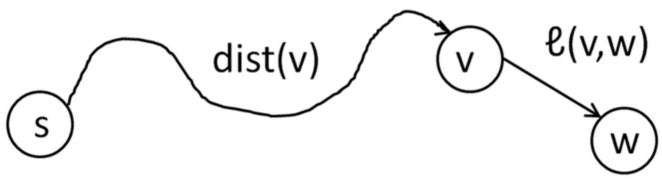
1 Fundamental Shortest Paths Formula

For any vertex w that isn't the source s, $w \neq s$,

$$\text{dist}(w) = \min_{(v,w)\in E} \text{dist}(v) + \ell(v, w)$$

This says that the distance of w is equal to the minimum over all edges v to w in E of the distance to v plus the length of the edge from v to w.



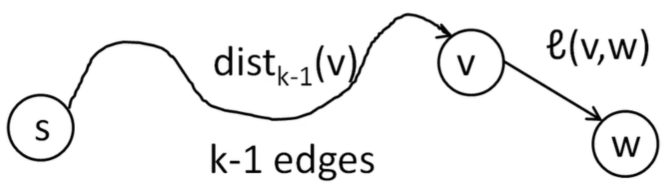
Looking at the visualization above, we're saying that the path from s to v has length $\text{dist}(v)$; this is the shortest path from s to v.

We can use a system of equations to solve for the distances from s to ever other vertex in the graph. When $\ell \geq 0$, Dijsktra gives an order to solve in. But, with negative edge weights, this order is no longer clear.

1.1 Algorithm Idea

Instead of finding the shortest paths, which may not exist due to a negative edge cycle, we instead find the shortest paths of length at most k edges. So, for $w \neq s$, we have:

$$\text{dist}_k(w) = \min_{(v,w)\in E} \text{dist}_{k-1}(v) + \ell(v, w)$$



If we look at a path from s to w with at most k edges, this is a path from s to v that uses at most $k-1$ edges for some v plus a single edge from v to w. The best length this path could have is $\text{dist}_{k-1}(v) + \ell(v, w)$, where our $\text{dist}_{k-1}(v)$ is minimized.

1.2 Bellman-Ford Algorithm

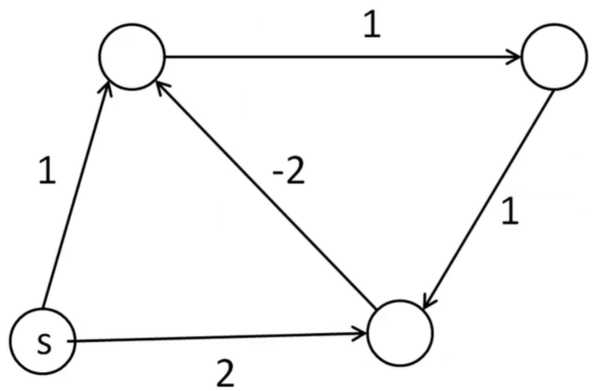
This formula gives rise to the *Bellman-Ford* algorithm.

```
Bellman-Ford(G, s, l)
    dist_{0}(v) = infinity for all v
    dist_{0}(s) = 0
    For k = 1 to n
        For w in V
            dist_{k}(w) = min(dist_{k}(w), dist_{k - 1}(v) + l(v, w))
        dist_{k}(s) = min(dist_{k}(s), 0)
```

We note that each iteration of the outer-loop is $\mathcal{O}(|E|)$ time. However, what value of k do we use?

1.2.1 Example: Applying the Bellman-Ford Algorithm

Find the shortest path from s to every other vertex in the graph shown below. For convenience, let the top-left vertex be denoted A, the top-right vertex be B, and the bottom-right vertex be C.



- First, when $k = 0$, $s = 0$ and everything else is assigned ∞; you can't get to anywhere else with no edges. Therefore, the distances are:

k	s	A	B	C
0	0	∞	∞	∞

- When $k = 1$, we consider all vertices that we can reach with one edge. So, you still can't reach vertex B since it needs at least two edges. But, we can get from s to A with path length 1, and we can get from s to C with path length 2. So:

k	s	A	B	C
0	0	∞	∞	∞
1	0	1	∞	2

  In other words, these are the shortest distances we can get from a path of length one to each of our vertices.

- When $k = 2$, we consider all vertices that we can reach with two edges. First, we can reach vertex B with just two edges (from s to A and from A to B). A has distance 1 so B must have distance 2. We can't do any better with s or C, but note that we can actually improve A (if we go from s to C and from C to A) by getting it down to distance 0. Therefore:

k	s	A	B	C
0	0	∞	∞	∞
1	0	1	∞	2
2	0	0	2	2

- When $k = 3$, we consider all vertices that we can reach with three edges. So, we can again reach every vertex. In particular, note we can update vertex B's distance. This is because if A is 0, then there is a path from A to B that has length 1 ($s \to C \to A \to B$). So:

k	s	A	B	C
0	0	∞	∞	∞
1	0	1	∞	2
2	0	0	2	2
3	0	0	1	2

- Past $k = 3$, we notice that the distances no longer change. In other words, the process has stabilized.

