

Algorithms - Spring '25

Shortest
Paths



Recap

Next: Shortest paths

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

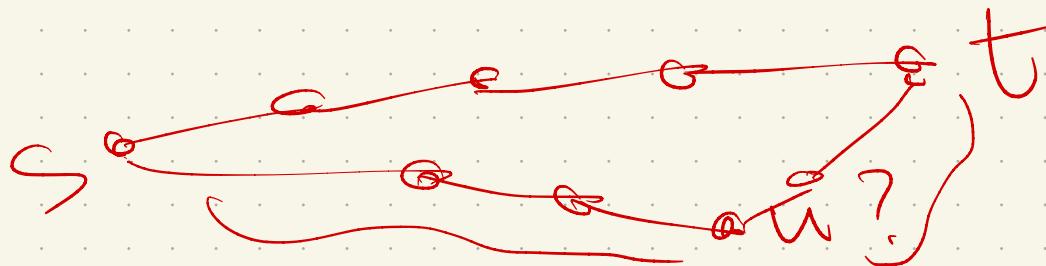
Motivation:

- roads
- routine
- 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.)

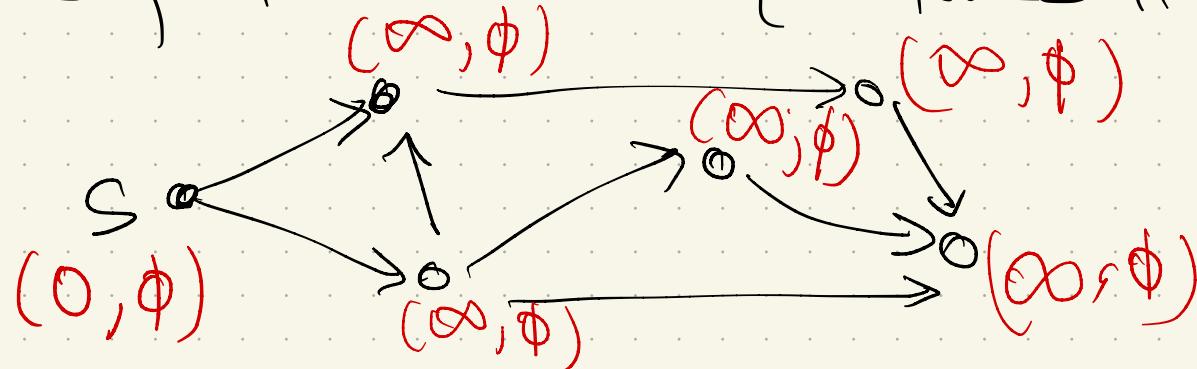
(dist, pred)

- $\text{dist}(v)$ is length of tentative shortest path $S \rightsquigarrow v$

(or ∞ if don't have an option yet)

- $\text{pred}(v)$ is the predecessor of v on that
tentative path $S \rightsquigarrow v$ (or NULL if none)

Initially:

