Dynamic Programming IV: Shortest Path Revisited

Admin

- Additional office hours tomorrow: **12.30 - 2 pm**
- Purpose of help hours today and tomorrow:
  - Come with any questions you have about topics so far
  - Or any topic that you'd want reviewed
- No lecture on Friday: 24-hour open-book midterm
  - Can be turned in latest by 10 am Sunday
  - No late work permitted on exam
- Review item: Solving recurrences with an $O()$ term
  - By definition, constants do not change as input size changes

Partitioning Books

Reading: Linked on GLOW

Partitioning Work

- Suppose we have to scan through a shelf of books, and each book has a different size
- We want to divide the shelf into k region of books, and each region is assigned one of the workers
- Order of books fixed by cataloging system: cannot reorder/ rearrange the books
- **Goal**: divide the work is a fair way among the workers

Linear Partition Problem

- **Input.** A input arrangement S of nonnegative integers $\{s_1, \ldots, s_n\}$ and an integer k
- **Problem.** Partition S into k ranges such that the **maximum sum** over all the ranges is **minimized**
- Example.
  - Consider the following arrangement

    100 200 300 400 500 600 700 800 900

  - If $k = 3$, a partition that minimizes the maximum sum:

    100 200 300 400 500 | 600 700 | 800 900

Subproblem

- **Subproblem**

$M(i, j)$ be the optimal cost of partitioning elements $s_1, s_2, \ldots, s_i$ using j partitions, where $1 \leq i \leq n$, $1 \leq j \leq k$

- **Final answer**

$M(n, k)$

Base Cases

- Let us think about which rows/columns can we fill initially
- What about the first row corresponding to item 1?
- Remember that optimal cost is max sum over all partitions
- $M(1, j)$: optimal cost of partitioning s_1 across j partitions
- For $j = 1,2,\ldots,k$ we can fill out the first column as:

$$\boxed{M(1, j) = s_1}$$

Base Cases

- Let us think about which rows/columns can we fill initially
- What about the first row corresponding to item 1?
- Remember that optimal cost is max sum over all partitions
- $M(i,\ 1)$: optimal cost of partitioning $s_1, s_2, \ldots, s_i$ using only 1 partition
- For $i = 1,2,\ldots,n$ we can fill out the first column as:

$$\boxed{M(i,\ 1) = \sum_{\ell=1}^{i} s_\ell}$$

Base Cases Summary

- For $j = 1,2,\ldots,k$ we can fill out the first column as:

$$\boxed{M(1, j) = s_1}$$

- For $i = 1,2,\ldots,n$ we can fill out the first column as:

$$\boxed{M(i, 1) = \sum_{\ell=1}^{i} s_\ell}$$

Towards a Recurrence

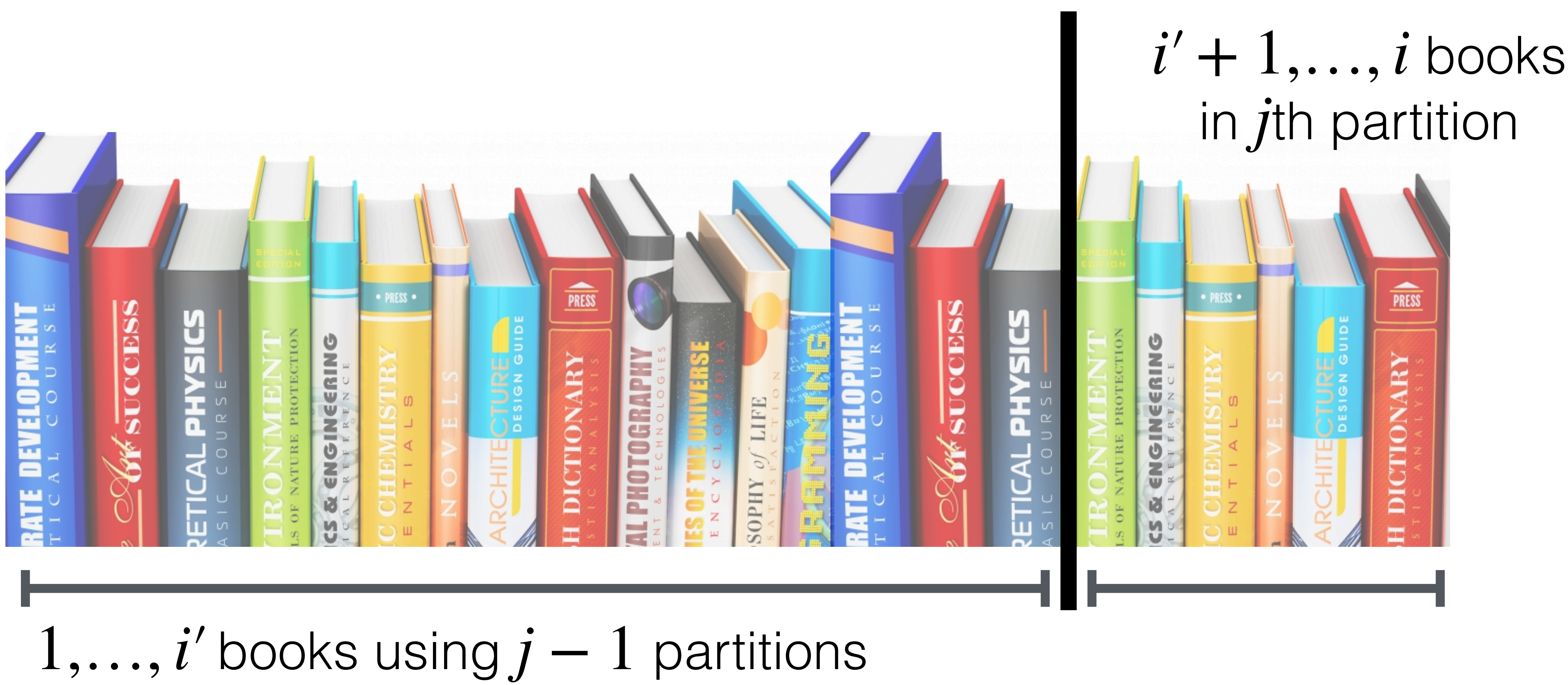
- Want a recurrence for $M(i, j)$
- Notice that the jth partition starts after we place the $(j-1)$st "divider"
- Where can we place the $j-1st$ divider?

