Algorithms - Spring '25

Shortest Paths

Recap

- HW due today
- Next: over MST + SSSPs
- No reading Monday
- Resume next Wed w/ readings

Next: **Shortest paths**

Goal: given $s, t \in V$, compute the shortest path from s to t.

Motivation: roads
routing
cost

To solve this, we need to solve a more general problem:
find shortest paths from s to every vertex.

Why?

Computing a SSSP.

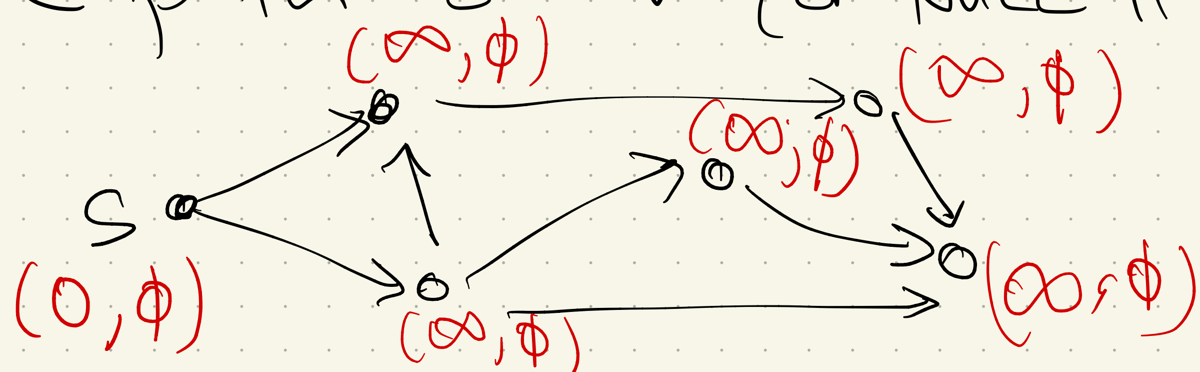
(Ford 1956 + Dantzig 1957)

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:

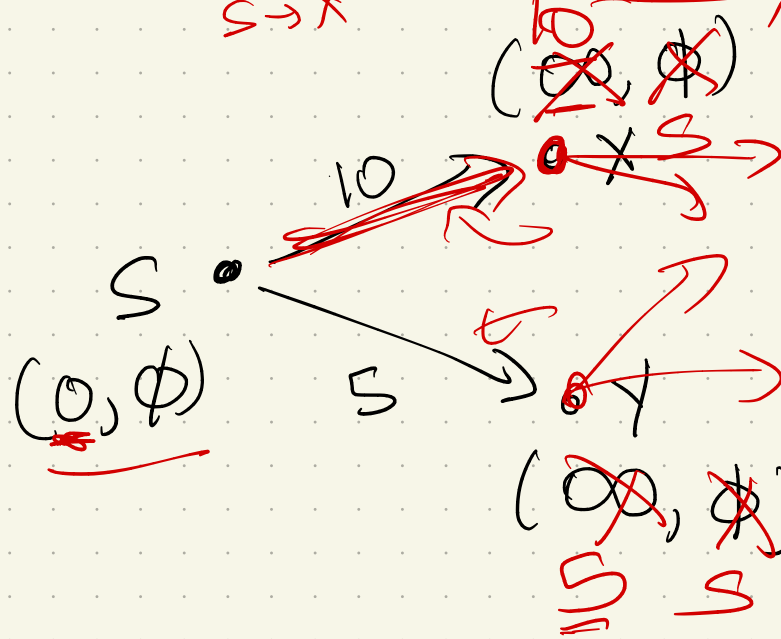


We say an edge $\overrightarrow{uv}$ is <u>tense</u> if

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

s s→x x

Initially:

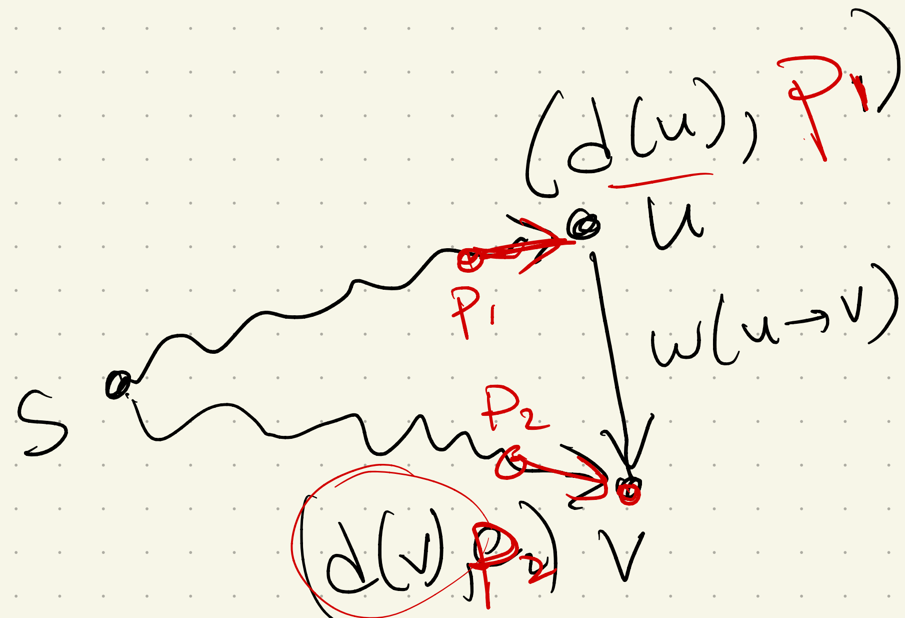


$\overrightarrow{sx}$: $0 + 10 < \infty$

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

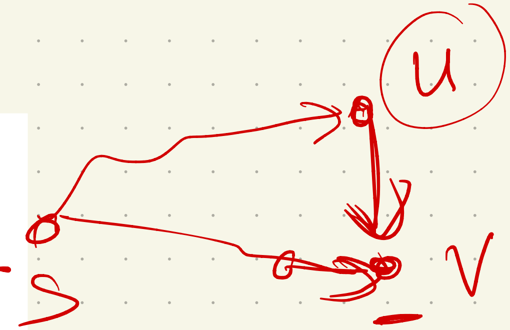
Here:

In general:



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

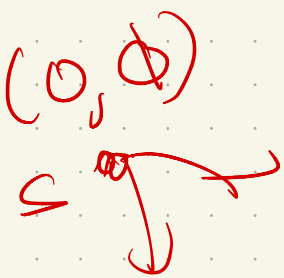
```
RELAX(u→v):
    dist(v) ← dist(u) + w(u→v)
    pred(v) ← u
```



