(s) = 0$$

$$OPT(v) = \min_{\substack{u: \\ (u,v) \in E}} \left\{ OPT(u) + \ell_{u,v} \right\}$$

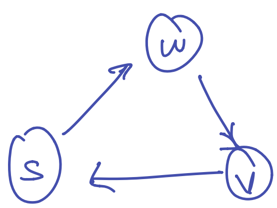
What Goes Wrong?

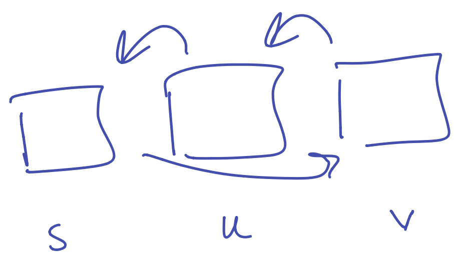


Base Case

In order to implement bottom-up DP there needs to be an ordering to fill the table

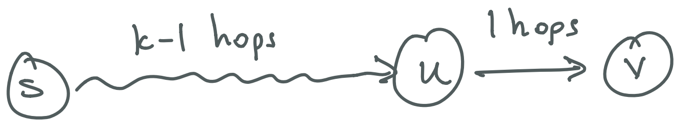
If each subproblem is OPT(v) and G has cycles then there is no ordering of the nodes





Dynamic Programming Take II

- Consider the shortest $s \to v$ path
  - the last edge is some (u, v)
  - it makes k hops for some k



- Let $OPT(v, i)$ be the length of the shortest $s \to v$ path making $\leq i$ hops
- $OPT(s, i) = 0$
- $OPT(v, 0) = \infty \quad \forall v \neq s$
- $$OPT(v, i) = \min \left\{ \begin{array}{l} OPT(v, i-1) \\ \min\limits_{(u,v) \in E} OPT(u, i-1) + l_{u,v} \end{array} \right\}$$

Recurrence

- $OPT(v, i)$ is the length of the shortest path from s to v that uses at most i hops
- Want to compute $\underbrace{OPT(v, n-1)}$ for all v

The shortest path has no cycles
$\Rightarrow$ it has $\leq n-1$ hops

- $\forall i \; OPT(s, i) = 0$
- $\forall v \neq s \; OPT(v, 0) = \infty$

$$OPT(v, i) = \min\left\{OPT(v, i-1), \min_{w \in V}\{OPT(w, i-1) + \ell_{w,v}\}\right\}$$

Finding the paths

- $OPT(v, i)$ is the length of the shortest path from s to v that uses at most i hops
- $P(v, i)$ is the last hop on the shortest path from s to v that uses at most i hops

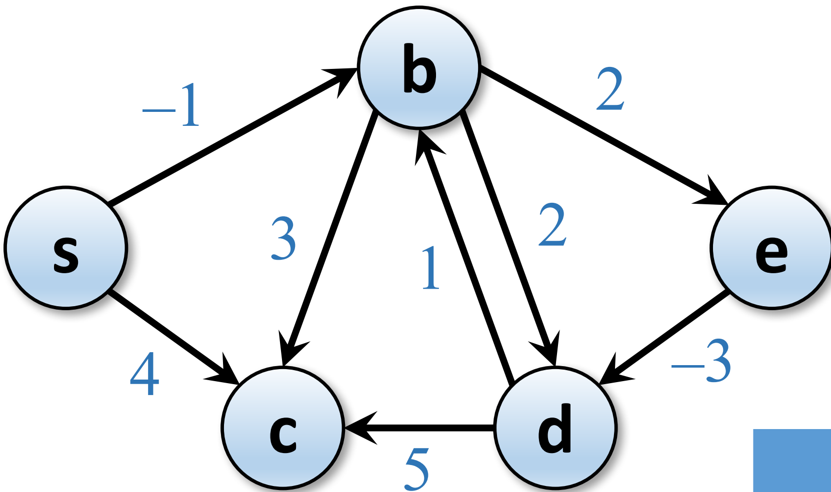
$$OPT(v, i) = \min\left\{OPT(v, i-1), \min_{w \in V}\{OPT(w, i-1) + \ell_{w,v}\}\right\}$$

