Algorithms - Spring '25

MSSPs

Recap

- HW posted, due next Wed
  - ↳ MST & Shortest Paths
- Next week's readings are up
- Algorithm's visualizer: demo

Shortest paths so far

Key:

Time to compute shortest path tree rooted at any $s \in V$.

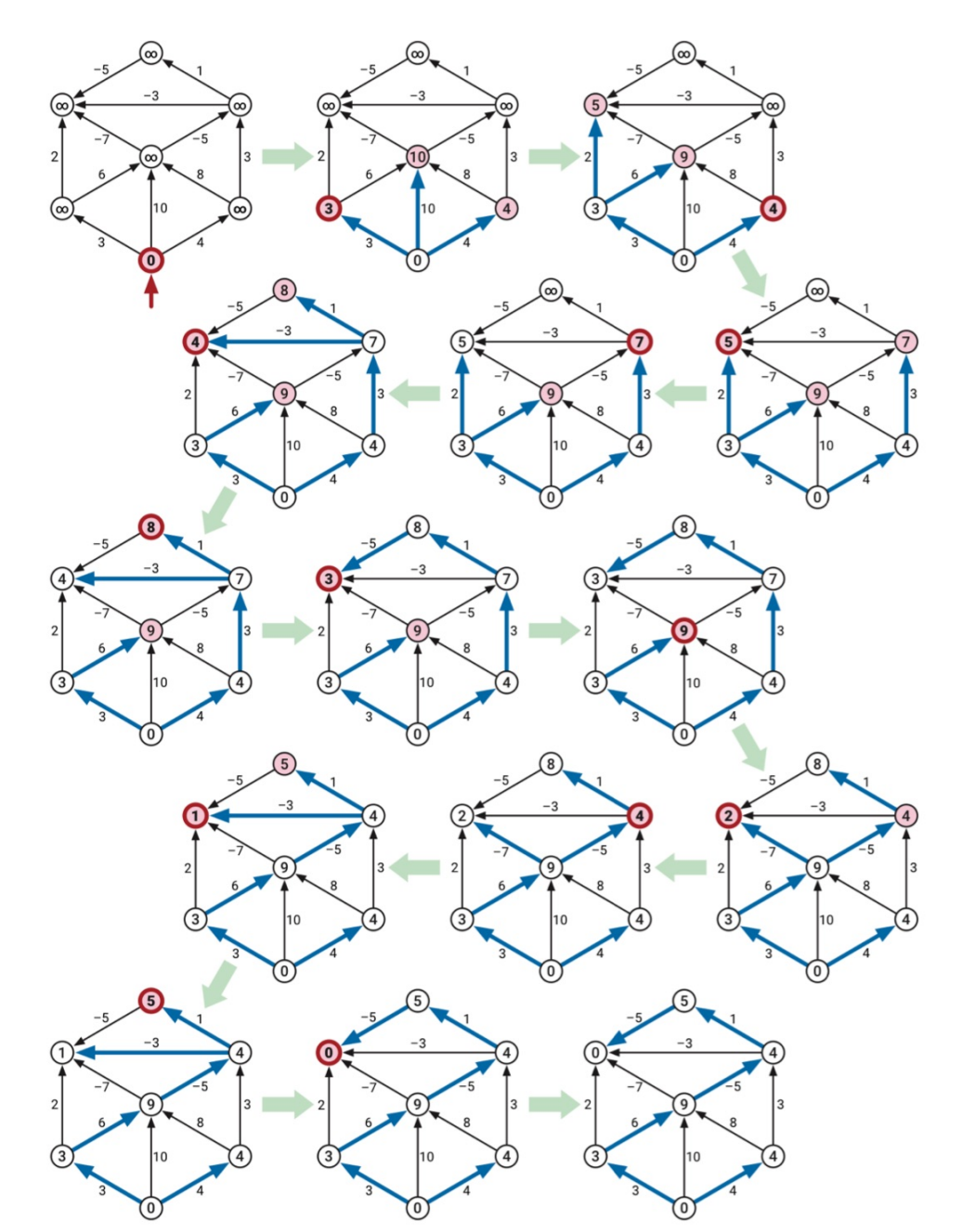
- If your graph is unweighted:
- If your grph is a DAG:
- If no negative edges!

