Algorithms

SP3

Recap

- HW: Friday

Today: What about negative edges??

Bellman-Ford:

Relax ALL the edges!

(Then repeat until nothing is tense.)

```
BellmanFord(s)
  InitSSSP(s)
  while there is at least one tense edge
      for every edge u→v
          if u→v is tense
              Relax(u→v)
```

O(E)

Correctness:

If nothing is tense, must have SP-tree!

Runtime? Problem - while loop

Well, still that negative cycle issue

So, revised:

If no negative cycles, then it will work.

Otherwise, give up.

```
BellmanFord(s)
    InitSSSP(s)
    repeat V − 1 times
        for every edge u→v
            if u→v is tense
                Relax(u→v)
    for every edge u→v
        if u→v is tense
            return "Negative cycle!"
```

$O(VE)$

Key: For loop!!

Why?

Notation:

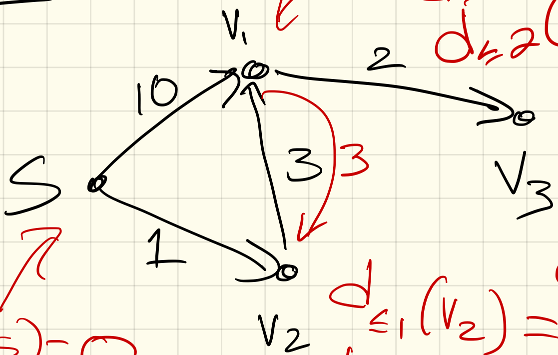
Let $dist_{\le i}(v) :=$ length of the shortest s-to-v path using at most i edges

Ex:

$d_{\le 0}(v_1) = \infty$

$d_{\le 1}(v_1) = 10$

$d_{\le 2}(v_1) = 4 = d_{\le 1}(v_2) + w(e)$



$d_{\le 1}(v_3) = \infty$

$d_{\le 2}(v_3) = 12$

$d_{\le 3}(v_3) = 6$

$d_{\le 1}(v_2) = 1$

$d_{\le 2}(v_2) = 1$

$d_0(s) = 0$

$d_1(s) = 6$

$d_2(s) = 0$

$\vdots$

So: $dist_{\leq 0}(s) = 0$ & for all $v \neq s$,

$dist_{\leq 0}(v) = \infty$

Claim: $\forall v$ & i, after i iterations of B-F,

$$dist(v) \leq dist_{\leq i}(v)$$

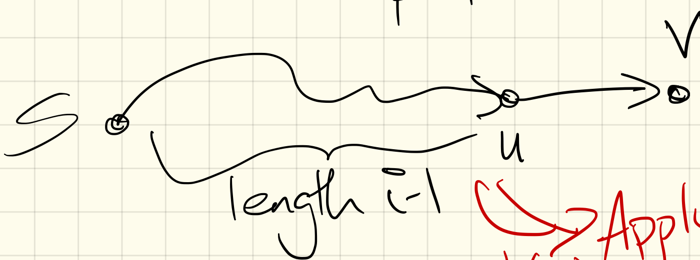
Why?

Induction on i:

BC: $d_{\leq 0}(s)$ & $d_{\leq 0}(v)$ are good

IH: $d_{\leq i-1}(v)$ is $\geq dist(v)$

IS: $d_{\leq i}(v)$: Take shortest path of length i to v:



Apply IH:

$d(u) \leq dist_{\leq i-1}(u)$

Either $u \to v$ is tense:

$$\text{dist}(v) > \text{dist}_{\le i-1}(u) + w(u \to v)$$

set $\text{dist}(v) = \underbrace{\text{dist}_{\le i-1}(u) + w(u \to v)}_{=\ \text{dist}_{\le i}(v)}$

<u>OR</u>: not

$$\text{dist}(v) < \underbrace{\text{dist}_{\le i-1}(u) + w(u \to v)}_{=\ d_{\le i}(v)}$$

Takeaway

Since any path has length $\leq V-1$, don't need to repeat more than that!

```
BELLMANFORD(s)
  INITSSSP(s)
  repeat V − 1 times
      for every edge u→v
          if u→v is tense
              RELAX(u→v)
  for every edge u→v
      if u→v is tense
          return "Negative cycle!"
```

Runtime: $O(VE)$

Why is B-F in practice slower?

