Algorithms - Spring 2025

SSSPs

Recap

- Posted next few weeks of readings
- Posted next HW: due next Wed.

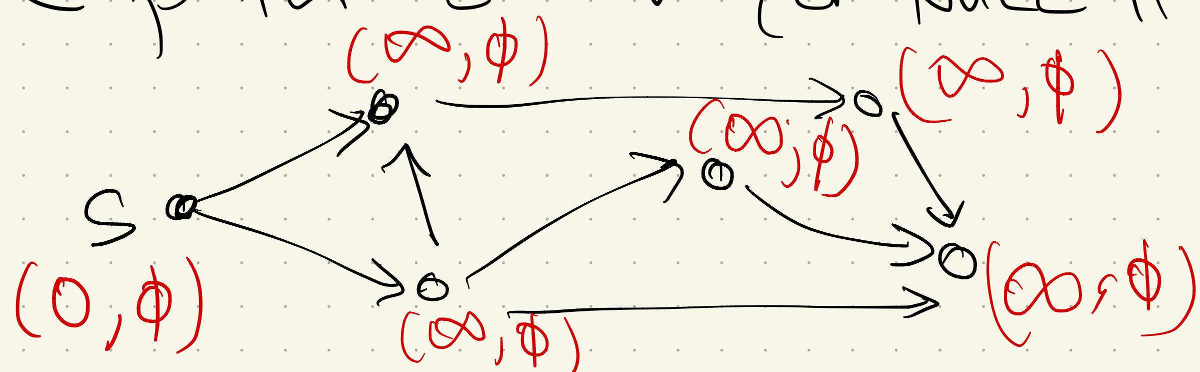
Computing a SSSP.

(Ford 1956 + Dantzig 1957)

Each vertex will store 2 values.
(Think of these as tentative shortest paths) (dist, pred)

- dist(v) is length of tentative shortest path $s \rightsquigarrow v$
  (or ∞ if don't have an option yet)
- pred(v) is the predecessor of v on that tentative path $s \rightsquigarrow v$ (or NULL if none)

Initially:

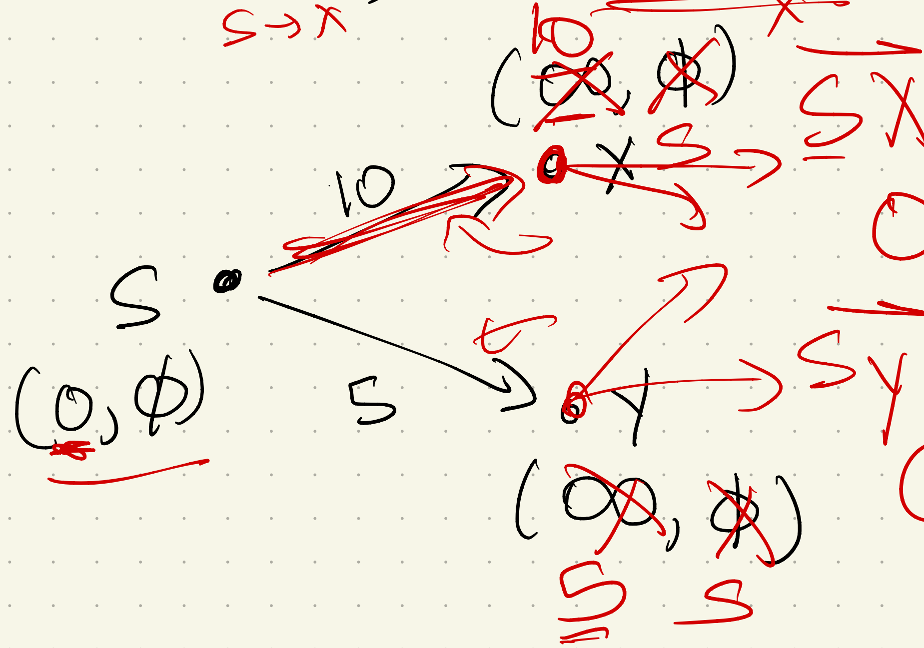


We say an edge $\overrightarrow{uv}$ is __tense__ if

$$\text{dist}(u) + w(u \to v) < \text{dist}(v)$$

s $s \to x$ x

Initially:

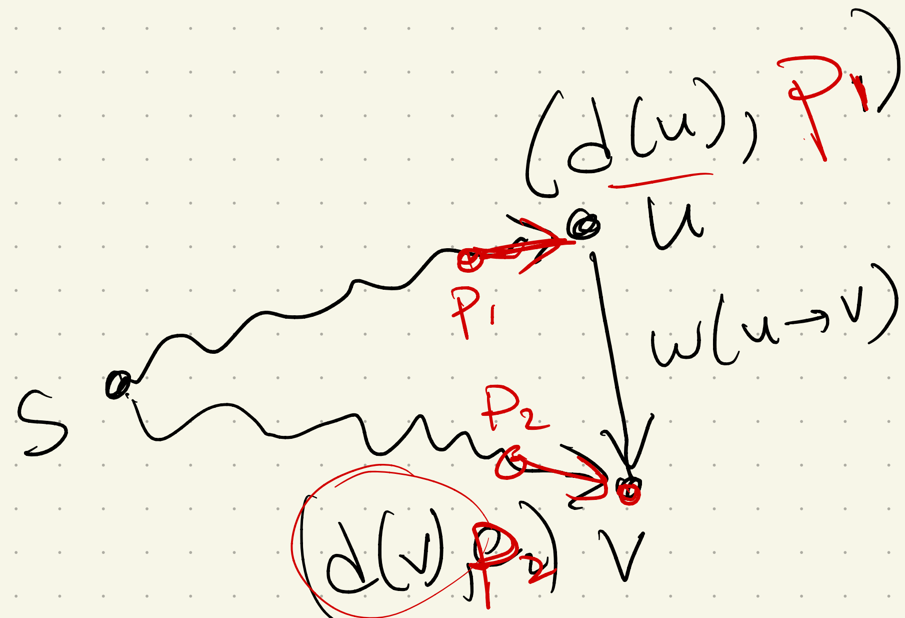


$\overrightarrow{sx}$: $0 + 10 < \infty$

$\overrightarrow{sy}$: $0 + 5 < \infty$

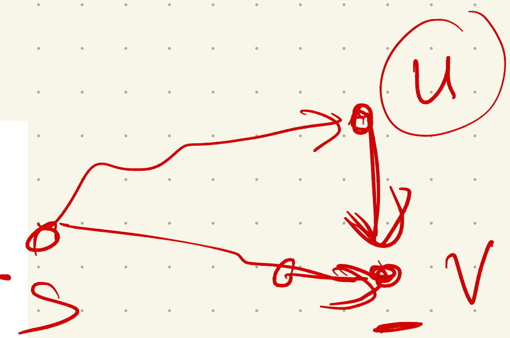
Here:

In general:



Key idea for algorithm:
Find tense edges & relax them:

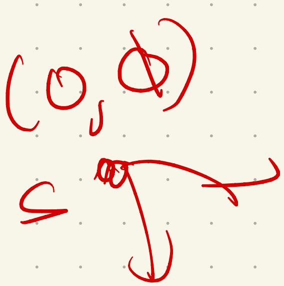
```
RELAX(u→v):
    dist(v) ← dist(u) + w(u→v)
    pred(v) ← u
```



Then:

```
INITSSSP(s):
    dist(s) ← 0
    pred(s) ← NULL
    for all vertices v ≠ s
        dist(v) ← ∞
        pred(v) ← NULL
```

```
GENERICSSSP(s):
    INITSSSP(s)
    put s in the bag
    while the bag is not empty
        take u from the bag
        for all edges u→v
            if u→v is tense
                RELAX(u→v)
                put v in the bag
```



Dijkstra ('59) → assume pos edges

(actually Leyzorek et al '57, Dantzig '58)

Make the bag a priority queue:

Keep "explored" part of the graph, S

Initially, $S = \{s\}$ + $dist(s) = 0$

(all others NULL + ∞)

While $S \neq V$:

select node $v \notin S$ with one edge from S to v with:

$$\min_{e=(u,v),\, u \in S} \left(dist(u) + w(u \to v) \right) \text{ } \} \text{ tension!}$$

Add v to S, set $dist(v)$ + $pred(v)$

Let's formalize this a bit...

Correctness (w/ pos. edge weights!)

