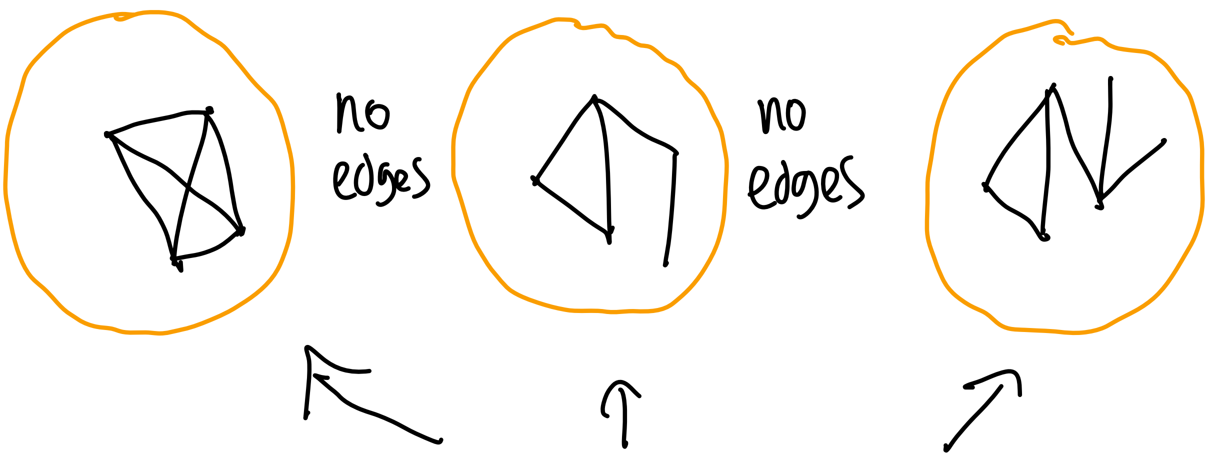
CS 331, Fall 2025
Lecture 12 (10/8)

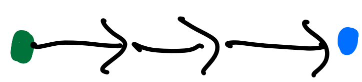
Today:
- SCCs
- Dijkstra
- Bellman-Ford

SCCs (Part V, Section 2.2)

Recall connectivity structure in undirected:



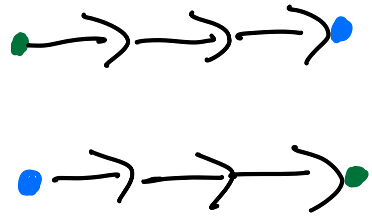
Key fact: Connectivity is <u>equivalence relation</u>

Not so for directed! 

