CPSC 365 / ECON 365:

Algorithms

Lecture 12: Dynamic Programming 2

Andre Wibisono

Yale University

March 3, 2022

Plan

Last time: Dynamic Programming

- Fibonacci Numbers
- DAG and Topological Sort

Today:

- Dynamic Programming 2
- 4:30 – 5:30 pm: PS review with TF (virtual)

Notes:

- Thu 03/10: Midterm 1 (in class)
  - You can bring 1-page (single-side) handwritten sheet
  - Cover all material until dynamic programming (next Tuesday)
- Review sessions Tue 03/08, Wed 03/09

Plan

Shortest Path in a DAG

Shortest Path in a General Weighted Graph

Detecting Negative Cycles

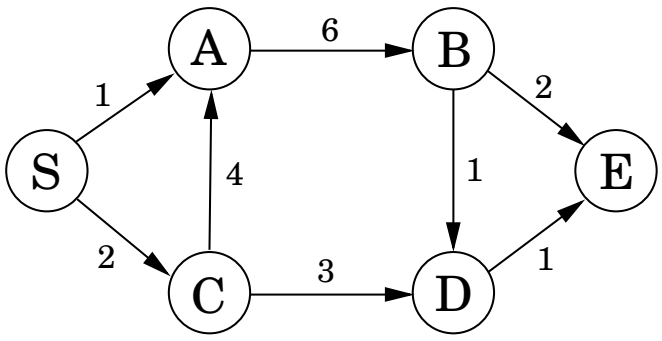
Longest Increasing Subsequence

Shortest Path in a DAG

Let $G = (V, E)$ be a DAG with *weights* $w(e) \in \mathbb{R}$ on each edge.

- **DAG:** G is directed and acyclic
- Weights $w(e) \in \mathbb{R}$ can be positive or negative
- Shortest distance $\delta(s, v)$ from s to v is:

$$\delta(s, v) = \min_{P:\ s \to v} \sum_{e \in P} w(e).$$

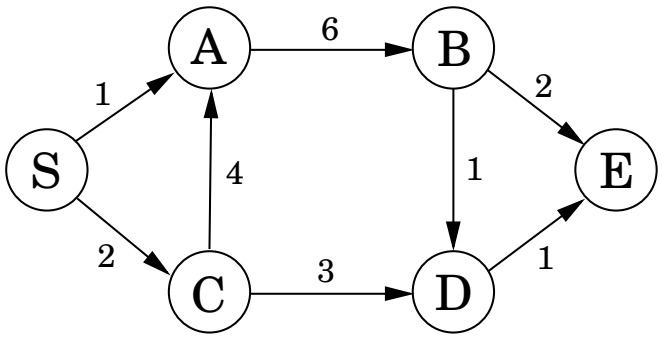


Shortest Path in a DAG

How to compute $\delta(s, v)$?

- Observation: Each path from s to v must come from one of the *incoming* edges to v.
- So the shortest path to v must be composed of a shortest path to a neighbor u, followed by the edge $e = (u, v)$.

$$\delta(S, D) = \min\{\delta(S, B) + 1,\ \delta(S, C) + 3\}$$

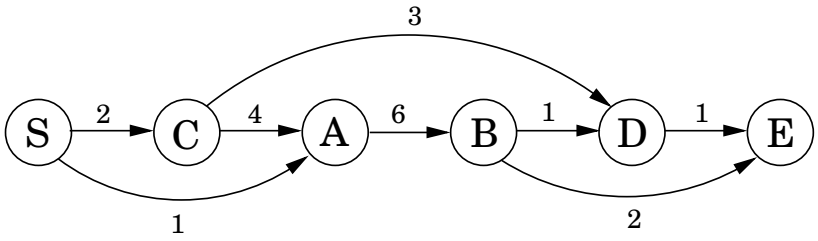


Shortest Path in a DAG

How to compute $\delta(s, v)$?

- Let's sort the vertices in a topological ordering.
- Then compute $\delta(s, v)$ in increasing order of v.
- If $s > v$, then there is no path from s to v, so $\delta(s, v) = \infty$.
- If $s < v$: A shortest path from s to v must pass through one of the neighbors of v. So:

$$\delta(s, v) = \min_{u:\ (u,v) \in E} \delta(s, u) + w(u, v)$$



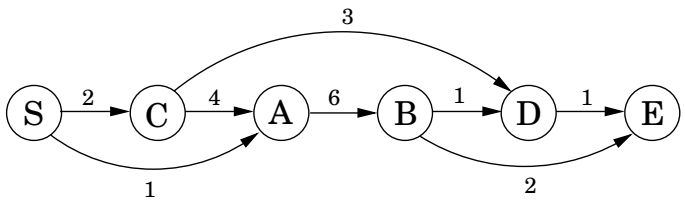
Shortest Path in a DAG

ShortestPath(G):

```
Sort vertices in topological order
Initialize dist[s] = 0, and dist[v] = ∞ for all v ∈ V
for each v ∈ V \ {s} (in increasing order):
    dist[v] = min_{u : (u,v)∈E} dist[u] + w(u, v)
```

Running time:

- Topological sort (e.g. via DFS): $O(|V| + |E|)$
- Computing `dist`$[v]$: $O(d_v)$, where $d_v =$ incoming degree of v
  $\Rightarrow$ Computing `dist`: $\sum_{v \in V} d_v = |E|$ (by the handshake lemma)
- So total running time is $O(|V| + |E|)$.



Plan

Shortest Path in a DAG

Shortest Path in a General Weighted Graph

Detecting Negative Cycles

Longest Increasing Subsequence

Shortest Path in a General Weighted Graph

Let $G = (V, E)$ be a directed graph with edge weights $w(e) \in \mathbb{R}$.

- Weights $w(e) \in \mathbb{R}$ can be positive or negative
- G may have a cycle (G is not necessarily a DAG)
- Shortest distance $\delta(s, v)$ from s to v is:

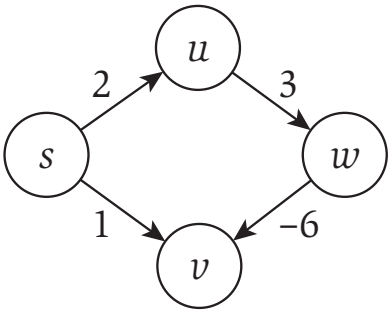
$$\delta(s, v) = \min_{P:\ s \to v} \sum_{e \in P} w(e).$$

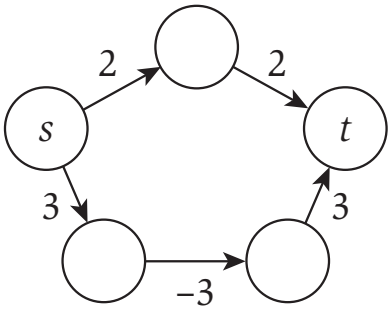
- Issue 1: Greeedy doesn't work if there is negative weight
- Issue 2: What if there is a negative cycle?

Issue 1: Failure of Greedy

Let $G = (V, E)$ be a directed graph with edge weights $w(e) \in \mathbb{R}$

- If some $w(e) < 0$, greedy methods (e.g. Dijkstra) can fail



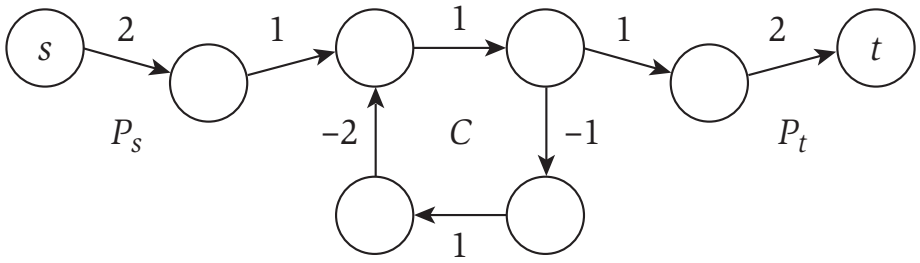


Issue 2: Negative Cycle

A negative cycle is a cycle C with negative total weights:

$$w(C) = \sum_{e \in C} w(e) < 0.$$

- If there is a negative cycle C, and it is on the way from s to t, then we can loop around C infinitely many times to reduce cost to $-\infty$.



Graph with No Negative Cycles

Let $G = (V, E)$ be a directed weighted graph on $n = |V|$ vertices.

- Assume G has no negative cycles (can detect later)
- **Lemma:** If G has no negative cycles, then for any $s, t \in V$, there is a shortest path from s to t that is simple (i.e. does not repeat vertices), and hence has at most $n - 1$ edges.

  Proof: Let P be a shortest path from s to t (i.e. with minimum total weight), which uses the *fewest* number of edges.
  We claim P is simple. Suppose not, so P repeats a node v. Then we can remove the part of P between consecutive visits to v, resulting in a path with fewer number of edges, and the total weight cannot increase (since the cycle has nonnegative weight). This contradicts the choice of P. Therefore, P is simple. □

Computing Shortest Paths

Let $G = (V, E)$ be directed, weighted with no negative cycles.

- Given $s \in V$, want to compute $\delta(s, v)$ for all $v \in V$.
- Recall Lemma: $\forall\, v$, $\exists$ shortest path P_{sv} with $\leq n - 1$ edges.
- Let's define subproblems, for all $v \in V$ and $0 \leq i \leq n - 1$:

$$\texttt{dist}(v, i) = \text{length of shortest path from } s \text{ to } v \text{ using } \leq i \text{ edges}$$

- Then what we want to find is:

$$\delta(s, v) = \texttt{dist}(v, n - 1).$$

Recurrence among Subproblems

$\texttt{dist}(v, i) =$ length of shortest path from s to v using $\leq i$ edges

Consider a shortest path P from s to v that uses $\leq i$ edges.

1. If P uses $\leq i - 1$ edges, then

$$\texttt{dist}(v, i) = \texttt{dist}(v, i - 1)$$

2. If P uses i edges, and the last edge is (u, v), then

$$\texttt{dist}(v, i) = \texttt{dist}(u, i - 1) + w(u, v).$$

Therefore, the recurrence is, for $i > 0$:

$$\texttt{dist}(v, i) = \min\{\texttt{dist}(v, i-1), \min_{u:\ (u,v)\in E} \texttt{dist}(u, i-1)+w(u, v)\}.$$

Bellman-Ford

Bellman-Ford: Dynamic programming to compute shortest paths. Assume G is directed, weighted, with no negative cycles.

BellmanFord(G):

$\texttt{dist}(s, 0) = 0$, $\texttt{dist}(v, 0) = \infty$ for all $v \neq s$

for $i = 1, \ldots, n-1$:

  for $v \in V$:

    $$\texttt{dist}(v, i) = \min\{\texttt{dist}(v, i-1), \ldots \min_{u:\,(u,v)\in E} \texttt{dist}(u, i-1) + w(u, v)\}$$

return $\texttt{dist}(v, n-1)$ for all $v \in V$

Notes:

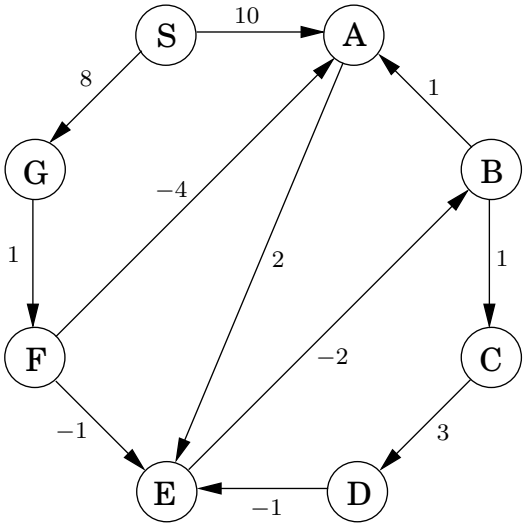
- This gives the *length* of the path. To reconstruct the path, we can add a `prev` pointers like in Dijkstra.
- By Lemma, $\texttt{dist}(v, n-1) = \texttt{dist}(v, n)$ for all $v \in V$.

Bellman-Ford Running Time

- We are filling in a table of size $|V| \times n$.
- Each entry $\texttt{dist}(v, i)$ takes $O(d_v)$, $d_v =$ in-degree of v.
- For each $i = 0, 1, \ldots, n-1$, computing $\{\texttt{dist}(v, i) : v \in V\}$ takes on the order of $\sum_{v \in V} d_v = m$, where $m = |E|$ is number of edges.
- So the running time of Bellman-Ford is:

$$O(n \cdot m)$$

Bellman-Ford Example



	Iteration							
Node	0	1	2	3	4	5	6	7
S	0	0	0	0	0	0	0	0
A	∞	10	10	5	5	5	5	5
B	∞	∞	∞	10	6	5	5	5
C	∞	∞	∞	∞	11	7	6	6
D	∞	∞	∞	∞	∞	14	10	9
E	∞	∞	12	8	7	7	7	7
F	∞	∞	9	9	9	9	9	9
G	∞	8	8	8	8	8	8	8

Plan

Shortest Path in a DAG

Shortest Path in a General Weighted Graph

Detecting Negative Cycles

Longest Increasing Subsequence

Detecting Negative Cycles

How to detect if G has a negative cycle?

- Recall Lemma: If G has no negative cycles, then for all v, there is a shortest path $P_{s \to v}$ with $\leq n-1$ edges.
  - The latter is equivalent to: $\texttt{dist}(v, n) = \texttt{dist}(v, n-1)$.
- Therefore, Lemma says: If G has no negative cycles, then $\texttt{dist}(v, n) = \texttt{dist}(v, n-1)$ for all $v \in V$.
- Equivalent to the contrapositive: If there is $v \in V$ with $\texttt{dist}(v, n) \neq \texttt{dist}(v, n-1)$, then G has a negative cycle
- So to detect negative cycle, can do the following:

Run Bellman-Ford n times, and check whether there is $v \in V$ that changes: $\texttt{dist}(v, n) \neq \texttt{dist}(v, n-1)$.