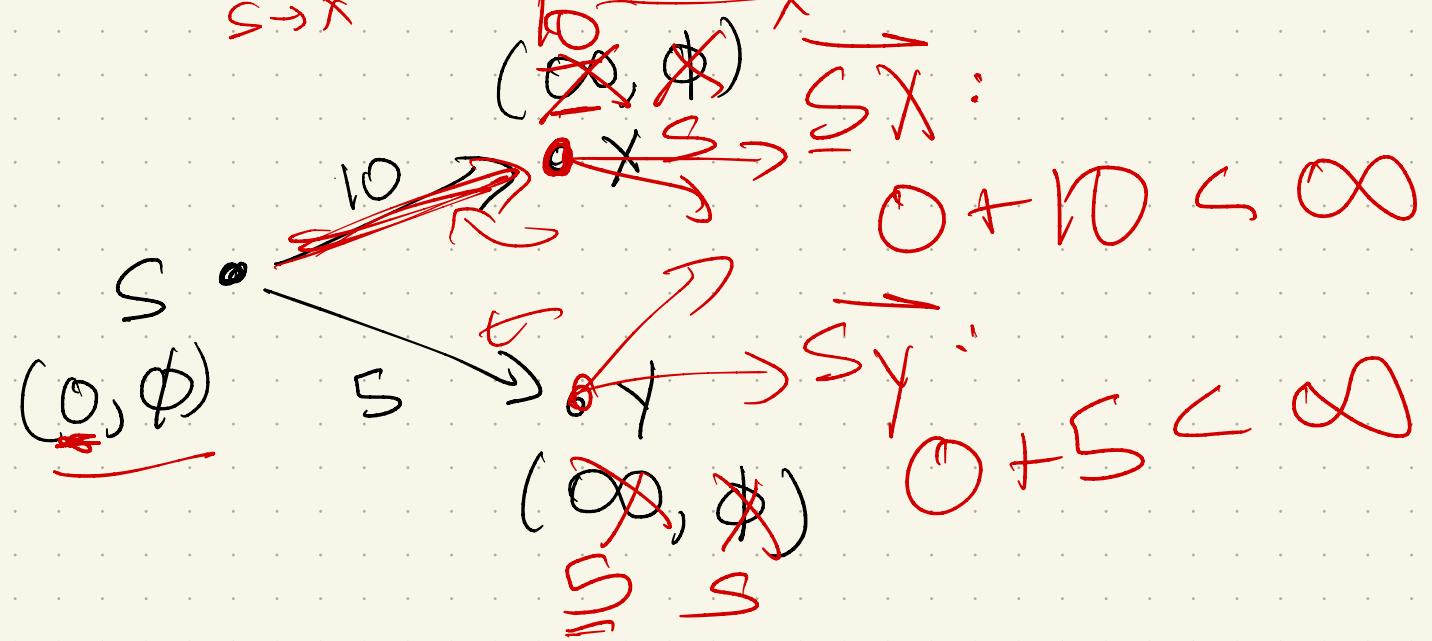


We say an edge \vec{uv} is tense if

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

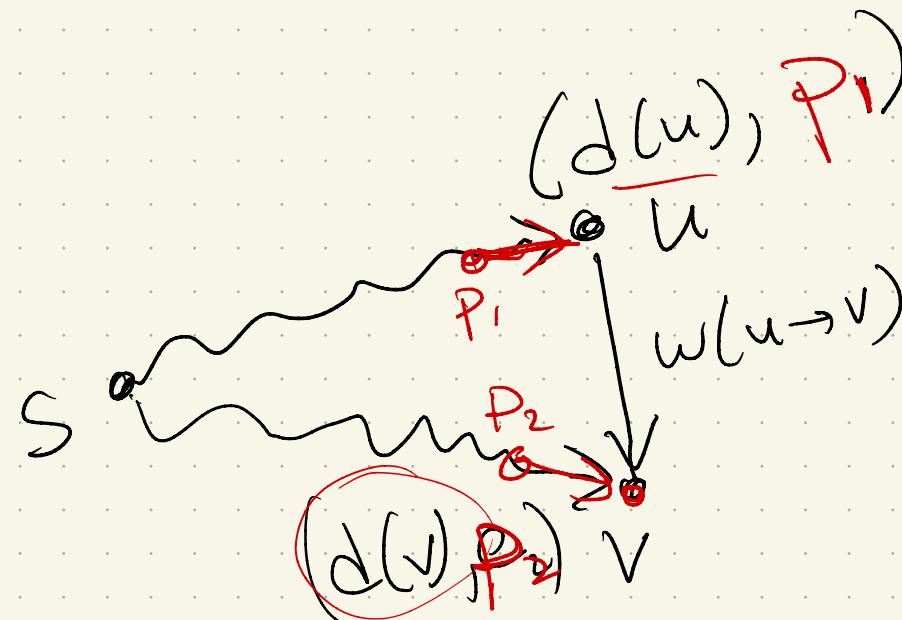
$\downarrow S \quad \downarrow S \rightarrow \nabla$