$i' + 1, \ldots, i$ books in jth partition

$1, \ldots, i'$ books using $j-1$ partitions

Towards a Recurrence

- Where can we place the $j-1st$ divider?
    - Between books i' and $i'+1$ for some $i' < i$



Towards a Recurrence

- Finally: for to choose the partition point i' for starting the jth partition
  - Let us consider all possibilities $1 \leq i' < i$
  - Take min cost option among them

Final Recurrence

- For $2 \leq i \leq n$ and $2 \leq j \leq k$, we have:

$$M(i, j) = \min_{1 \leq i' < i} \text{cost of starting } j\text{th parition at book } i' + 1$$

Towards a Recurrence

- Cost of this way of partitioning?
  - (Remember cost is max sum across all partitions)

$i' + 1, \ldots, i$ books in jth partition

$1, \ldots, i'$ books using $j - 1$ partitions

Towards a Recurrence

- Cost of jth partition itself: $\sum_{t=i'+1}^{i} s_i$
- Cost of remaining partitions? $M[i', j-1]$

$i' + 1, \ldots, i$ books in jth partition

$1, \ldots, i'$ books using $j-1$ partitions

Final Recurrence

- For $2 \leq i \leq n$ and $2 \leq j \leq k$, we have:

$$\boxed{M(i, j) = \min_{1 \leq i' < i} \max\{M(i', j-1), \sum_{\ell = i'+1}^{i} s_t\}}$$

- **Memoization structure**: We store $M[i, j]$ values in a 2-D array or table using space $O(nk)$
- **Evaluation order**: In what order should we fill in the table?

Final Pieces

- Evaluation order.
    - To fill out $M[i, j]$, I need the previous column filled in for rows less than i, that is, $M[i', j-1]$ for all $1 \leq i' < i$
    - Can compute using column major order: column by column
- Running time?
    - Size of table (space): $O(k \cdot n)$
    - How long to compute a single cell?
        - Depends on n other cells
        - $O(n)$ time to fill in one cell

Running Time

- Running time
    - $O(n^2 \cdot k)$
- Is this a polynomial running time?
    - Not as stated, not polynomial in k
    - But lets think if we can upper bound k using n
- How big can k get?
    - At most n non-empty partitions of n elements
    - $O(n^3)$ algorithm in the worst case

Last Topic in Dynamic Programming:
Shortest Paths Revisited

Shortest Path Problem

- **Single-Source Shortest Path Problem**.
  Given a directed graph $G = (V, E)$ with edge weights w_e on each $e \in E$ and a a source node s, find the shortest path from s to to all nodes in G.

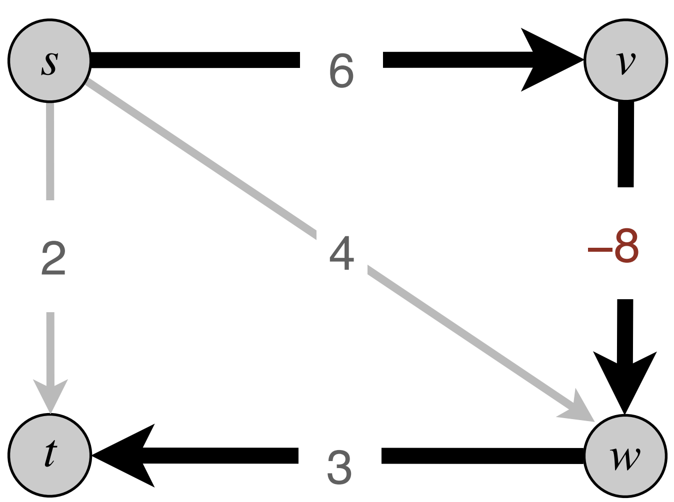
- **Negative weights**. The edge-weights $w_e$in G can be negative. (When we studied Dijkstra's, we assumed non-negative weights.)