Dijkstra: Saw 2 versions.

One of them works on negative edges (no cycles):

```
DIJKSTRA(s):
  INITSSSP(s)
  INSERT(s, 0)
  while the priority queue is not empty
      u ← EXTRACTMIN( )
      for all edges u→v
          if u→v is tense
              RELAX(u→v)
              if v is in the priority queue
                  DECREASEKEY(v, dist(v))
              else
                  INSERT(v, dist(v))
```

Figure 8.11. Dijkstra's algorithm.



If negative edges and (potentially) negative cycles:

Bellman-Ford

```
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!"
```

– or –

```
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)
```

Runtime:

Multiple Source Shortest paths:

First attempt: [for each vertex v
Run SSSP(v)

Runtime : $O(V \times (SSSP_{alg}))$

- if unweighted or a DAG, SSSP was $O(V+E) \Rightarrow$
- no negative edge weights, Dijkstra was $O(E \log V) \Rightarrow$
- Bellman-Ford was $O(VE)$

$\Rightarrow$

Side question:

Since negative edges are bad, why can't we just re-weight?

Idea: increase all edge weights by some amount, so all positive.

Doesn't work.

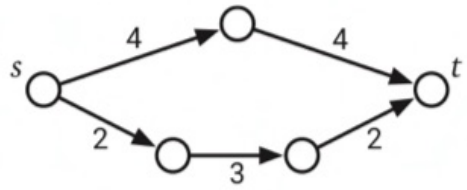


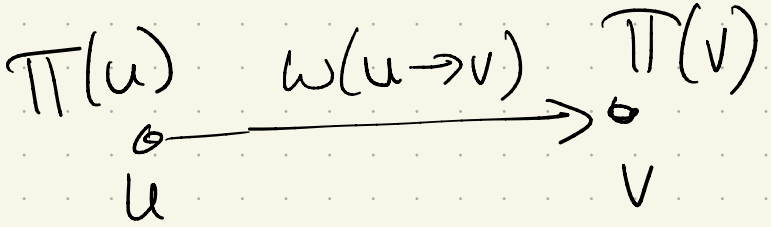
Figure 9.1. Increasing all the edge weights by 2 changes the shortest path from s to t.

Why?

Another idea (that works):

Suppose each v has a price attached, $\pi(v)$.

(Still have edge weights.)



Define a new function w':

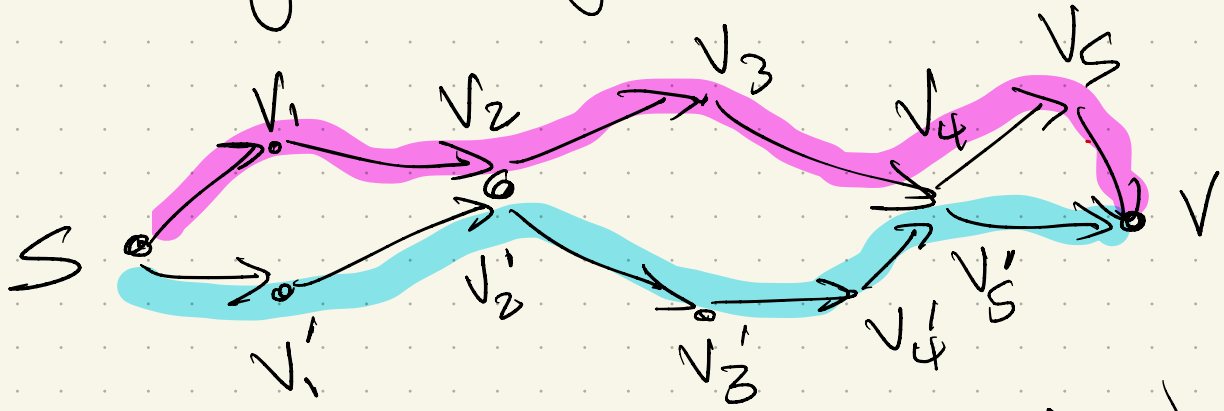
$$w'(u \to v) = \pi(u) + w(u \to v) - \pi(v)$$

Claim:

Claim: under w', shortest paths are the same as under w.

