Recap:

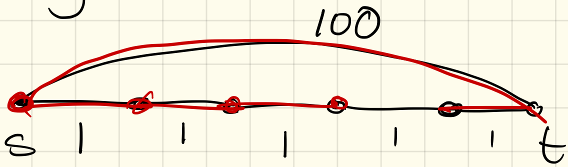
- HW due
  - ↳ extended until Friday
- Lab tomorrow
  - ↳ set to be due next Friday
- Last HW, likely on zybooks due next Saturday (no extentions please!)
- Review last Monday of class
- Let me know ASAP about conflicts (final, Wed at 2pm)

Last time:

In some sense, BFS trees are "short".

But – what if graph is weighted?

Ex:



BFS tree:

ignores weights entirely

Need to examine how to get minimum things when graph is weighted.

Today:

Weighted graph: a graph $G = (V, E)$ plus a function $w : E \to \mathbb{R}$

The length of a path $=$ sum of $w(e)$ for all e on the path

distance $= d(u, v)$

$=$ min weight path from u to v

Shortest path trees:

First, note that set of shortest paths form a tree.

Why?

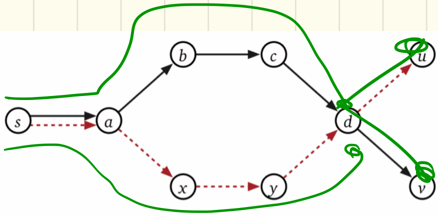


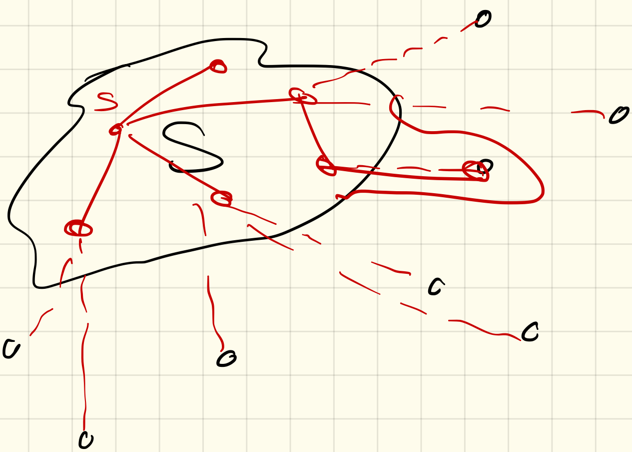
Figure 8.1. If $s \to a \to b \to c \to d \to v$ (solid) and $s \to a \to x \to y \to d \to u$ (dashed) are shortest paths, then $s \to a \to b \to c \to d \to u$ (along the top) is also a shortest path.

$s \to d$ paths

So, can find a set of shortest paths from s to all other vertices that will be a tree.

Algorithm:

- Keep a set of vertices whose shortest path from s is unknown
- At each step in loop, add 1 more vertex



Key: Can always guarantee min possibility will be shortest.

Pseudocode:

- Saw in Zybook.
- Implementation: depends on graph representation!
- Key data structure: heap containing all current distances (initially all ∞)
- Then pick min value from heap
  - this is guaranteed shortest
- Upate all distances + repeat

modifying heap

Picture:

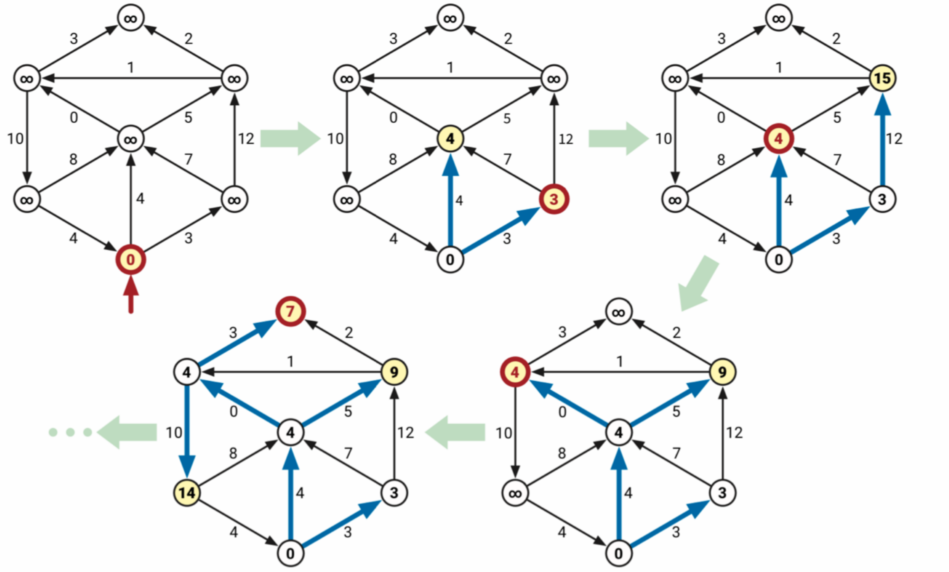


Figure 8.12. The first four iterations of Dijkstra's algorithm on a graph with no negative edges. In each iteration, bold edges indicate predecessors; shaded vertices are in the priority queue; and the bold vertex is about to be scanned. The remaining iterations do not change the distances or the shortest-path tree.

Details:

Need no negative cycles.

Why?

What is shortest s-t route?

There isn't one!

Need "sensible" weights.

Runtime:

$O(\log n)$

- Get min from heap

Then "relax" adjacent edges $d(v)$

Need to update in heap $O(\log n)$ per neighbor

- repeats $\leq$ once for each vertex

Total: $\leq \sum_{v \in V} [\log n + d(v) \cdot \log n]$

$$= \log n \left[\sum_{v \in V} (1 + d(v)) \right] = O(\log n (n+m))$$

$$= O((n+m) \log n)$$