# 6.006 Lecture 18: Single-Source Single-Target & All Pairs

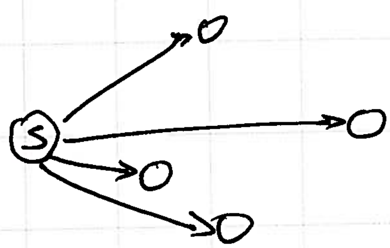
- ☐ SS-ST: Speeding up Dijkstra!? Wagner & Willhalm paper
- ☐ Preview (only) of All-Pairs Shortest Paths Ch. 25

## Speeding up Dijksta!?

- Binary heaps ⇒ Fibonacci Heaps helps

  we perform |V| Extract-Min's and up to |E| Decrease-Key's, but we do not care how much each takes in the worst case; we only care about the total (for a worst-case bound on Dijkstra). ~~Fibonna~~ Fibonacci heaps perform |E| Decrease-Key operations in $O(|E|)$ time, so total for Dijkstra is $O(V \lg V + E)$.

• I would be hard to do better, because we need to consider all the edges, and we extract vertices in sorted order (distance from s), so we essentially sort




If we input a graph like this to Dijkstra, it will process the edges/vertices in sorted order.

• Still, people invested effirt in speed-up techniques that do not change the worst-case asymptotic running time, but the constants and/or the average or "common" case

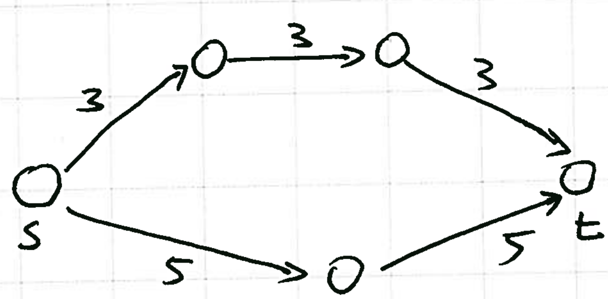
→ How much more would you pay for a computer that was "only" ×2 faster? probably somthing

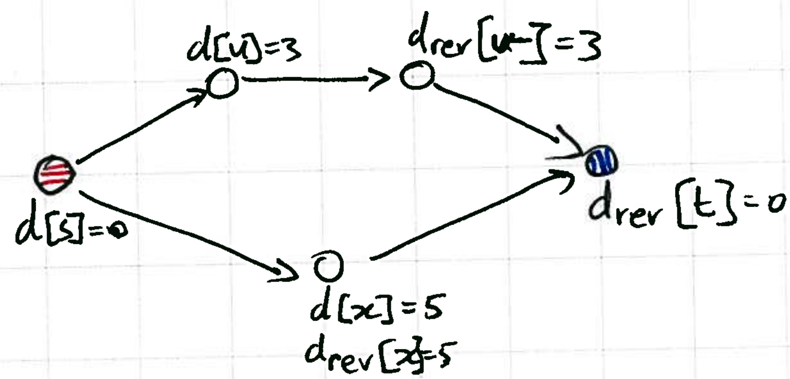
• Asymptotic worst-case improvements are the most valuable, but there is some value in weaker results.

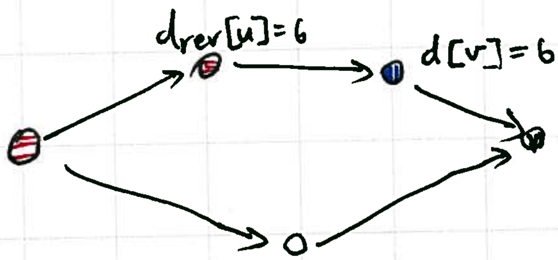
# Single sourse single target using bidi search

