# Last Topics in Dynamic Programming: Knapsack, and Shortest Paths Revisited

# Admin

- Assignment 5 largely graded, back soon
  - You did well!
- Midterm review tomorrow at 7PM
  - Bring questions!  Mostly Q&A
- 24 hour midterm starts Wed Oct 28 at 10:40 AM
- No office hours Wed or Thur this week (Mon still on)

# DP Explanations

- Some recipe points do not require explanation (subproblem, memoization data structure)

- Some should have a 1 sentence (maybe 2 sentence) explanation: the recurrence, the base cases

- Some it depends a bit on the context: final output, evaluation order (sometimes these are VERY obvious, but sometimes they can be tricky.  Best to include a very short sentence)

- When in doubt: is your rationale for a choice completely obvious?  If not, should probably write something

# Knapsack Problem

- **Problem**. Pack a knapsack to maximize total value
- There are $n$ items, each with weight $w_i$ and value $v_i$, where $v_i, w_i > 0$. Weights must be integers!
- Knapsack has total capacity $C$
- Output: subset $S$ of items fit in the knapsack, that is, $\sum_{i \in S} w_i \leq C$ and maximizes the total value $\sum_{i \in S} v_i$
- **Assumption**. All **weights** are integral

# Idea #1: Capacity Table

- Let's create a table $T$ where $T[c]$ contains the optimal solution using capacity $\leq c$.
- Optimal solution: $T[C]$
- How do come up with a recurrence?
- Not obvious with just  capacities

| capacity | items | value |
|---|---|---|
| c = 0 | | $0 |
| c = 1 | $2/1kg | $2 |
| c = 2 | $2/1kg $1/1kg | $3 |
| c = 3 | $2/1kg $2/2kg | $4 |
| c = 4 | $10/4kg | $10 |
| c = 5 | $2/1kg $10/4kg | $12 |
| c = 6 | $2/1kg $10/4kg $1/1kg | $13 |
| c = 7 | [activity] | |
| ... | ... | ... |

Table for the item set

$4/12kg $2/1kg $10/4kg $1/1kg $2/2kg

# DP: Right Recurrence

- What else can we keep track of to get a recurrence with an optimal substructure?

- Let $T[j, c]$ be the optimal solution using items $[1,\ldots j]$ with total capacity $\leq c$

- What are our two cases?
- Case 1. If item $j$ is not in the optimal solution
    - $T[j, c] = T[j-1, c]$
- Case 2. If item $j$ is in the optimal solution then
    - $T[j, c] = v_j + T[j-1, c-w_j]$

# Recurrence & Memoization

- **Base case**.
  - $T[j, c] = 0$ if $j = 0$, or $c = 0$
- **Recurrence**
  - For $j, c > 0$,
    $T[j, c] = \max\{T[j-1,\ c],\ v_j + T[j-1,\ c - w_j]\}$
- Now that we have the recurrence, we can memoize and figure out the evaluation order
- We will store $T[j, c]$ for $1 \le j \le n, \quad 1 \le c \le C$ in a 2D array
- **Evaluation order?**
  - Row by row (i.e. item by item: for each item fill in each capacity one by one)
- **Final answer?** $T[n, c]$

# Running Time

- Takes $O(1)$ to fill out a cell, $O(nC)$ total cells
- Is this polynomial?  By which I mean polynomial in the *size of the input*
- How large is the input to knapsack?
  - Store $n$ items, plus need to store $C$
  - $O(n + \log C)$
- Is $O(nC)$ polynomial?
  - No!
  - “Pseudopolynomial” - polynomial in the *value* of the input
- To think about: does this work if the weights are not integers?