directed path not symmetric

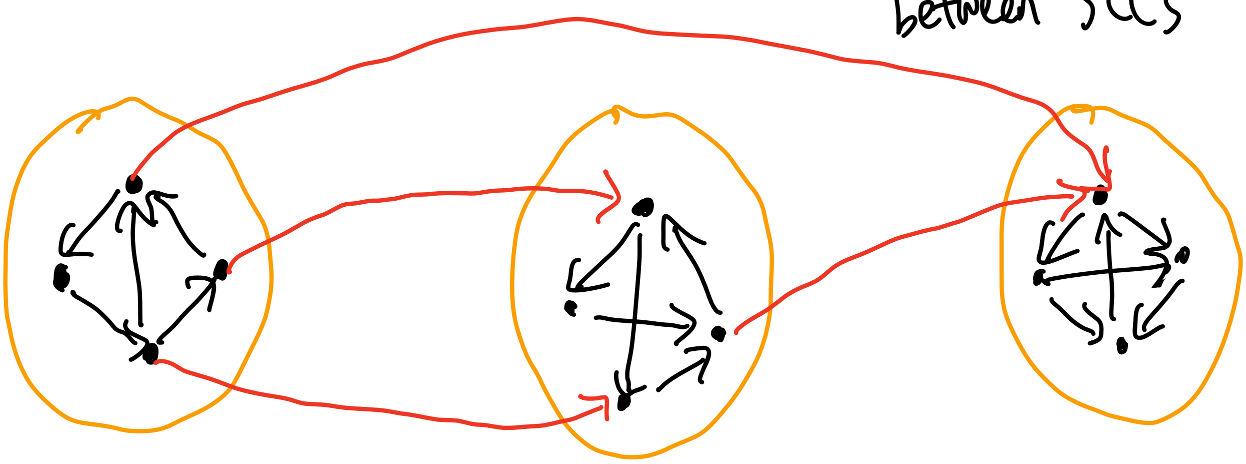
Fix: Strong connectivity

We say • ~ • if ∃



Equivalence relation, partitions G into SCCs

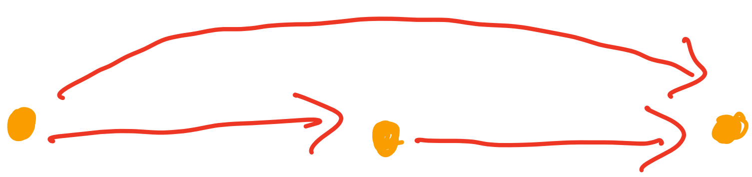
Overall structure of G:



SCC(G): Vertices = $1, 2, \ldots, k$ #SCCs

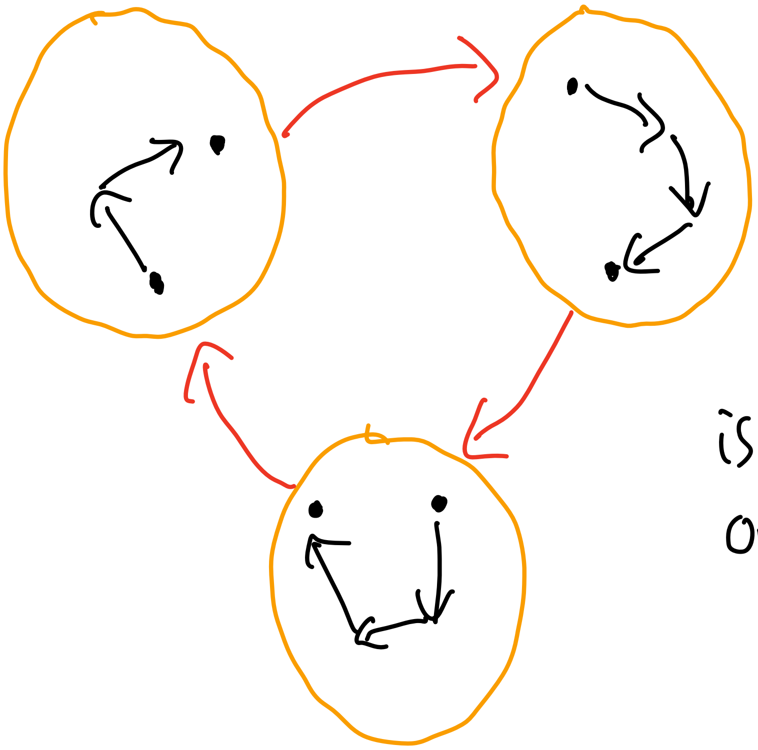
edge = (i, j) iff ∃ some $SCC_i \to SCC_j$

e.g.



Claim: SCC(G) is a DAG

Proof:

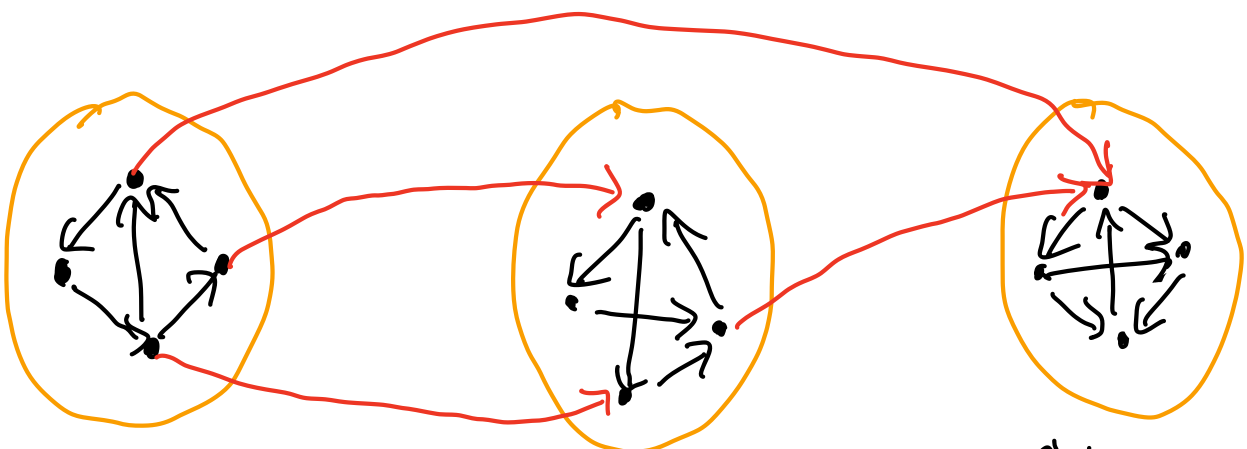


is actually one large SCC

How to compute SCCs?

Idea: every DAG has a source and sink

no inc. edges — no out edges



Source component

Sink component

Algo: Repeat following.

1) Find vertex s in sink component

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

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

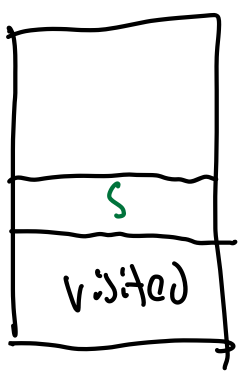
How about 1)? Claim: Can implement in $O(m+n)$.

Proof sketch: In postordering of all vertices,

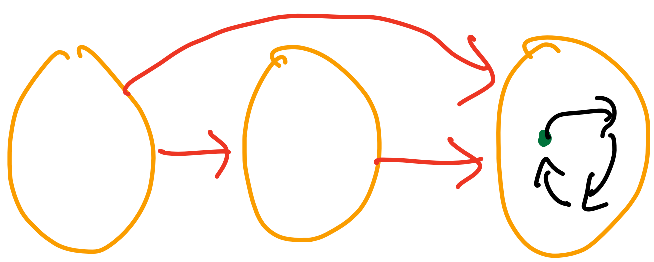
last vertex is in source component.

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

Goal: Compute all shortest path distances:

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

We've already seen two variants.

1) DAGs (Part III)

2) Unweighted (Last class)

Today: 3) Positive edge weights 4) any weights

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

Two key ingredients in Dijkstra's algo:

- relaxing tense edges
- priority queue

$\boxed{\text{Relaxing tense edges}}$

Let $D[v] \geq d(s, v)$

overestimate, keep finding shorter paths

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

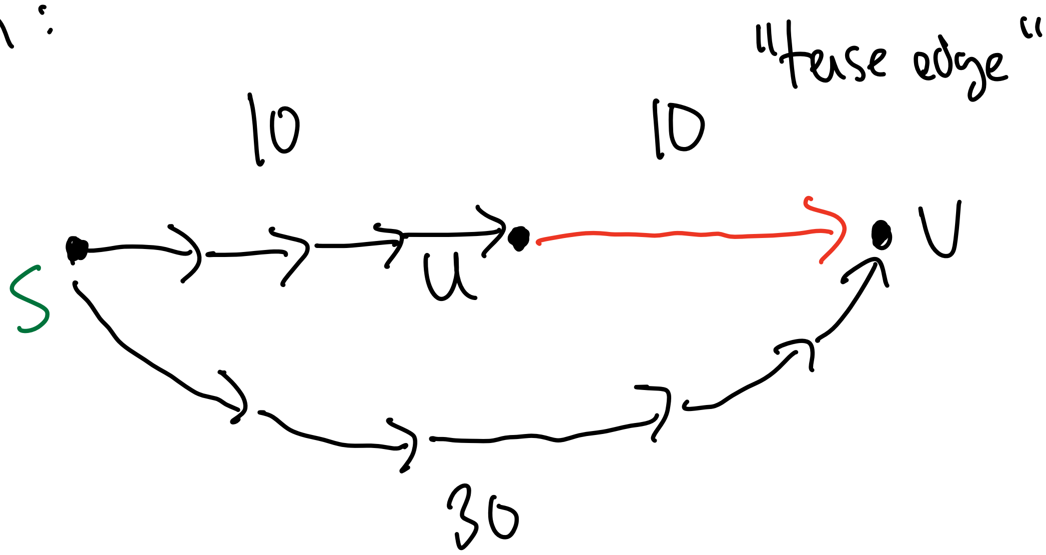
remains an overestimate.

We call it <u>relaxing</u> the edge (u, v)

Proof: $D[u] + w(u,v) \geq \delta(s,u) + w(u,v)$

$\geq \delta(s,v)$

Intuition:



Priority queue

values	1	3	7	...
objects	x	y	z	

- Insert (x, val)
- Delete (x)
- ExtractMin ()

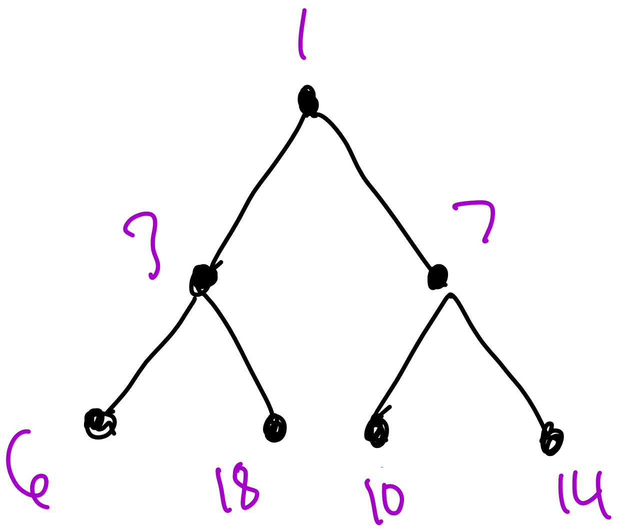
take out smallest-val object from set.

You get to cut in line.

We can implement with <u>Heap</u>.

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

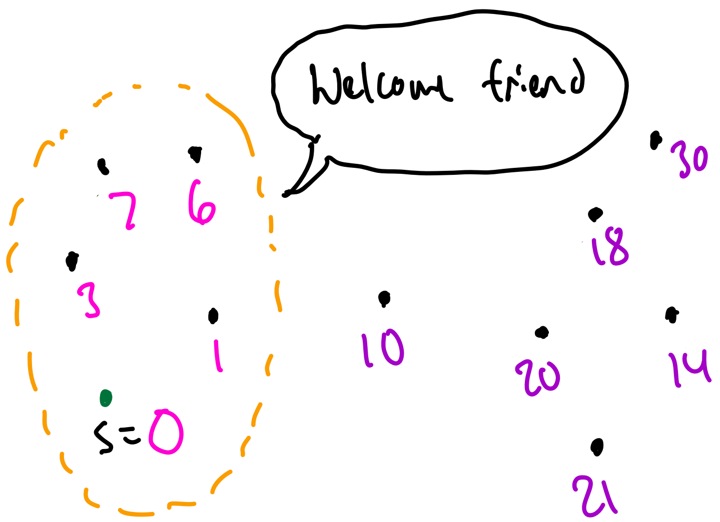
Review: Section 7.2, Part I



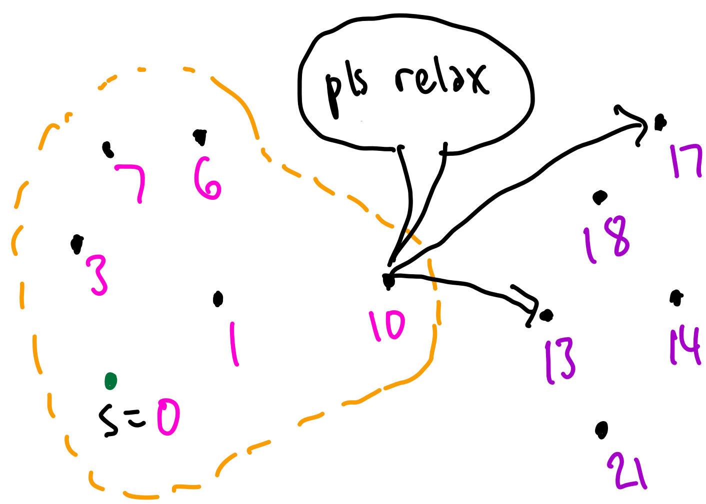
<u>Key ideas</u>

- 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)

SSSPPositive(G, s):

$H \leftarrow$ Heap.Init()

H.Insert(s, 0)

For $v \in V$, $v \neq s$: H.Insert(v, ∞)

$\partial \leftarrow$ Array.Init(n)

While $|H| > 0$:

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

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

Return ∂

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

[Thorup, 2000]

Correctness: Induction on steps

Step 1  Correct (positive weights)

Step $k+1$

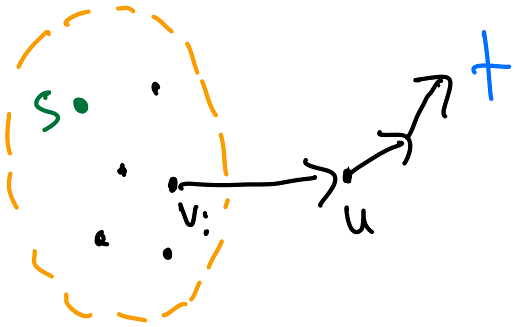
(assume $1, 2, \ldots k$)

Every unfrozen label is

$$v.\mathrm{val} = \underbrace{\min_{\substack{i \in [k] \\ (v_i, v) \in E}} \delta(s, v_i) + w_{(v_i, v)}}_{\text{all candidates in algo}}$$

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

How can another path beat $t.val$?



We have

(Positivity)

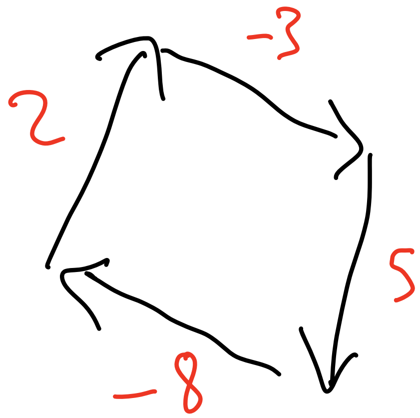
$$\text{path length} \geq \partial(s, v_i) + w_{(v_i, u)} + \partial(u, t)$$
$$\geq u.val \geq t.val$$

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 $mn^2 \Rightarrow mn$)

$S[v][\ell]$ = shortest $s \to v$ path, $\leq \ell$ 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 $\underbrace{n-1}_{\substack{\text{upper bound} \\ \text{on path length.} \\ \text{no cycles}}}$ times

Return D

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 $(v_1, v_2), \ldots, (v_k, v_1)$

Suppose for <u>any</u> D

$$D[v_i] \le D[v_{i-1}] + w(v_{i-1}, v_i) \quad \forall i \in [k]$$

$(k \equiv 0)$

Sum both sides,

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

cancels

not NWC!

$\Rightarrow\!\Leftarrow$