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
```



Claim: At any point in time, dist(v) is either ∞ or the length of some $s \rightsquigarrow v$ walk.

Proof: Induction on while loop iterations.

Base case: loop iteration 1

at beginning, s has dist = 0 + all others = ∞

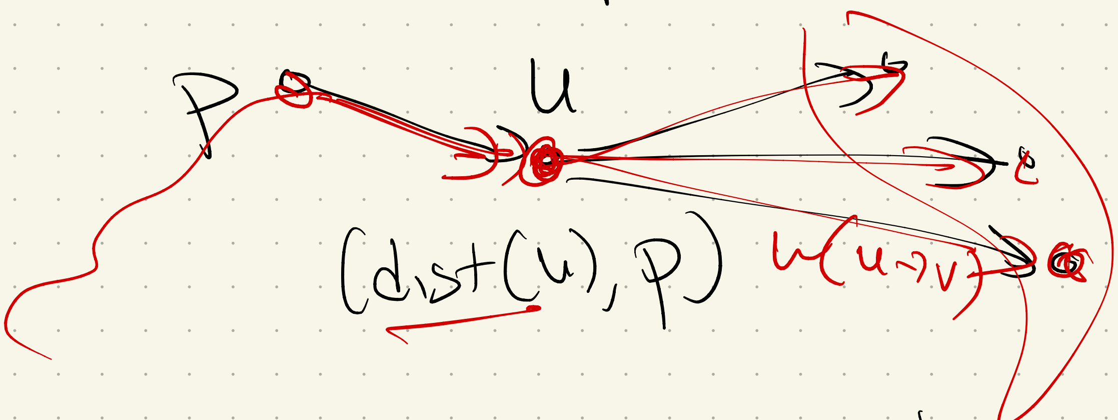
at end, s has dist = 0 still, + all neighbors u now have dist(u) = w(s→u), which is a length 1 walk. (Others are ∞)

Ind hyp:

In iteration $k-1$, the claim is true (all vertices v have $dist(v) = \infty$ or $=$ length of an $s \leadsto v$ walk)

Ind Step:

In iteration k: At beginning, we take out some vertex u.



By IH, $dist(u)$ is the weight of some $s \leadsto u$ walk.

At end, all nbrs v of u are either unchanged (& so by IH are still either ∞ or length of $s \leadsto v$ walk)

or $u \to v$ was tense, &
now $dist(v) = dist(u) + w(u \to v)$.
Since $dist(u)$ is a $s \rightsquigarrow u$ walk,