Why? Consider 2 $s \rightsquigarrow v$ paths:

(in original weights)



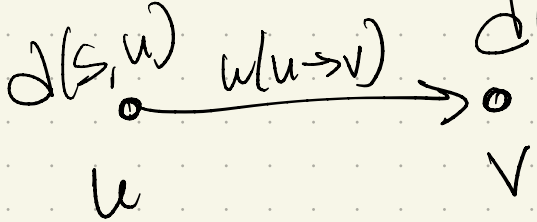
weights: $\pi(v_i)$

purple path after reweighting:

= blue path:

Johnson's algorithm for MSSP: use this new kind of weight function!

- Run Bellman-Ford once from some $s \in V$
  - ↳ no negative cycles or give up
- Reweight (if it works): let $\pi(v) = d(s,v)$

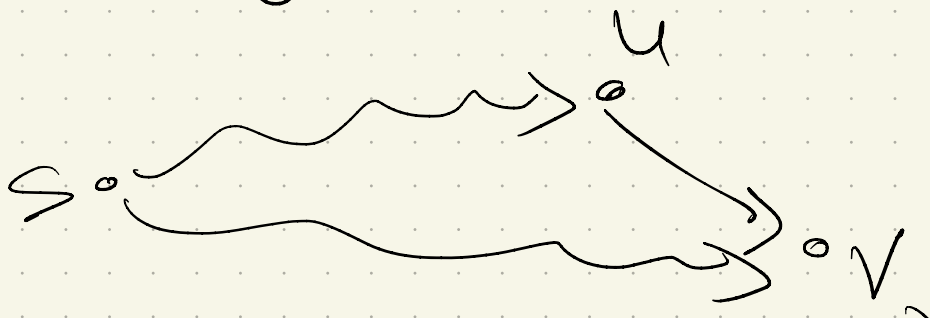


$w'(u \to v) =$

- Now, all paths are positive, so can use faster algorithm for other SSSPs!

Aside: Why positive?

After SSSP, no edges are tense:



so $d(s,u) + w(u \to v) \geq d(s,v)$

rearrange:

So - Algorithm:

Runtime:

```
JOHNSONAPSP(V, E, w) :
  ⟨⟨Add an artificial source⟩⟩
  add a new vertex s
  for every vertex v
      add a new edge s→v
      w(s→v) ← 0
  ⟨⟨Compute vertex prices⟩⟩
  dist[s, ·] ← BELLMANFORD(V, E, w, s)
  if BELLMANFORD found a negative cycle
      fail gracefully
  ⟨⟨Reweight the edges⟩⟩
  for every edge u→v ∈ E
      w'(u→v) ← dist[s, u] + w(u→v) − dist[s, v]
  ⟨⟨Compute reweighted shortest path distances⟩⟩
  for every vertex u
      dist'[u, ·] ← DIJKSTRA(V, E, w', u)
  ⟨⟨Compute original shortest-path distances⟩⟩
  for every vertex u
      for every vertex v
          dist[u, v] ← dist'[u, v] − dist[s, u] + dist[s, v]
```

Figure 9.2. Johnson's all-pairs shortest paths algorithm

(Compare to naive: $O(V^2E) = O(V^4)$

More Improvements

Set up recursive approach:

Let $dist(u, v, \ell)$ = best $u \rightsquigarrow v$ path using at most ℓ edges.

