Shortest Paths with Negative Edge Weights

- Applications of negative edge weights (see slides).

- Recall Dijkstra's - updating occurs per edge.

- Can replace steps 9-11 with call to update \((v, u)\) where:

  \[
  \text{update} \quad \text{if (} \neg u.\text{processed and } u.\text{dist} > v.\text{dist} + w_{v,u})
  \]

  \[
  u.\text{dist} = v.\text{dist} + w_{v,u}
  \]

  \[
  u.\text{pred} = v
  \]

- We update edges out of most recently processed node \((v)\) because we know \(v.\text{dist}\) won't be improved.

What changes when there are negative edge weights?

- Node with min distance is no longer "processed" since its distance may be improved later.

![Diagram](attachment:image.png)
With negative edge weights:

⇒ should update not just edges out of most recently processed node but all edges in alphabetic order.

Order doesn’t matter, so just use alphabetic order:

ex: (b,a) (s,a) (s,b)

"How many times should we update all edges? Let's find out..."

Final dist's should be: ("eyeball it")

ex:

Final dist's:

- s: dist: 0
- a: dist: -3
- b: dist: 1
- c: dist: -2

After each round of updates which nodes have their final distances? Remember updates are in alphabetic order.

⇒ (a,c) (b,a) (s,a) (s,b) (s,c).

After 1st round of updates: b dist is final

2nd: a dist
3rd: c dist

See a pattern? "After 1st, 2nd, 3rd, which nodes dist's were final?"
*After i rounds of updates all nodes whose shortest paths have i edges have final distances.

"How many rounds of updates should we do to ensure we get distances for all possible shortest paths?"

* Max # edges in a path = |V| - 1, so |V| - 1 rounds of updates:

This is actually more than we need *(HW ques) (*)

/* Finds shortest path distances from s to every other node in G */
Bellman Ford (G, s) // G = (V, E) where W_{uv} may be < 0
// for (u,v) ∈ E

1. For all v ∈ V, v.dist = ∞, v.pred = null (no more processed!)
2. s.dist = 0
3. for i = 1 to |V| - 1
   // Check all edges (to see if dist to end vertex can improve)
4. for each edge (u,v) ∈ E
5. update (v,u)
6. <Leave Space>
7.
8.
Bellman-Ford (6, s)

14.3 \[ 1v1 - 1 = 5 - 1 = 4 \] (rounds of updates)

Alphabetic order:

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

13.75

1st round: a = 9, e = 6
2nd round: b = 13.7, c = 14.3
3rd round: a = 1
4th round: b = 5.

What could go wrong?

9.87 (5)

\[ a \rightarrow c \rightarrow a \] 14.13

\[ (5) \]

Negative cycle!

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

How to detect at the end of algorithm?

\[ \Rightarrow If \ any \ node's \ distance \ is \ improved \ after \ one \ more \ set \ of \ updates, \ this \ node's \ shortest \ path \ has \ 1v1 \ edges, \ so \ contains \ a \ cycle. \]
Since traversing the cycle improved the distance, it must have been a negative cycle!

\[ (v) \rightarrow (w) \]

6. **For each edge** \((v, u) \in E\): 
   
   if \((u \text{ dist} > v \text{ dist} + w_{v,u})\)
   
   print("Error! Negative cycle!")

---

**Run Time:** 

Call update on each edge \(|V|-1\) times 

\(\Rightarrow O(|V| \cdot |E|) \)  \text{Note: Much slower than Dijkstra's.}