then $dist(v)$ is weight of the
walk $(s \rightsquigarrow u) + (u \to v)$, which
is a walk with one more edge
at end.
(All other vertices are unchanged,
so by IH are still ∞ or a $s \rightsquigarrow v$
walk.)

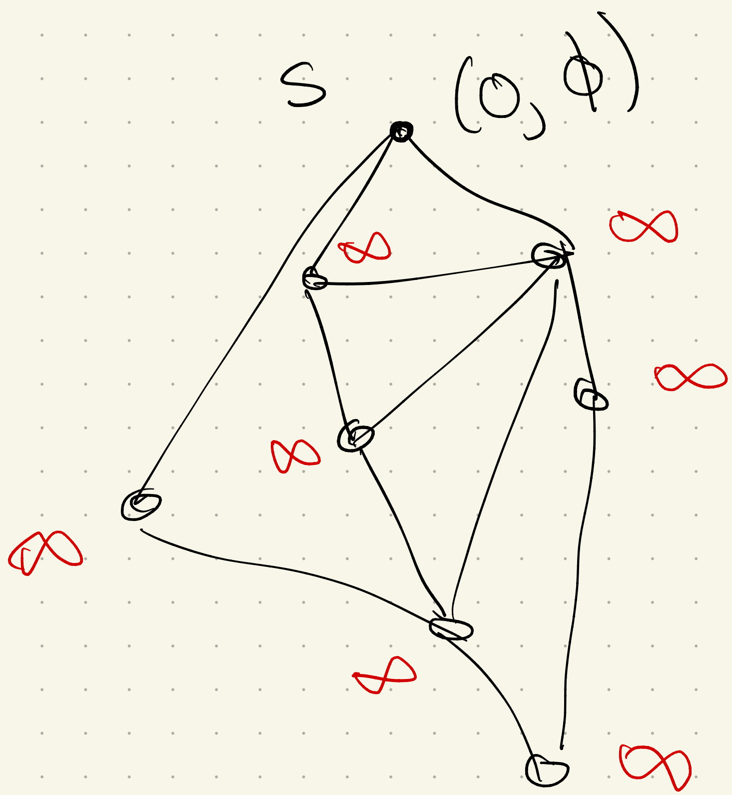
$\blacksquare$

Warm-up: Unweighted graphs

→ use a queue

How does "tense" work?

(Hint: think BFS!)



all nbrs of s have tense incoming edges:

$d(s) + w(s \to u)$
$= 0 + 1 < \infty$

What the heck is his token??

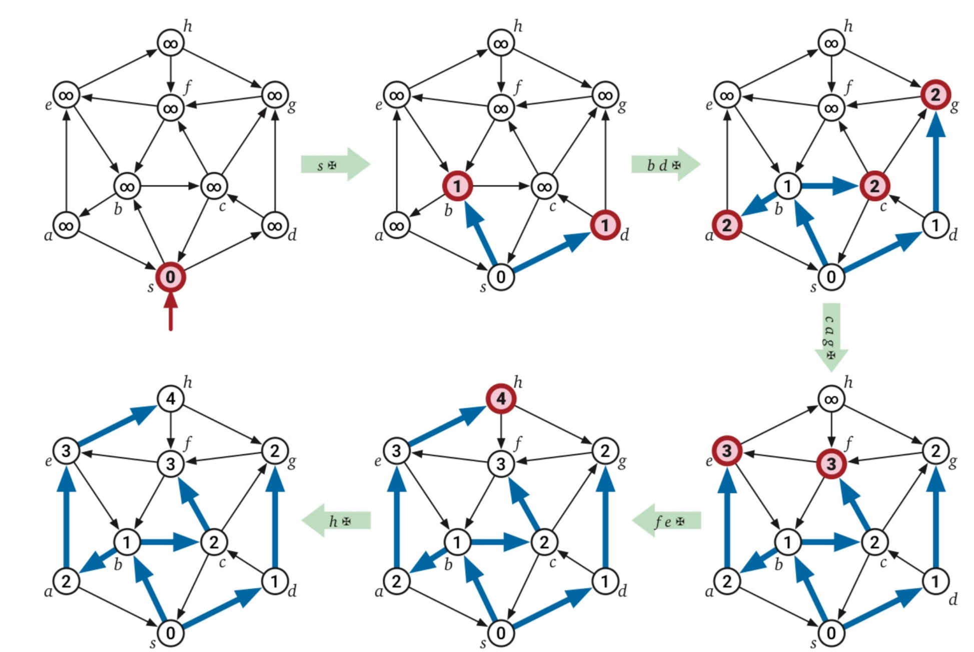
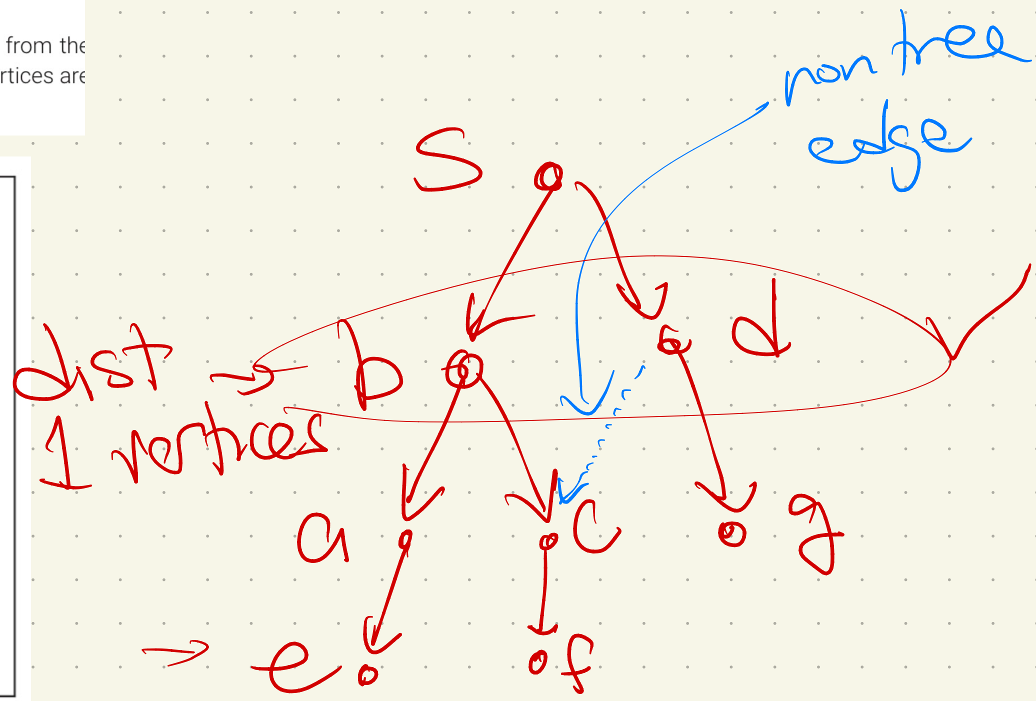


Figure 8.6. A complete run of breadth-first search in a directed graph. Vertices are pulled from the queue in the order s ✠ b d ✠ c a g ✠ f e ✠ h ✠ ✠, where ✠ is the end-of-phase token. Bold vertices are in the queue at the end of each phase. Bold edges describe the evolving shortest path tree.

queue:
~~s ✠ b d ✠ a c g~~ ✠
ef

u = ~~s ✠ b d ✠~~