# Shortest Path Problem

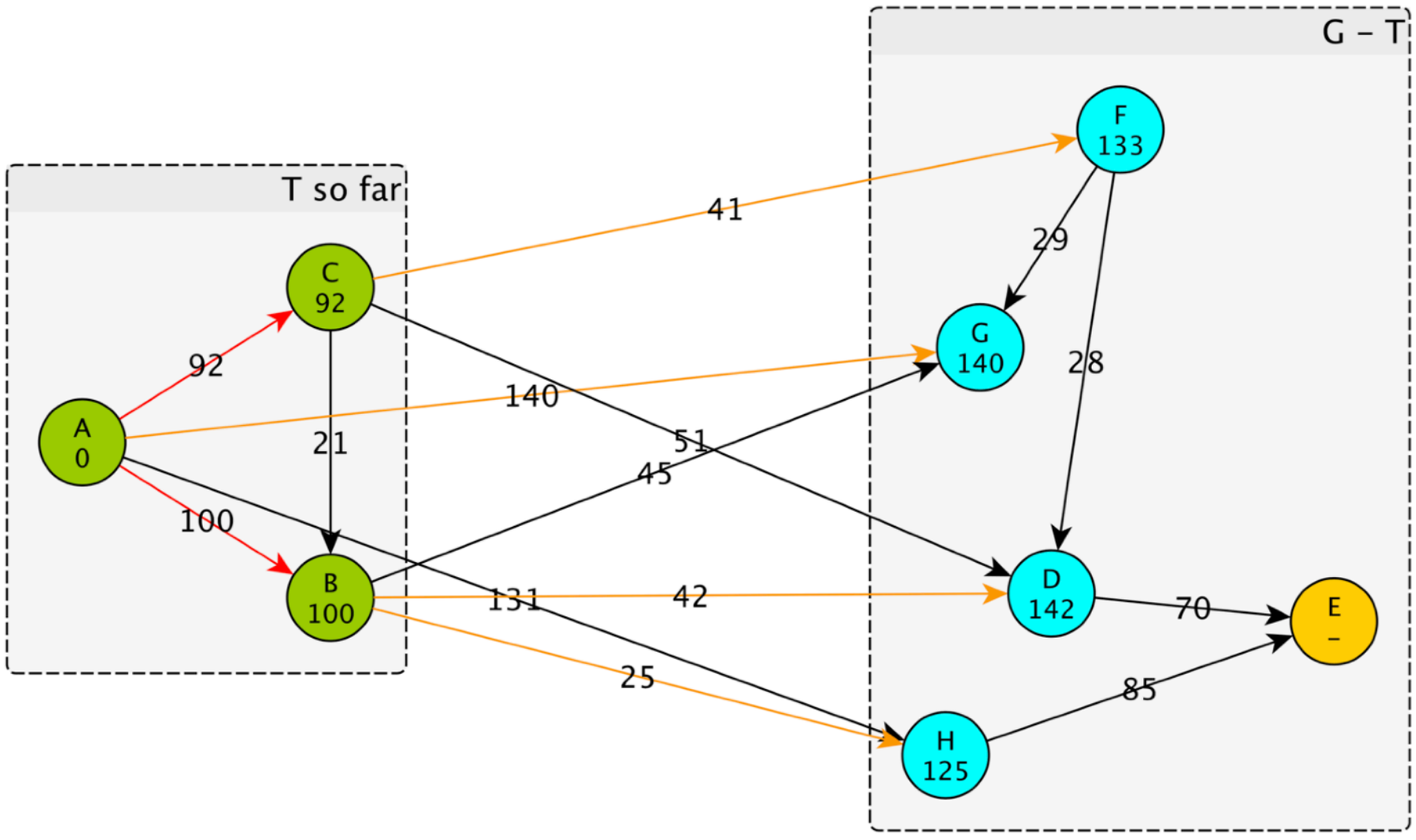
- **Single-Source Shortest Path Problem**.
  Given a directed graph $G = (V, E)$ with edge weights $w_e$ on each $e \in E$ and a a source node $s$, find the shortest path from $s$ to to all nodes in $G$.

- **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

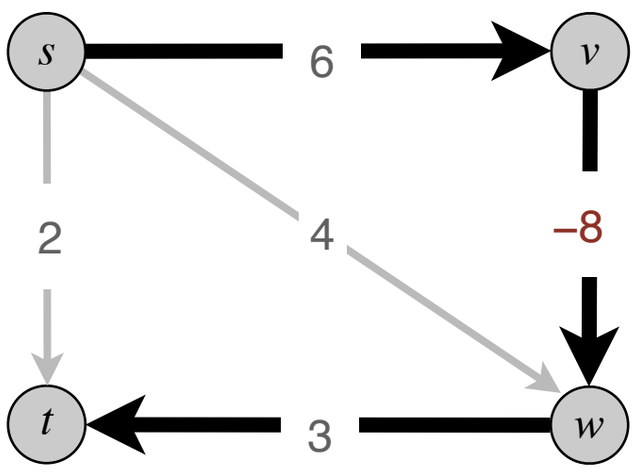
# Remember Dijkstra's Algorithm?




