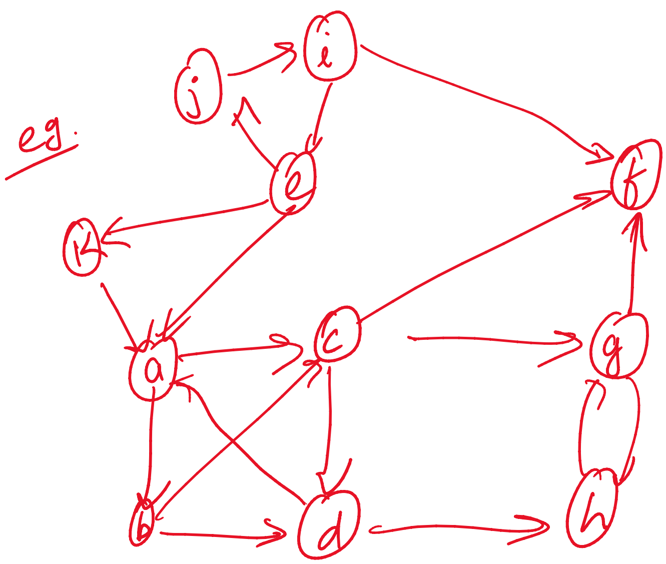
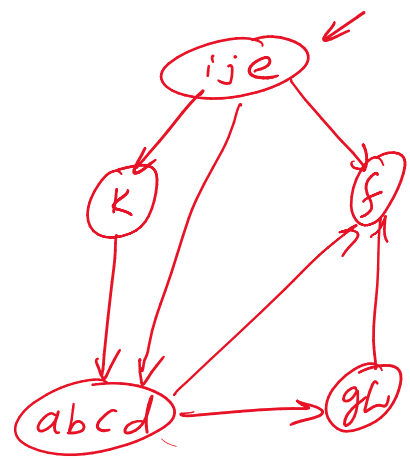
Monday, March 28, 2022 8:59 PM

## Last lec: Strongly Connected Components (SCC)

Given a directed graph $G=(V,E)$
Find a partition of it's vertex set s.t.
$u,v \in V$ are in the same partition
iff they are strongly connected $\equiv u \overset{}{\underset{\text{path}}{\rightsquigarrow}} v$ & $v \underset{\text{path}}{\rightsquigarrow} u$

eg.




Meta-graph is acyclic (DAG).

$\{j, i, e\}$, $\{a\ b\ c\ d\}$, $\{g, h\}$, $\{f\}$, $\{k\}$

---

**First idea:**

1. Find a vertex u in the source component of the meta-graph
2. Find u's component
3. Remove & repeat.

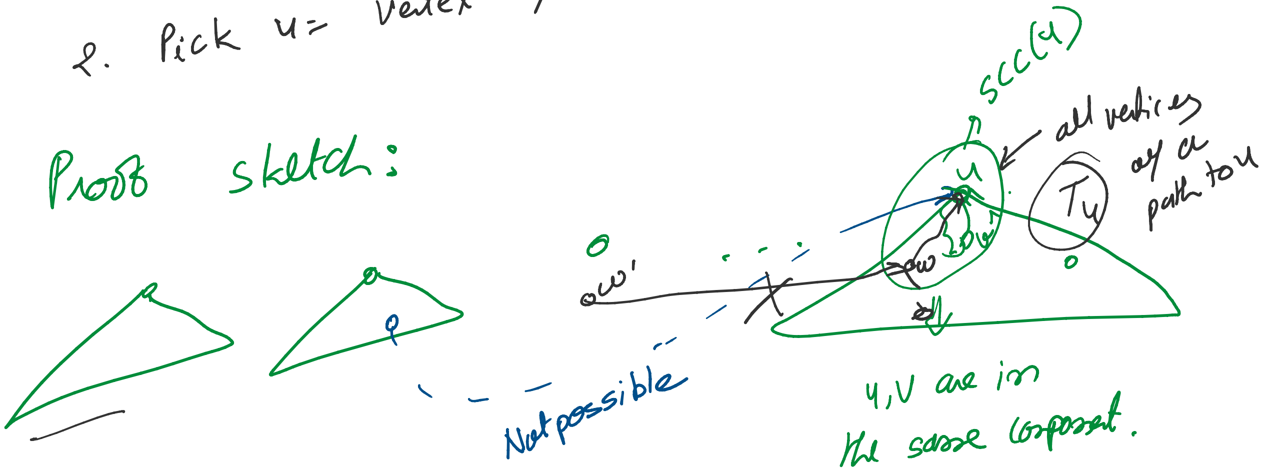
Q: How to find a vertex in the source component?