```
BFSWithToken(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 dist(v) > dist(u) + 1      ⟨⟨if u→v is tense⟩⟩
          dist(v) ← dist(u) + 1       ⟨⟨relax u→v⟩⟩
          pred(v) ← u
          Push(v)
```

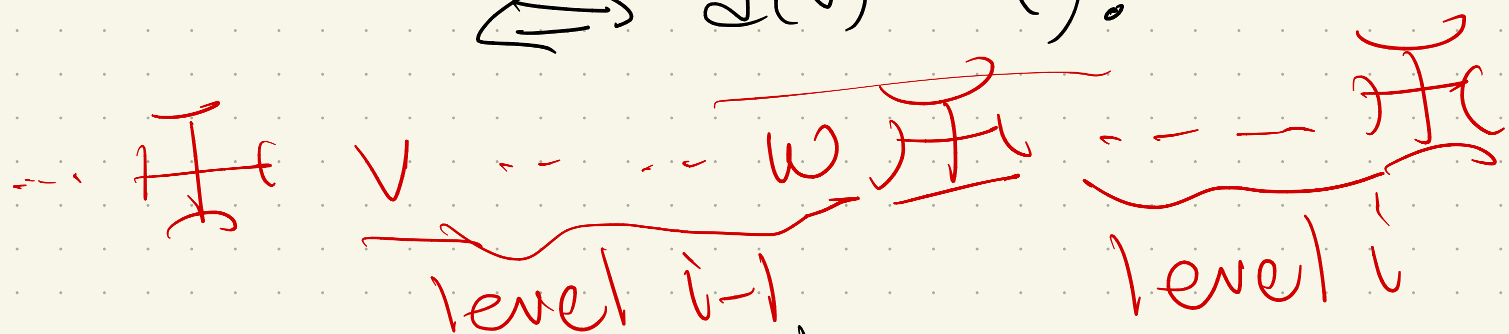


Lemma

At the end of the ith phase (when ⌶ comes off the queue), for every vertex v,

either $\bullet$ $d(v) = \infty$ (not found yet)

or $\bullet$ $d(v) \le i$ (and v is only in queue $\iff d(v) = i$).



Proof: induction on phase

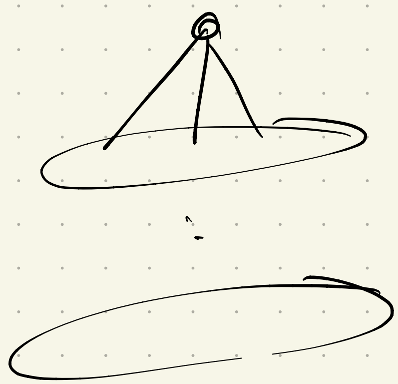
Base case: phase 0: s's nbrs

$d(s) = 0$ $\quad$ $d(s\text{'s nbrs}) = \text{all} = 1$

Inductive Hyp: lemma holds for phases $\leq i-1$

IS: phase i: we know by the IH, when last phase ended:

BFS tree



$i-1$

What now?

$$Q: \underbrace{\cancel{v_k} \cdots \cancel{v_\ell}}_{dist = i-1} \quad \text{nbrs} \ \{\text{dist } i-1$$

In this phase, any undiscovered nbr of level $i-1$ vertex will go at end of queue

2^{nd} version: DAGs

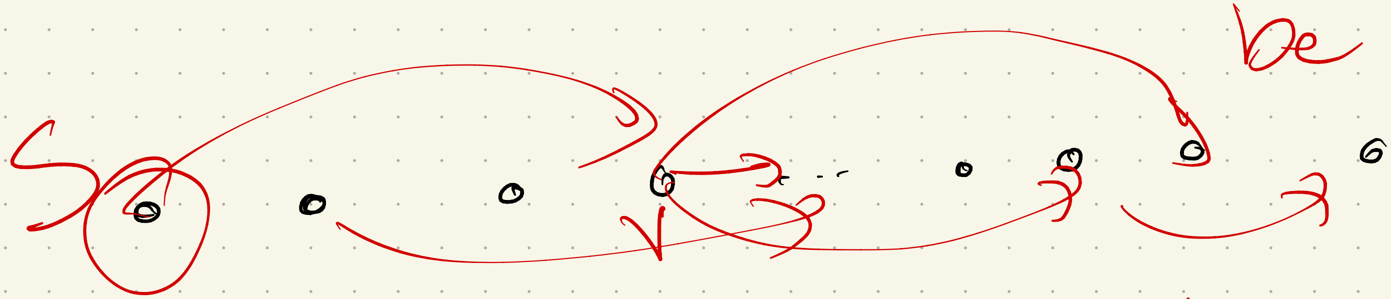
What if directed + acyclic?

Remember: helps to have all "closer" vertices done before computing your distance.

Well, know something about DAG-orders:

↳ topological order!

only vertices which can be in SP to v will be. Define



edges: later in ordering v

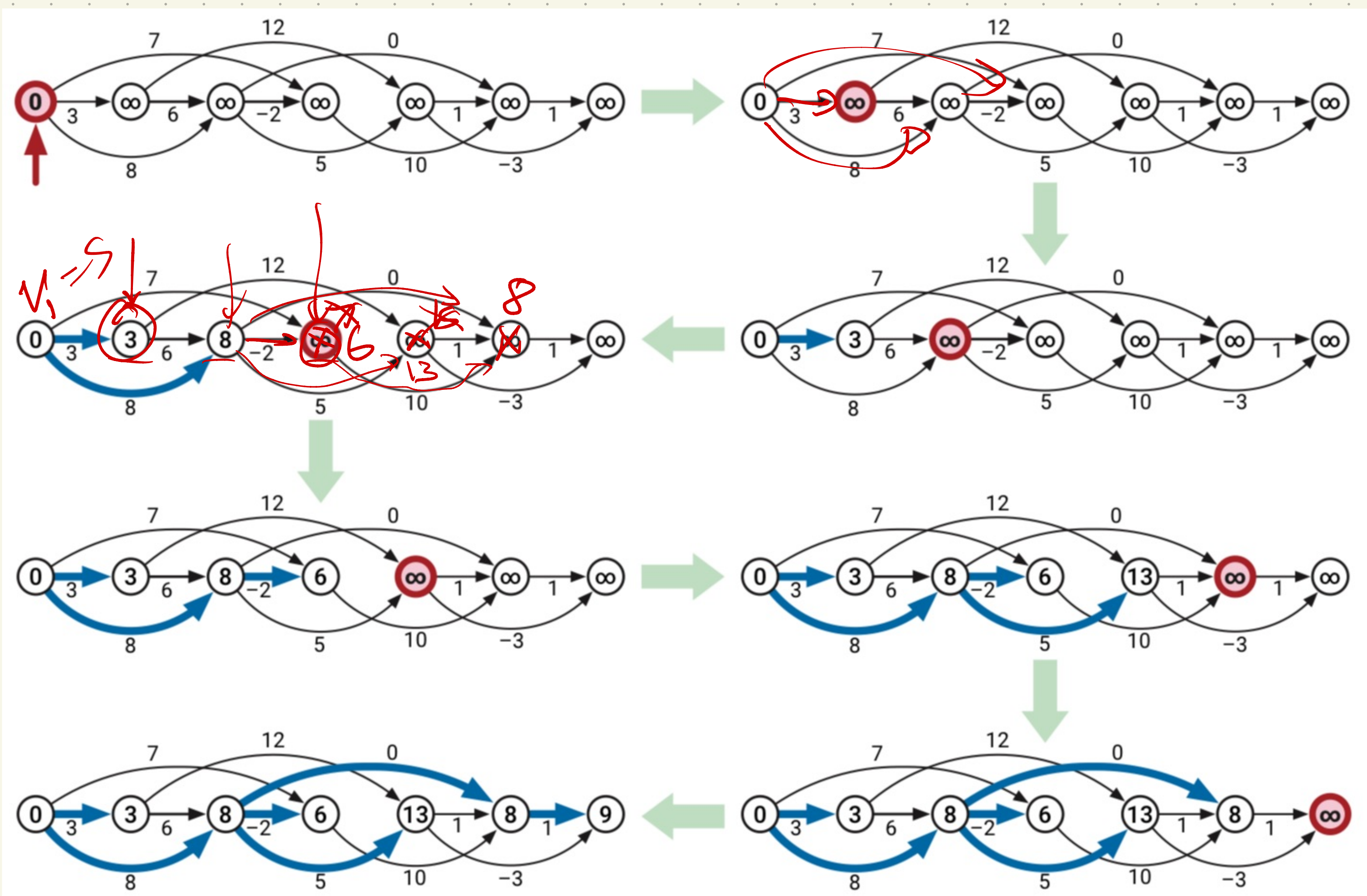
So, use it!