Initially:



Here:

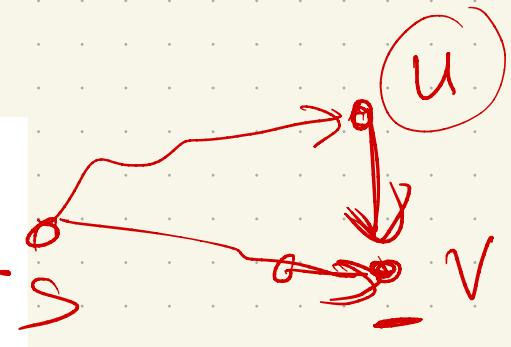
In general:



Key Idea for algorithm:

Find tense edges & relax them:

```
RELAX( $u \rightarrow v$ ):  
     $dist(v) \leftarrow dist(u) + w(u \rightarrow v)$   
     $pred(v) \leftarrow u$ 
```



Then:

```
INITSSSP( $s$ ):  
     $dist(s) \leftarrow 0$   
     $pred(s) \leftarrow \text{NULL}$   
    for all vertices  $v \neq s$   
         $dist(v) \leftarrow \infty$   
         $pred(v) \leftarrow \text{NULL}$ 
```

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

(0,0)
s ↗

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

Proof: Induction on while loop iterations.

Base case: loop iteration 1

at beginning, s has $\text{dist}=0$ +
all others = ∞

at end, s has $\text{dist}=0$ still
& all neighbors u now have

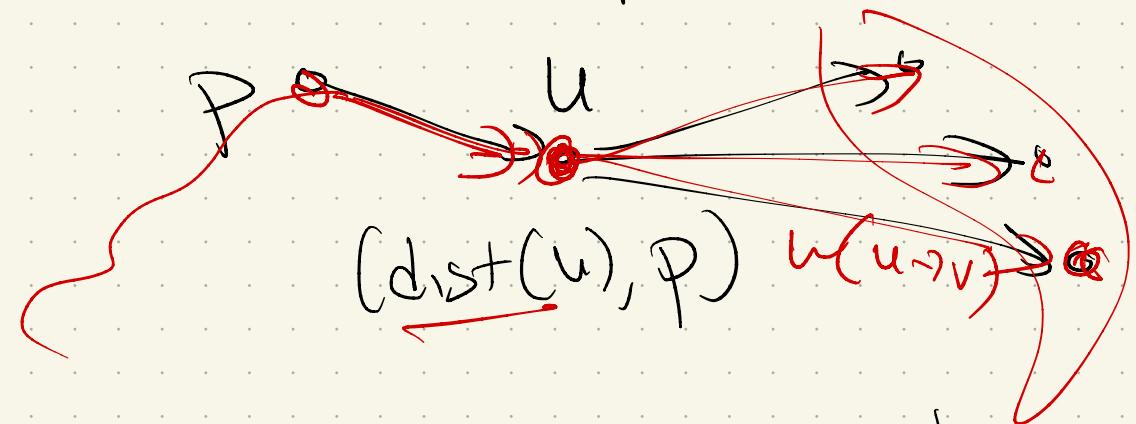
$\text{dist}(u) = w(s \xrightarrow{*} u)$, which is a
length 1 walk. (Others are ∞)

Ind hyp:

In iteration $k-1$, the claim is true

Ind Step:

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



By IH, $\text{dist}(u)$ is the weight of some $u \rightarrow u$ walk.

At end, all nbrs v of u are either unchanged (by IH) or length of $u \rightarrow v$ walk either ∞ or length of $u \rightarrow v$ walk

or $u \rightarrow v$ was tense, &
now $\text{dist}(v) = \text{dist}(u) + w(u \rightarrow v)$.

Since $\text{dist}(u)$ is a $\text{sm} \rightarrow u$ walk,
then $\text{dist}(v)$ is weight of the
walk $(\text{sm} \rightarrow u) + (u \rightarrow v)$, which
is a walk with one more edge
at end.

