VE281

Data Structures and Algorithms

Shortest Path

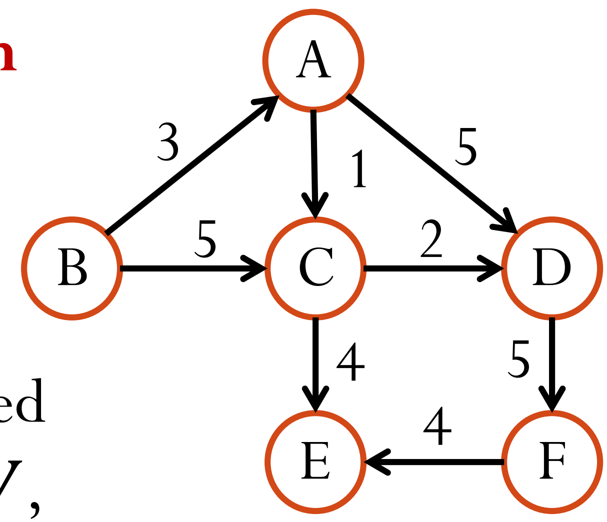
Outline

- Shortest Path Problem
  - Unweighted Graph
  - Dijkstra's Algorithm
  - Bellman-Ford Algorithm

Shortest Path Problem

Introduction

- Given a weighted graph $G = (V, E)$, **path length** is defined as the sum of weights of edges on the path.
  - E.g., length of the path B, C, D, F is 12.
- **Shortest path problem**: given a weighted graph $G = (V, E)$ and two nodes $s, d \in V$, find the shortest path from s to d.
  - Assume G is a directed graph without parallel edges of the same direction
  - For an undirected graph, we can replace each edge by two edges of the same weight but of different directions.



What is the shortest path from B to F?

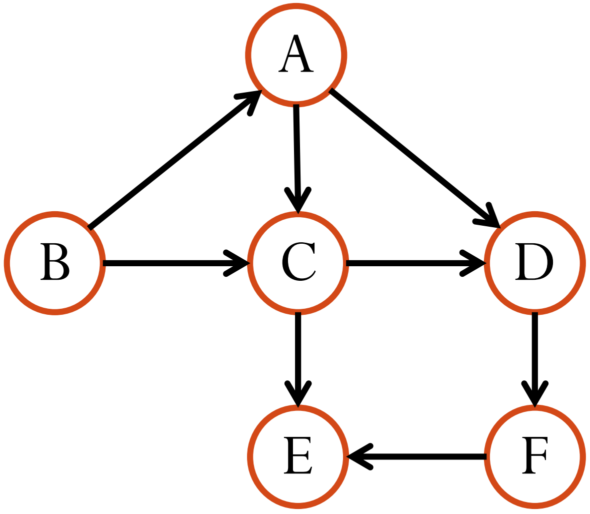
Shortest Path Problem

- The starting node on the path is the **source** node and the ending node is the **destination** node.
- The previous problem is a **single source single destination** problem.
- What we will solve is a **single source all destinations** problem: Given $G = (V, E)$ and a node $s \in V$, find the shortest path from s to **every other** node in G.
  - Single source single destination problem can be solved by solving the single source all destinations problem.
  - However, single source single destination problem is not much easier than the single source all destinations problem.

Shortest Path Problem

A Simple Version: Unweighted Graphs

- For an unweighted graph, path length is defined as the number of edges on the path.
- How do you obtain the shortest path between a pair of nodes?

