Single-Source All-Destinations
Shortest Paths With Negative Costs

• Directed weighted graph.
• Edges may have negative cost.
• No cycle whose cost is < 0.
• Find a shortest path from a given source vertex \( s \) to each of the \( n \) vertices of the digraph.

Single-Source All-Destinations
Shortest Paths With Negative Costs

• Dijkstra’s \( O(n^2) \) single-source greedy algorithm doesn’t work when there are negative-cost edges.
• Floyd’s \( \Theta(n^3) \) all-pairs dynamic-programming algorithm does work in this case.
Bellman-Ford Algorithm

- Single-source all-destinations shortest paths in digraphs with negative-cost edges.
- Uses dynamic programming.
- Runs in $O(n^3)$ time when adjacency matrices are used.
- Runs in $O(ne)$ time when adjacency lists are used.

Decision Sequence

- To construct a shortest path from the source to vertex $v$, decide on the max number of edges on the path and on the vertex that comes just before $v$.
- Since the digraph has no cycle whose length is $< 0$, we may limit ourselves to the discovery of cycle-free (acyclic) shortest paths.
- A path that has no cycle has at most $n-1$ edges.
Problem State

- Problem state is given by \((u,k)\), where \(u\) is the destination vertex and \(k\) is the max number of edges.
- \((v,n-1)\) is the state in which we want the shortest path to \(v\) that has at most \(n-1\) edges.

Cost Function

- Let \(d(v,k)\) be the length of a shortest path from the source vertex to vertex \(v\) under the constraint that the path has at most \(k\) edges.
- \(d(v,n-1)\) is the length of a shortest unconstrained path from the source vertex to vertex \(v\).
- We want to determine \(d(v,n-1)\) for every vertex \(v\).
Value Of $d(\ast,0)$

- $d(v,0)$ is the length of a shortest path from the source vertex to vertex $v$ under the constraint that the path has at most 0 edges.

- $d(s,0) = 0$.
- $d(v,0) = \infty$ for $v \neq s$.

Recurrence For $d(\ast,k)$, $k > 0$

- $d(v,k)$ is the length of a shortest path from the source vertex to vertex $v$ under the constraint that the path has at most $k$ edges.
- If this constrained shortest path goes through no edge, then $d(v,k) = d(v,0)$. 
Recurrence For $d(*,k)$, $k > 0$

- If this constrained shortest path goes through at least one edge, then let $w$ be the vertex just before $v$ on this shortest path (note that $w$ may be $s$).

  $s \rightarrow w \rightarrow v$

  - We see that the path from the source to $w$ must be a shortest path from the source vertex to vertex $w$ under the constraint that this path has at most $k-1$ edges.
  - $d(v,k) = d(w,k-1) + \text{length of edge } (w,v)$.

- We do not know what $w$ is.
- We can assert
  - $d(v,k) = \min \{d(w,k-1) + \text{length of edge } (w,v)\}$, where the $\min$ is taken over all $w$ such that $(w,v)$ is an edge of the digraph.
- Combining the two cases considered yields:
  - $d(v,k) = \min \{d(v,0), \min \{d(w,k-1) + \text{length of edge } (w,v)\}\}$
Pseudocode To Compute $d(*,*)$

// initialize $d(*,0)$
\[
d(s,0) = 0;
\]
\[
d(v,0) = \text{infinity}, \ v \neq s;
\]

// compute $d(*,k)$, $0 < k < n$
\[
\text{for (int } k = 1; k < n; k++)
\{
\]
\[
d(v,k) = d(v,0), 1 \leq v \leq n;
\]
\[
\text{for (each edge (u,v))}
\]
\[
d(v,k) = \min\{d(v,k), d(u,k-1) + \text{cost}(u,v)\}
\]
\[
\}
\]

Complexity

- \text{Theta}(n) to initialize $d(*,0)$.
- \text{Theta}(n^2) to compute $d(*,k)$ for each $k > 0$ when adjacency matrix is used.
- \text{Theta}(e) to compute $d(*,k)$ for each $k > 0$ when adjacency lists are used.
- Overall time is \text{Theta}(n^3) when adjacency matrix is used.
- Overall time is \text{Theta}(ne) when adjacency lists are used.
- \text{Theta}(n^2) space needed for $d(*,*)$. 
Let \( p(v,k) \) be the vertex just before vertex \( v \) on the shortest path for \( d(v,k) \).
- \( p(v,0) \) is undefined.
- Used to construct shortest paths.

**Example**

Source vertex is 1.
Observations

- \(d(v,k) = \min\{d(v,0), \min\{d(w,k-1) + \text{length of edge } (w,v)\}\}\)
- \(d(s,k) = 0\) for all \(k\).
- If \(d(v,k) = d(v,k-1)\) for all \(v\), then \(d(v,j) = d(v,k-1)\), for all \(j \geq k-1\) and all \(v\).
- If we stop computing as soon as we have a \(d(*,k)\) that is identical to \(d(*,k-1)\) the run time becomes
  - \(O(n^3)\) when adjacency matrix is used.
  - \(O(ne)\) when adjacency lists are used.
Observations

• The computation may be done in-place.
  \[ d(v) = \min\{d(v), \min\{d(w) + \text{length of edge (w,v)}\}\} \]
  instead of
  \[ d(v,k) = \min\{d(v,0), \min\{d(w,k-1) + \text{length of edge (w,v)}\}\} \]
• Following iteration \( k \), \( d(v,k+1) \leq d(v) \leq d(v,k) \)
• On termination \( d(v) = d(v,n-1) \).
• Space requirement becomes \( O(n) \) for \( d(*) \) and \( p(*) \).