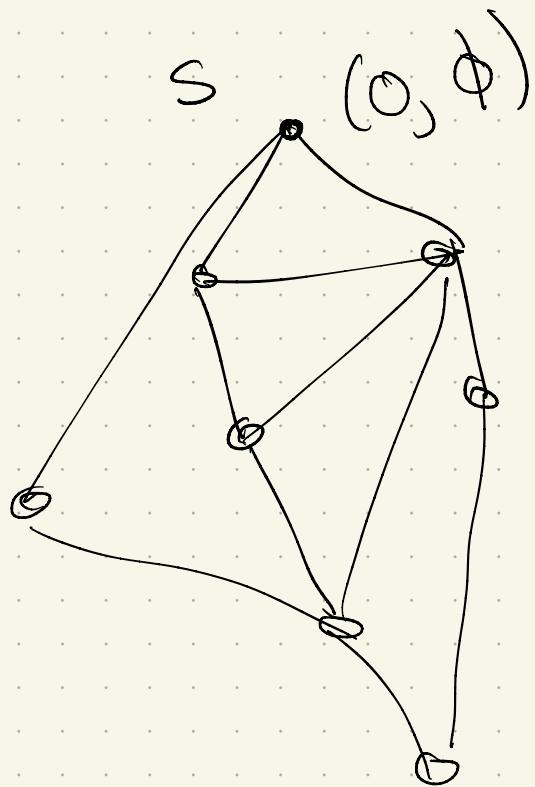
(All other vertices are unchanged,
so by TH are still ∞ or a $\text{sm} \rightarrow v$
walk.) 

Warm-up: Unweighted graphs

→ use a queue

How does "fence" work?

(Hint: think BFS!)



What the heck is his token??