- Let P be a path from s to t, denoted $s \rightsquigarrow t$.
  - The **length** of P is the number of edges in P
  - The cost or weight of P is $w(P) = \sum_{e \in P} w_e$

- Goal: **cost** of the shortest path from s to all nodes

Negative Weights & Dijkstra's

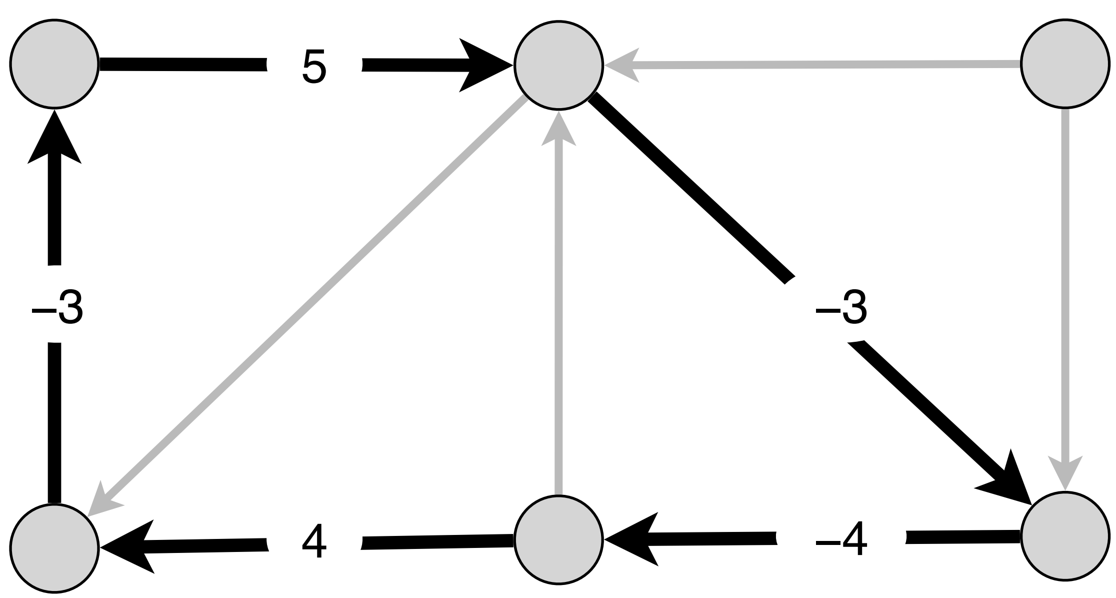
- **Dijkstra's Algorithm**. Does the greedy approach work for graphs with negative edge weights?
  - Dijkstra's will explore s's neighbor and add t, with $d[t] = w_{sv} = 2$ to the shortest path tree
  - Dijkstra assumes that there cannot be a "longer path" that has lower cost (relies on edge weights being non-negative)



Dijkstra's will find $s \rightarrow t$ as shortest path with cost 2
But the shortest path is $s \rightarrow v \rightarrow w \rightarrow t$ with cost 1

Negative Cycles

- **Definition**. A negative cycle is a directed cycle C such that the sum of all the edge weights in C is less than zero
- **Question**. How do negative cycles affect shortest path?



a negative cycle W : $\ell(W) = \sum_{e \in W} \ell_e < 0$

Negative Cycles & Shortest Paths

- **Claim.** If a path from s to some node v contains a negative cycle, then there does not exist a shortest path from s to v.

- **Proof**.
  - Suppose there exists a shortest $s \rightsquigarrow v$ path with cost d that traverses the negative cycle t times for $t \geq 0$.
  - Can construct a shorter path by traversing the cycle $t + 1$ times

  $\Rightarrow\!\Leftarrow$ $\blacksquare$

- **Assumption.** G has no negative cycle.

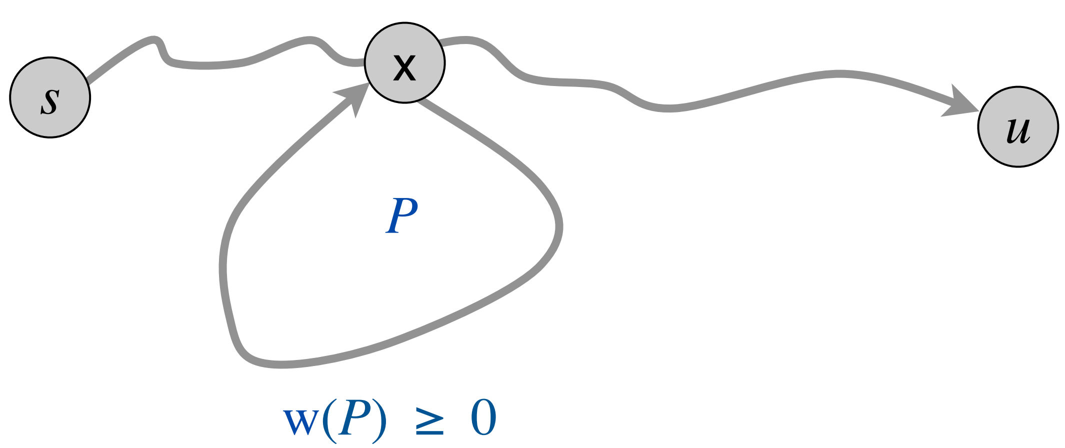
- Later in the lecture: how can we detect whether the input graph G contains a negative cycle?

Dynamic Programming Approach

- First step to a dynamic program? Recursive formulation
  - Subproblem with an “optimal substructure”
- **Structure of the problem**. With negative edge weights, the optimal cost can have any length
  - Let's keep track of **length of paths** considered so far
- How long can the shortest path from s to any node u be, assuming no negative cycle?
- **Claim**. If G has no negative cycles, then exists a shortest path from s to any node u that uses at most $n-1$ edges.