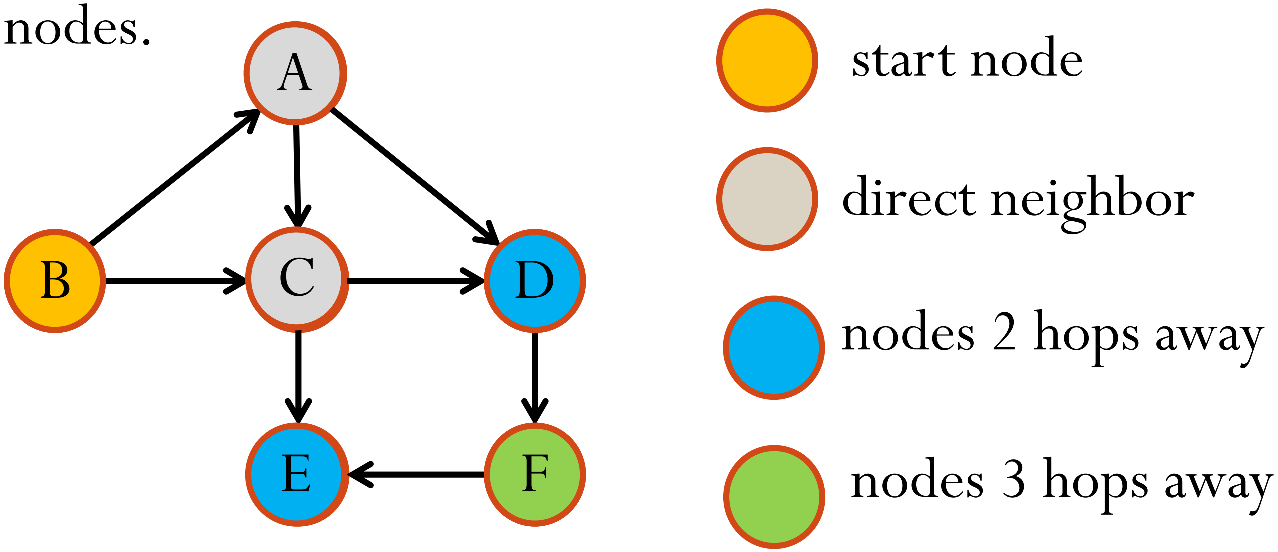


What is the shortest path from B to F?

Using breadth-first search!

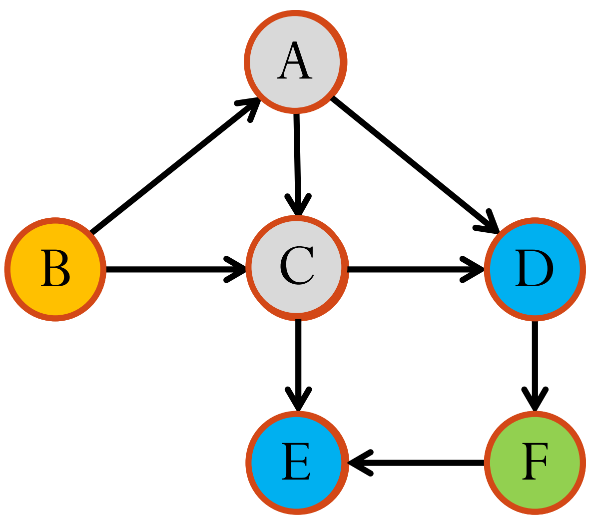
Shortest Path for Unweighted Graphs

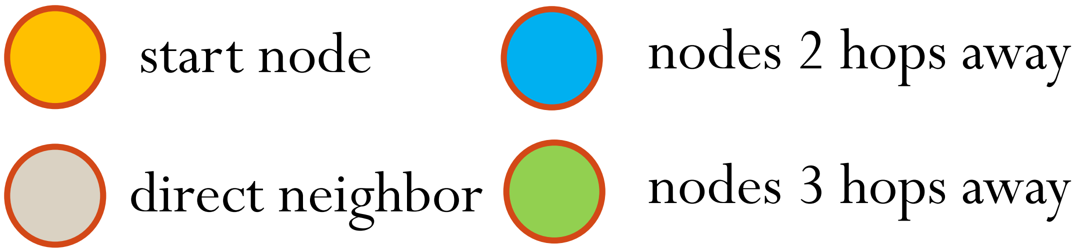
- Recall breadth-first search (BFS): Given a start node, visit all directly connected neighbors first, then nodes 2 hops away, 3 hops away, and so on.
  - This is exactly what we want!
  - When the node visited is the destination node, we stop.
  - When the queue becomes empty, there is no path between the two nodes.