1.3 Analysis

Proposition. *If $n \geq |V| - 1$ and if G has no negative weight cycles, then for all v,*

$$dist(v) = dist_n(v)$$

This says that if we run the Bellman-Ford algorithm, there is a limit. Assuming there are no negative weight cycles[1], we only need to run the algorithm for $|V| - 1$ rounds for a final runtime of $\mathcal{O}(|V||E|)$.

Proof. We need to show that the shortest path has fewer than $|V|$ edges. Suppose that there is a path that has at least $|V|$ edges, then by the pigeonhole principle, it must contain the same vertex twice. This means that there is a loop. If we remove the loop (which we assume has non-negative total weight, i.e. not a negative weight cycle), then we get a shorter path. This new path has at most $|V| - 1$ edges since we can only hit each vertex at most once. Note that this path is no longer than the one with the loop since we're considering the *shortest* distance. □

1.4 Revised Bellman-Ford Algorithm

With this analysis in mind, our algorithm looks like:

```
Bellman-Ford(G, s, l)
    dist_{0}(v) = infinity for all v
    dist_{0}(s) = 0
    For k = 1 to |V|
        For w in V
            dist_{k}(w) = min(dist_{k}(w), dist_{k - 1}(v) + l(v, w))
        dist_{k}(s) = min(dist_{k}(s), 0)
    Return dist_{|V|}(t)
```

Which we now know computes the shortest paths *if no negative weight cycles* in $\mathcal{O}(|V||E|)$ time.

1.5 Detecting Negative Cycles

If there are no negative weight cycles, Bellman-Ford computes the shortest paths. Suppose there *are* negative weight cycles. Well, Bellman-Ford will calculate some distances which will probably be garbage values.

How do we know whether or not there are any negative weight cycles?

1.5.1 Negative Cycle Detection

Proposition. *For any $n \geq |V| - 1$, there are* ***no*** *negative weight cycles reachable from s if and only if, for every $v \in V$:*

$$dist_n(v) = dist_{n+1}(v)$$

So, to detect negative cycles, all we need to do is run one more round of Bellman-Ford and see if any distances change.

[1]If there is a negative weight cycle, there is probably no shortest path.

Proof. Suppose no negative weight cycles exist. Then, for any $n \geq |V| - 1$, $\text{dist}_n(v) = \text{dist}(v)$. So, $\text{dist}_n(v) = \text{dist}(v) = \text{dist}_{n+1}(v)$ by the transitive property (since $n + 1 \geq n \geq |V| - 1$).

Suppose $\text{dist}_n(v) = \text{dist}_{n+1}(v)$ for all v. Then:

$$\begin{aligned}\text{dist}_{n+2}(w) &= \min_{(v,w)\in E}(\text{dist}_{n+1}(v) + \ell(v,w)) \\ &= \min_{(v,w)\in E}(\text{dist}_{n}(v) + \ell(v,w)) \\ &= \text{dist}_{n+1}(w)\end{aligned}$$

We essentially apply the idea that if $\text{dist}_n(v) = \text{dist}_{n+1}(v)$, then the same idea holds for $n+1$. In other words, if the distances are the same for one round from n to $n+1$, then it will be the same for $n+1$ to $n+2$ and so on. If the distance functions stabilize for one round, they will stabilize forever. So:

$$\text{dist}_n(v) = \text{dist}_{n+1}(v) + \text{dist}_{n+2}(v) + \text{dist}_{n+3}(v) + \ldots$$

However, if there were a negative weight cycle, the distances would decrease eventually. □

1.6 Shortest Paths in DAGs

We saw that shortest paths is harder when we needed to deal with negative weight cycles. For general graphs, we needed to use Bellman-Ford which is much slower than our other algorithms. In our case here, if we're working with a DAG, then there are faster algorithms that we can apply.

Recall that, for any vertex w that isn't the source s, $w \neq s$,

$$\text{dist}(w) = \min_{(v,w)\in E} \text{dist}(v) + \ell(v,w)$$

We can use topological ordering for DAGs to compute the shortest distance.

1.6.1 Algorithm

The algorithm is as follows:

```
ShortestPathsInDAG(G, s, l)
    TopologicalSort(G)
    For w in V in topological order
        If w = s
            dist(w) = 0
        Else
            dist(w) = min(dist(v) + l(v, w))
```

This has runtime $\mathcal{O}(|V| + |E|)$.

1.7 Shortest Path Algorithms Summary

Path Type	**Algorithm**	**Runtime**
Unit Weights, General Graph	`Breadth First Search`	$\mathcal{O}(\|V\| + \|E\|)$
Non-Negative Weights, General Graph	`Dijsktra`	$\mathcal{O}(\|V\|\log(\|V\|) + \|E\|)$
Arbitrary Weights, General Graph	`Bellman-Ford`	$\mathcal{O}(\|V\|\|E\|)$
Arbitrary Weights, DAG	`ShortestPathsInDAG`	$\mathcal{O}(\|V\| + \|E\|)$