Thm: Consider the set S at any point in the algorithm

For each $u \in S$, the distance dist(u) is the shortest path distance

(so pred(u) traces a shortest path).

pf: Induction on $|S|$:

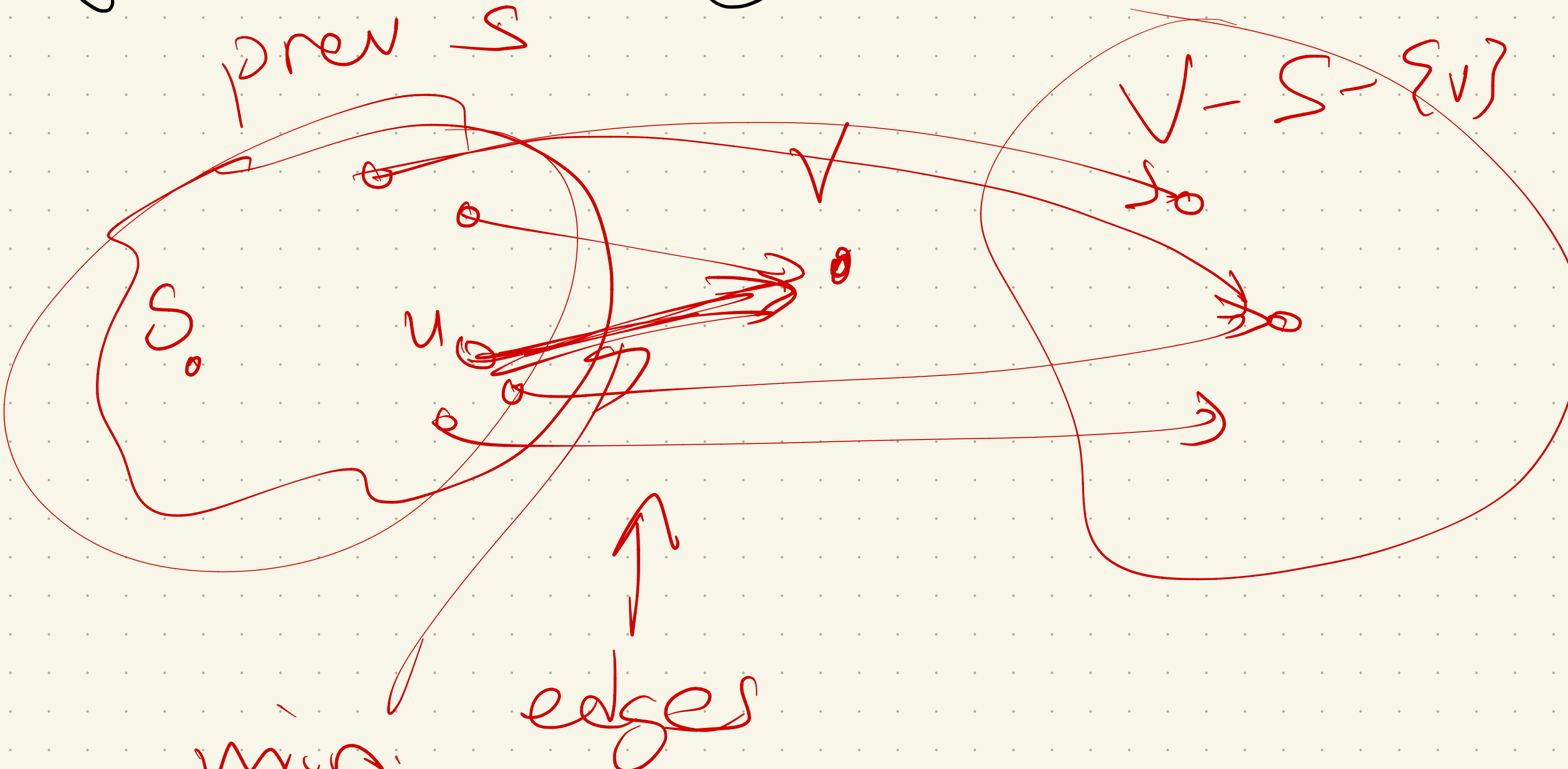
Base Case: $|S| = 1$

dist(s) = 0 ✓

IH: Spps claim holds when $|S| = k-1$.