Shortest Path for Unweighted Graphs

- Additional bookkeeping
  - Store the distance.
  - Store the **predecessor** on the shortest path, i.e., the previous node on the path.



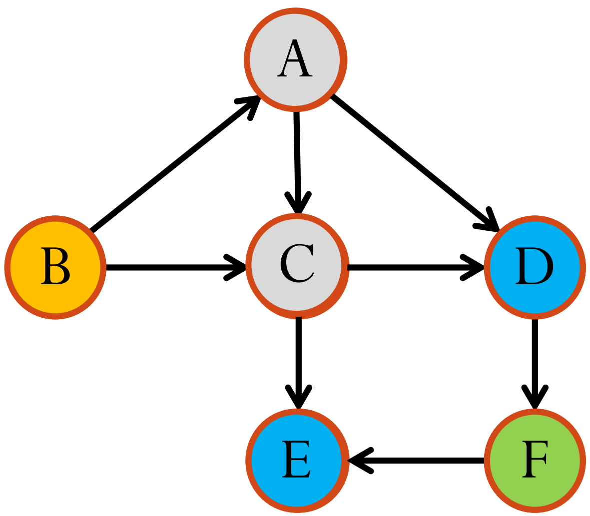


	A	B	C	D	E	F
dist	1	0	1	2	2	3
pred	B	-	B	A	C	D

Shortest Path for Unweighted Graphs

- We can obtain the shortest path by backtracking.
  - E.g., shortest path from B to F

$\boxed{B \rightarrow A \rightarrow D \rightarrow F}$



start node

direct neighbor

nodes 2 hops away

nodes 3 hops away

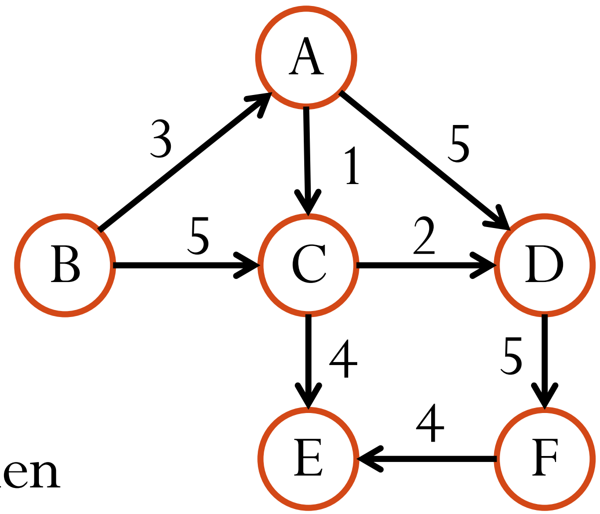
	A	B	C	D	E	F
dist	1	0	1	2	2	3
prev	B	-	B	A	C	D

Outline

- Shortest Path Problem
  - Unweighted Graph
  - Dijkstra's Algorithm
  - Bellman-Ford Algorithm

Shortest Path for Weighted Graphs

- The problem becomes more difficult when edges have different weights.
  - Breadth-first search won't work!
  - What is the shortest path from B to F?



- If the weights are **non-negative**, then we can apply **Dijkstra's Algorithm** (more details & examples from Ve203)
  - Works only when all weights are non-negative
  - A greedy algorithm for solving single source all destinations shortest path problem

Dijkstra's Algorithm

- Keep **distance estimate** $D(v)$ and **predecessor** $P(v)$ for each node v.
  - Predecessor: the previous node on the shortest path.

