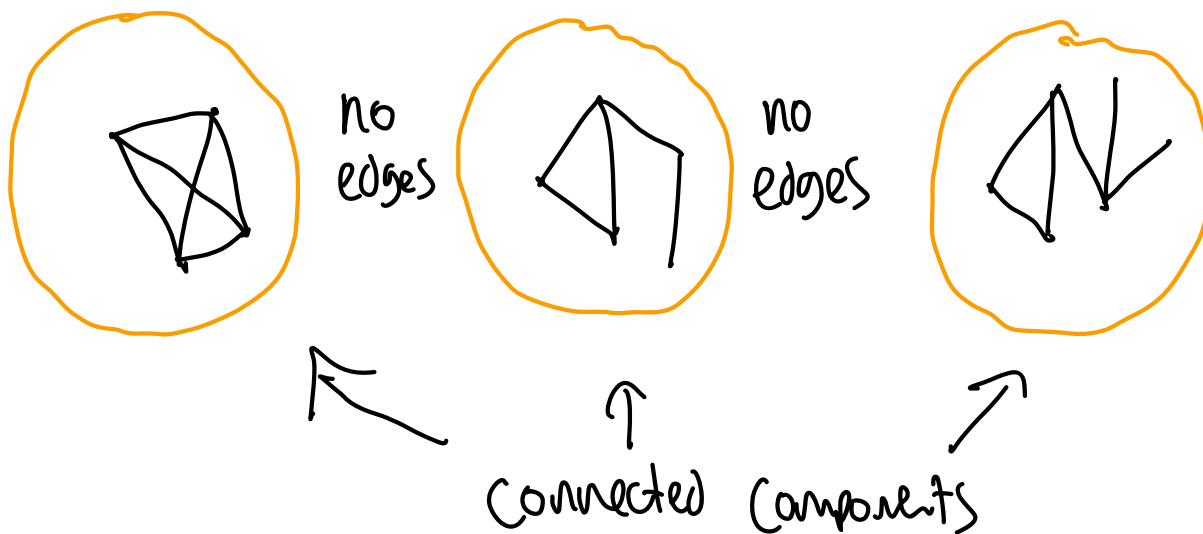


CS 331, Fall 2026
Lecture 12 (10/7)

Today: - SCCs
- Dijkstra
- Bellman
- Ford

SCCs (Part V, Section 2.2)

Recall Connectivity Structure in undirected:



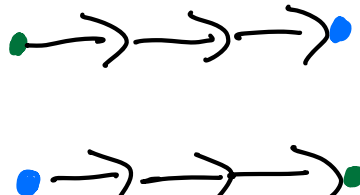
Key fact: Connectivity is equivalence relation

Not so for directed!