1. Run DFSAll(G)
2. Let u = vertex of largest finish order.

1. Run DFSAll( ...

2. Pick u = vertex of largest finish order.

Proof sketch:




Q: How to find u's component?

All vertices v that can reach $v \rightsquigarrow u$

$\Downarrow$

in $T_u$

Run DFS starting at u using "in-coming" edge reverse edges.

Q: How to remove u's component & repeat?

Do nothing! Just pick vertex in $V \setminus SCC(u)$ of max finish time.

$G^R$: G of every edge direction reversed. $\leftarrow O(m)$

---

Final Algo:

1. Run DFSAll(G). Order the vertices in decreasing order of finish time. $O(m+n) \leftarrow$

2. Run DFSAll($G^R$), while preferring vertices in the above order whenever there is a choice. $O(m+n) \leftarrow$

3. Output trees from step 2 as connected components.

3. Output trees from step 2 as connected components.

$O(n) \leftarrow$

Running Time: $O(m+n)$.

Exe: Run the algo on the above example.

## Shortest Paths:

Given a directed graph $G=(V,E)$, $s,t \in V$

$w: E \to \mathbb{R}_+$ (eg. Google maps)

find a path $P$ from $s$ to $t$

minimizes $\sum_{e \in P} w(e)$.

Special case 1: Unweighted ($w(e)=1, \forall e \in E$)

BFS(s). level(u) is the shortest path length from $s$ to $u$.

Solves Single Source Shortest path (SSSP) (finds mindist(s, u), $\forall u \in V$).

Special case 2: DAG (weighted).

Dynamic Programming. (solve SSSP)

- Define subproblems: $\forall v \in V$

$d(v)$ = mindist (s, v)

= shortest path length from s to v.

- Ans: $d(t)$

- Base case: $d(s)=0$

$m(\sum \text{indeg}(v))$ - Recursive Formula:

$d(v) = \min \quad d(u) + w(u,v)$

$O\left(\sum_{v \in V} \text{in-deg}(v)\right)$

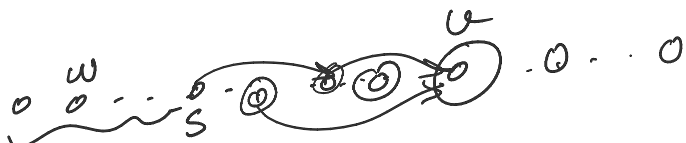
$\Rightarrow O(m)$


- Recursive Formula :

$$d(v) = \min_{u:(u,v) \in E} d(u) + w(u,v)$$

- Evaluation Order : Topological Sort.

$O(m+n)$




$d = \infty$

- Running Time : $O(m+n)$

Note : - Works only for DAGs .

- Also works w/ -ve weights!

$\Downarrow$

can be used to find "longest path" in DAGs .

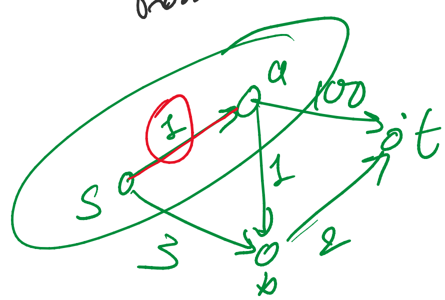
---

Dijkstra's Algo (1959) :

- Assume no -ve weights : $w(e) \geq 0$ , $\forall e \in E$ .
- Solve SSSP.

Compute $d[v]$ = shortest-path length , $\forall v \in V$ from s to v,

Greedy!




Idea :- Consider vertices in increasing order of distance from s.

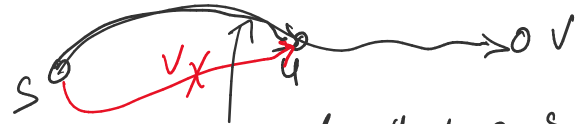
Observations :

shortest path

## Observations:

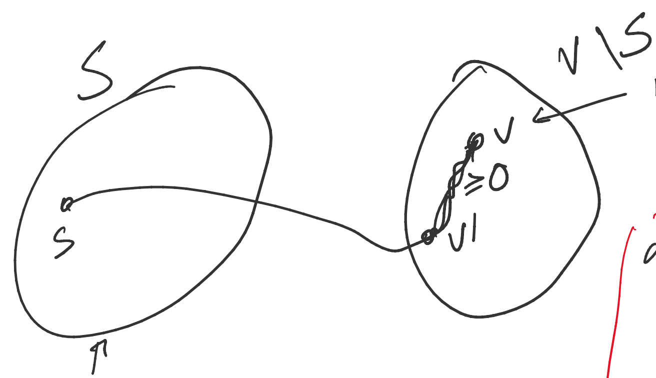
①

shortest path




shortest path from s to u.