1. Initially, $D(s) = 0$; $D(v)$ for other nodes is $+\infty$; $P(v)$ is unknown.
2. Store all the nodes in a set R.
3. While R is not empty
   1. Choose node v in R such that $D(v)$ is the **smallest**. Remove v from the set R.
   2. Declare that v's shortest distance is known, which is $D(v)$.
   3. For each of v's **neighbors** u that is **still in** R, update distance estimate $D(u)$ and predecessor $P(u)$.

Updating

- For each of v's **neighbors** u that is **still in** R, if $D(v) + w(v, u) < D(u)$, then update $D(u) = D(v) + w(v, u)$ and the predecessor $P(u) = v$.
  - I.e., update $D(u)$ if the path going through v is shorter than the best path so far to u.

Dijkstra's Algorithm v.s. Prim's Algorithm

- Dijkstra's algorithm is similar to Prim's algorithm
  - Prim's algorithm: grow the set of nodes we add to the MST.
  - Dijkstra's algorithm: grow the set of nodes to which we know the shortest path.

Dijkstra's Algorithm

Time Complexity

- Number of times to find the smallest $D(v)$: $|V|$.
  - Each cost? Linear scan: $O(|V|)$; Binary heap: $O(\log|V|)$; Fibonacci heap: $O(\log|V|)$
- Total number of times to update the neighbors: $|E|$.
  - Since each neighbor of each node could be potentially updated.
  - Each cost? Linear scan: $O(1)$; Binary heap: $O(\log|V|)$; Fibonacci heap: $O(1)$
- Total time complexity
  - Linear scan: $O(|E| + |V|^2) = O(|V|^2)$
  - Binary heap: $O(|V|\log|V| + |E|\log|V|)$
  - Fibonacci heap: $O(|V|\log|V| + |E|)$

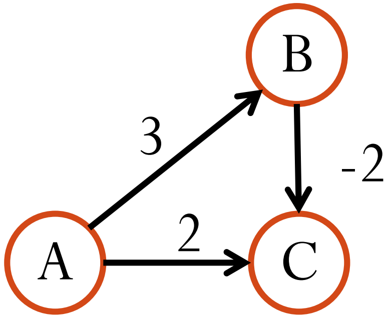
Outline

- Shortest Path Problem
  - Unweighted Graph
  - Dijkstra's Algorithm
  - Bellman-Ford Algorithm

Limitation of Dijkstra's Algorithm

- Not always correct on graphs with negative edge weights

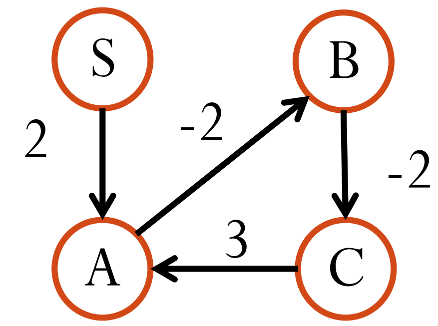
Find shortest paths from source node A



- Solution: The Bellman-Ford algorithm

Negative Cycles

- If the graph contains a negative cycle reachable from the source s, then the shortest paths for some destinations are $-\infty$
  - Can traverse the negative cycle infinite times
  - Thus, if there is any negative cycle reachable from s, the short path problem is not well defined



- If there is no negative cycle reachable from s, Bellman-Ford algorithm reports the correct shortest paths for all destinations
- If there is such a negative cycle, the algorithm will report it

