Shortest Paths

- Finding the Shortest Paths in a graph arises in many different application:
  - Transportation Problems: Finding the cheapest way to travel between two locations.
  - Motion Planning: What is the most natural way to travel a robot in an environment.
  - Communication Problems:
    - The shortest set of hubs to get a message between two nodes in a network.
    - Which two locations are farthest apart, i.e., what is the diameter of a network.
The Single Source Shortest Path

- We are given a graph \( G=(V,E) \) and a real weight function \( w \) from \( E \) to \( R \), define the weight of a path \( p=<v_0, v_1, \ldots, v_k> \) as the sum of weight of its edges:
  \[
  w(p) = \sum_{i=1}^k w(v_{i-1}, v_i).
  \]

- We define the shortest-path weigh between \( u \) and \( v \) by:
  \[
  d(u,v) = \begin{cases} 
  \min \{ w(p) : u \xrightarrow{p} v \} & \text{if there is a path between } u \text{ and } v \\
  \infty & \text{otherwise}
  \end{cases}
  \]

The Single Source Shortest Path

- Given a graph \( G=(V,E) \), we want to find a shortest path from a source “s” to every vertex \( v \) in \( V \).

- Variants:
  - Single destination shortest path.
  - Single pair shortest path.
  - All-pairs shortest path.

- We will make different assumptions about edge weights!
Uniqueness

Sub-optimality

- Given a weighted graph \( G=(V,E) \) and weight function \( w \) on edges. Let \( P=\langle v_1,v_2,\ldots,v_k \rangle \) be a shortest path from \( v_1 \) to \( v_k \) and for any \( i \) and \( j \) such that \( 1 \leq i \leq j \leq k \) let \( P_{ij}=\langle v_i,v_{i+1},\ldots,v_j \rangle \) be the sub path of \( P \) from \( v_i \) to \( v_j \). Then, \( P_{ij} \) is the shortest path from \( v_i \) to \( v_j \).
Sub-optimality

- Let $G=(V,E)$ be a weighted, directed graph with weight function $w$. Suppose that a shortest path $P$ from source $s$ to a vertex $v$ can be decomposed into a path $P'$ and an edge $(u,v)$ as follows:

$$d(s,v) = d(s,u) + w(u,v)$$

Triangle Inequality

- Let $G=(V,E)$ be a weighted, directed graph with weight function $w$. Then for all edges $(u,v)$, we have

$$d(s,v) \leq d(s,u) + w(u,v)$$
Relaxation Techniques

- For a vertex \( v \) in \( V \), we maintain an attribute \( d[v] \), which is an upper bound on the shortest path from \( s \) to \( v \). We call \( d[v] \) a shortest path estimate.
- Initially, all the shortest path estimates are infinity.
- As algorithms proceeds this values gets closer and closer to actual value \( d(s,v) \) of shortest path between \( s \) and \( v \).

Relaxation

- The process of relaxing an edge \( (u,v) \) consists of testing whether we can improve the shortest path found so far to \( u \), by extending it to \( v \).
- A relaxation step may decrease the of shortest path estimate \( d[v] \) using the triangle inequality:

\[
\text{Relax}(u,v,w) \\
\text{if } d[v] > d[u]+w(u,v) \text{ the} \\
d[v] = d[u]+ w(u,v) \\
\text{End}
\]
The Effect of relaxation

Properties of Relaxation

- Let $G=(V,E)$ be a weighted graph with weight function $w$. Then
  - Immediately after relaxing $(u,v)$, we have $d[v] \leq d[u] + w(u,v)$
  - For every vertex $v$ in $G$: $d[v] \geq d(s,v)$
  - If there is no path in $G$ between $s$ and $v$: $d[v] = \infty$
  - Let $p=<s,u,v>$ be the shortest path between $s$ and $v$. If $d[u] = d(s,u)$ at any time prior to relaxation of $(u,v)$ then $d[v] = d(s,v)$ after the relaxation.
Bellman-Ford Algorithm

- This is the most basic single-source shortest path algorithm:
  - The algorithm finds the shortest path from source $s$ to every vertex $v$ in the graph.
  - The actual shortest path can be constructed easily.
  - Starts with an estimate of shortest distance and eventually converges to shortest weight paths.

Algorithm

SSSP(G)

```
for each $v \in V$ do
  $d[v] = \infty$
  $d[s] = 0$

for $i = 1$ to $|V|$ do
  for each edge $(u, v) \in E$ do
    if $d[v] > d[u] + w(u, v)$ then
      $d[v] = d[u] + w(u, v)$

for each edge $(u, v) \in E$ do
  if $d[v] > d[u] + w(u, v)$ then
    Output "NoSolution!"
```

- Initialize $d[ ]$ to infinity, which will converge to shortest-path value.
- Relaxation: will relax each edge $|V|-1$ times.
- At the end, test to see if a solution is found (gets solution if no negative-weight cycles exits.)
Well-definedness

- If the graph contains a negative cycle then some shortest path may not exist. Consider the following example, as we go around the cycle we always get shorter path.