Then, back to "keep relaxing edges" approach:

$$dist(u, v, \ell) = \begin{cases} 0 & \text{if } \ell = 0 \text{ and } u = v \\ \infty & \text{if } \ell = 0 \text{ and } u \neq v \\ \min \left\{ \begin{array}{c} dist(u, v, \ell - 1) \\ \min\limits_{x \to v} \left(dist(u, x, \ell - 1) + w(x \to v) \right) \end{array} \right\} & \text{otherwise} \end{cases}$$

Code:

```
SHIMBELAPSP(V, E, w):
  for all vertices u
      for all vertices v
          if u = v
              dist[u, v, 0] ← 0
          else
              dist[u, v, 0] ← ∞

  for ℓ ← 1 to V − 1
      for all vertices u
          for all vertices v ≠ u
              dist[u, v, ℓ] ← dist[u, v, ℓ − 1]
              for all edges x→v
                  if dist[u, v, ℓ] > dist[u, x, ℓ − 1] + w(x→v)
                      dist[u, v, ℓ] ← dist[u, x, ℓ − 1] + w(x→v)
```

} paths of length 0

Can improve

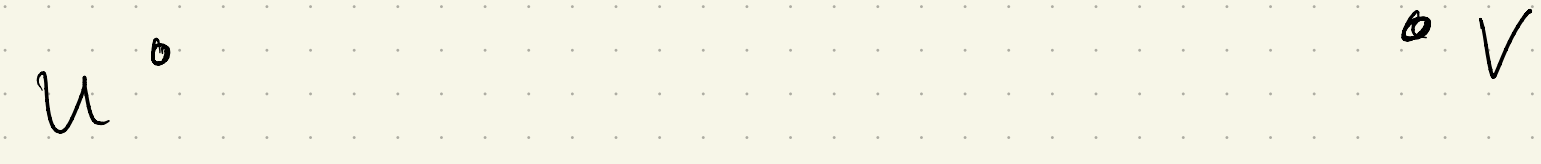
```
ALLPAIRSBELLMANFORD(V, E, w):
  for all vertices u
      for all vertices v
          if u = v
              dist[u, v] ← 0
          else
              dist[u, v] ← ∞

  for ℓ ← 1 to V − 1
      for all vertices u
          for all edges x→v
              if dist[u, v] > dist[u, x] + w(x→v)
                  dist[u, v] ← dist[u, x] + w(x→v)
```

Runtime:

Wait → that's worse!

But, let's try a more balanced divide & conquer!

Instead of going $n-1 \rightarrow n$, can we add longer paths?



$$dist(u, v, \ell) = \begin{cases} w(u \rightarrow v) & \text{if } i = 1 \\ \min_x \left(dist(u, x, \ell/2) + dist(x, v, \ell/2) \right) & \text{otherwise} \end{cases}$$

Code:

```
FischerMeyerAPSP(V, E, w):
  for all vertices u
      for all vertices v
          dist[u, v, 0] ← w(u→v)

  for i ← 1 to ⌈lg V⌉                ⟨⟨ℓ = 2^i⟩⟩
      for all vertices u
          for all vertices v
              dist[u, v, i] ← ∞
              for all vertices x
                  if dist[u, v, i] > dist[u, x, i − 1] + dist[x, v, i − 1]
                      dist[u, v, i] ← dist[u, x, i − 1] + dist[x, v, i − 1]
```

→ Can also save space

```
LeyzorekAPSP(V, E, w):
  for all vertices u
      for all vertices v
          dist[u, v] ← w(u→v)

  for i ← 1 to ⌈lg V⌉                ⟨⟨ℓ = 2^i⟩⟩
      for all vertices u
          for all vertices v
              for all vertices x
                  if dist[u, v] > dist[u, x] + dist[x, v]
                      dist[u, v] ← dist[u, x] + dist[x, v]
```

Can we get better?