Ind Step: Consider $|S| = k$:

algorithm is adding some v to S



min:

$d(u) + w(u \to v)$

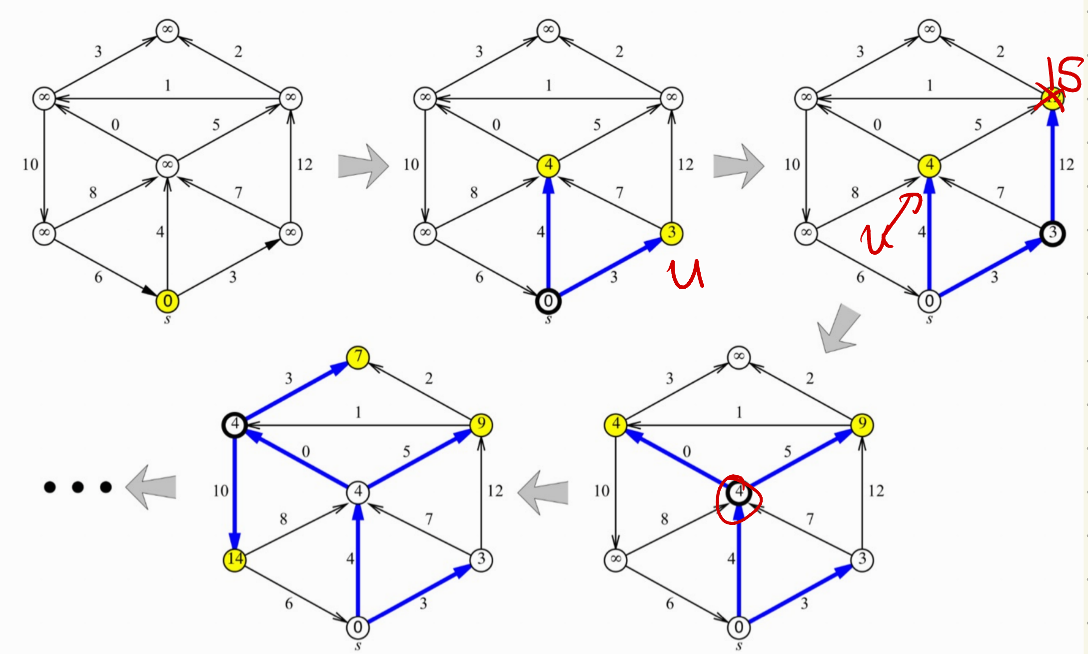
Book's implementation:

When v is added to S:

- look at v's edges and either insert w with key dist(v) + w(v→w)
- or update w's key, if dist(v) + w(v→w) beats current one

```
NonnegativeDijkstra(s):
  InitSSSP(s)
  for all vertices v
      Insert(v, dist(v))
  while the priority queue is not empty
      u ← ExtractMin()
      for all edges u→v
          if u→v is tense
              Relax(u→v)
              DecreaseKey(v, dist(v))
```

or delete & reinsert



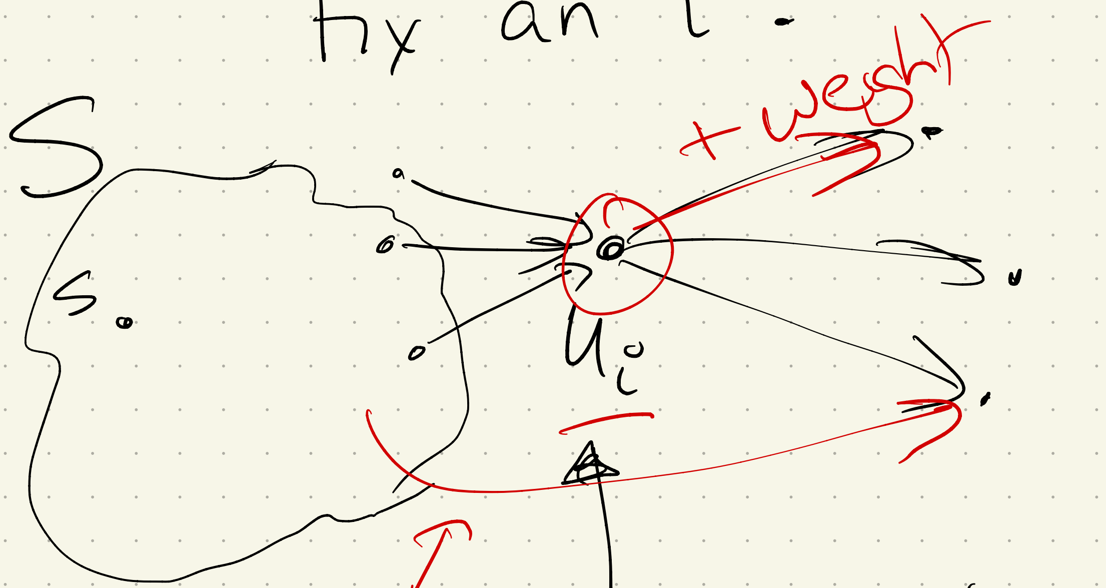
Four phases of Dijkstra's algorithm run on a graph with no negative edges.
At each phase, the shaded vertices are in the heap, and the bold vertex has just been scanned.
The bold edges describe the evolving shortest path tree.

