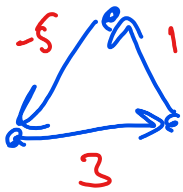
Last time we were considering the shortest path problem. We will be considering the same problem again today.

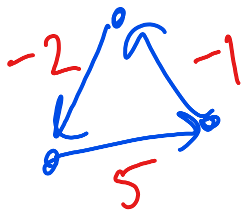
Recall that if there is a negative-weight cycle then a shortest path may not exist.

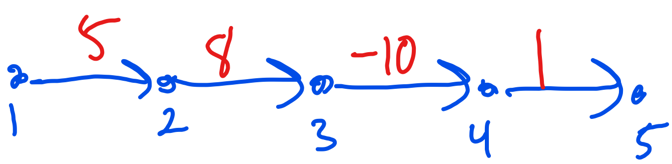
- We can repeatedly follow the cycle to reduce the length of our path.



When all of the edge weights are nonnegative then there certainly cannot exist a negative-weight cycle, and we considered Dijkstra's algorithm for single-source shortest path problem on such graphs.

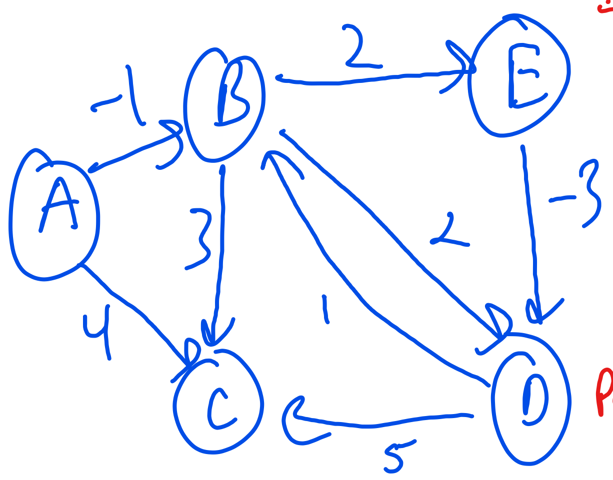
It is possible to have graphs with negative edge weights but no negative cycles. Shortest paths are then well-defined. It would be nice to have an algorithm which could compute shortest paths for these types of graphs (and determine that there is a negative cycle if there is one).





Such an algorithm is the **Bellman-Ford algorithm**. We again maintain a $d[v]$ value for each vertex v (again an upper bound of $\delta(s, v)$).

1. $d[s] = 0, d[v] = \infty$ for all $v \neq s$.
2. For $i = 1$ to $n - 1$
   (a) For each edge (u, v), check if $d[v] > d[u] + w(u, v)$. If so, then $d[v] = d[u] + w(u, v)$.
3. For each edge (u, v), if $d[v] > d[u] + w(u, v)$ then a negative cycle exists. If this is not true for each of the edges, then $d[v] = \delta(s, v)$.



	A	B	C	D	E
Initial	0	∞	∞	∞	∞
Pass 1		-1	4		
	0	-1	2	∞	∞
Pass 2				1	1
	0	-1	2	-2	1

Order of edges:

(B,E), (D,B), (B,D), (A,B), (A,C), (D,C), (B,C), (E,D)

Example of Bellman-Ford:

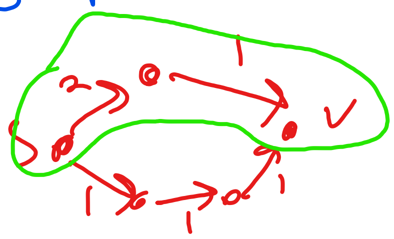
Proof of correctness:

Thm

If the graph does not contain a negative cycle, then Bellman-Ford terminates with $d[v] = \delta(s,v)$ for all $v \in V$.

Proof:

Let v be any vertex, and consider a shortest path p from s to v with the minimum # of edges.



p: $\underset{s}{v_0} \to v_1 \to v_2 \to v_3 \to \cdots \to \underset{v}{v_k}$

there are k edges in the path.

Since p is a shortest path, we have

$$\delta(s, v_i) = \delta(s, v_{i-1}) + w(v_{i-1}, i)$$

Initially $d[v_0] = 0 = \delta(s, v_0)$. $d[s]$ is unchanged because there are no negative weight cycles.

After 1 pass through E, we have $d[v_1] = \delta(s, v_1)$.
" 2 " " ", " " $d[v_2] = \delta(s, v_2)$.

⋮

After k " " ", " " $d[v_k] = \delta(s, v_k)$

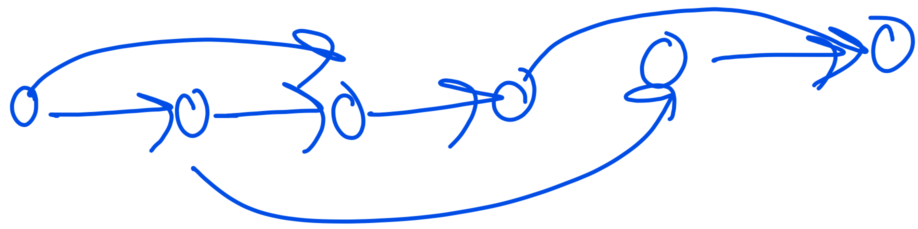
Since G contains no negative cycles, p is simple.
Then there are at most $n-1$ edges in p.
So after $n-1$ passes through E, we have discovered the path p.

Corollary

If $d[v]$ fails to converge after $n-1$ passes through E then there is a negative weight cycle in G that is reachable from s.

Shortest paths on a directed acyclic graph (DAG):

1. Compute a topological sort.
2. Do one pass of Bellman-Ford (considering the edges "in order" according to the topological sort).



Summary of algorithms considered:

- Single-source shortest paths:
  - Nonnegative edge weights. Dijkstra: $O(m \log n)$
  - Arbitrary edge weights. Bellman-Ford: $O(nm)$
  - DAG: Bellman-Ford single pass: $O(n + m)$

Suppose now we want to compute the shortest paths between any two pairs of vertices. We could run each of the previous algorithms with n different sources to accomplish this:

- All-pairs shortest paths:
  - Nonnegative edge weights. Dijkstra n times: $O(nm \log n)$
  - Arbitrary edge weights. Bellman-Ford n times: $O(n^2 m)$

In a dense graph, we have $m = \Omega(n^2)$, so thus far our best algorithm for all-pairs shortest paths with arbitrary edge weights would be $O(n^4)$. Can we do better?

The **Floyd-Warshall algorithm** is a dynamic programming algorithm for the all-pairs shortest path problem.

Suppose the graph is given as an *adjacency matrix* $A = (a_{ij})$ where a_{ij} is the weight of the edge from i to j.

Let $c_{ij}^{(k)}$ denote the weight of a shortest path from i to j with intermediate vertices on the path belonging to the set $\{1, 2, \ldots, k\}$.

- Note that $\delta(i, j) = c_{ij}^{(n)}$.

The algorithm is to show that $c_{ij}^{(k)}$ for each $1 \leq i, j, k \leq n$ can be computed using dynamic programming.