Key Idea of Bellman-Ford Algorithm

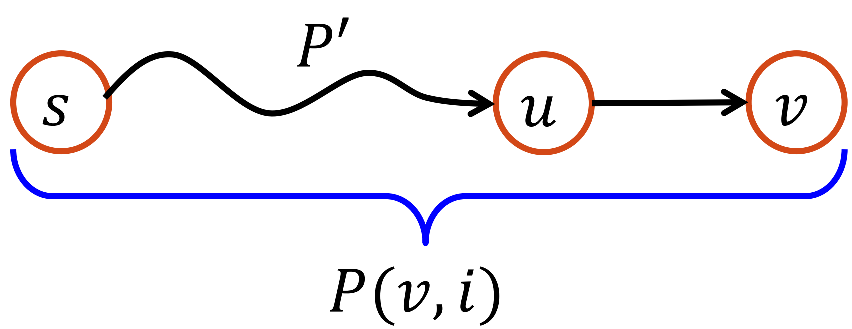
- Claim: Suppose there is no negative cycles. Then, for each node $v \in V$, there is a shortest s-v path with $\leq |V| - 1$ edges
  - Proof: If shortest path has $\geq |V|$ edges, then there is a cycle. By assumption, the cycle has non-negative length. We can drop the cycle and obtain a no-worse path
- Key idea: consider shortest s-v path with **at most** i edges
  - Denoted as $P(v, i)$
  - It can be recursively obtained (show next)
  - If there is no negative cycles, then $P(v, |V| - 1)$ is a shortest s-v path

Optimality

- Define $P(v, i)$ as the shortest s-v path with **at most** i edges
- Case 1: $P(v, i)$ has $\leq i - 1$ edges. Then $P(v, i)$ is also a shortest s-v path with $\leq i - 1$ edges.
  - Proof: By contradiction, assume Q is an s-v path with $\leq i - 1$ edges shorter than $P(v, i)$. Then $P(v, i)$ cannot be the shortest s-v path with $\leq i$ edges

Optimality

- Case 2: $P(v,i)$ has i edges. Suppose the first $i-1$ edges of $P(v,i)$ form the path P' and the last edge is (u,v), then P' is a shortest s-u path with $\leq i-1$ edges.
  - Proof: By contradiction, suppose Q is a s-u path with $\leq i-1$ edges shorter than P'. Then $Q+(u,v)$ is shorter than $P'+(u,v)$. Then $P(v,i)$ cannot be the shortest s-v path with $\leq i$ edges



Recurrence

- Possible candidates for $P(v,i)$:
  - $P(v,i-1)$
  - $P(u,i-1)+(u,v)$, for all $(u,v) \in E$
- $P(v,i)$ is the shortest one of the above candidates

- Define the length of $P(v,i)$ as $L(v,i)$
- $L(v,i) = \min\{L(v,i-1), \min_{(u,v)\in E}[L(u,i-1)+w(u,v)]\}$

The Bellman-Ford Algorithm

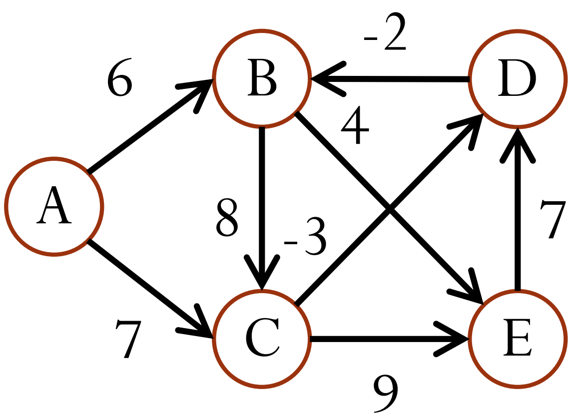
- Initially, $L(s,0) = 0$; $L(v,0) = +\infty$ for other $v \in V$
- For $i = 1,2, \ldots, |V| - 1$
  - For each $v \in V$,

