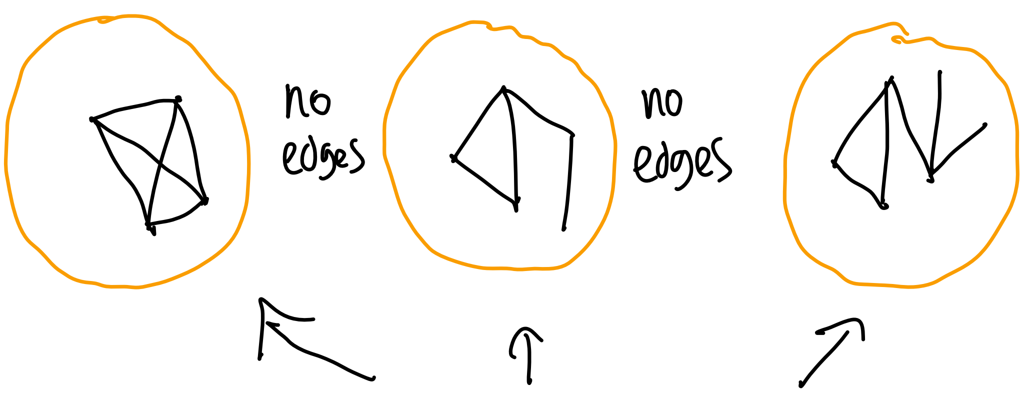
CS 331, Fall 2024
Lecture 12 (10/9)

Today:
- SCCs
- Dijkstra
- Bellman-Ford

SCCs (Part V, Section 2.2)

Recall connectivity structure in undirected:



connected components

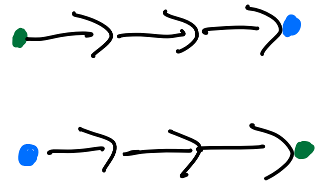
Key fact: Connectivity is <u>equivalence relation</u>

Not so for directed!

directed path not symmetric

Fix: <u>Strong connectivity</u>

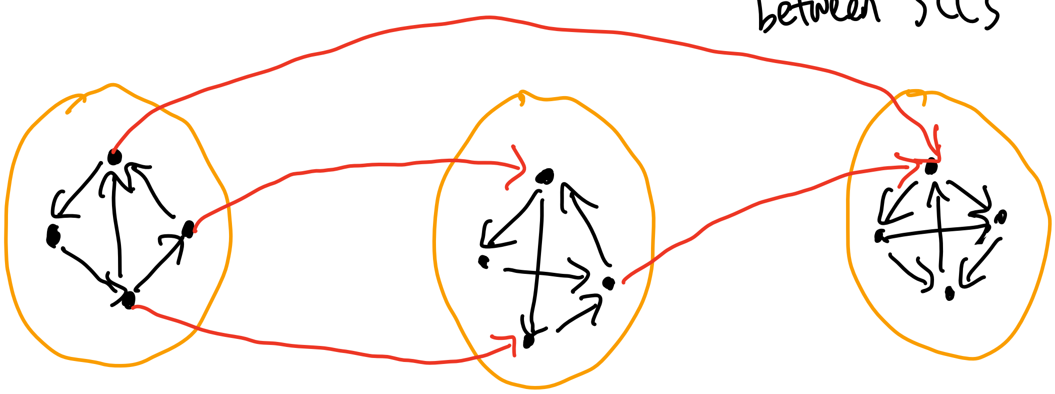
We say $\bullet \sim \bullet$ if $\exists$



