SINGLE SOURCE SHORTEST PATHS
SINGLE SOURCE SHORTEST PATHS

Generally assume a directed graph (can make undirected→directed easily)
SINGLE SOURCE SHORTEST PATHS

\[ S \]
SINGLE SOURCE  SHORTEST PATHS

paths from $s$ to $t$

$3 + 2 + 1 + 20$
SINGLE SOURCE  SHORTEST PATHS

paths from $s$ to $t$

$3 + 2 + 1 + 20$

$3 + 2 + 4 + 15$

not greedy or BFS

although this is an extension of BFS
(weights = 1)
SINGLE SOURCE SHORTEST PATHS

paths from $s$ to $t$

3 + 2 + 1 + 20
3 + 2 + 4 + 15
3 + 2 + 4 + 5 + 8 = 22

not greedy or BFS

tot. length of path from $s$
SINGLE SOURCE SHORTEST PATHS

paths from $s$ to $t$

\[ 3 + 2 + 1 + 20 \]
\[ 3 + 2 + 4 + 15 \]
\[ 3 + 2 + 4 + 5 + 8 \]

not greedy or BFS

multiple options:
\[ 5 + 1 + \ldots \]
\[ + 4 + \ldots \]  

= 22
SINGLE SOURCE SHORTEST PATHS

paths from $s$ to $t$

$3 + 2 + 1 + 20$

$3 + 2 + 4 + 15$

$3 + 2 + 4 + 5 + 8 = 22$

not greedy or BFS

multiple options

$5 + 1 + ...$

$+ 4 + ... = 22$

disconnected from $s$
SINGLE SOURCE SHORTEST PATHS

Observations
- No cycles in \( s \to t \) (shortest path)

assumption?
SINGLE SOURCE SHORTEST PATHS

Observations

- No cycles in $s \rightarrow t$
- Negative weights are OK, unless they form a negative cycle in $G$

$\sum (\text{cycle}) < 0$
SINGLE SOURCE SHORTEST PATHS

Observations
- No cycles in $s \rightarrow t$
- Negative weights are OK, unless they form a negative cycle in $G$

Any vertex reachable from a negative cycle gets a score of $-\infty$ assuming cycle can be reached from $s$
**SINGLE SOURCE SHORTEST PATHS**

![Diagram of a graph with labeled edges and vertices]

**Observations**
- No cycles in $s \rightarrow t$
- Negative weights are OK, unless they form a negative cycle in $G$

**Any vertex reachable from a negative cycle gets a score of $-\infty$**
Observations

- No cycles in $s \rightarrow t$
- Negative weights \( \sim \text{OK, unless they form a negative cycle in } G \)
- Shortest path $s \rightarrow v \rightarrow t$
- Shortest path $s \rightarrow v \leftarrow (9)$
- Shortest path $v \rightarrow t \leftarrow (13)$
there may be multiple shortest paths
e.g. $s \rightarrow t$ or $s \rightarrow x \rightarrow y \rightarrow t$
there may be multiple shortest paths
e.g. $s \rightarrow t$ or $s \rightarrow x \rightarrow y \rightarrow t$

All shortest paths from $s$ to $V$
can be represented in a DAG
DAG → tree: arbitrarily keep one path to each vertex

"shortest paths tree"

there may be multiple shortest paths
e.g. \( s \rightarrow t \) or \( s \rightarrow x \rightarrow y \rightarrow t \)

All shortest paths from \( s \) to \( V \) can be represented in a DAG

(similar to picking one BFS/DFS search)
By exploring some path from $s$ to $t$ we get a path score (e.g. 26)
By exploring some path from $s$ to $t$ we get a path score (e.g. 26) the score of $t$ is 26, which may only decrease as we explore more options.
By exploring some path from $s$ to $t$ we get a path score (e.g. 26) the score of $t$ is 26, which may only decrease as we explore more options.