Analysis: Let u_i be i^{th} vertex extracted from queue, & let d_i = value of $dist(u_i)$ when extracted.

Lemma: If G has no negative edges, then for all $i<j$, $d_i \le d_j$.

Proof

Fix an i:

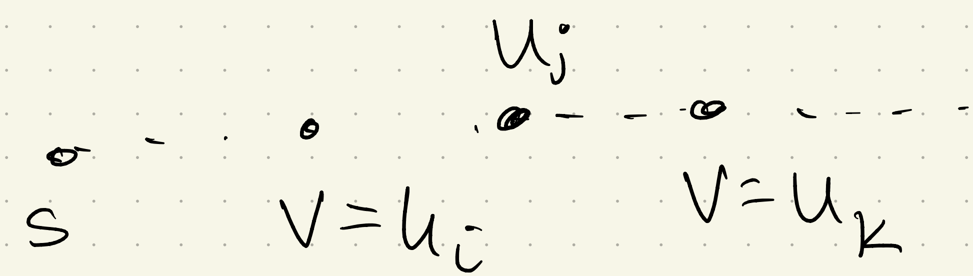


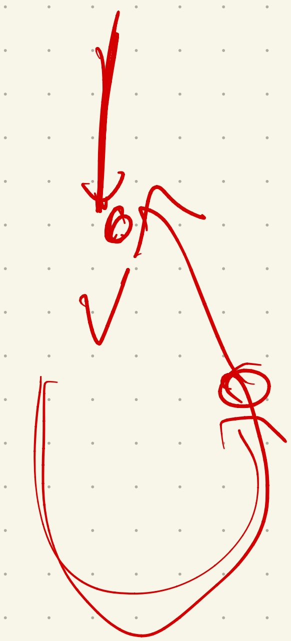
current best in heap

any later j could use u_i as parent (& hence is d_i + weight) or not: use some vertex in S, & those were larger in heap.

Lemma: Each vertex is extracted from the heap once (or less)

Proof: Spps not:





prev lemma $\Rightarrow$ know $d_i \le d_k$

But: v was readded to queue means some edge $u_j \to v$ became tense.

won't be

Runtime: In the end, runtime is $O(E \log V)$ ← $O((E+V)\log V)$

Why? Heap ops:

decreasekey: $\le E$ times

insert: V times

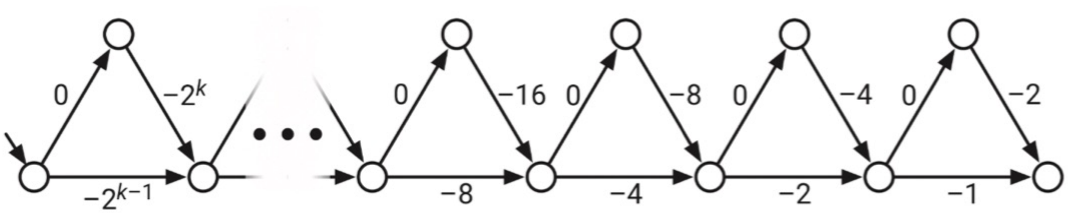
ExtractMin: V times

each heap op takes $O(\log V)$ time

```
NonnegativeDijkstra(s):
  InitSSSP(s)
  for all vertices v
      Insert(v, dist(v))
  while the priority queue is not empty
      u ← ExtractMin()
      for all edges u→v
          if u→v is tense
              Relax(u→v)
              DecreaseKey(v, dist(v))
```

readd to queue

Main downside: negative edges



a lot

Figure 8.14. A directed graph with negative edges that forces Dijkstra to run in exponential time.

How to deal with negative edges?

