Shortest Paths (w/ negative weights).

Input: directed n-node m-edge graph
\[ G = (V, E, c) \]
each edge \((i, j) \in E\) has cost \(C_{ij}\) \(\gamma\) which might be negative.
two nodes \(s, t \in V\)
Output: (the length of) the shortest path from \(s\) to \(t\) in \(G\).

a path \(P\) from \(s\) to \(t\) that minimizes \(\sum_{e \in P} C_e\).

Failed attempt 1. Dijkstra's algorithm no longer works. (see HW3).

\[ \begin{array}{ccc}
S=1 & \rightarrow & E=1 \\
1 & \rightarrow & t=0 \\
\end{array} \]
Dijkstra will return \(S-t\) shortest path is \(S-u-t\).

Failed attempt 2. Add a large number \(M\) to all edge weights
st. \(C_e \geq 0\) after this change.,
and then run Dijkstra.

\[ \begin{array}{ccc}
S=2 & \rightarrow & E=3 \\
2 & \rightarrow & U=2+5 \\
3 & \rightarrow & Y=3+5 \\
\end{array} \]
Shortest \(S-t\) path is \(S \rightarrow x \rightarrow y \rightarrow t\) with length 3.

If we add \(M=5\) to all edge weights
the shortest path is now
length \((S \rightarrow u \rightarrow t) = 4 + 5 \cdot 2 = 14\)
\(\) length \((S \rightarrow x \rightarrow y \rightarrow t) = 3 + 5 \cdot 3 = 18\)

Remark. \(\) Q: What is the length of the shortest path from \(S\) to \(t\) ?
\[ \begin{array}{ccc}
S & \rightarrow & t \\
1 & \rightarrow & E=1 \\
\end{array} \]
A: \(-\infty\)

Why negative weights?

For example, \(C_{ij}\) could represent the cost
of buying an item from agent \(i\) and sell it to agent \(j\).

negative cost = profit.
The Bellman-Ford Algorithm.

single-source shortest path algorithm, i.e., compute the shortest paths from \( s \) to every other node in the graph.

We assume there is no negative cycles in \( G \).

Claim 6.22. If \( G \) has no negative cycles, then there exists a shortest path from \( s \) to \( t \) that is simple, and thus has at most \( n-1 \) edges.

Proof. Fix a shortest path \( P \) from \( s \) to \( t \). If \( P \) visits a node more than once, \( P \) has a cycle. Removing this cycle from \( P \) does not make \( P \) longer.

Dynamic programming. (slightly different from the Bellman-Ford presented in Chapter 6.8 of the textbook on page 290)

\[ \text{OPT}(i,v) : \text{the length of the shortest path from } s \text{ to } v \]
\[ \forall 0 \leq i \leq n-1 \text{ using at most } i \text{ edges.} \]
\[ \forall v \in G . \]

Final answer = \( \text{OPT}(n-1,v) \)

(i.e., the length of the shortest \( s \rightarrow v \) path)

Recursive formula:

\[ \text{OPT}(i,v) = \begin{cases} 0 & \text{if } v = s \\ +\infty & \text{otherwise} \end{cases} \]

when \( i = 0 \): for each edge \((u,v) \in E \)

\[ \text{OPT}(i,v) = \min \left( \text{OPT}(i-1,v), \text{OPT}(i-1,u) + w(u,v) \right) \]

1: "at most \( i \) edges" gives the same solution as "at most \( i-1 \) edges".

2: we first go from \( s \) to \( u \) using \( \leq i-1 \) edges and then take the edge \( u \rightarrow v \).
Runtime

\(\tilde{O}(mn)\)

- \(n-1\) iterations (or \(n\) iterations if need to detect negative cycles)
- each iteration takes \(O(m)\) time to loop over every edge.

Slower than Dijkstra's algorithm, but allow negative weight.

Improving the memory requirement: \(O(mn)\) additional space → \(O(n)\).

Idea: Instead of \(OPT[i][v]\), we use \(OPT[v]\) and update it repeatedly.

Algorithm 1: Bellman-Ford Algorithm

\[
\text{Input}: \text{An } n\text{-node directed graph } G \text{ with edge weights } w \text{ and a source node } s. \\
\text{Output}: \text{The length of the shortest path from } s \text{ to every node in } G. \\
\text{for } u = 1 \text{ to } n \text{ do} \\
\hspace{1cm} d[u] \leftarrow +\infty; \\
\hspace{1cm} d[s] \leftarrow 0; \\
\text{for } i = 1 \text{ to } n \text{ do} \\
\hspace{1cm} \text{change} \leftarrow 0; \\
\hspace{2cm} \text{for each edge } e = (u, v) \in E \text{ with weight } w_e \text{ do} \\
\hspace{3cm} \text{if } d[u] + w_e < d[v] \text{ then} \\
\hspace{4cm} d[v] \leftarrow d[u] + w_e; \\
\hspace{4cm} \text{change} \leftarrow 1; \\
\hspace{2cm} \text{if change} = 0 \text{ then break; } \\
\hspace{2cm} \text{if change} = 1 \text{ and } i = n \text{ then} \\
\hspace{3cm} \text{return error } "G \text{ contains a negative cycle}"; \\
\text{return } d
\]

\(d[v] = OPT[v]\)

(see page 295 of the textbook for a proof of the correctness of this algorithm)