```
DagSSSP(s):
    InitSSSP(s)
    for all vertices v in topological order
        for all edges u→v
            if u→v is tense
                Relax(u→v)
```

s has dist $= 0$
all others ∞

$O(V+E)$ → top sort

$\sum_v (1+d(v)) = V+2E$



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, $S = \{s\}$ + $\mathrm{dist}(s) = 0$
(all others NULL + ∞)

While $S \neq V$:

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

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

Add v to S, set $\mathrm{dist}(v)$ + $\mathrm{pred}(v)$

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 $|S|$:

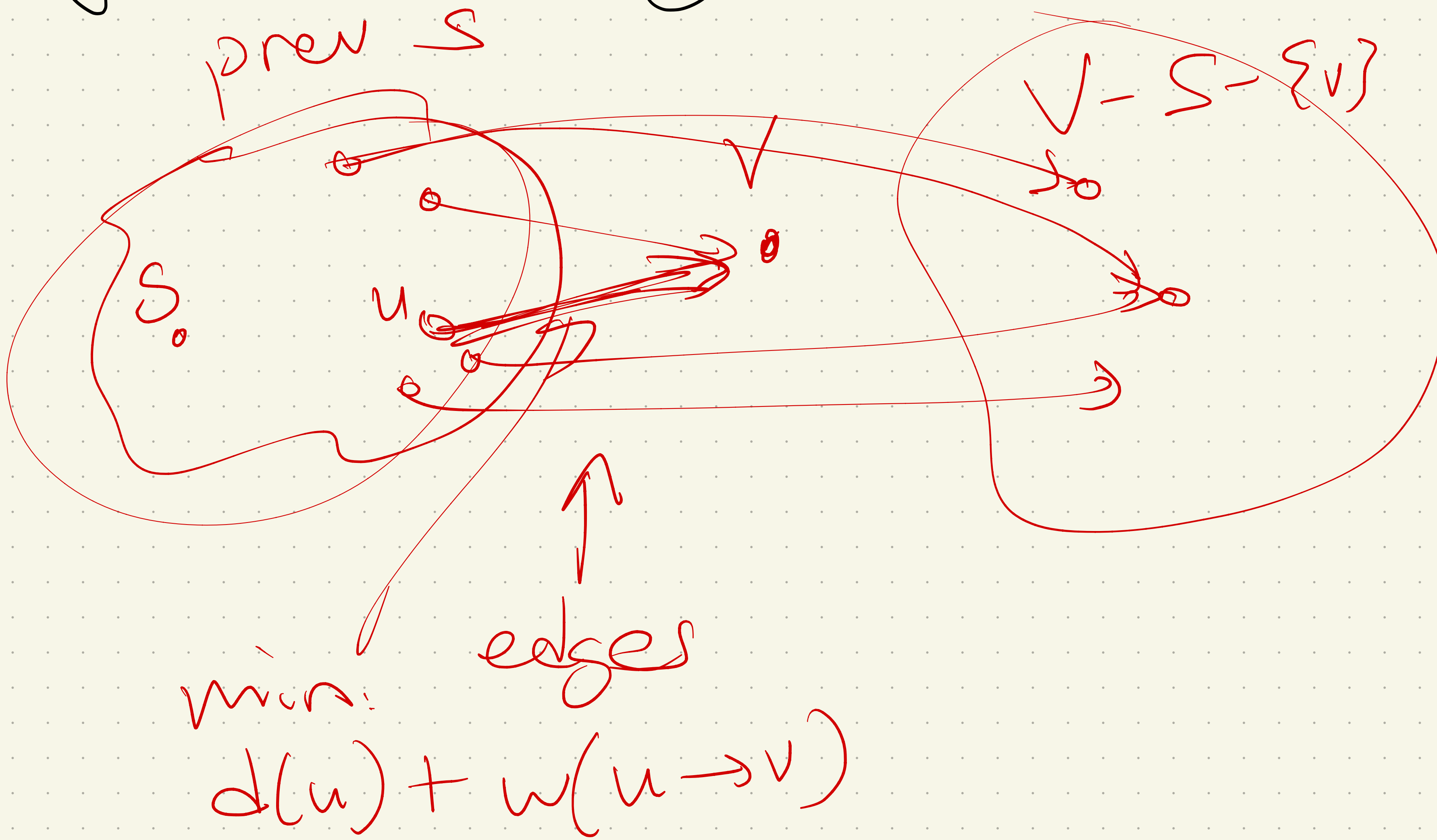
Base Case: $|S| = 1$

$dist(s) = 0$ ✓

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

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

algorithm is adding some v to S

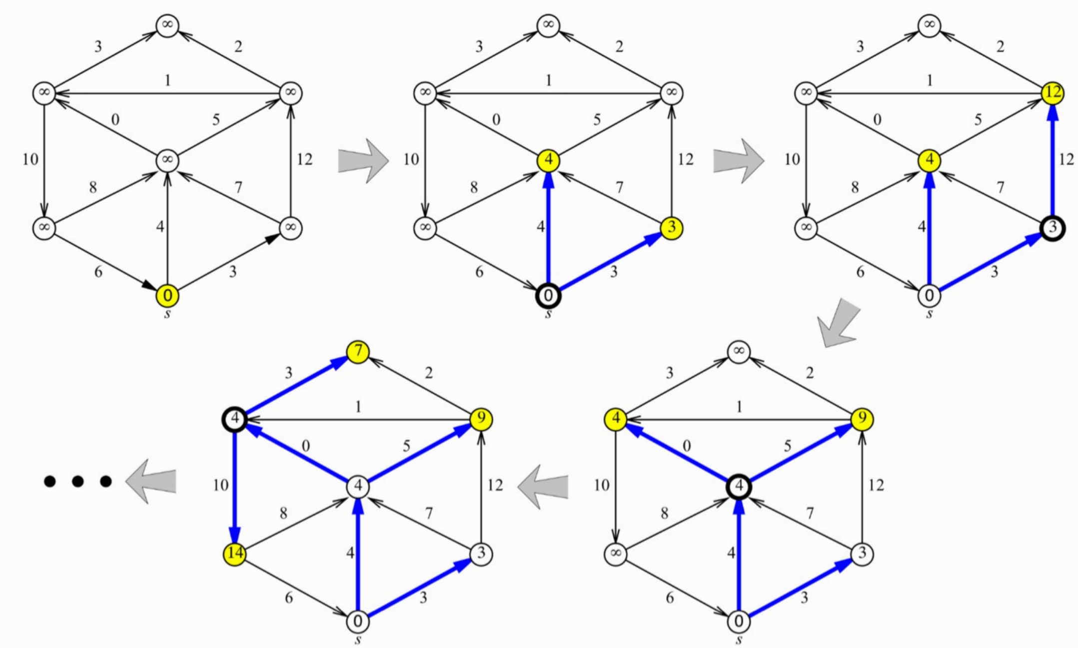


Book's implementation:

When v is added to S:

- look at v's edges and either insert w with key $dist(v) + w(v \to w)$
- or update w's key, if $dist(v) + w(v \to w)$ 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))
```



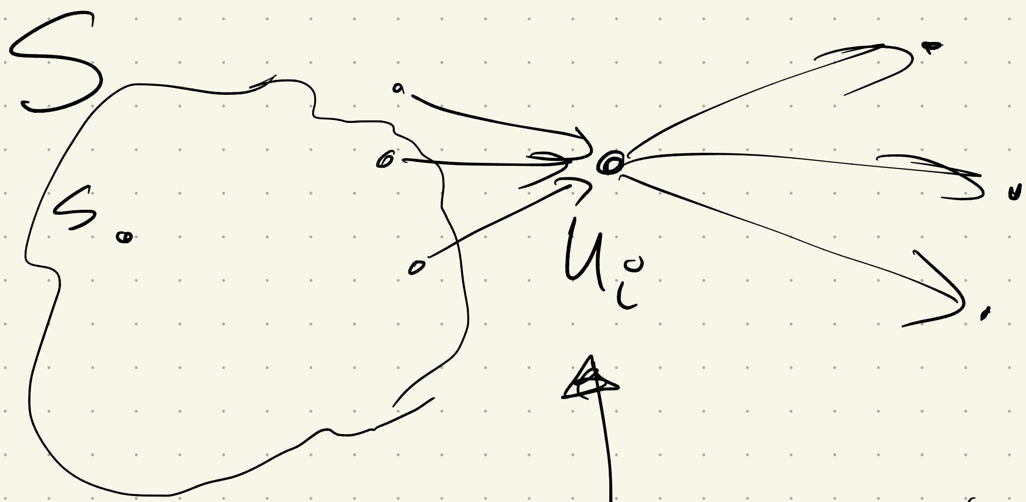
Four phases of Dijkstra's algorithm run on a graph with no negative edges.
At each phase, the shaded vertices are in the heap, and the bold vertex has just been scanned.
The bold edges describe the evolving shortest path tree.