Equivalence relation, partitions G into <u>SCCs</u>

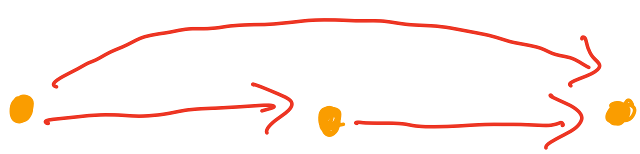
Overall structure of G:

edges between SCCs



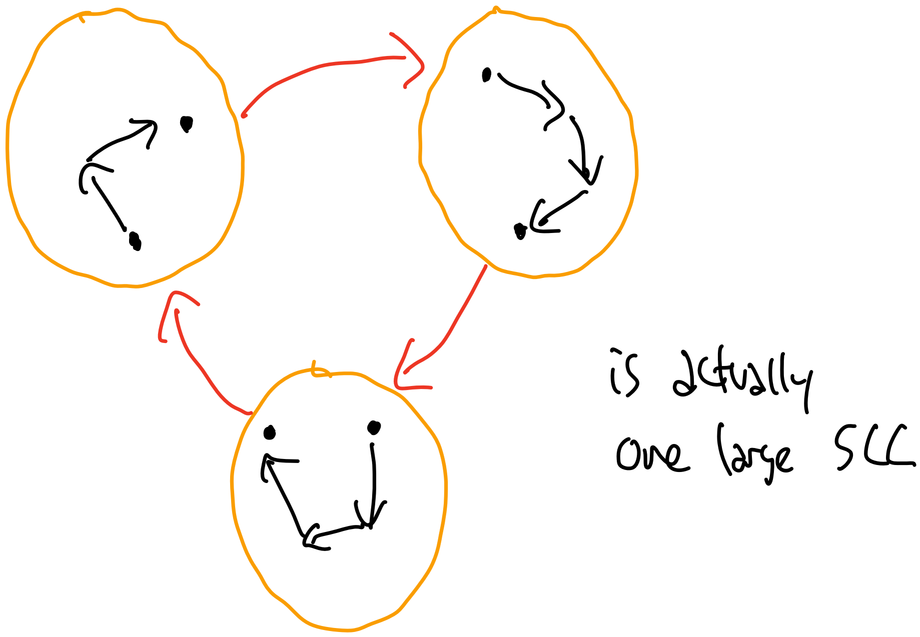
$SCC(G)$: Vertices $= 1, 2, \ldots, k$ #SCCs
edge $= (i, j)$ iff $\exists$ some $SCC_i \to SCC_j$

e.g.



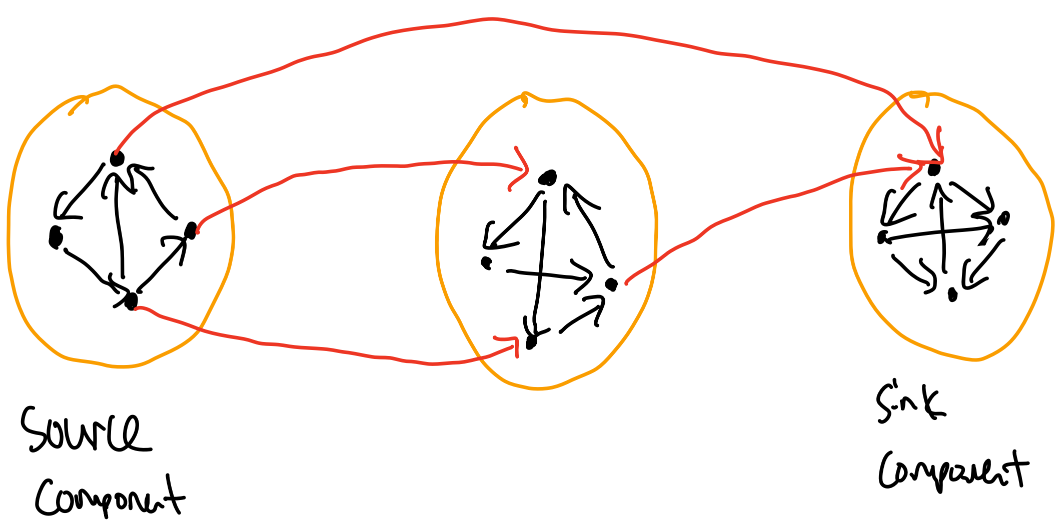
Claim: SCC(G) is a DAG

Proof:



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) $\underbrace{\text{Run DFS}(G, s) \text{ (only discover sink)}}$

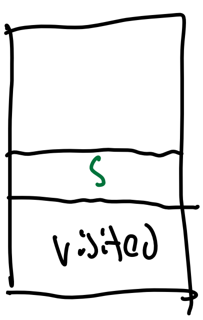
total runtime: $O(n) + \sum O(m_{C_S}) = O(m+n)$.

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

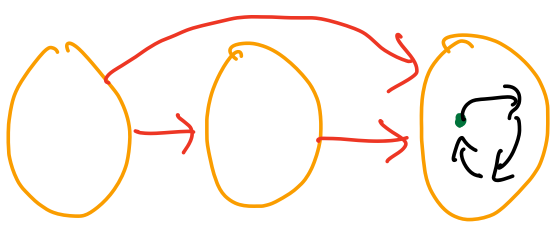
Proof sketch: In postordering of all vertices,
last vertex is in source component.

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:

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

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

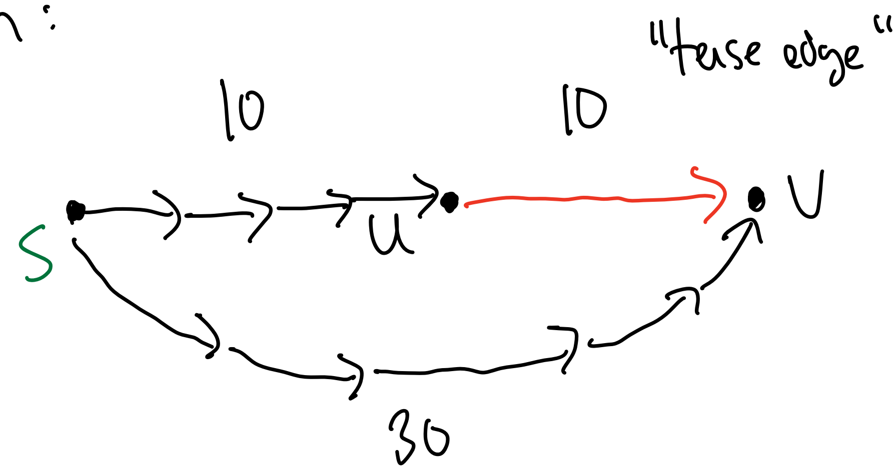
remains an overestimate.

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:



$\boxed{\text{Priority queue}}$

values	1	3	7	...
objects	X	Y	Z	

You get to cut in line.

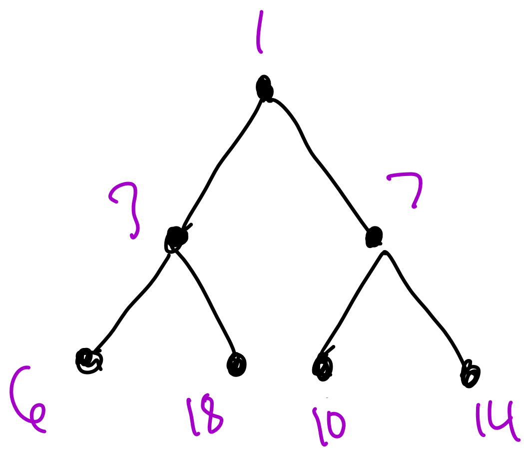
- Insert (x, val)
- Delete (x)
- ExtractMin ()

take out smallest-val object from set.