Dijkstra: $O(E \log V)$ ← Dij uses a cool data structure

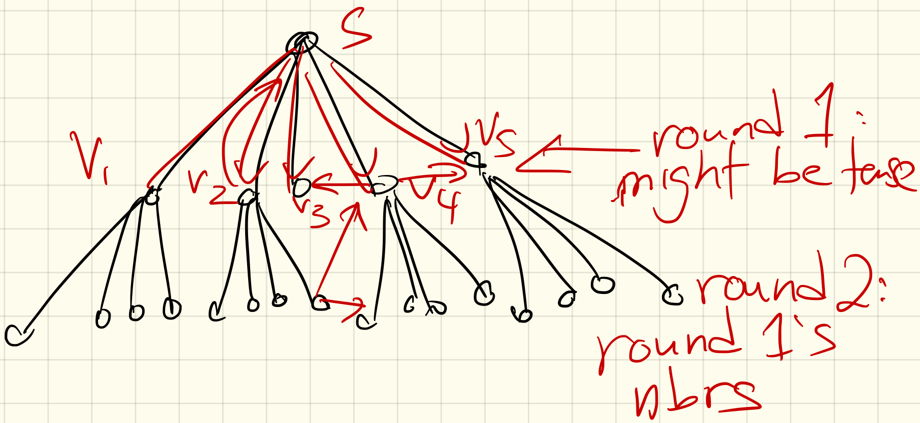
B-F : $O(EV)$ ← does naive loop

really:

- negative edges
  both work, but Dij gets slower
- negative cycles:
  Dij. fails

The rest: an (in practice) speed-up

Think of a BFS tree




```
MOORE(s):
  INITSSSP(s)
  PUSH(s)
  PUSH(✠)                      ⟨⟨start the first phase⟩⟩
  while the queue contains at least one vertex
      u ← PULL()
      if u = ✠
          PUSH(✠)              ⟨⟨start the next phase⟩⟩
      else
          for all edges u→v
              if u→v is tense
                  RELAX(u→v)
                  if v is not already in the queue
                      PUSH(v)
```

Queue

~~s~~ ✠

~~✠~~ v_1 v_2 v_3 v_4 v_5 ✠

~~✠ all nbrs of v_1 v_2 v_3 v_4 v_5~~ ✠

✠ all dist 3 vertices

Final version: Bellman's!

$$dist_{\le i}(v) = \begin{cases} 0 & \text{if } i = 0 \text{ and } v = s \\ \infty & \text{if } i = 0 \text{ and } v \ne s \\ \min \left\{ \begin{array}{c} dist_{\le i-1}(v) \\ \min\limits_{u \to v} \left(dist_{\le i-1}(u) + w(u \to v) \right) \end{array} \right\} & \text{otherwise} \end{cases}$$

Why??

Using i again as # of edges in the path!

Since all paths are $\le V-1$,

$dist_{V-1}(v)$ <u>is</u> $dist(v)$

(assuming no negative cycles)

Nicer:

```
BellmanFordDP(s)
  dist[0,s] ← 0
  for every vertex v ≠ s
      dist[0,v] ← ∞
  for i ← 1 to V − 1
      for every vertex v
          dist[i,v] ← dist[i−1,v]
          for every edge u→v
              if dist[i,v] > dist[i−1,u] + w(u→v)
                  dist[i,v] ← dist[i−1,u] + w(u→v)
```

V×V array

Later observation:

Really don't need the i.

Just update those "tentative" distances, & trust it'll halt.

```
BellmanFordFinal(s)
  dist[s] ← 0
  for every vertex v ≠ s
      dist[v] ← ∞
  for i ← 1 to V − 1
      for every edge u→v
          if dist[v] > dist[u] + w(u→v)
              dist[v] ← dist[u] + w(u→v)
```

initialize SSSD

if tense, relax

Next time: NP-Hardness!

Fundamental question:

Are there "harder" problems?
How do we rank?

- Polynomial
- Exponential
- Unsolvable?

Undecidabily:

Some problems are impossible to solve!

The Halting Problem: (Turing)

Given a program P and input I, does P halt or run forever if given I?

Output: True / False

(Utility should be obvious!)

Note: Can't just simulate P on I. Why?

Thm [Turing 1936]:

The halting problem is undecidable.

(That is, no such algorithm can exist.)

Proof: by contradiction - suppose we have such a program h:

$$h(P, I) = \begin{cases} \text{True} & \text{if } P \text{ halts on } I \\ \text{False} & \text{otherwise} \end{cases}$$

Need a contradiction now...

Now define a program g that uses h:

$g(X) :=$ if $h(X, X) =$ False
return False
else
loop forever

The contraction: What does g(g) do?

Calls $h(g, g)$:
Does g halt on itself
if yes - return false
if no, loop forever

So... what next?

Clearly, many things are solvable in polynomial time.

Some things are impossible.

But – what is in between?

Idea: