• **DISCLAIMER:** These notes are not necessarily an accurate representation of what I said during the class. They are mostly what I intend to say, and have not been carefully edited.

• Main topics:
  - Reachability algorithm, speedup to $O(m)$.
  - Shortest paths: Bellman-Ford algorithm:
    * Repeatedly improve distances.
    * If goes more than $n$ steps, exists negative cycle.

• Comments from Last Time:
  - Variable names inconsistent: all edges (including directed ones) are $e$.
  - Cover math in more depth.

We continue from the algorithm for finding reachability in a directed graph $G = (V, E)$:

1. Mark $s$ as reachable, all other vertices as unreachable.

2. Repeat $n$ times
   (a) Loop through all edges $e = u \rightarrow v$
      i. If $u$ is reachable, mark $v$ as reachable

This algorithm as stated runs in $O(nm)$ time. To make it run faster, notice that once we reach $u$, we only need to mark all neighbors of $u$ once. To do this, we need to:

1. Store edges in adjacency lists.

2. Use a queue to store visited vertices which we’ve yet to mark neighbors. Only add a vertex to a queue if we have yet to update all its neighbors.

This algorithm runs in $O(m)$ time because it goes through the neighborhood of each vertex once. It has an additional property: it visits the vertices in order of their distances to the sources.

For the following example that’s also in Figure 4.14 of the text book:
If we run this search from $S$, we will visit the vertices in order

$$SGAFEBCD.$$  

We will formally prove that this method also computes distances when we discuss faster shortest path algorithms. For now we will give a more general algorithm that works for all distances. For a graph with distances, we associate a length, $l_e$ for each edge. For example, the above example can incorporate lengths as:

- $l_{S \to A} = 10$
- $l_{S \to G} = 8$
- $l_{G \to F} = 1$
- $l_{F \to E} = -1$
- $l_{F \to A} = -4$
- $l_{D \to E} = -1$
- $l_{C \to D} = 3$
- $l_{A \to E} = 2$
- $l_{E \to B} = -2$
- $l_{B \to A} = 1$
- $l_{B \to C} = 1$

We can compute distances by modifying the reachability algorithm to main distances instead of ‘reached’ flags.

1. Initialize $d[s] \leftarrow 0$, $d[u] \leftarrow \infty$ for all other vertices.
2. Repeat $n$ times
   a. Loop through all edges $e = u \to v$
      i. $d[v] \leftarrow \min(d[v], d[u] + l_{u \to v})$.

The correctness of this algorithm can be proven in two steps:

**Lemma 0.1.** At all steps, we have $\text{dist}(s, u) \leq d[u]$.

*Proof.* The proof is by induction. The base case follows from the initialization. The inductive case follows from $\text{dist}(s, v) \leq \text{dist}(s, u) + l_{u \to v}$ since a path from $stou$ can be extended to $v$ using the edge $u \to v$. \qed

**Lemma 0.2.** If $G$ has no negative cycles, then the $d[u]$ values returned are correct.

*Proof.* This can be proven in two ways:

1. **This is what the textbook does, without proof.** You only need to know how to find the negative cycle. By tracing back the ‘parent’ of the updates that led to updates in $d[u]$. There are at least $n$ of these updates, so some vertex is repeated. This forms a cycle, whose length must be negative.

2. **This is required because it also gives how to recover the shortest path**

   Proving by induction that if the shortest path from $s$ to $u$ is a length $k$ path, then $d[u] = \text{dist}(s, u)$ after the $k^{th}$ outer loop.

   Base case: $k = 0$, $d[s] = 0$. Inductive case: let the path be $s = u_0 \ldots u_{k-1}, u_k = k$.

   By inductive hypothesis at the start we have $d[u_{k-1}] = \text{dist}(s, k - 1)$. Then the update step gives

   $$d[u_k] \leq \text{dist}(s, u_{k-1}) + l_{u_{k-1}u_k} = \text{dist}(s, u_k).$$ \qed