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

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

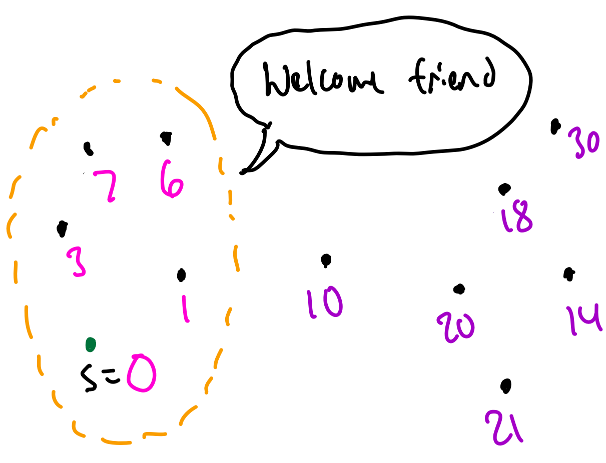
Review: Section 7.2, Part I



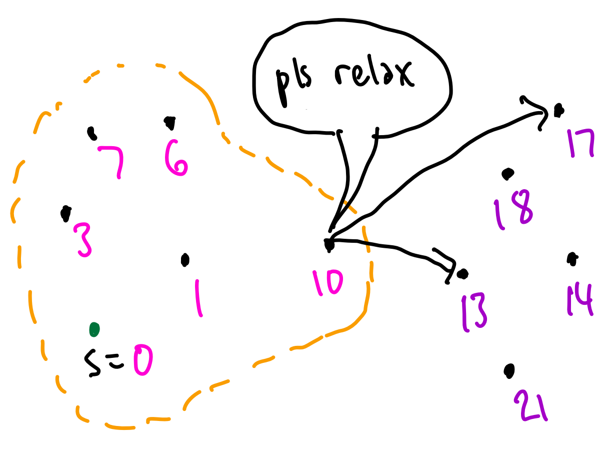
<u>Key ideas</u>

- 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)

SSSP Positive(G, s):

$H \leftarrow$ Heap.Init()

H.Insert$(s, 0)$

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

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

While $|H| > 0$:

locking distance

$v \longleftarrow$ H.ExtractMin()

$\delta[v] \leftarrow v.val$

$= O(n \log(n))$ time

relax edges

For $(v, u) \in E$:

H.Delete(u)

H.Insert$(u, \min(u.val, v.val + w_{(u,v)}))$

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

Return δ

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 $k+1$

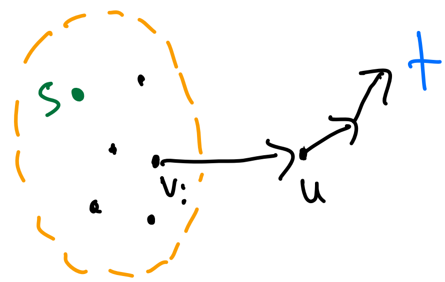
(assume $1, 2, \ldots k$)

Every unfrozen label is

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

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

How can another path beat $t.val$?



We have (positivity)

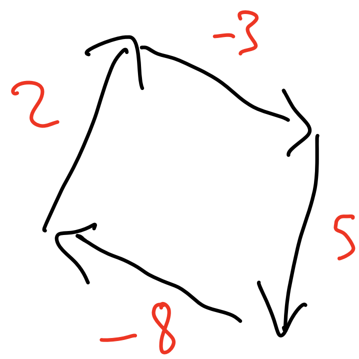
$$\text{path length} \geq \delta(s, v_i) + w_{(v_i,u)} + \delta(u, t)$$
$$\geq u.val \geq t.val$$

There's no better path.

Bellman-Ford (Part V, Section 3.2)

The gloves are off. Arbitrary graph, arbitrary edge weights

... remember, no <u>negative-weight cycles</u>.



Shortest path distance not well-defined.

Wait, didn't we solve <u>APSP</u> 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 $\underbrace{n-1}$ times

Return D

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

Suppose for <u>any</u> D

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

$(k \equiv 0)$

Sum both sides,

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

cancels

not NWC!

$\Rightarrow\Leftarrow$