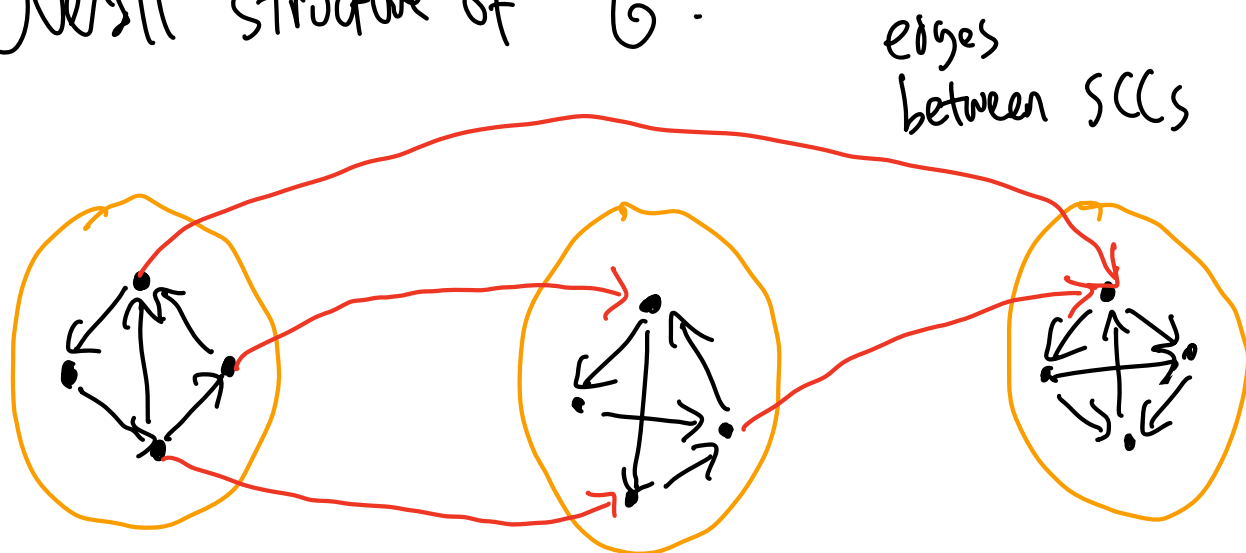
directed path not symmetric

Fix: Strong connectivity

We say $\bullet \sim \bullet$ if $\exists$ 

Equivalence relation, partitions G into SCCs

Overall structure of G :



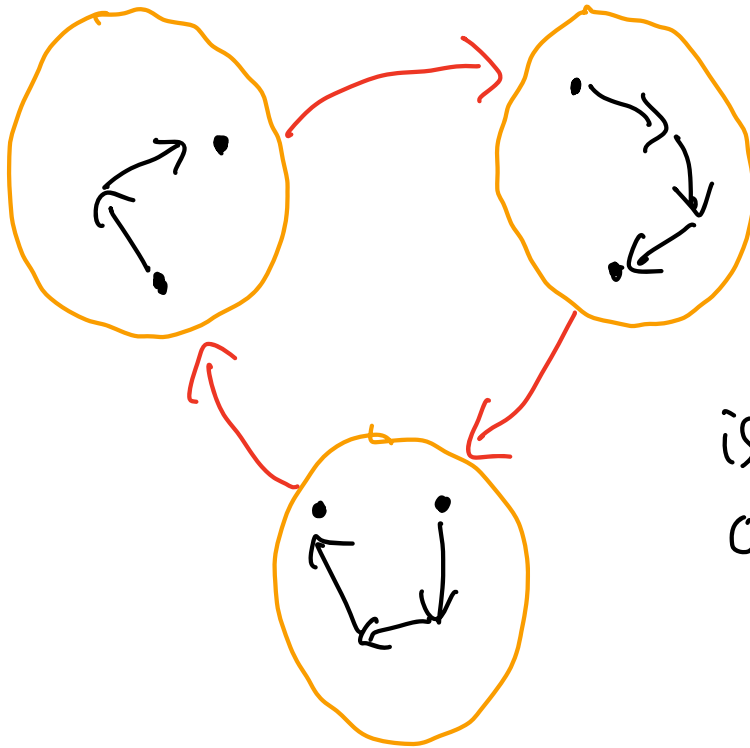
$\text{SCC}(G)$: Vertices = $1, 2, \dots, k$ # SCCs
edge = (i, j) iff $\exists$ some $\text{SCC}_i \rightarrow \text{SCC}_j$

e.g.