No. of Edges in Shortest Path

- **Claim.** If G has no negative cycles, then exists a shortest path from s to any node u that uses at most $n-1$ edges.
- **Proof**. Suppose there exists a shortest path from s to u made up of n or more edges
- A path of length at least n must visit at least $n+1$ nodes
- There exists a node x that is visited more than once (**pigeonhole principle**). Let P denote the portion of the path between the successive visits.
- Can remove P without increasing cost of path. ∎



Shortest Path Subproblem

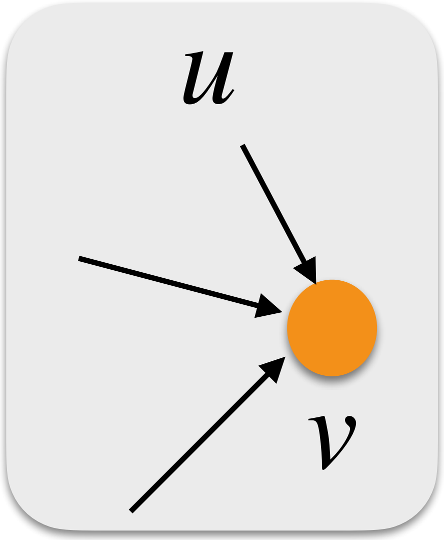
- **Subproblem**. $D[v, i]$: (optimal) cost of shortest path from s to v using $\leq i$ edges

- **Base cases**.
  - $D[s, i] = 0$ for any i
  - $D[v, 0] = \infty$ for any $v \neq s$

- **Final answer** for shortest path cost to node v
  - $D[v, n - 1]$

Recurrence

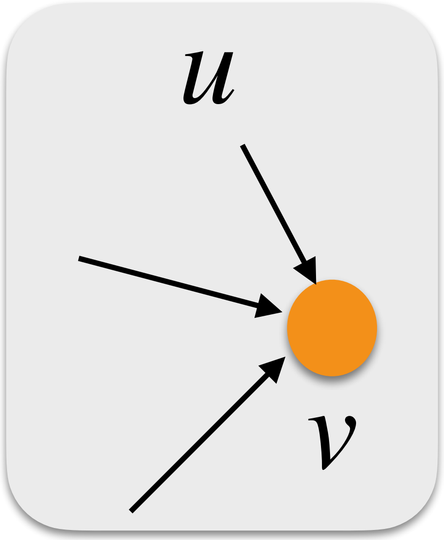
- Suppose we have found shortest paths to all nodes of length at most $i - 1$
- We are now considering shortest paths of length i
- Cases to consider for the **recurrence** of $D[v, i]$
    - **Case 1**. Shortest path to v was already found (is same as $D[v, i-1]$)
    - **Case 2**. Shortest path to v is "longer" than paths found so far:
        - Look at all nodes u that have incoming edges to v
        - Take minimum over their distances and add w_{uv}



Bellman-Ford-Moore Algorithm

- **Recurrence.** For all nodes $v \neq s$, and for all $1 \leq i \leq n-1$,

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



- Called the **Bellman-Ford-Moore** algorithm

Bellman-Ford-Moore Algorithm

- **Subproblem**. $D[v, i]$: (optimal) cost of shortest path from s to v using $\leq i$ edges

- **Recurrence.**
  $$D[v, i] = \min\{D[v, i-1], \min_{(u,v)\in E} \{D[u, i-1] + w_{uv}\}\}$$

- **Memoization structure**. Two-dimensional array

- **Evaluation order**.
  - $i : 1 \to n - 1$ (column major order)
  - Starting from s, the row of vertices can be in any order

Running Time

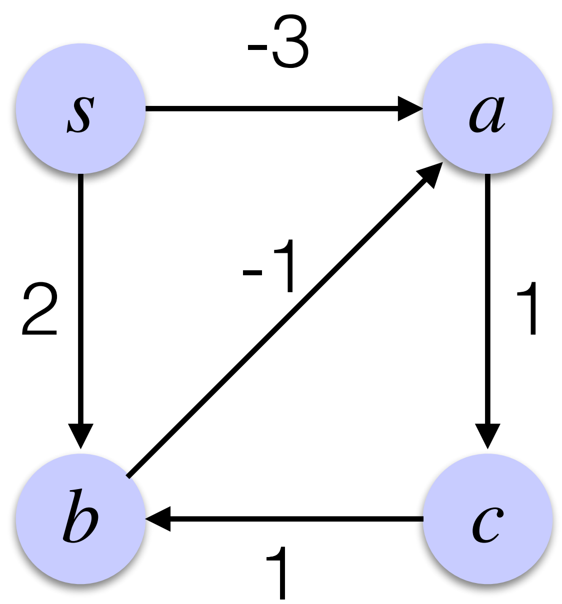
- **Recurrence**.
  $$D[v,i] = \min\{D[v,i-1], \min_{(u,v)\in E} \{D[u,i-1] + w_{uv}\}\}$$
- **Naive analysis**. $O(n^3)$ time
  - Each entry takes $O(n)$ to compute, there are $O(n^2)$ entries
- **Improved analysis**. For a given i, v, $d[v,i]$ looks at each incoming edge of v
  - Takes indegree(v) accesses to the table
  - For a given i, filling $d[-,i]$ takes $\sum_{v\in V}$ indegree(v) accesses
  - At most $O(n+m) = O(m)$ accesses for connected graphs where $m \geq n-1$
- Overall running time is $O(nm)$

- **Shortest-Path Summary.** Assuming there are no negative cycles in G, we can compute the shortest path from s to all nodes in G in $O(nm)$ time using the Bellman-Ford-Moore algorithm