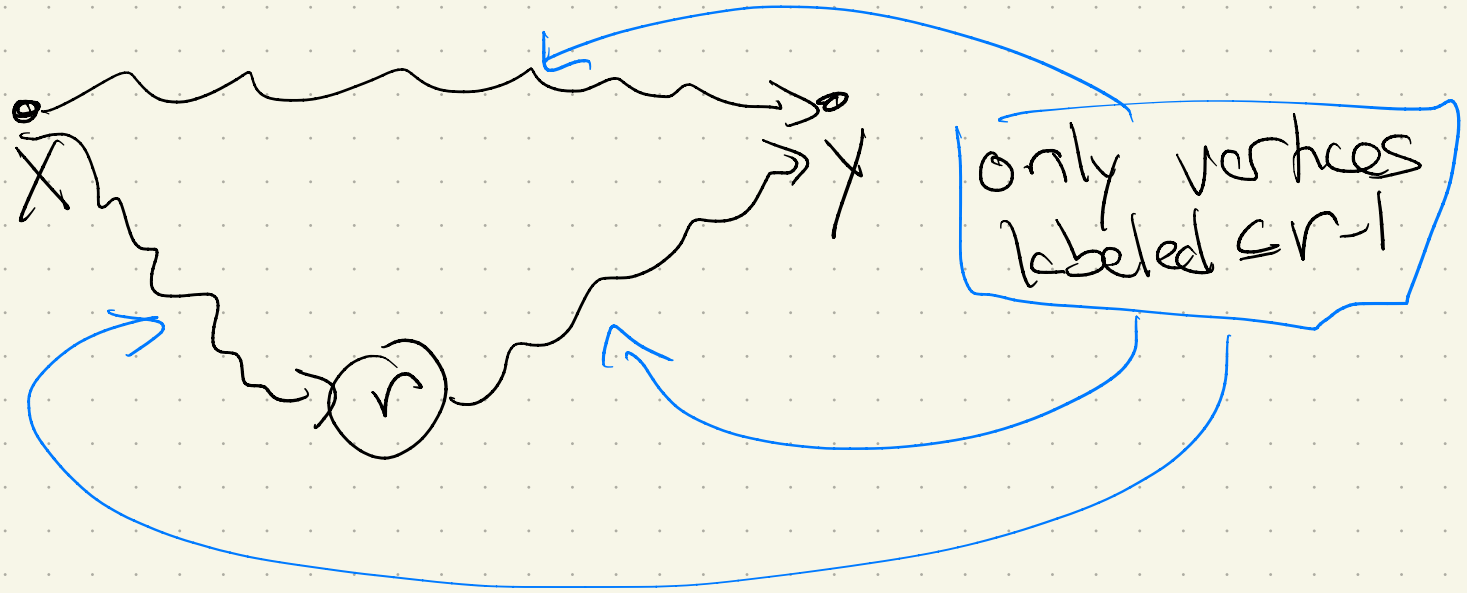
Floyd-Warshall:

order vertices $1, \ldots, V$

Let $d(x, y, r) =$

best length path via vertices labeled $1 \ldots r$

Then:



Recursion:

$$dist(u,v,r) = \begin{cases} w(u \to v) & \text{if } r = 0 \\ \min \begin{cases} dist(u,v,r-1) \\ dist(u,r,r-1) + dist(r,v,r-1) \end{cases} & \text{otherwise} \end{cases}$$

don't use vertex r

use vertex r

code ⤳

```
KleeneAPSP(V, E, w):
  for all vertices u
      for all vertices v
          dist[u, v, 0] ← w(u→v)

  for r ← 1 to V
      for all vertices u
          for all vertices v
              if dist[u, v, r − 1] < dist[u, r, r − 1] + dist[r, v, r − 1]
                  dist[u, v, r] ← dist[u, v, r − 1]
              else
                  dist[u, v, r] ← dist[u, r, r − 1] + dist[r, v, r − 1]
```

Runtime?

save space

```
FloydWarshall(V, E, w):
  for all vertices u
      for all vertices v
          dist[u, v] ← w(u→v)

  for all vertices r
      for all vertices u
          for all vertices v
              if dist[u, v] > dist[u, r] + dist[r, v]
                  dist[u, v] ← dist[u, r] + dist[r, v]
```

Next topic: Flows & cuts