Recall:

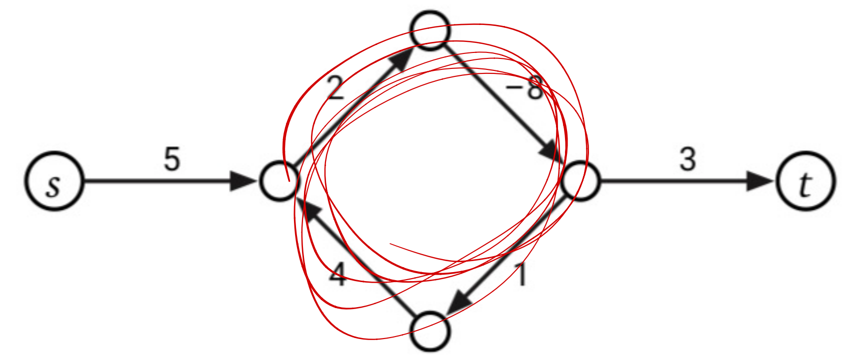


Figure 8.3. There is no shortest walk from s to t.

So two issues:

- Negative edges: Dijkstra might take along time
- Negative cycles: no finite shortest path

Bellman-Ford:

Relax edges for a while.

Stop when every edge has been relaxed at least once

If any one is still tense:

you've relaxed ≥ 2 times!

At end: track paths of length V

Runtime: $V \cdot E$

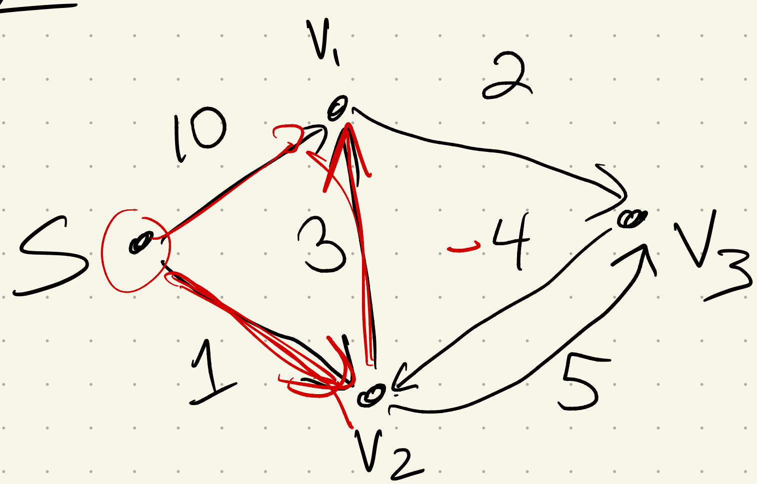
```
BellmanFord(s)
  InitSSSP(s)
  repeat V − 1 times
      for every edge u→v
          if u→v is tense            } O(1)
              Relax(u→v)
  for every edge u→v
      if u→v is tense
          return "Negative cycle!"
```

How to prove correctness?

Notation:

Let $\text{dist}_{\leq i}(v) :=$ length of shortest s to v path using $\leq i$ edges

Ex:



	S:	V_1:	V_2:	V_3:
≤ 1	0	10	1	∞
≤ 2	0	4	1	6
≤ 3	0	4	1	6

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

$\text{dist}(v) \leq \text{dist}_{\leq i}(v)$

Induction on i:

BC: $i = 0$: only s is reachable with length 0 path, all others are ∞

IH: After $i-1$ iterations, all tentative guesses are $\leq d_{\leq i-1}(v)$.

IS: Now consider $d_{\leq i}(v)$: relax a bunch of edges

built from a path $\longrightarrow$

$$\underbrace{s \to \circ \to \circ \to \circ \cdots \to u}_{\le i \text{ edges}} \to v$$

We know in round $i-1$, $\text{dist}(x) \le d_{\le i-1}(x)$ $\forall x \in V$.

Consider $u \to v$ in next round:

It was tense: gotten smaller, building path via vertex u. $\to$ path of length $(i-1)+1$

or not: keep length $i-1$ path: no new edge makes thing shorter

The rest: an (in practice) speed-up

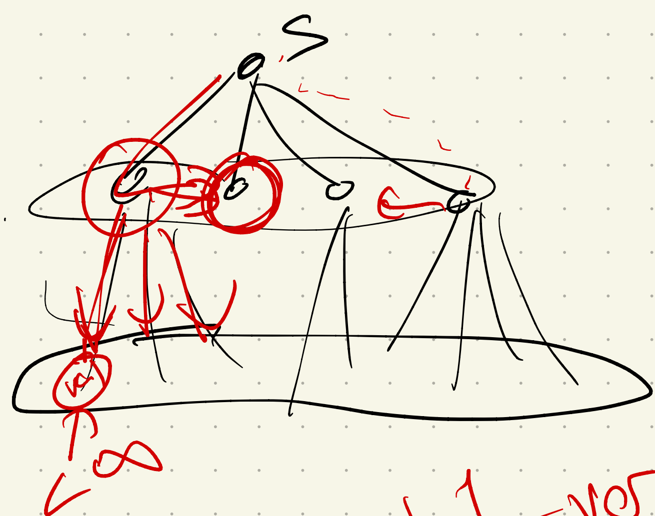
BF looks at every edge.