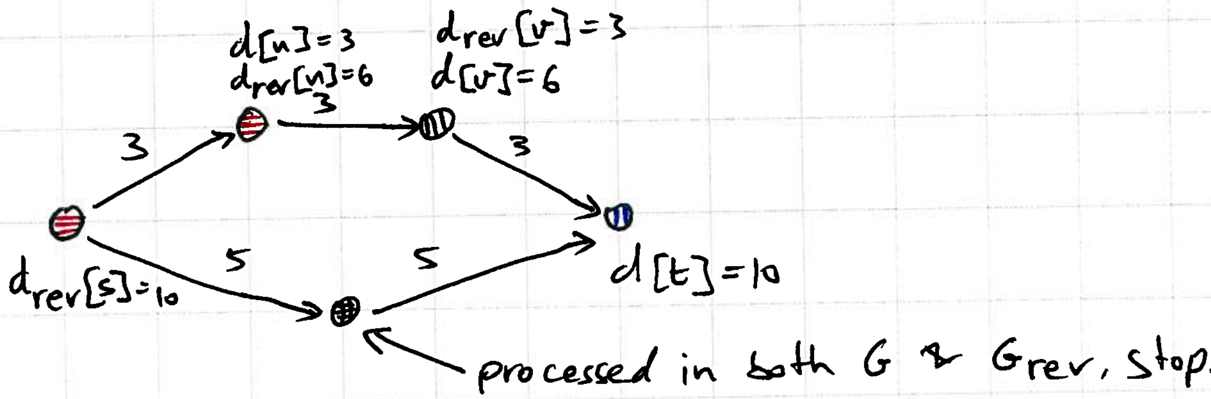
Run two copies of Dijksta simulteneously, alternating steps, one ~~for~~ from s on G, and one from the target t on $G_{rev} = (V, \{(v,u) : (u,v) \in E\})$.

stop when some vertex is extracted from both heaps.



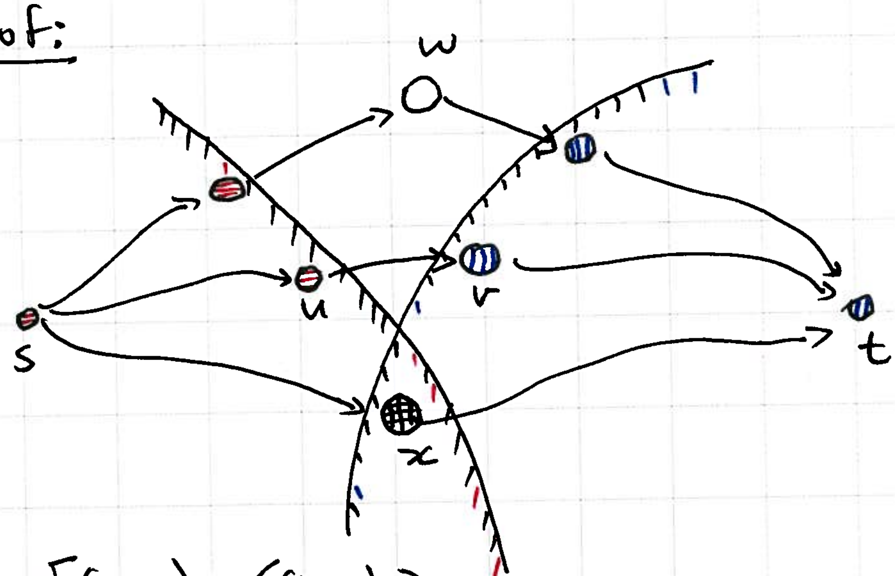






$d[t]$ is <u>not</u> the shortest path!

But $\min_w \{d[w] + d_{rev}[w]\}$ is.