$$L(v,i) = \min\{L(v,i-1), \min_{(u,v)\in E}[L(u,i-1) + w(u,v)]\}$$

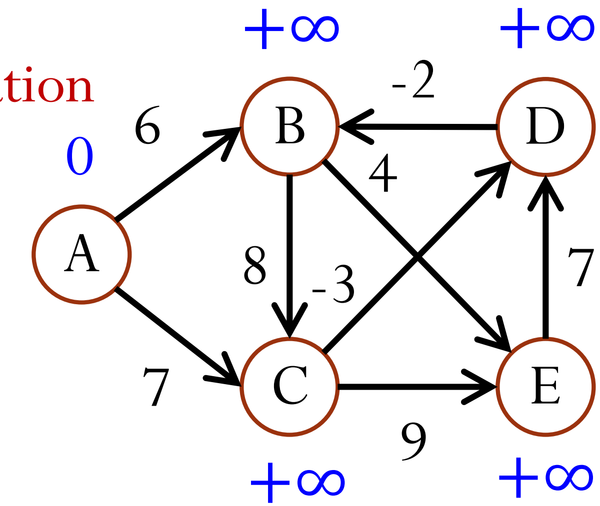
- If there is no negative cycle, the length of shortest s-v path is $L(v,|V|-1)$ for all $v \in V$
- Can keep the predecessor to track shortest path