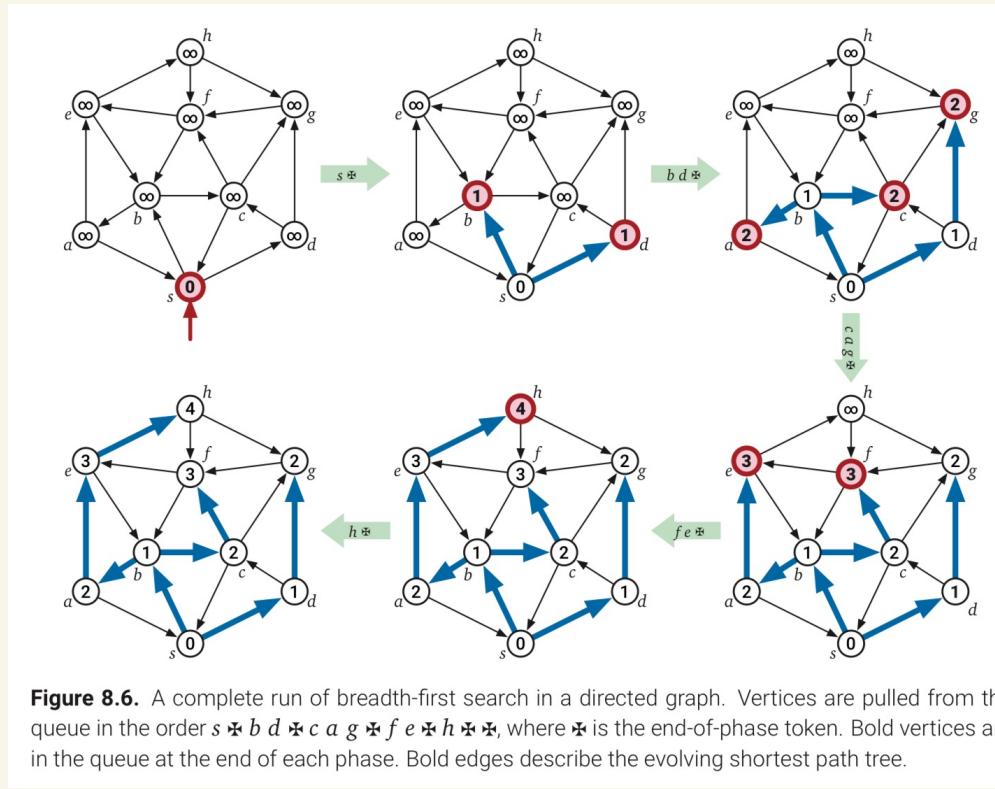


Figure 8.6. A complete run of breadth-first search in a directed graph. Vertices are pulled from the queue in the order $s \star b d \star c a g \star f e \star h \star \star$, where \star is the end-of-phase token. Bold vertices are in the queue at the end of each phase. Bold edges describe the evolving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

queue is
S

U =

```
BFSWITHTOKEN( $s$ ):
  INITSSSP( $s$ )
  PUSH( $s$ )
  PUSH( $\star$ )           «start the first phase»
  while the queue contains at least one vertex
     $u \leftarrow \text{PULL}()$ 
    if  $u = \star$ 
      PUSH( $\star$ )       «start the next phase»
    else
      for all edges  $u \rightarrow v$ 
        if  $\text{dist}(v) > \text{dist}(u) + 1$     «if  $u \rightarrow v$  is tense»
           $\text{dist}(v) \leftarrow \text{dist}(u) + 1$ 
           $\text{pred}(v) \leftarrow u$                 «relax  $u \rightarrow v$ »
          PUSH( $v$ )
```

Lemma

At the end of the i^{th} phase (when \mathbb{H} comes off the queue), for every vertex v ,

either

• $d(v) = \infty$
(not found yet)

or

• $d(v) \leq i$

(and v is only in queue
 $\Leftrightarrow d(v) = i$).

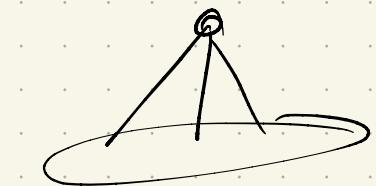
Proof : induction on phase

Base case:

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

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

BFS tree



$Q = v_k \dots v_e \text{ } \check{v}_e$
 $\text{dist} = i-1$

What now?

2nd 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!

○ ○ ○ ○ ⋯ ○ ○ ⋯ ○

edges

So, use it!

DAGSSP(s):