Dynamic Programming
Shortest Path:
Bellman-Ford-Moore Example

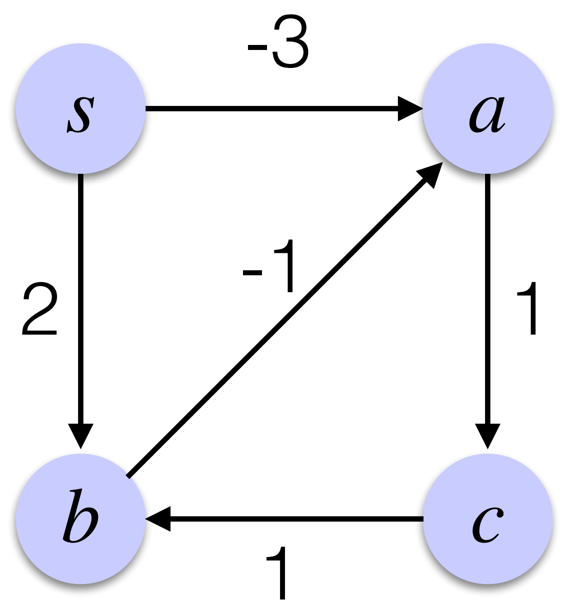
- $D[s, i] = 0$ for any i
- $D[v,0] = \infty$ for any $v \neq s$

	0	1	2	3
s	0	0	0	0
a	inf			
b	inf			
c	inf			



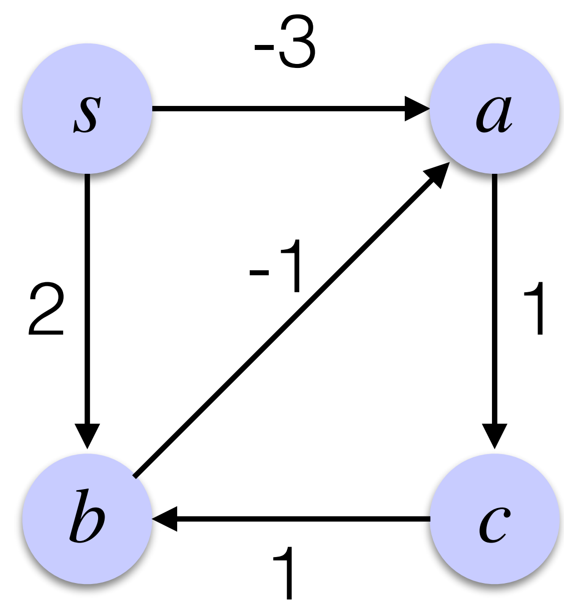
- $D[v,1] = \min\{D[v,0], \min_{u,v \in E}\{D[u,0] + w_{uv}\}$

	0	1	2	3
s	0	0	0	0
a	inf			
b	inf			
c	inf			



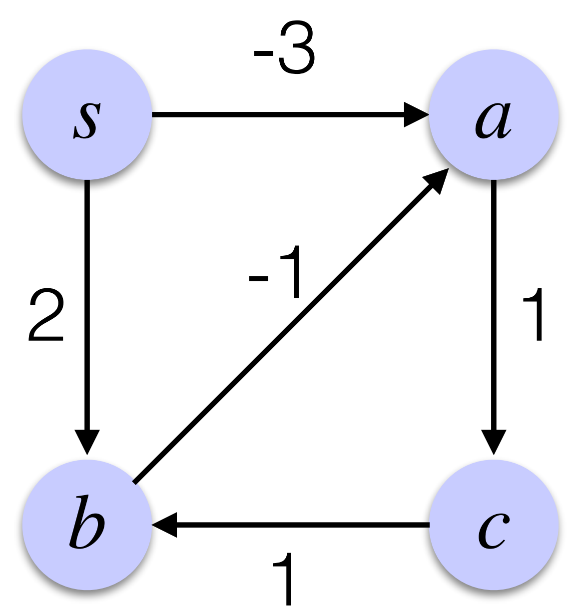
- $D[v,1] = \min\{D[v,0], \min_{u,v \in E} \{D[u,0] + w_{uv}\}$

	0	1	2	3
s	0	0	0	0
a	inf	-3		
b	inf			
c	inf			



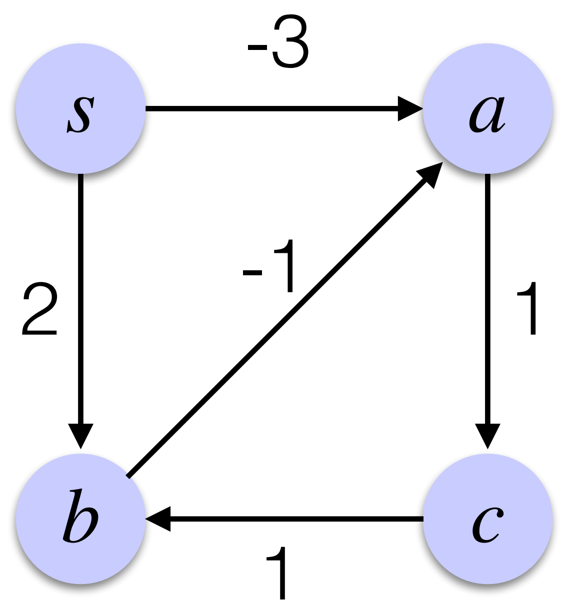
- $D[v,1] = \min\{D[v,0], \min_{u,v \in E} \{D[u,0] + w_{uv}\}$