Estimate at vertex $v$ is the weight of shortest path in $T$ followed by a single edge from $T$ to $G - T$

# 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)

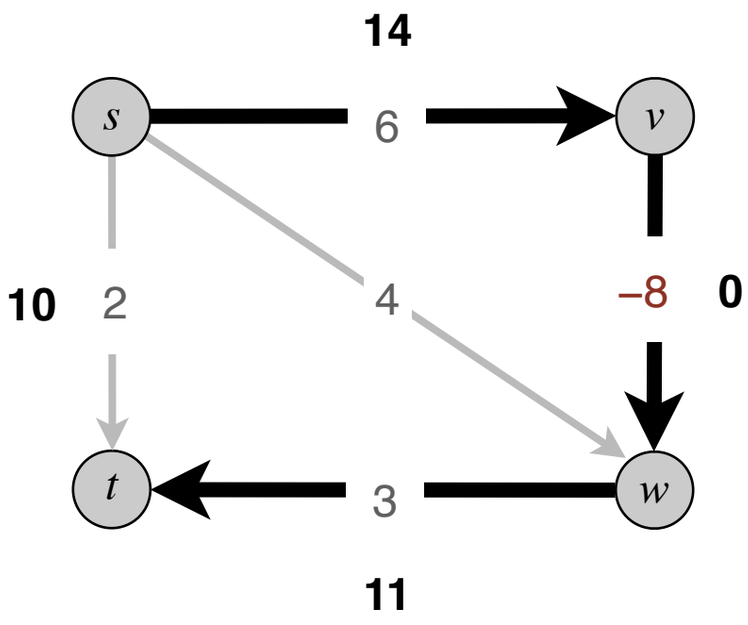



Dijkstra's will fi
But the shortes

We fixed it later—why is this not OK in general??

# Negative Weights: Failed Attempt

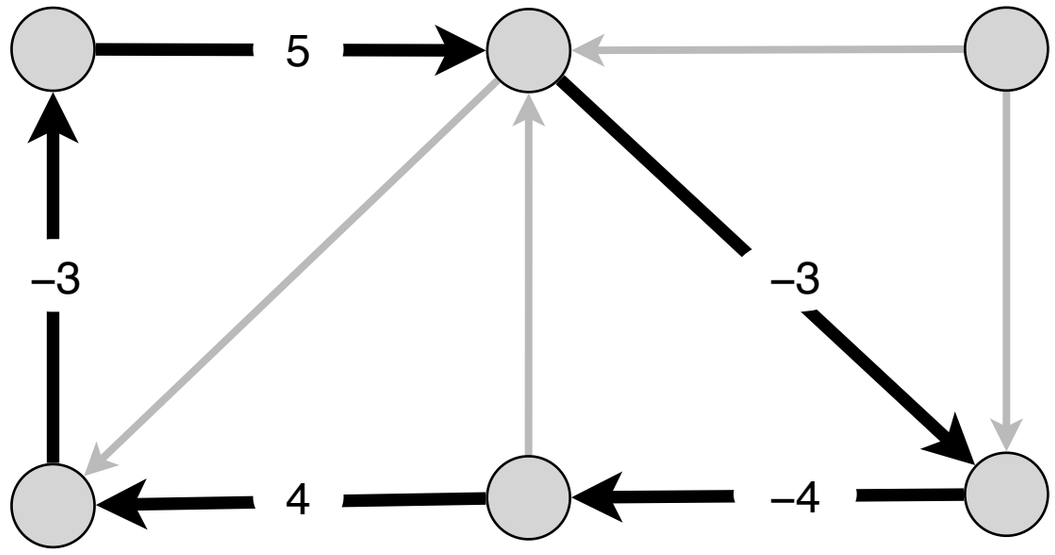
- What if we add a large enough constant $C$ such that all weights become positive
  - $w'_{ij} = w_{ij} + C > 0$
  - Run Dijkstra's algorithm based with $w'$
- Does this give us the shortest path in the original graph?