Claim: $\text{SCC}(G)$ is a DAG

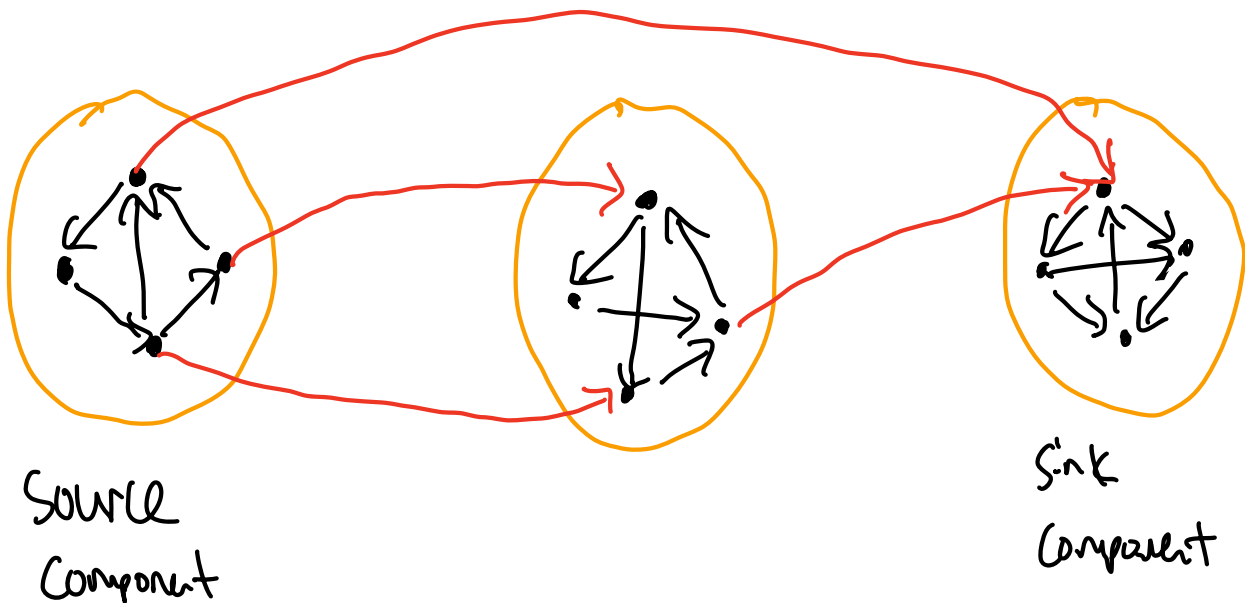
Proof:



is actually
one large SCC

How to compute SCCs?

Idea: every DAG has a source and sink
no inc. edges no out edges



Algo: Repeat following.

1) Find vertex s in sink component

2) Run DFS(G, s) (only discover sink)

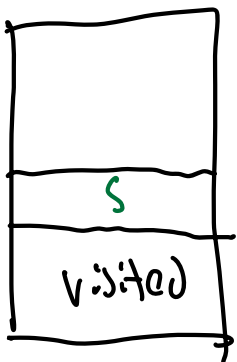
total runtime: $O(n) + \sum O(m_{cs}) = O(m+n)$.

How about 1)? Claim: Can implement in $O(m+n)$.

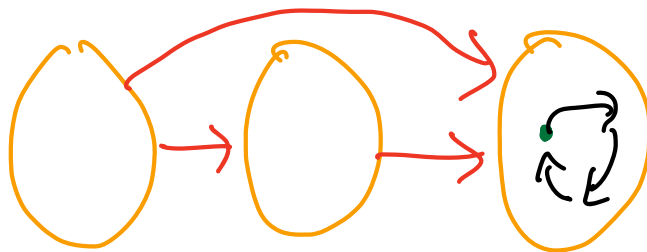
Proof sketch: In postordering of all vertices,
last vertex is in source component.

We want sink component!

Fix: postorder $rev(G)$ = reverse all edges.



postorder $rev(G)$



Dijkstra (Part V, Section 3.1)

UT Austin
1984-1999

Rest of today: SSSP.

Input: $G = (V, E, \underbrace{W}_{\text{edge weights}})$, $\underbrace{s \in V}_{\text{source}}$

Goal: Compute all shortest path distances:

$$d(s, v) = \min_{\substack{p \subseteq E \\ p \text{ is } s \rightarrow v \text{ path}}} \sum_{e \in p} w_e$$

We've already seen two variants.

1) DAGs (Part III)

2) Unweighted (Last class)

Today: 3) Positive edge weights 4) Any weights

For now: $w \in \mathbb{R}_{\geq 0}^E$ (positive)

Two key ingredients in Dijkstra's algo:

- relaxing tense edges
- priority queue

Relaxing tense edges

$$\text{let } D[v] \geq d(s, v)$$

overestimate, keep finding shorter paths

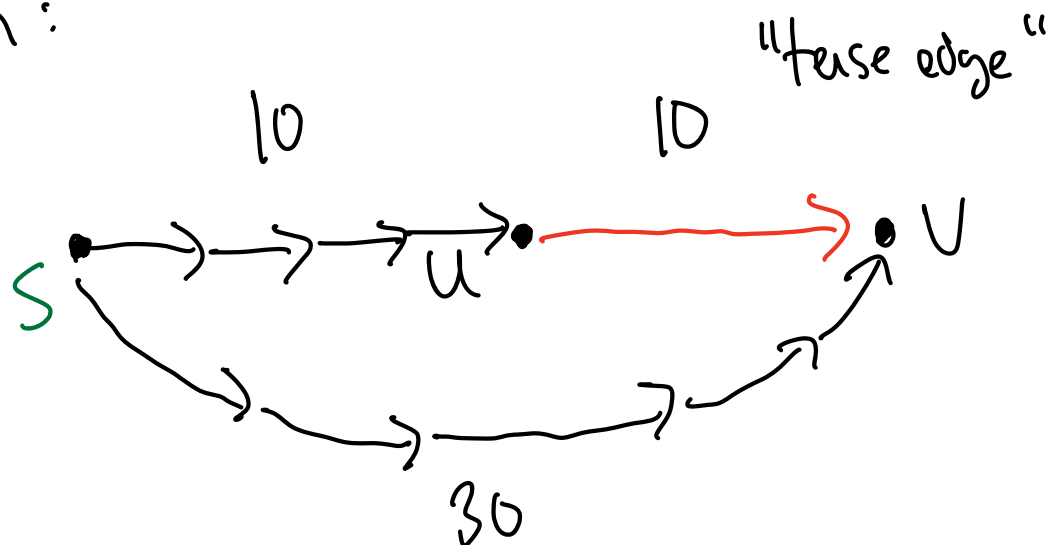
$$\text{Claim: } D[v] \leftarrow \min \left(D[v], D[u] + w_{(u,v)} \right)$$

remains an overestimate.

We call it relaxing the edge (u, v)

Proof: $D(u) + w_{(u,v)} \geq d(s, u) + w_{(u,v)}$
 $\geq d(s, v)$

Intuition:



Priority queue

values	1	3	7	...
objects	x	y	z	

You get to cut in line.

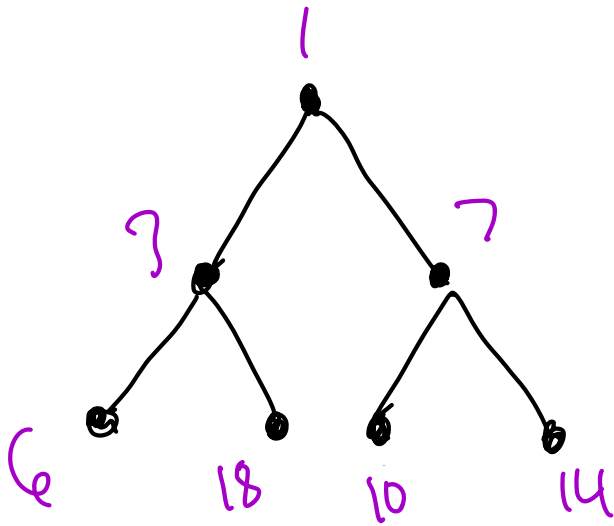
- Insert (x, val)
- Delete (x)
- Extract Min()

take out smallest- val object from set.

We can implement with Heap.

All ops $O(\log(n))$ time, $n = \text{max size}$

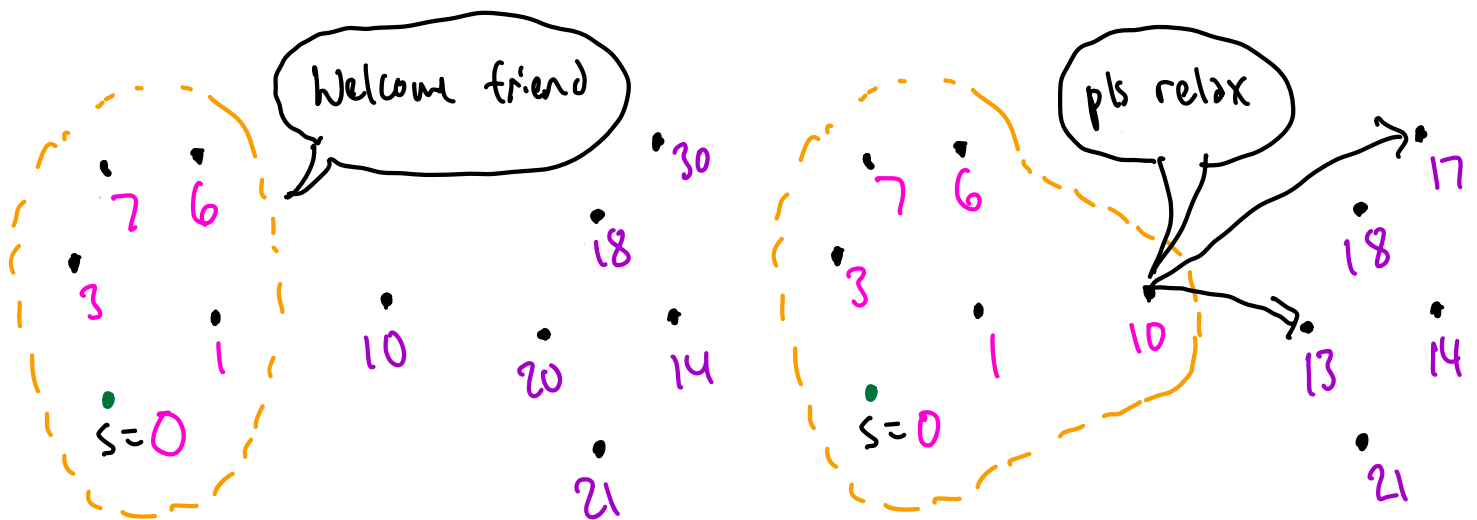
Review: Section 7.2, Part I



Key ideas

- Heap property
- Balance ($O(\log(n))$ depth)

Idea of Dijkstra: graph search w/ priority queue



Step k start

Step k end

Assume all of V reachable from s
(for-free graph primitive: BFS)

SSSP Positive (G, s) :

$H \leftarrow \text{Heap.Init}()$

$H.\text{Insert}(s, 0)$

For $v \in V, v \neq s$: $H.\text{Insert}(v, \infty)$

$d \leftarrow \text{Array.Init}(n)$

While $|H| > 0$:

locking
state $\left\{ \begin{array}{l} v \leftarrow H.\text{ExtractMin}() \\ d[v] \leftarrow v.\text{val} \end{array} \right. \quad = O(n \log n) \text{ time}$

relax
edges $\left\{ \begin{array}{l} \text{For } (v, u) \in E: \\ H.\text{Delete}(u) \\ H.\text{Insert}(u, \min(u.\text{val}, v.\text{val} + w_{(u,v)})) \end{array} \right. \quad = O(m \log n) \text{ time}$

Return d

Runtime: $O((m+n) \log(n))$

Fun facts: Improvable using Fibonacci heap

"Decrease Val" in $O(1)$ time

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

SOTA: $O(m \log \log(n))$ in Word RAM
[Thmp, 2000]

Correctness: Induction on steps

Step 1  correct (positive weights)

Step $k+1$ Every unfrozen label is

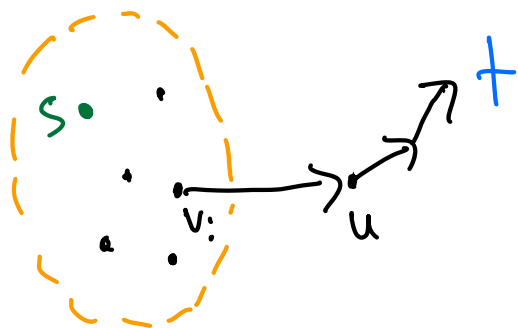
(assume $1, 2, \dots, k$)

$$V.val = \min_{\substack{i \in (k) \\ (v, u) \in E}} d(s, v_i) + w_{(v, u)}$$

all candidates in algo

Let us extract $t = v_{k+1}$

How can another path beat $t.val$?



We have

(Positivity)

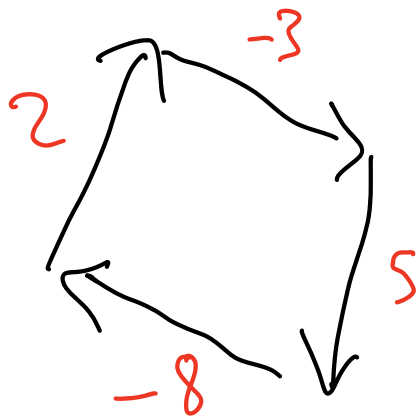
$$\begin{aligned} \text{path length} &\geq d(s, v:) + w_{(v:, u)} + d(u, t) \\ &\geq u.val \geq t.val \end{aligned}$$

There's no better path.

Bellman-Ford (Part V, Section 3.2)

The gloves are off. Arbitrary graph,
arbitrary edge weights

... Remember, no negative-weight cycles.



Shortest path distance
not well-defined.

Wait, didn't we solve APSP in $O(n^3)$???

Yes. The exact same algo gets $O(mn)$ SSSP.

(n^4 version is secretly $mn^2 \Rightarrow mn$)

$S[v][\ell] = \text{Shortest } S \rightarrow v \text{ path, } \leq \ell \text{ edges}$

$$S[v][\ell] = \min \left(S[v][\ell-1], \min_{(u,v) \in E} S[u][\ell-1] + w_{(u,v)} \right)$$

Bellman-Ford: Initialize $D[v] \leftarrow \infty$

$\forall v \in V \setminus \{s\},$

$D[s] \leftarrow 0$

Relax all edges $n-1$ times

Return D

$\underbrace{\hspace{1cm}}$
Upper bound
on path length.
no cycles

Cool consequence: detecting negative-weight cycles

Algo: Run Bellman-Ford

Check if any tense edges

$D[v] > D[u] + w(u,v)$

Yes iff negative-weight cycle.

No NWC $\Rightarrow$ no tense edge

Proof: BF computes $D = d$

If tense edge, can decrease D

(Earlier showed D stays overestimate $\Rightarrow \Leftarrow$)

NWC $\Rightarrow$ tense edge

Proof: let NWC be $(v_1, v_2), \dots, (v_k, v_1)$

Suppose for any D

$$D[v_i] \leq D[v_{i-1}] + W_{(v_{i-1}, v_i)} \quad \forall i \in [k] \\ (k \geq 0)$$

Sum both sides,

$$\sum_{i \in [k]} D[v_i] \leq \sum_{i \in [k]} D[v_{i-1}] + W_{(v_{i-1}, v_i)}$$

cancels $\nearrow$

not NWC!
 $\Rightarrow \Leftarrow$