if min $\Rightarrow$ $P(v,i) \leftarrow P(v,i-1)$

if min for some w

$\Rightarrow$ $P(v,i) \leftarrow w$

Example



	0	1	2	3	4
s	0	0	0	0	0
b	∞	−1	−1	−1	−1
c	∞	4	2	2	2
d	∞	∞	1	−2	−2
e	∞	∞	1	1	1

Implementation (Bottom Up)

```
Shortest-Path(G, s)
   foreach node v ∈ V
      M[0,v] ← ∞
      P[0,v] ← ϕ
   M[0,s] ← 0

   for i = 1 to n-1
      foreach node v ∈ V
        M[i,v] ← M[i-1,v]
        P[i,v] ← P[i-1,v]
        foreach edge (v, w) ∈ E
           if (M[i-1,w] + ℓwv < M[i,v])
              M[i,v] ← M[i-1,w] + ℓwv
              P[i,v] ← w
```

$O(n)$

n-1 times

$\forall v$

$\deg(v)$

$O(1)$

main loop: $\sum_{v \in V} O(\deg(v)) = O(m)$ per iteration

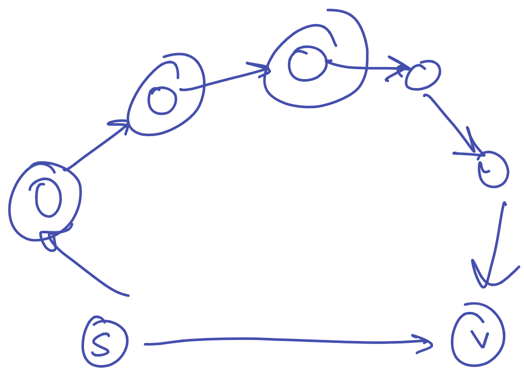
$\times (n-1)$ iterations

$= O(nm)$ time

Optimizations

- One array $M[v]$ containing shortest $s - v$ path found so far
- No need to check edges (w, v) unless $M[w]$ has changed
- Stop if no $M[w]$ has changed for a full pass through V

- Theorem:
  - Throughout the algorithm $M[v]$ is the length of some $s - v$ path
  - After i passes through the nodes, $M[v] \leq OPT(v, i)$


s
v

Implementation II

```
Efficient-Shortest-Path(G, s)
   foreach node v ∈ V
      M[v] ← ∞
      P[v] ← ϕ
   M[s] ← 0

   for i = 1 to n-1
      foreach node w ∈ V
        if (M[w] changed in the last iteration)
           foreach edge (w,v) ∈ E
           if (M[w] + ℓwv < M[v])
               M[v] ← M[w] + ℓwv
               P[v] ← w
        if (no M[w] changed): return M
```

Worst-case running time is $O(nm)$

In practice only make a small # of passes $\Rightarrow O(m)$

Summary

- Can solve shortest path w/ negative length (but no negative cycles) in $O(mn)$ time
  - Faster in practice
- Can implement in a distributed/asynchronous fashion
  - Roughly how routing tables are kept

Negative Cycle Detection

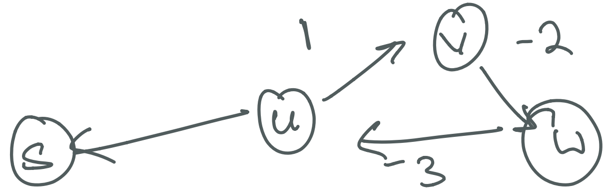
$\forall v$

- Claim 1: if $OPT(v, n) = OPT(v, n - 1)$ then there are no negative cycles reachable from s

- Claim 2: if $OPT(v, n) < OPT(v, n - 1)$ then any shortest $s - v$ path contains a negative cycle

If $OPT(v, n) = OPT(v, n-1)$

then $OPT(v, i) = OPT(v, n-1) \quad \forall\, i \geq n-1$

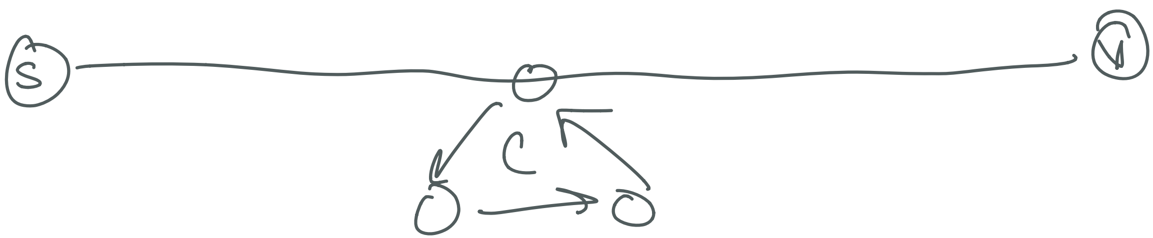


Negative Cycle Detection

$\forall v$

- Claim 1: if $OPT(v, n) = OPT(v, n-1)$ then there are no negative cycles reachable from s