Adding C = 8 to all weights does not work

# 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

$\Rightarrow\Leftarrow$ ∎

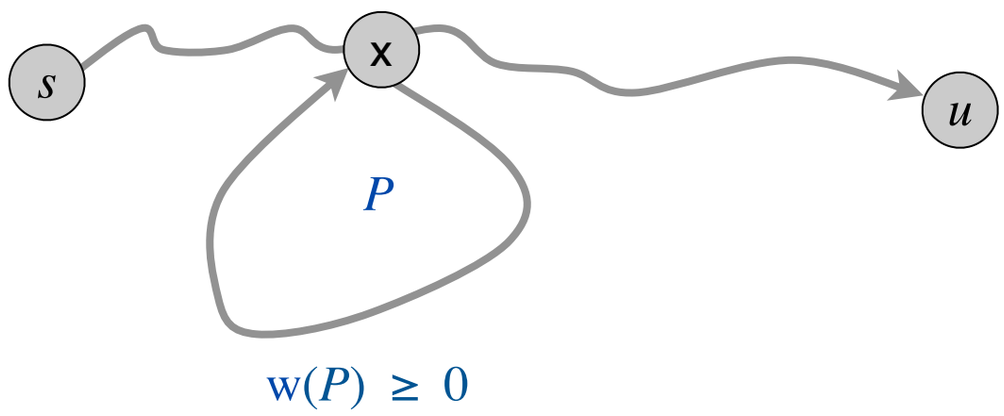
- **Assumption.** $G$ has no negative cycle.
- Later in the lecture: how can we detect whether the input graph $G$ contains a negative cycle?

# Dynamic Programming Approach

- First step to a dynamic program? Recursive formulation
  - Subproblem with an “optimal substructure”
- **Structure of the problem**. Interested in optimal cost path (can have any length)
    - Easier to build on subproblems if we 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. ■



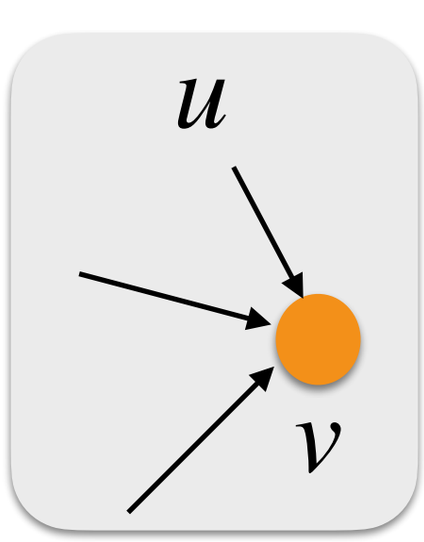
# Shortest Paths: Dynamic Program

- **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 $v$
  - $D[v, n-1]$
- How do we formulate the **recurrence**?
  - **Case 1**. Shortest path to $v$ uses exactly $i$ edges
  - **Case 2**. Shortest path to $v$ uses less than $i$ edges (that is, uses $\leq i-1$ edges)

# Shortest Paths: Recurrence

- **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.**

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




- 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
- **Base cases.** $D[s, i] = 0$ for any $i$ and $D[v,0] = \infty$ for any $v \neq s$
- **Final answer** for shortest path cost to node $v$: $D[v, n-1]$
- **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**.
  - $i : 1 \rightarrow n-1$ (column major order)
  - Starting from $s$, the row of vertices can be in any order

# Bellman-Ford: 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 $O(nm)$

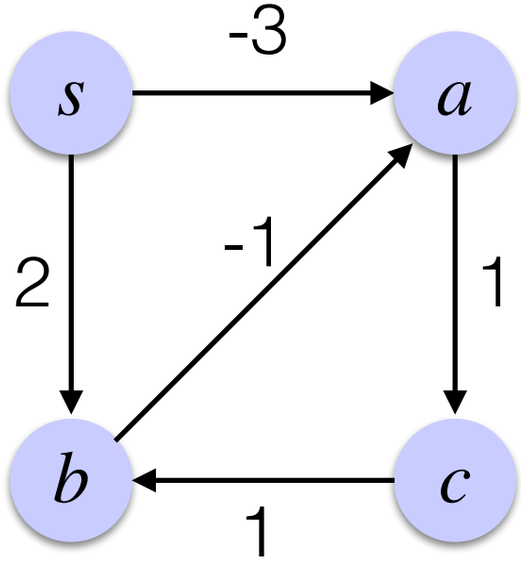
- **Shortest-Path Summary.** Assuming there are no negative cycles in $G$, we can compute the shortest path from $s$ to all nodes in $G$ in $O(nm)$ time using the Bellman-Ford-Moore algorithm

# Dynamic Programming Shortest Path:

## Bellman-Ford-Moore Example

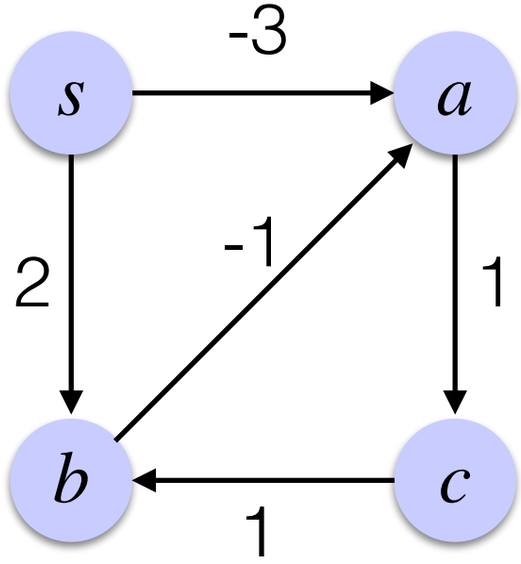
- $D[s, i] = 0$ for any $i$
- $D[v,0] = \infty$ for any $v \neq s$

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



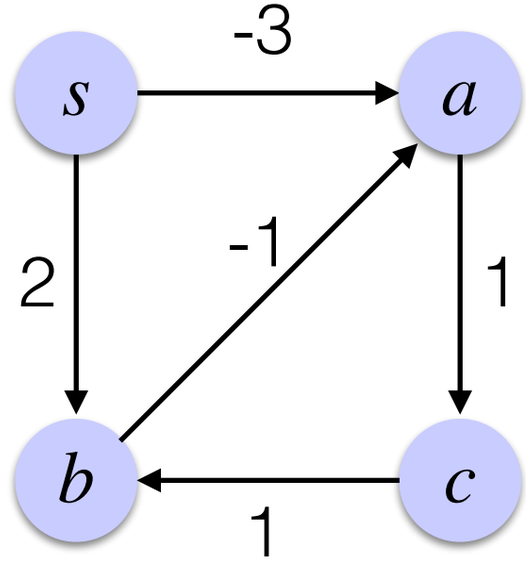
- $D[v,1] = \min\{D[v,0], \min_{u,v \in E} \{D[u,0] + w_{uv}\}$

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



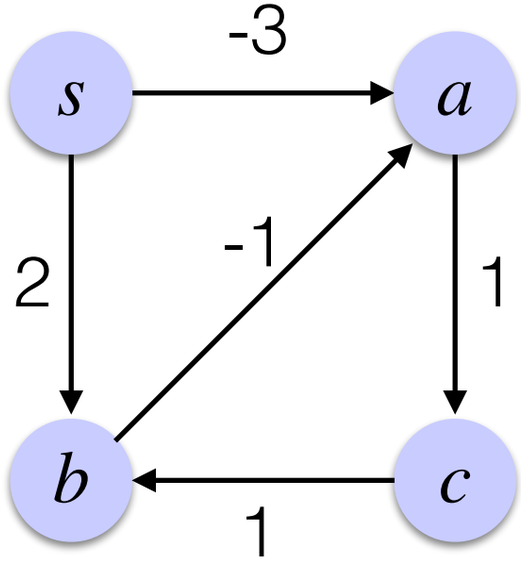
- $D[v,1] = \min\{D[v,0], \min_{u,v \in E} \{D[u,0] + w_{uv}\}$