	0	1	2	3
s	0	0	0	0
a	inf	-3		
b	inf	2		
c	inf			



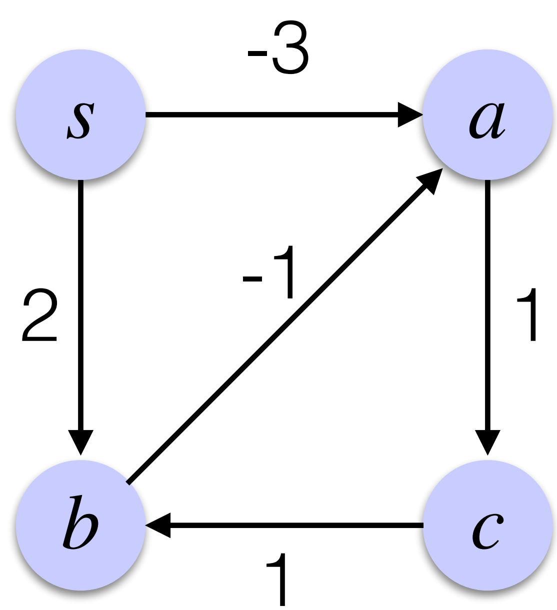
- $D[v,1] = \min\{D[v,0], \min_{u,v \in E} \{D[u,0] + w_{uv}\}$

	0	1	2	3
s	0	0	0	0
a	inf	-3		
b	inf	2		
c	inf	inf		



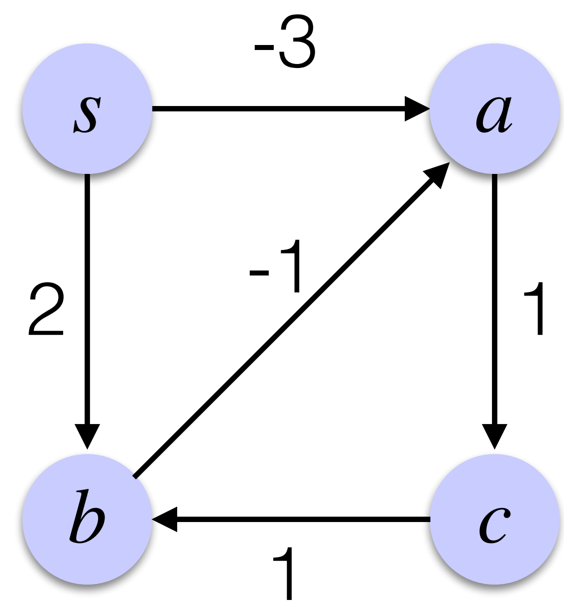
- $D[v,2] = \min\{D[v,1], \min_{u,v \in E} \{D[u,1] + w_{uv}\}$

	0	1	2	3
s	0	0	0	0
a	inf	-3		
b	inf	2		
c	inf	inf		



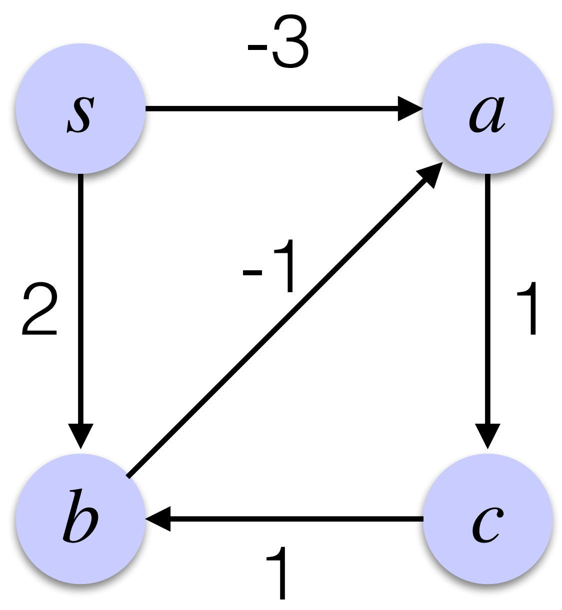
- $D[v,2] = \min\{D[v,1], \min_{u,v \in E} \{D[u,1] + w_{uv}\}$

	0	1	2	3
s	0	0	0	0
a	inf	-3	-3	
b	inf	2		
c	inf	inf		



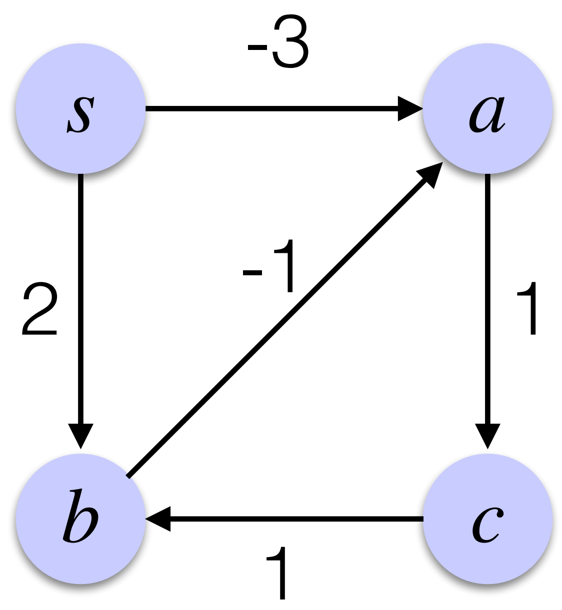
- $D[v,2] = \min\{D[v,1], \min_{u,v \in E} \{D[u,1] + w_{uv}\}$