Example

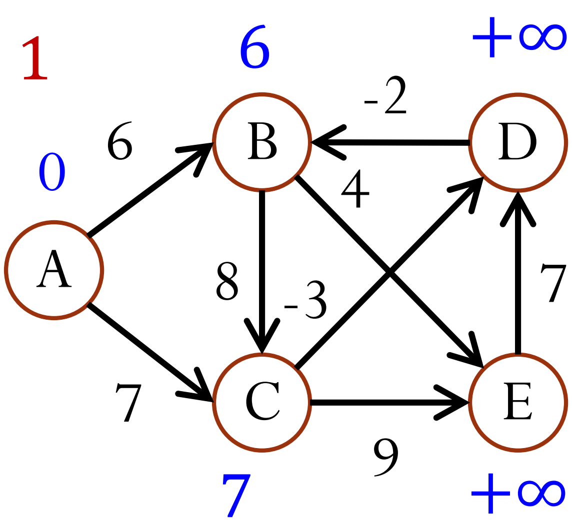
- Let start point be A



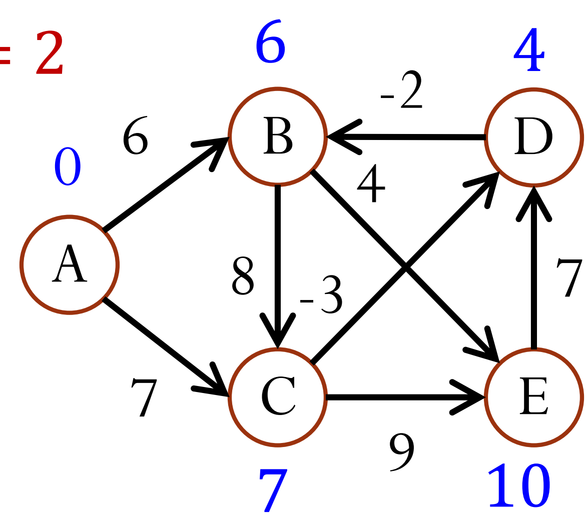
Initialization



$i = 1$

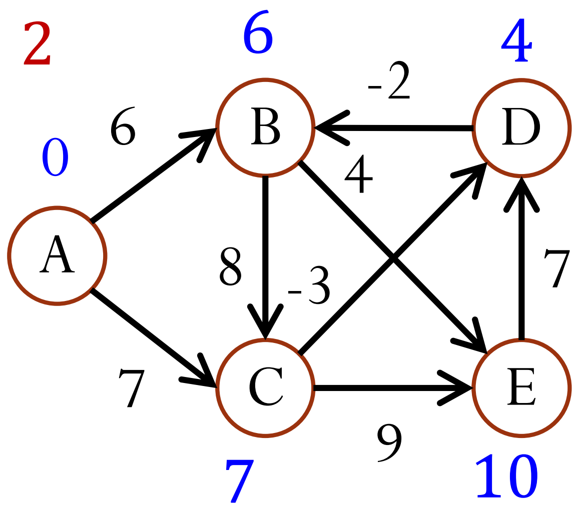


$i = 2$

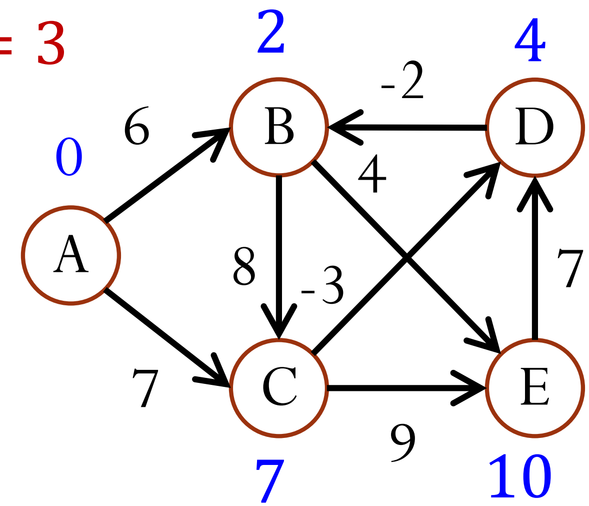


Example

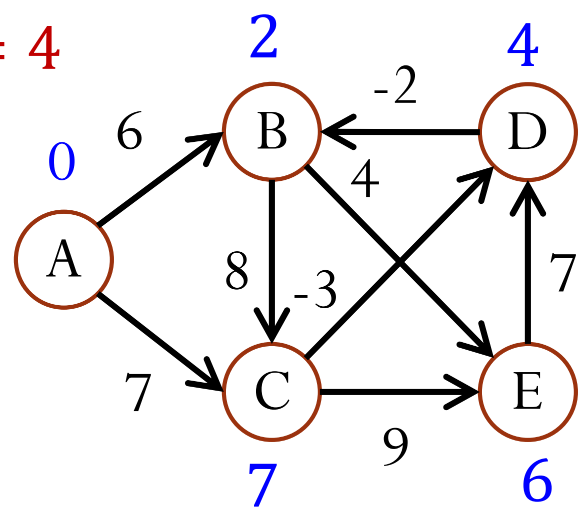
$i = 2$



$i = 3$



$i = 4$



Final shortest path length

Time Complexity of Bellman-Ford

- Initially, $L(s,0) = 0$; $L(v,0) = +\infty$ for other $v \in V$
- For $i = 1,2,\ldots,|V|-1$
  - For each $v \in V$,

$$L(v,i) = \min\{L(v,i-1), \min_{(u,v)\in E}[L(u,i-1) + w(u,v)]\}$$

- Assume adjacency list representation. What's the time complexity?
  - $O((|V| + |E|)|V|)$ (worse than Dijkstra's algorithm)

Detecting Negative Cycle

- We can modify Bellman-Ford algorithm to report the existence of negative cycle by running it for one more iteration to obtain $L(v, |V|)$ for all $v \in V$

- Claim: G has no negative cycle reachable from $s \Leftrightarrow$ $L(v, |V| - 1) = L(v, |V|)$ for all $v \in V$

- Proof of $\Rightarrow$: There exists a shortest s-v path with $\leq |V| - 1$ edges. Thus $L(v, |V| - 1) = L(v, |V|)$ for all $v \in V$

Detecting Negative Cycle

- Claim: G has no negative cycle reachable from $s \Leftrightarrow L(v, |V| - 1) = L(v, |V|)$ for all $v \in V$

- Proof of $\Leftarrow$:
  - Let $d(v) = L(v, |V| - 1)(= L(v, |V|))$
  - By the recurrence

    $$\underbrace{L(v, |V|)}_{d(v)} = \min\left\{L(v, |V| - 1), \min_{(u,v)\in E}\left[\underbrace{L(u, |V| - 1)}_{d(u)} + w(u, v)\right]\right\},$$

  - … we have for any $(u, v) \in E$, $d(v) \leq d(u) + w(u, v)$, or $d(v) - d(u) \leq w(u, v)$
  - Now consider any cycle C, we have $\sum_{(u,v)\in C} w(u, v) \geq 0$