Proof:




$$\left.\begin{array}{l}\delta(s,w) \geq \delta(s,x) \\ \delta(w,t) \geq \delta(x,t)\end{array}\right\} \Rightarrow s \leadsto w \leadsto t \text{ longer than } s \leadsto x \leadsto t$$

But we do have the correct weights of

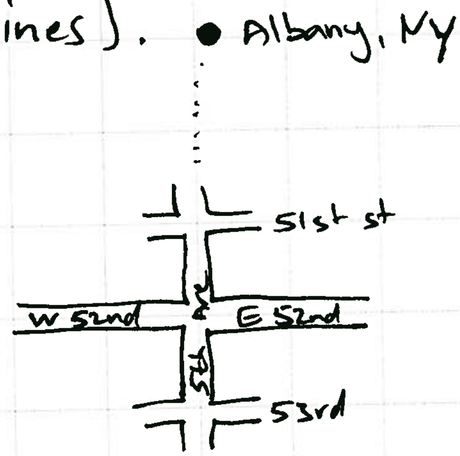
$s \leadsto u \to v$ and $v \leadsto t$

$s \leadsto u$ and $u \to v \leadsto t$

- Simple idea but a tricky detail; beware.

## Single-Source Single Target using Potentials

Suppose the vertices represents points in the plane (e.g., on a map) and $w(u,v)$ is the Euclidean distance from u to v. (Edges are straight lines).

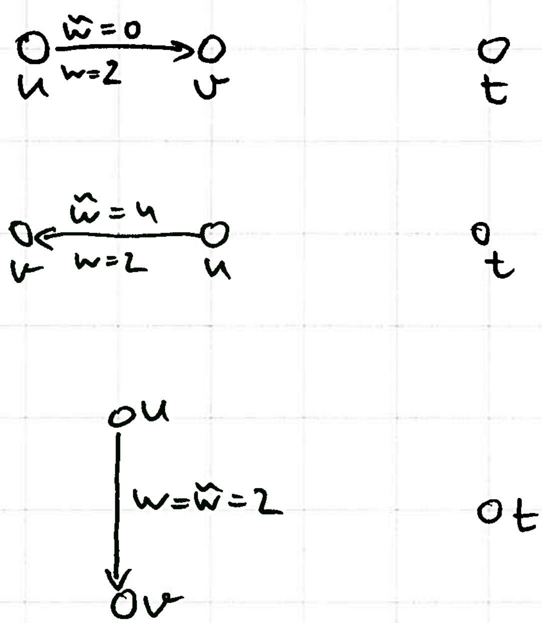



E.g., you start walking from 5th Av & 52nd st in NY to Albany, up north. Should the next vertex be $51^{st}$ & 5th or $53^{rd}$ & 5th (its a tie on the map).

We'll show you can do the logical thing and not increase the asymptotic running time (maybe reduce it a bit)

Let $p(w) = \|w - t\|$ (Euclidean distance from $w$ to $t$).

Define $\hat{w}(u,v) = w(u,v) - p(u) + p(v)$




$\hat{w} \geq 0$, can run Dijkstra

$$\hat{w}(\text{path}) = \hat{w}(\langle \overset{s}{v_1}, v_2 \dots v_{k-1}, \overset{t}{v_k} \rangle)$$

$$= \sum_{i=2}^{k} \hat{w}(v_{i-1}, v_i)$$

$$= \sum_{i=2}^{k} w(v_{i-1}, v_i) - p(v_{i-1}) + p(v_i)$$

$$= -p(s) + p(t) + \sum_{i=2}^{k} w(v_{i-1}, v_i)$$

$$= w(\text{path}) - p(s)$$

relative rank of paths is preserved

# All Pairs Shortest Paths (Preview)

Clearly more expensive (at least not cheaper) but there is so much to compute that algorithms become simpler.

Floyd-Warshall: $O(V^3)$

Set up an $n \times n$ matrix $D$

Initialize $D_{ij} = w(i,j)$ lengh of SP that do not go through any intermediate vertex

Shorten paths by going through vertex 1, if possibe


Repeat for vertices 2 through n.

Johnson: $O(V^2 \lg V + EV)$ same technique again

Run Dijkstra from ~~ver~~ every vertex after finding a potential p that makes edges non-negative (using Bellman-Ford) using building blocks