	0	1	2	3
s	0	0	0	0
a	inf	-3	-3	
b	inf	2	2	
c	inf	inf		



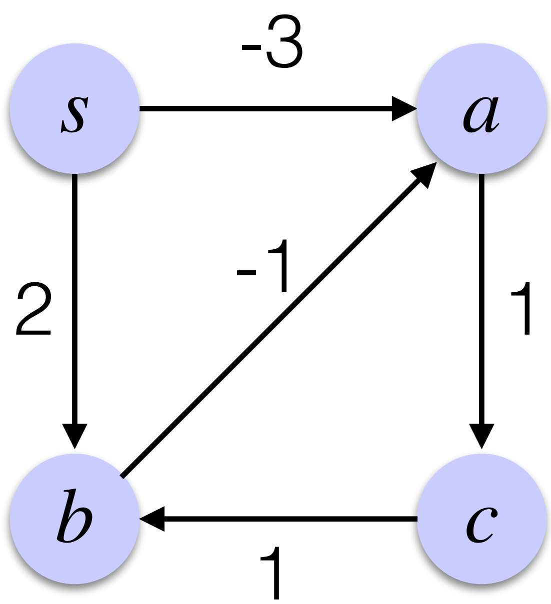
- $D[v,2] = \min\{D[v,1], \min_{u,v \in E} \{D[u,1] + w_{uv}\}$

	0	1	2	3
s	0	0	0	0
a	inf	-3	-3	
b	inf	2	2	
c	inf	inf	-2	



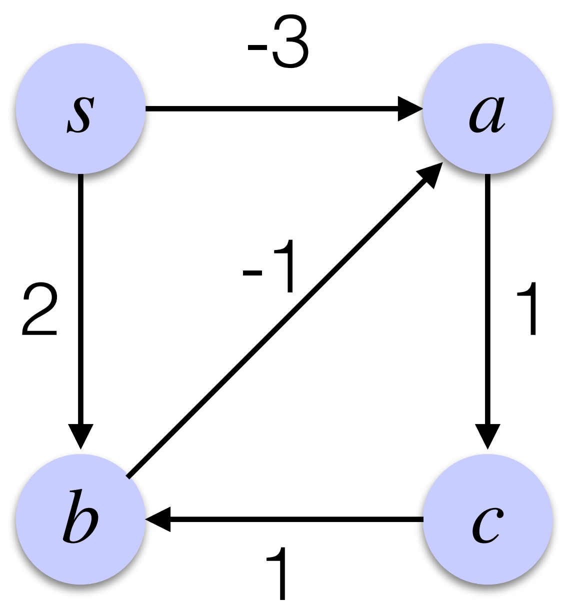
- $D[v,3] = \min\{D[v,2], \min_{u,v \in E} \{D[u,2] + w_{uv}\}$

	0	1	2	3
s	0	0	0	0
a	inf	-3	-3	-3
b	inf	2	2	
c	inf	inf	-2	



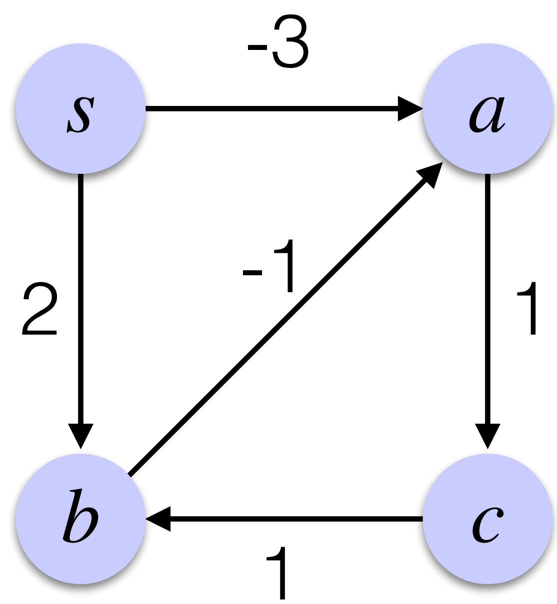
- $D[v,3] = \min\{D[v,2], \min_{u,v \in E} \{D[u,2] + w_{uv}\}$