If we update the score of $v$:

$$d(v)$$

& \exists \text{ edge } v \rightarrow t

then we can possibly improve $d(t)$:

$$d(v) + w(v, t) < d(t) \quad (15)$$
By exploring some path from \( s \) to \( t \) we get a path score (e.g. 26) the score of \( t \) is 26, which may only decrease as we explore more options.

If we update the score of \( v \): 
\[
d(v) 
\]

where there exists an edge \( v \rightarrow t \) then we can possibly improve \( d(t) \): 
\[
d(v) + w(v,t) < d(t) \tag{15}
\]
By exploring some path from $s$ to $t$ we get a path score (e.g. 26) the score of $t$ is 26, which may only decrease as we explore more options.

If we update the score of $v$:

$$d(v)$$

& $\exists$ edge $v \rightarrow t$

then we can possibly improve $d(t)$:

$$d(v) + w(v, t) < d(t) \quad (15)$$
Relax($v,t$): checking if score of $t$ can be improved (lowered) by using $s \rightarrow v \rightarrow t$
Relax\((v,t)\): checking if score of \(t\) can be improved (lowered) by using \(s \rightarrow v \rightarrow t\).
Relax($v,t$): checking if score of $t$ can be improved (lowered) by using $s \rightarrow v \rightarrow t$.

Keep min of $d(t)$ vs. $d(v) + w(v,t)$.

If $v$ helps, then parent$(t) = v$. 

Diagram with nodes and edges labeled with weights.
Assume this is a shortest path from $s$ to $t$.
Assume this is a shortest path from $s$ to $t$. unknown but exists
Assume this is a shortest path from $s$ to $t$ but unknown exists.

Suppose we have an algorithm based on relaxing edges.
Assume this is a shortest path from $s$ to $t$ unknown but exists.

Suppose we have an algorithm based on relaxing edges.

If we relax $e_1$ before $e_2$ before $\ldots$ before $e_{k-1}$ before $e_k$ then $\ldots$ ?
Assume this is a shortest path from $s$ to $t$ (unknown but exists).

Suppose we have an algorithm based on relaxing edges.

If we relax $e_1$ before $e_2$ before ... before $e_{k-1}$ before $e_k$ then we will correctly compute $d(t)$. 
Assume this is a shortest path from \( s \) to \( t \) unknown but exists.

Suppose we have an algorithm based on relaxing edges.
If we relax \( e_1 \) before \( e_2 \) before \( \ldots \) before \( e_{k-1} \) before \( e_k \) arbitrary then we will correctly compute \( d(t) \).

Relax sequence: \( e^*e_1e^*e_2e^*e_1e_k e_{k-1}e_1e^*e_k e^*e^*e_k e^* \)
Assume this is a shortest path from $s$ to $t$ unknown but exists

Suppose we have an algorithm based on relaxing edges. If we relax $e_1$ before $e_2$ before ... before $e_{k-1}$ before $e_k$ then we will correctly compute $d(t)$.

Relax sequence: $e^x e_1 e^j e^y e_2 e^x e^i e_k e_{k-1} e_i e^x e_k e^y$ : OK (don't care if we relax other edges or the same ones repeatedly)
Assume this is a shortest path from $s$ to $t$ unknown but exists

Suppose we have an algorithm based on relaxing edges.

If we relax $e_1$ before $e_2$ before $\ldots$ before $e_{k-1}$ before $e_k$

then we will correctly compute $d(t)$ why?

Relax sequence: $e^x, e_1, e^y, e_2, e^x, e_1, e^y, e_k, e_{k-1}, e_1, e^x, e_k, e^y$ : ok

(don't care if we relax other edges or the same ones repeatedly)
Assume this is a shortest path from $s$ to $t$ unknown but exists

Suppose we have an algorithm based on relaxing edges. If we relax $e_1$ before $e_2$ before $\ldots$ before $e_{k-1}$ before $e_k$ then we will correctly compute $d(t)$ by induction.

Relax sequence: $e^* e_1 e^* e_2 e^* e_1 e_k e_{k-1} e_1 e^* e_k e^*$ is OK (don't care if we relax other edges or the same ones repeatedly)