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



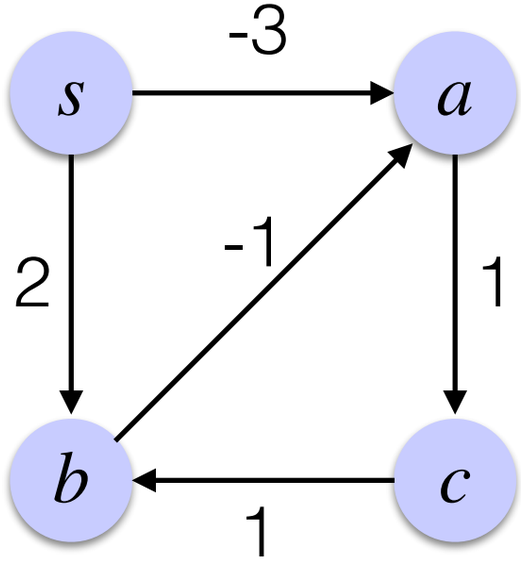
- $D[v,1] = \min\{D[v,0], \min_{u,v \in E} \{D[u,0] + w_{uv}\}$

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



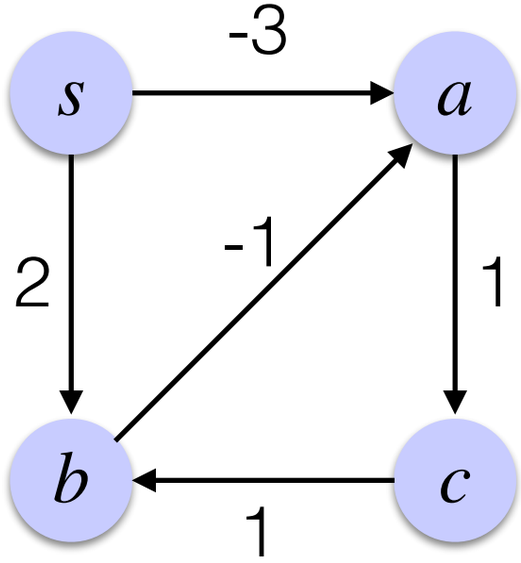
- $D[v,1] = \min\{D[v,0], \min_{u,v \in E} \{D[u,0] + w_{uv}\}$

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



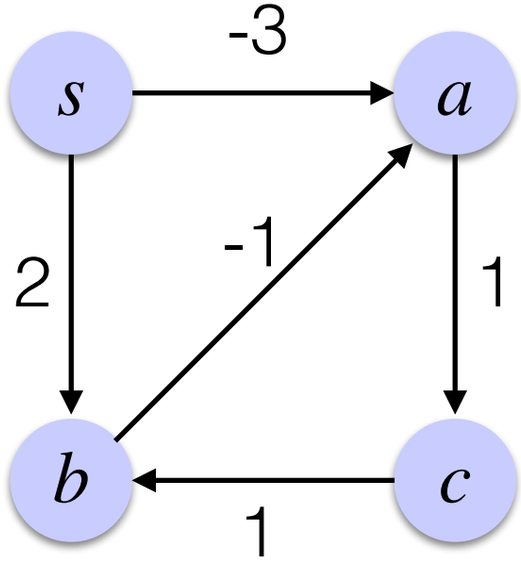
- $D[v,2] = \min\{D[v,1], \min_{u,v \in E} \{D[u,1] + w_{uv}\}$

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



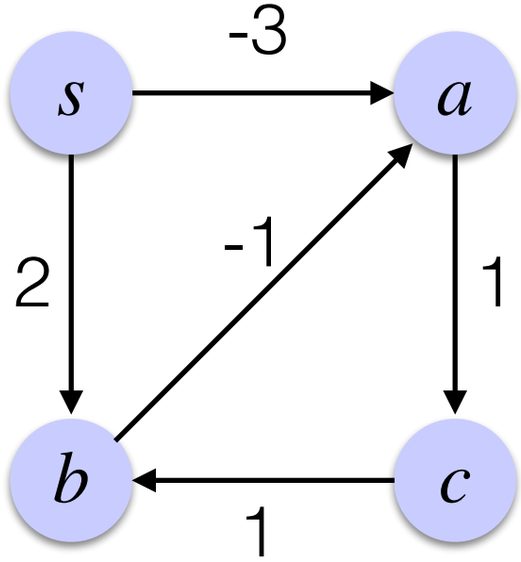
- $D[v,2] = \min\{D[v,1], \min_{u,v \in E} \{D[u,1] + w_{uv}\}$