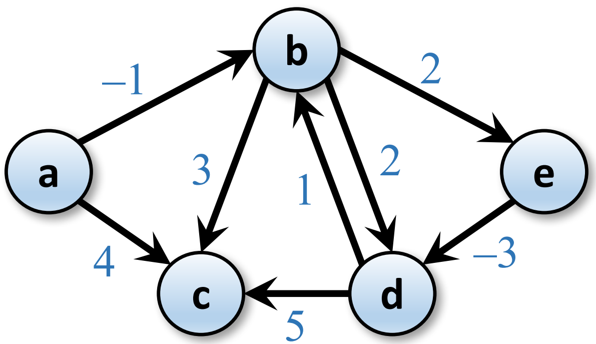
- Claim 2: if $OPT(v, n) < OPT(v, n-1)$ then any shortest $s - v$ path contains a negative cycle



C must have negative length

Negative Cycle Detection

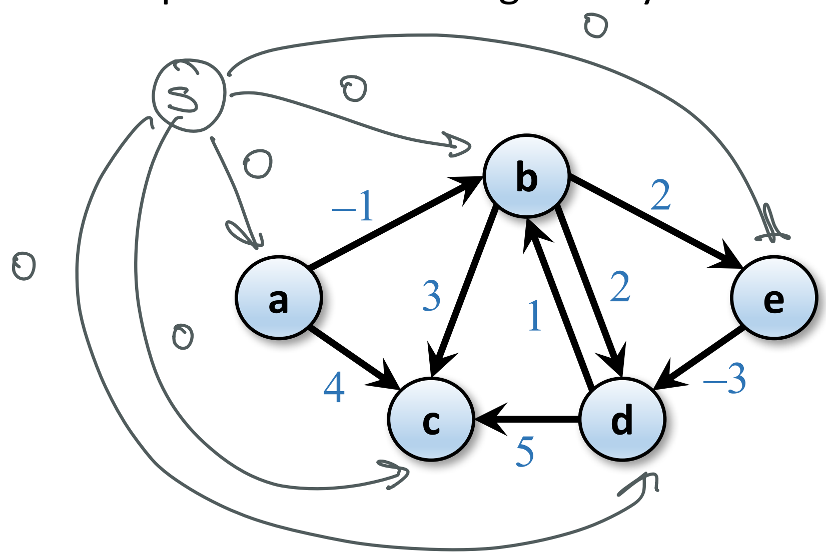
- Algorithm:
  - Pick a node $a \in V$
  - Run Bellman-Ford for n iterations
  - Check if $OPT(v, n) \neq OPT(v, n-1)$ for some $v \in V$
    - If no, then there are no negative cycles
    - If yes, the shortest $a - v$ path contains a negative cycle



Negative Cycle Detection

- Algorithm:
  - Add a new node $s \in V$, add edges (s, v) for every $v \in V$
  - Run Bellman-Ford for n iterations
  - Check if $OPT(v, n) \neq OPT(v, n-1)$ for some $v \in V$
    - If no, then there are no negative cycles
    - If yes, the shortest $s - v$ path contains a negative cycle

$O(nm)$ time
to run Bellman-Ford



Negative Cycle Detection

- Claim 1: if $OPT(v, n) = OPT(v, n-1)$ then there are no negative cycles

- Claim 2: if $OPT(v, n) < OPT(v, n-1)$ then any shortest $s - v$ path contains a negative cycle

Summary

- Bellman-Ford finds shortest paths in $O(nm)$
- Can be modified to find negative cycles also in $O(nm)$.

Implementation (Bottom Up)

```
Shortest-Path(G)
    foreach pair of nodes i,j ∈ V
        if (i = j): M[i,j,0] ← 0
        elseif ((i,j)∈ E): M[i,j,0] ← ℓ_ij
        else: M[i,j,0] ← ∞

    for k = 1 to n:
        for i = 1 to n:
            for j = 1 to n:
```

$$M(i,j,k) \leftarrow \min\left\{\begin{matrix} M(i,j,k-1), \\ M(i,k,k-1) + M(k,j,k-1) \end{matrix}\right\}$$