②




is the next nearest vertex to s? **NO!**

$$\boxed{dist(s, v') < dist(s, v) \quad (\text{if } w(e) > 0 \ \forall e)}$$

known vertices. ≡ all vertices v whose shortest paths are computed.

⇓

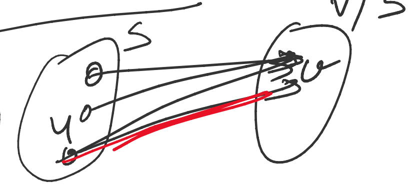
Next nearest vertex has to be "adjacent" to a vertex in S.

## Highlevel logic of Algo:

/maintain the set of "known vertices".

1. $S = \{s\}$, $d[s] = 0$
2. while $S \neq V$ {

$O(m) \to$ 3. Pick edge $(u, v) \in E$ s.t. $u \in S$, $v \in V \setminus S$ minimizes $d[u] + w(u, v)$




4. set $d[v] = d[u] + w(u, v)$
5. Insert v to S.

}

$O(mn)$.

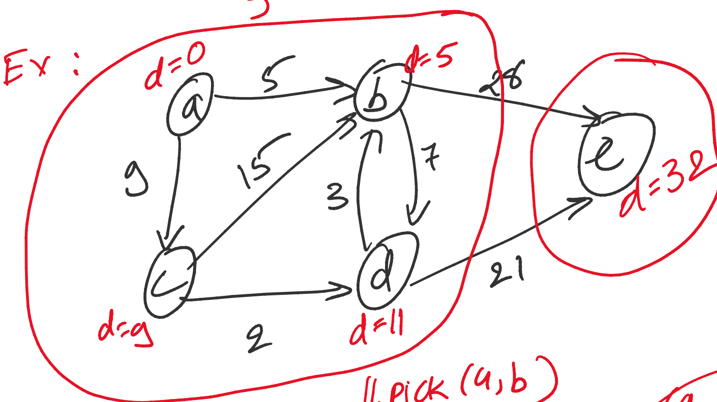
)

Running Time : $O(mn)$.
V/S

5

$s = a$, $t = e$

Ex :

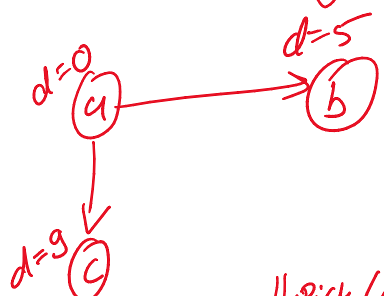



⇓ Pick (a,b)

(a,b) : 0 + 5
(a,c) : 0 + 9

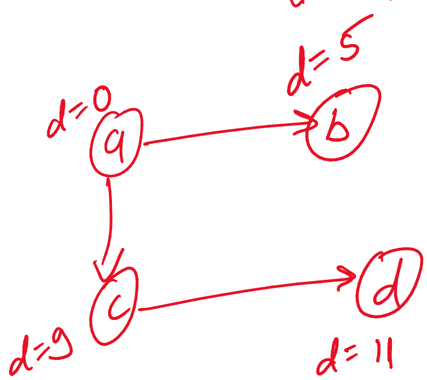

⇓ Pick (a,c)

(b,e) : 5 + 28
(b,d) : 5 + 7
(a,c) : 0 + 9



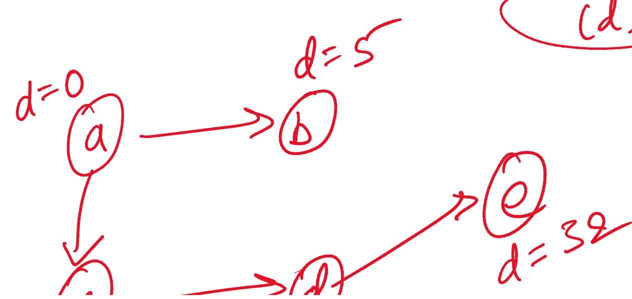

⇓ Pick (c,d)

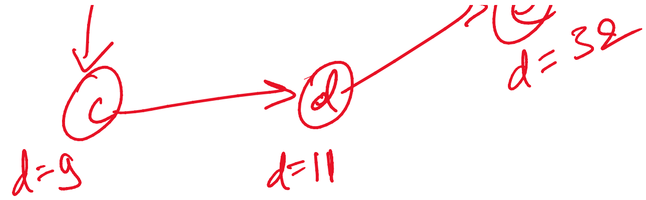
(c,d) : 9 + 2
(b,e) : 5 + 28
(b,d) : 5 + 7




⇓ Pick (d,e)

(b,e) : 5 + 28
(d,e) : 11 + 21






## Proof of correctness :

**Claim :** → Assume $d[u]$ value are correct $\forall u \in S$.

If $(u,v)$ is edge from $u \in S$ to $v \in V \setminus S$ with smallest $\underline{d[u] + w(u,v)}$ value.