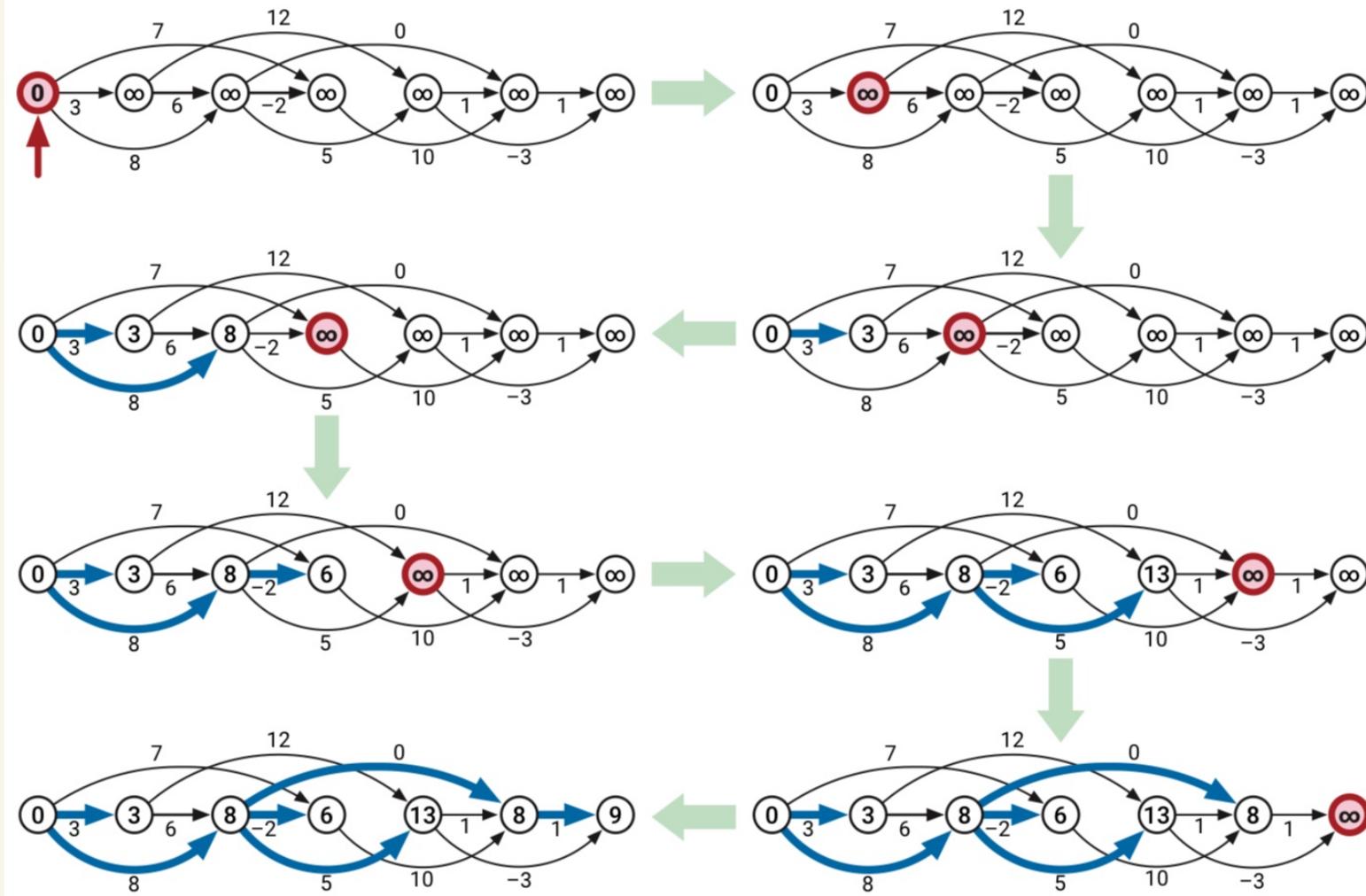
INITSSSP(s)

for all vertices v in topological order

for all edges $u \rightarrow v$

if $u \rightarrow v$ is tense

RELAX($u \rightarrow v$)



Dijkstra (59)

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

Make the bag a priority queue:

Keep "explored" part of the graph, S

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

(all others NULL + ∞)

While $S \neq V$:

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

$$\min_{e=(u,v), u \in S} \text{dist}(u) + w(u \rightarrow v)$$

Add v to S , set $\text{dist}(v) \leftarrow \text{pred}(v)$

Let's formalize this a bit...

Correctness

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

For each $u \in S$, the distance $\text{dist}(u)$ is the shortest path distance
(so $\text{pred}(u)$ traces a shortest path).

Pf: Induction on $|S|$:

Base Case: $|S|=1$

IH: Sups 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 $\text{dist}(v) + w(v \rightarrow w)$
- or update w 's key, if $\text{dist}(v) + w(v \rightarrow w)$ beats current one