Do we need to?

Think of a BFS tree + "token"

level 1

level 2




```
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)
```

level 1 vertices

queue:

s ✠ (s's nbrs) ✠ (level 2) ✠ (

Final version: Bellman's!

only using i-1 or less

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

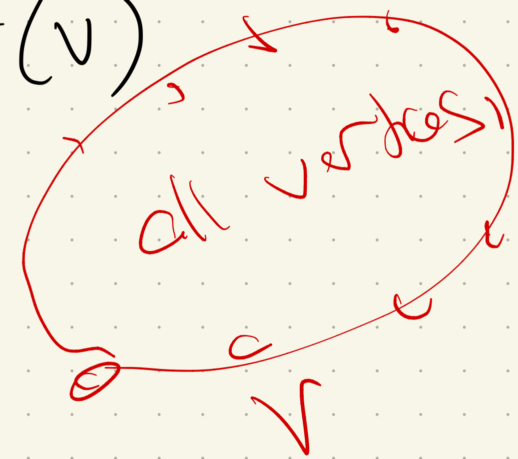
use some tense edge

Why?? Using i again as # of edges in the path!

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

$dist_{V-1}(v)$ is $dist(v)$

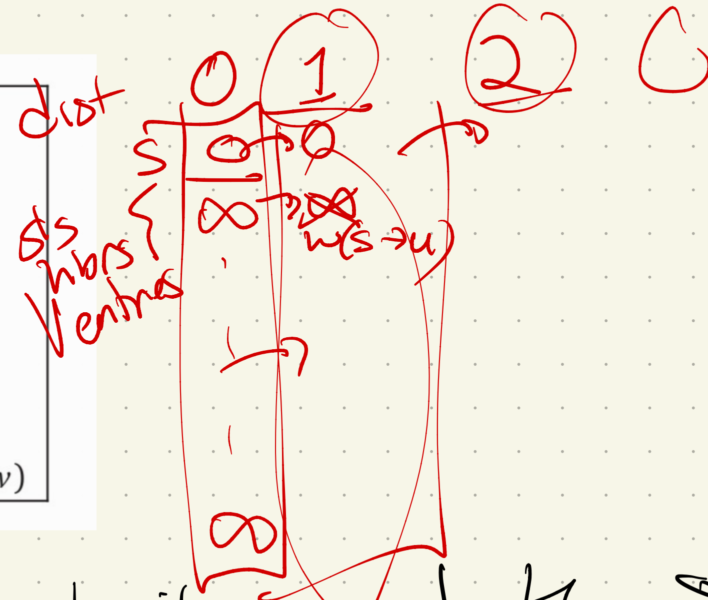
(assuming no negative cycles)



Nicer: DP!

```
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)
```

if tense



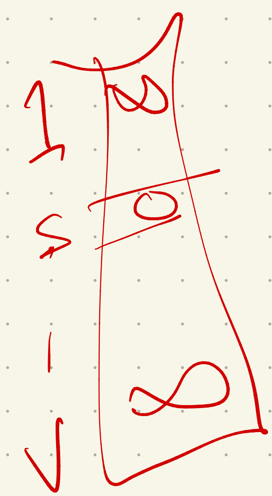
Later observation: Really don't need the i.

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)
```

tense

relax



Runtime:

Same: $V \cdot E$

Next time: MSSP

SSSPs are nice, but:
What if we are doing lots of shortest path computations?

Goal: Precompute these, & store them!

How to store?

v_i

	v_1	v_2	.....	v_n
v_1				
v_2				
v_3				
⋮				
v_n				

v_j

best route

Lookup time: $O(1)$

But: how to compute?

Obvious answer

Well, we just designed two or three SSSP algorithms — use them!

MSSP(G):

for each $v \in G$:
  run SSSP(v)
  store tree distances in dist[s, ·]

$V \cdot$ (SSSP comp)

Can we do better? Yes