	0	1	2	3
s	0	0	0	0
a	inf	-3	-3	-3
b	inf	2	2	-1
c	inf	inf	-2	



- $D[v,3] = \min\{D[v,2], \min_{u,v \in E} \{D[u,2] + w_{uv}\}$

	0	1	2	3
s	0	0	0	0
a	inf	-3	-3	-3
b	inf	2	2	-1
c	inf	inf	-2	-2

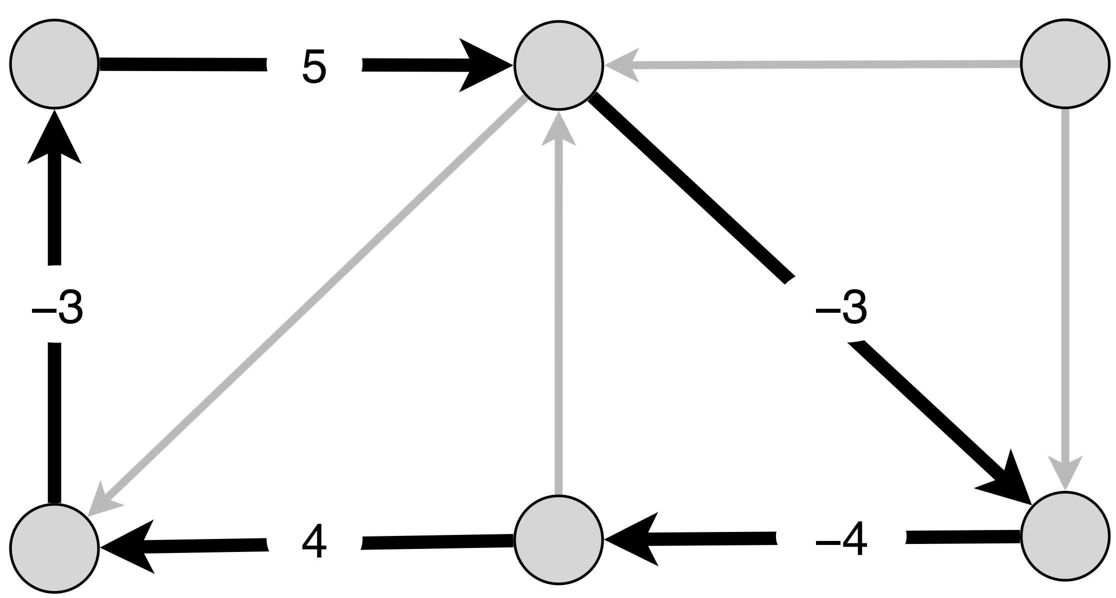


Dynamic Programming Shortest Path:

Detecting a Negative Cycle

Negative Cycle

- **Definition**. A negative cycle is a directed cycle C such that the sum of all the edge weights in C is less than zero
- **Claim.** If a path from s to some node v contains a negative cycle, then there does not exist a shortest path from s to v.



a negative cycle W : $\ell(W) = \sum_{e \in W} \ell_e < 0$

Detecting a Negative Cycle

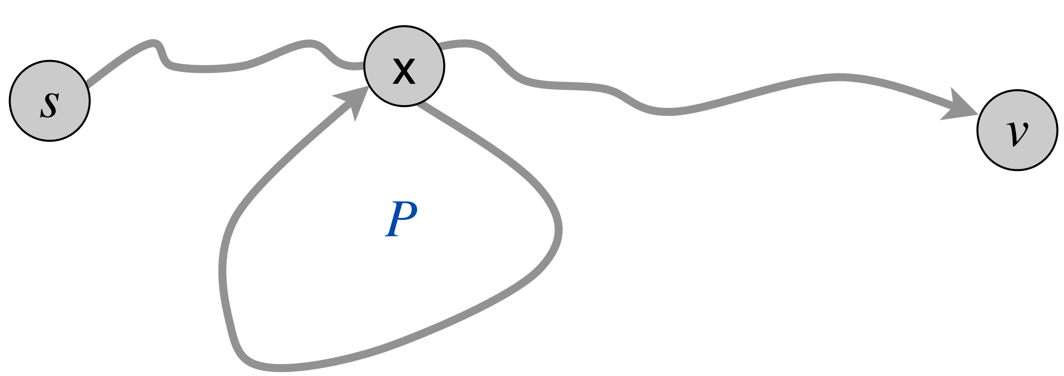
- **Question.** Given a directed graph $G = (V, E)$ with edge-weights w_e (can be negative), determine if G contains a negative cycle.
- Now, we don't a specific source node given to us
- Let's change this problem a little bit
- **Problem**. Given G and source s, find if there is negative cycle on a $s \rightsquigarrow v$ path for any node v.

Detecting a Negative Cycle

- **Problem**. Given G and source s, find if there is negative cycle on a $s \rightsquigarrow v$ path for any node v.
- $D[v, i]$ is the cost of the shortest path from s to v of length at most i
- Suppose there is a negative cycle on a $s \rightsquigarrow v$ path
    - Then $\lim_{i \to \infty} D[v, i] = -\infty$
- If $D[v, n] = D[v, n-1]$ for every node v then G has no negative cycles exists!
    - Table values converge, no further improvements possible

Detecting a Negative Cycle

- **Lemma.** If $D[v, n] < D[v, n-1]$ then any shortest $s \rightsquigarrow v$ path contains a negative cycle.
- **Proof**. [By contradiction] Suppose G does not contain a negative cycle
- Since $D[v, n] < D[v, n-1]$, the shortest $s \rightsquigarrow v$ path that caused this update has exactly n edges
- By pigeonhole principle, path must contain a repeated node, let the cycle between two successive visits to the node be P
- If P has non-negative weight, removing it would give us a shortest path with less than n edges $\Rightarrow\Leftarrow$



Problem Reduction

- Now we know how to detect negative cycles on a shortest path from s to some node v.
- How do we detect a negative cycle anywhere in G?
- **Reduction**. Given graph G, add a source s and connect it to all vertices in G with edge weight 0. Let the new graph be G'
- **Claim**. G has a negative cycle iff G' has a negative cycle from s to some node v.
- **Proof**. $\Rightarrow$ If G has a negative cycle, then this cycle lies on the shortest path from s to a node on the cycle in G'
- $\Leftarrow$ If G' has a negative cycle on a shortest path from s to some node, then that node is on a negative cycle in G

Acknowledgments

- Some of the material in these slides are taken from
    - Kleinberg Tardos Slides by Kevin Wayne (https://www.cs.princeton.edu/~wayne/kleinberg-tardos/pdf/04GreedyAlgorithmsI.pdf)
    - Jeff Erickson's Algorithms Book (http://jeffe.cs.illinois.edu/teaching/algorithms/book/Algorithms-JeffE.pdf)