NONNEGATIVEDIJKSTRA(s):

```
INITSSSP( $s$ )
for all vertices  $v$ 
    INSERT( $v, \text{dist}(v)$ )
```

while the priority queue is not empty

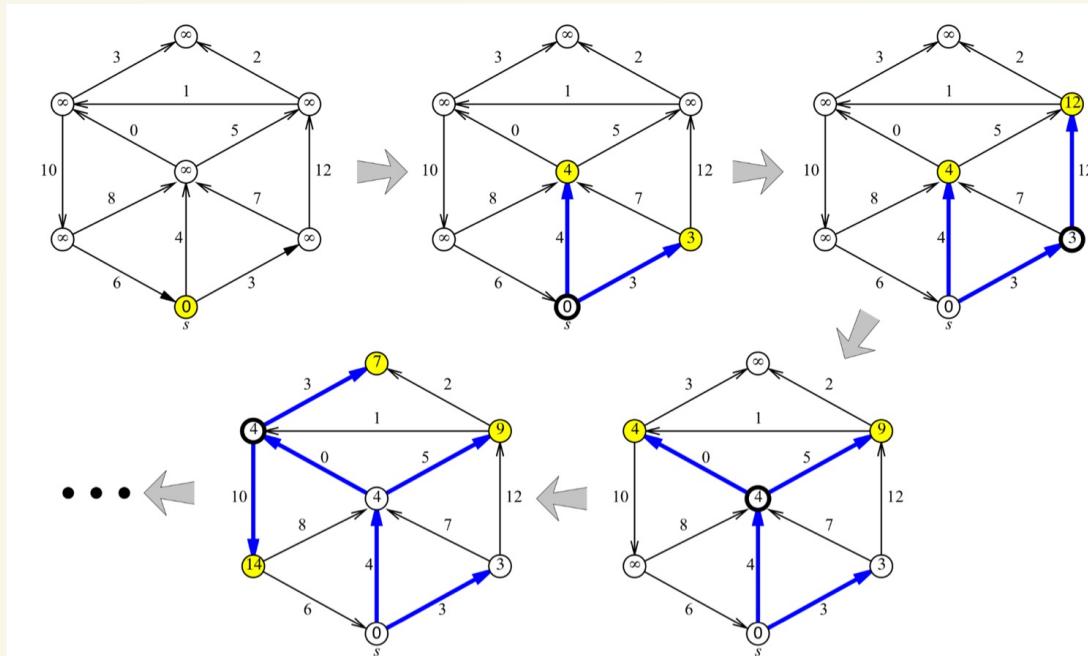
$u \leftarrow \text{EXTRACTMIN}()$

for all edges $u \rightarrow v$

if $u \rightarrow v$ is tense

RELAX($u \rightarrow v$)

DECREASEKEY($v, \text{dist}(v)$)



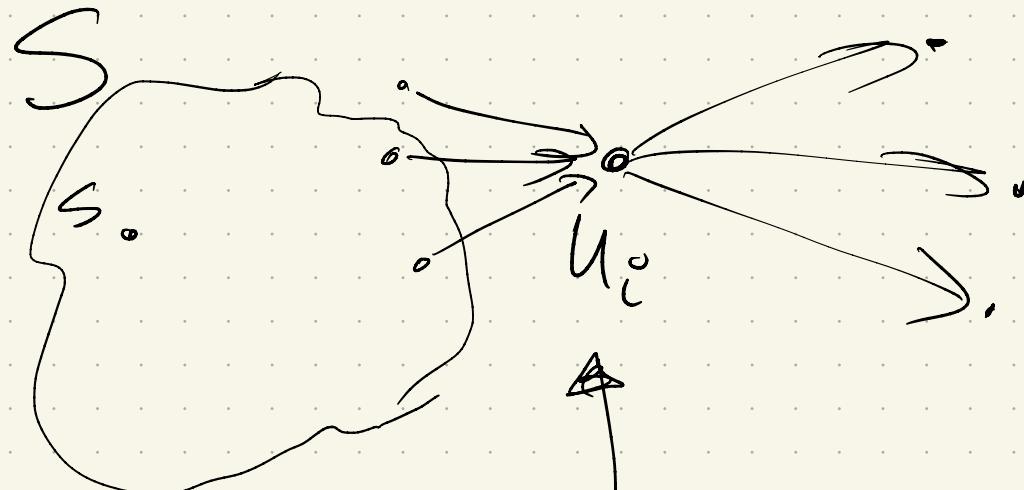
Four phases of Dijkstra's algorithm run on a graph with no negative edges.
At each phase, the shaded vertices are in the heap, and the bold vertex has just been scanned.
The bold edges describe the evolving shortest path tree.

