Bellman Ford (G, s)

1. 1v1 - 1 = 5 - 1 = 4

Alphabetic order:

(a, b)  (a, c)  (a, e)
(c, a)
(e, b)  (e, c)
(s, a)  (s, e).

1st check:

a = 9  e = 6

2nd:

b = 13, 7  c = 14, 3

3rd:

a = 1

4th:

b = 5.

What could go wrong?

What:

Cannot find shortest paths! (All distances are -∞).

How to detect at the end of algorithm?

⇒ If any node's distance is improved after one more set of updates, then node's shortest path
Since traversing the cycle improved the distance, it must have been a negative cycle.

$$V ightarrow V$$

6. For each edge $$(v, u) \in E$$. 
    
    If $$u\text{dist} > v\text{dist} + W_{v,u}$$ 
    
    - print("Error! Negative cycle!")

Run-Time: (Main loop):

- Check each edge $$|V| - 1$$ times.

$$= O(|V|^2 |E|)$$ Note: Much slower than Dijkstra's.

Correctness: (SKIP)

Proof by induction on distance found using some # of edges.

Let $$d_i(u)$$ = S-u distance using $$\leq i$$ edges found by BF

Claim: For all $$i \leq |V|-1$$, $$d_i(u)$$ is optimal distance using $$\leq i$$ edges.

Base Case: $$i=1$$, $$d_1(u)$$ is best distance using $$\leq 1$$ edges, ✓

Ind. Hyp: $$d_k(u)$$ is optimal for all $$u \in V$$

Ind. Step: Show $$d_{k+1}(u)$$ is optimal
(can apply graphs to things not so obvious to networks)

- Scheduling courses to satisfy part of major (e.g., systems)
  Courses with prereqs: 101, 200, 201, 314, 414, 701
  150, 190
  313, 413

How to find an ordering of courses?

- Node = course
- Edge (u, v): if should take u before v

Graph has to be directed. What else? No cycles

Topological sort - in a directed acyclic graph (DAG), ordering of vertices s.t. each edge points towards later vertex.

Not always so obvious:
[Hint: indegree of vertex \( v \) = number of edges \((u,v)\) (inconuing edges)].

[ex: indegree of A = 0, C = 5]

"Which node should be output first? Those with indegree = 0. Second? Hint: Look back at course graph, consider deleting nodes/edges. Nodes whose indegrees become 0 after deleting the first set of output nodes and their outgoing edges."

Note: Since graph is acyclic some node will have indegree = 0

If all nodes had indegree > 0 then there would be a cycle

To print in TopSort order:
1. Until all vertices printed.
2. Look for a new vertex \( u \), with indegree = 0.
3. Print \( u \).
4. Decrement indegree of nodes adjacent to \( u \).

One solution:

(Alphabetic) / Break ties alphabetically.

A B E F G D C

Run Time? Step 2 takes \( O(\text{V}) \) time, Performed \( O(\text{V}) \) times (Step 1)

\( \Rightarrow O(\text{V}^2 + |E|) = O(\text{V}^2) \)
How can we make this faster?

Notice in every step, we update indegrees and look for node with indegree = 0.

When a node has indegree = 0, store it to be next node to be deleted. Store in queue.

TopSort(G) // G=(V,E). Prints nodes of G in top-sort order
1. For all \( v \in V \)
2.   if \( (v, \text{indegree} = 0) \)
3.       Q.enqueue(v)
4. while (!Q.empty())
5.   \( u = Q.dequeue() \)
6.   print(u)

   //update indegrees of neighbors
7.   for all neighbors, \( v \), of \( u \)
8.   \( v, \text{indegree} -= \)
9.     if \( (v, \text{indegree} = 0) \)
10.    Q.enqueue(v).

\[ \text{printed: A B E F G D C} \]
Run Time:
Steps 1-3: $O(VI)$
Steps 4-10: $O(VI) + (\text{not } *)$
Step 7: $O(1E1)$

Consider each vertex linear $# \text{times}$, each edge once: $O(VI + 1E1)$

Recall: runtimes for Dijkstra's $(O(VI \log VI + 1E1 \log VI))$ & BF $(O(VI \cdot 1E1)$

Can we do better for DAGs?
For DAGs, top sort can help us find shortest paths. How?
Just put nodes in top sort order.

Order of processed nodes: $s, a, b, c, d, e$

For shortest paths in DAGs:
(1) Process nodes in top sort order (no MinHeap needed)
(2) Why? For a node $a$, only nodes that appear before
$a$ in top sort order can affect $a$. dist

Does this work for graphs with neg. edge weights? Yes!
Recall: problem with negative edges - Dijkstra's: future node
would help improve distance to already processed node

Dijkstra's:

T.S.:

We will

If nodes processed in top sort order, a negative edge cannot
improve the distance of a processed node (only unprocessed nodes).