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

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

<u>Proof</u>

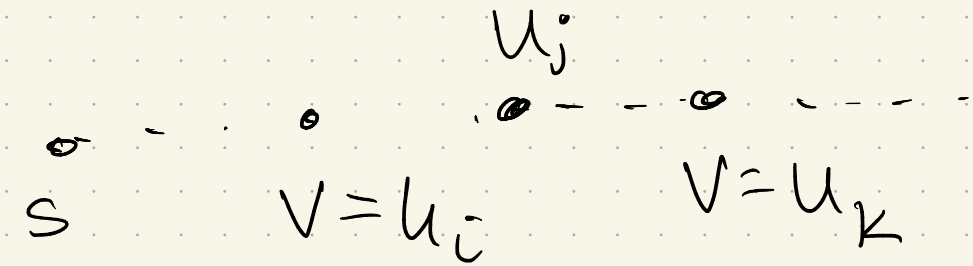
Fix an i:



current best 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.

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

Why?

decreaseKey:

insert:

ExtractMin:

```
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))
```

Main downside:

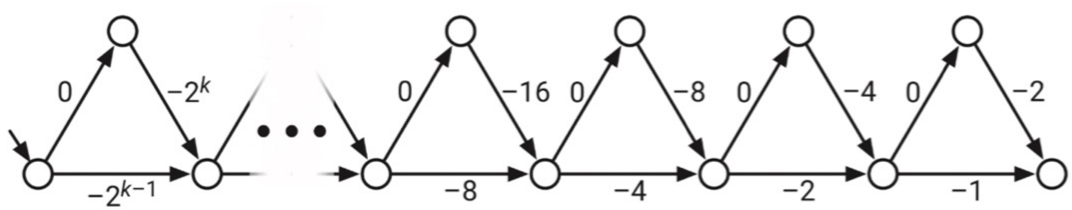


Figure 8.14. A directed graph with negative edges that forces DIJKSTRA to run in exponential time.

Next Monday:
How to deal with negative edges!

Note:

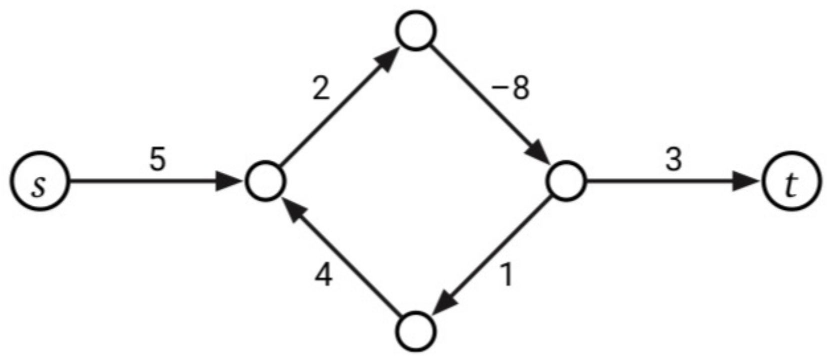


Figure 8.3. There is no shortest walk from s to t.