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

Lemmas: If G has no negative edges,
then for all $i < j$, $d_i^o \leq d_j$.

Proof

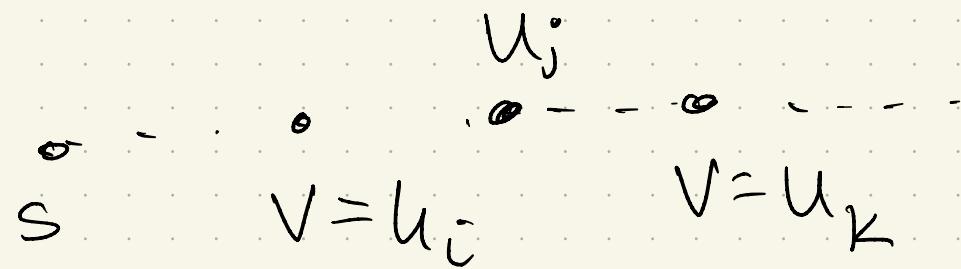
Fix an i :



current best in heap

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

Proof: Sups not:



prev lemma \Rightarrow know $d_i \leq d_k$

But: v was recadded to queue

means some edge $u_j \rightarrow v$
became tense.

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

Why?

decreasekey:

Insert:

Extract Min:

Main downside:

```
NONNEGATIVEDIJKSTRA( $s$ ):  
    INITSSSP( $s$ )  
    for all vertices  $v$   
        INSERT( $v, dist(v)$ )  
    while the priority queue is not empty  
         $u \leftarrow EXTRACTMIN()$   
        for all edges  $u \rightarrow v$   
            if  $u \rightarrow v$  is tense  
                RELAX( $u \rightarrow v$ )  
                DECREASEKEY( $v, dist(v)$ )
```

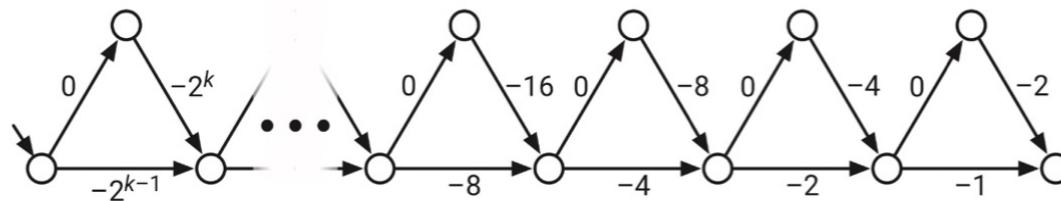


Figure 8.14. A directed graph with negative edges that forces DIJKSTRA to run in exponential time.

Next Monday:

How to deal with negative edges!

Note:

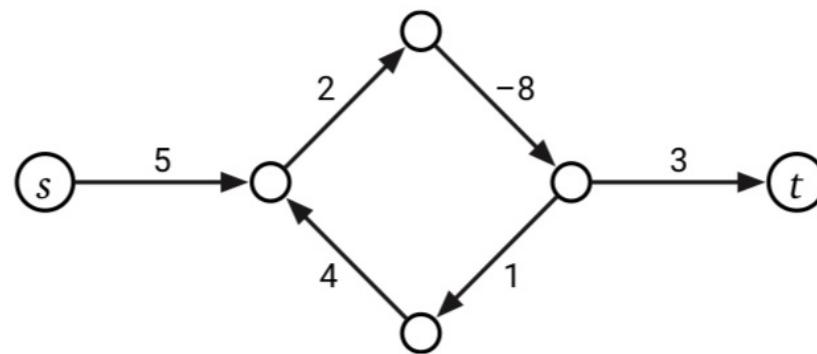


Figure 8.3. There is no shortest walk from s to t .