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



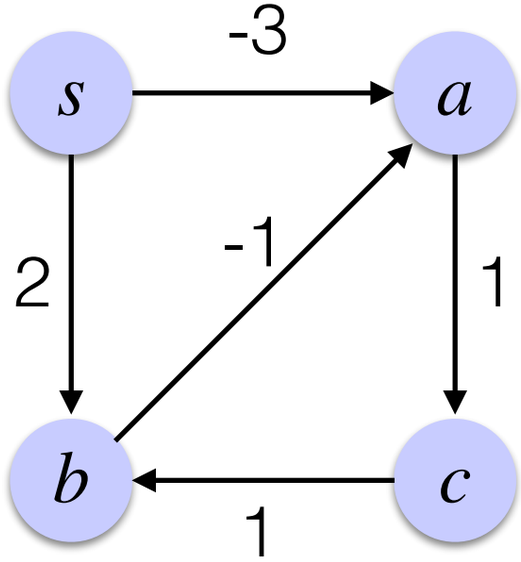
- $D[v,2] = \min\{D[v,1], \min_{u,v \in E} \{D[u,1] + w_{uv}\}$

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



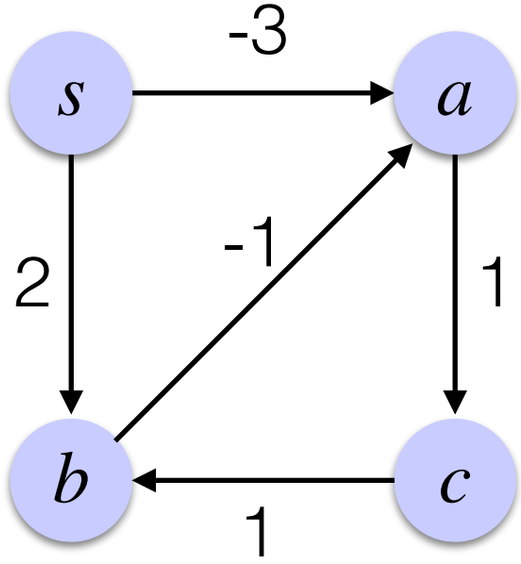
- $D[v,2] = \min\{D[v,1], \min_{u,v \in E} \{D[u,1] + w_{uv}\}$

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



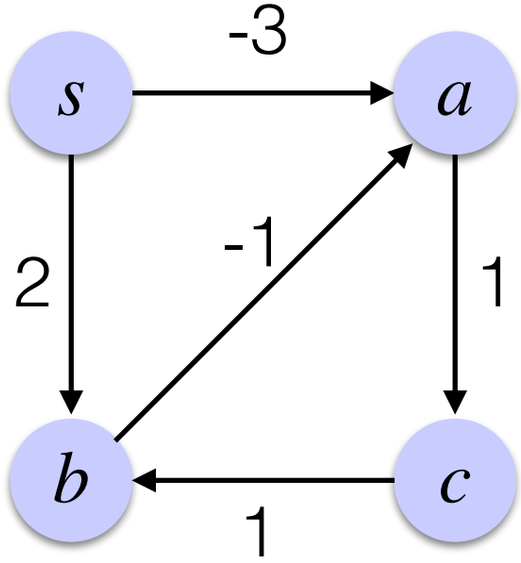
- $D[v,3] = \min\{D[v,2], \min_{u,v \in E} \{D[u,2] + w_{uv}\}$

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



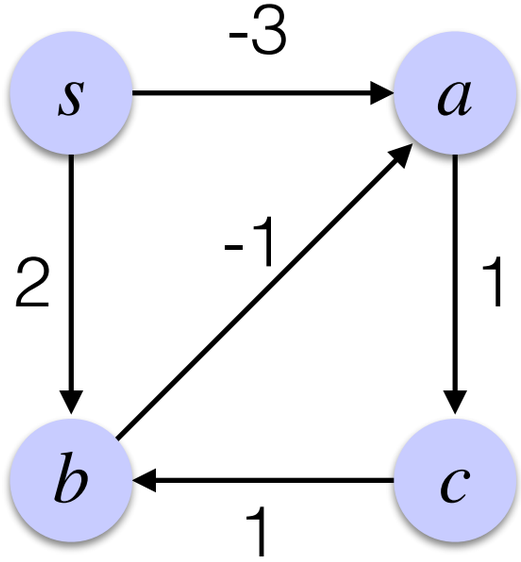
- $D[v,3] = \min\{D[v,2], \min_{u,v \in E} \{D[u,2] + w_{uv}\}$

| | 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 |



# 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)