Example

- Initialization:
  \[ d[A] = 0 \]
  \[ d[B] = d[C] = d[D] = d[E] = \infty \]

- 1st Relaxation: Process edges in order \((A,B), (A,C), (B,C), (B,D), (D,C), (E,D), (B,E)\).
Example

- 2st Relaxation: Same order of process for edges, i.e. (A,B), (A,C), (B,C), (B,D), (D,C), (E,D), (B,E). (there are 3 more relations but the $d[\cdot]$ values will not change).

Running time and Correctness

- Running time: $O(nm)$, where $n=|V|$ and $m=|E|$.
- After $|V|-1$ iteration the $d[v]$ represent the shortest path between $s$ and each vertex $v$:
  - Initially: $d[s]=0$
  - Let $s \rightarrow v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_k$ denote the shortest path between $s$ and $v_k$ then:
    - After 1st path $d[v_1]$ is correct, since $d[v_1] = d[s]+w(s,v_1)$.
    - After 2nd path $d[v_2]$ is correct, since $d[v_2] = d[v_1]+w(v_1,v_2)$.
    - … After $k$th path $d[v_k]$ is correct, since $d[v_k] = d[v_{k-1}]+w(v_{k-1},v_k)$.
  - This holds for all vertices, since the longest path in the graph has length $|V|-1$. 
Dijkstra’s Algorithm

- This algorithm works only for the graphs with non-negative edge weights.
- The result of this algorithm is similar to BFS if the graph is unweighted.
- Like Prim’s algorithm uses a priority queue.
- Has better running time than Bellman-Ford.

Algorithm

$$\text{SSSP}(G)$$

for each \( v \in V \) do
\[
d[v] = \infty
\]
\[
d[s] = 0
\]
\[
S = \emptyset
\]
\[
Q = V
\]
while \( Q \neq \emptyset \) do
\[
u = \text{Extract} - \text{Min}(Q)
\]
\[
S = S \cup \{u\}
\]
for each \( v \in \text{Adj}[u] \) do
\[
\text{if } d[v] > d[u] + w(u, v) \text{ then}
\]
\[
d[v] = d[u] + w(u, v)
\]

- Initialize of \( d[ ] \) is similar to Bellman-Ford.
- Relaxation: each vertex can be subject to relaxation as many times as its in-degree.
- The changes due to relaxation will be handled by Decrease-Key algorithm.
Example

Extract-Min: A
Decrease-Key: B,C

Q:

Extract-Min: C
Decrease-Key: B,D

Q:

Extract-Min: D
Decrease-Key: B

Q:
Example

Extract-Min: \( B \)

\( Q: \text{EMPTY} \)

Running Time

- Extract-Min: will be executed \(|V|\) times.
- Decrease-Key: will be executed \(|E|\) times.

\[ T(n, m) = O(n \log n + m \log n) \]
Correctness

At the termination of algorithm $d[u] = d(s,u)$.
Assume not, i.e. $d[u] \neq d(s,u)$. Right before $u$ is added to $S$.
Let $p$ be the shortest path between between $s$ and $u$. Let $y$
be the first vertex on $p$ outside $S$ and let $x$ be the vertex on $p$
right before $y$. Clearly when $u$ is inserted to $S$, $d[y] = d(s,y)$.
But this will result in a contradiction.

\[ d[y] = d(s,y) \]
\[ \leq d(s,u) \]
\[ \leq d[u] \]

Shortest Path in DAGs

- SSSP is well defined for DAGs, since DAGs can not have negative cycles.
- We are looking for a fast algorithm (as opposed to Dijkstra's and Bellman-Ford).
- Observe that, if there is a path from $u$ to $v$, then $u$ precedes $v$ in topological sort. Which means to find the SSSP we can just pass once over the vertices in the topologically sorted ordered.
Algorithm

DAG-Shortest-Path(G,w,s)
   Topologically Sort the vertices of G
   Initialize the d[ ] for all the vertices.
   For each vertex u taken in topologically sorted order do
      For each vertex v in Adj[u] do
         Relax(u,v,w)

Example
Example

Example
Example

Running Time

DAG-Shortest-Path($G, w, s$)

- **Topologically Sort** the vertices of $G$
- **Initialize** the $d[ ]$ for all the vertices.
- **For** each vertex $u$ taken in topologically sorted order **do**
  - **For** each vertex $v$ in $Adj[u]$ **do**
    - **Relax**($u, v, w$)

- Since every vertex will be looked at at most once, the outer loop will be executed $O(|V|)$ times.
- The inner loop will be executed only $O(|E|)$ times.
- The overall running time is $O(|V|+|E|)$. 