Then

$$\text{mindist}(s,v) = d[u] + w(u,v).$$

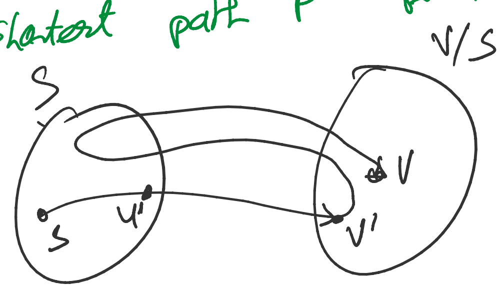
$(\leq)$

$(\geq)$

**pf :** $(\leq)$ There is a path from s to v


$$\text{min dist}(s,v) \leq d[u] + w(u,v)$$

$(\geq)$ Take shortest path $p^*$ from s to v.




length ( v' to v on $p^*$ ) $= l \geq 0$

$$\text{mindist}(s,v) = \text{length}(p^*) = d[u'] + w(u',v') + l$$

$$\geq d[u'] + w(u',v')$$

$$\geq d[u] + w(u,v)$$

$$\geq_{\left(\substack{\text{by def} \\ (u,v)}\right)} d[u] + w(u,v)$$

∎

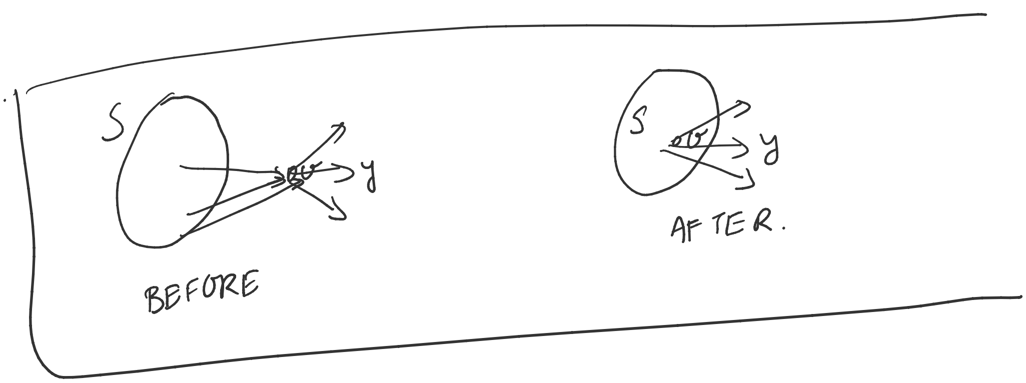
## * Efficient Implementation (using priority Queue):

// maintain $key[v] = \min_{u \in S} d(u) + w(u,v)$, $\forall v \in V \setminus S$

// $Q = V \setminus S$

1. $Q = V \setminus \{s\}$
2. $key[v] = \infty$, $\forall v \in V \setminus \{s\}$; $key[s] = 0$
3. while $Q \neq \emptyset$ {
4. Pick $v \in Q$ w/ smallest key.
5. $d[v] = key[v]$.
6. Remove $v$ from $Q$.
7. for each out-neighbor $y$ of $v$
8. if $y \in Q$ & $d[v] + w(v,y) < key[y]$
9. then $key[y] = d[v] + w(v,y)$; $Pred[y] = v$.

}




... data structure | Priority Queue

| ★ Running Time : | No data structure for Q. | Priority Queue for Q. |
|---|---|---|
| line 4 | $O(n)$ | $O(\log n)$ |
| line 6 | $O(1)$ | $O(\log n)$ |
| line 9 | $O(1)$ | $O(\log n)$ |
| Total: | $O\left(n^2 + \sum_{v \in V} \text{out-deg}(v) \cdot O(1)\right)$ $= O(n^2 + m) = O(n^2)$ | $O\left(n \cdot \log n + \left(\sum_{v \in V} \text{out-deg}(v)\right) \cdot \log n\right)$ $= O(n \log n + m \log n)$ $= O((n+m) \log n)$. |