Lecture 8

Wednesday, February 7, 2024 2:47 PM

Finding SCC in a directed Graph

Algorithm for finding SCCs

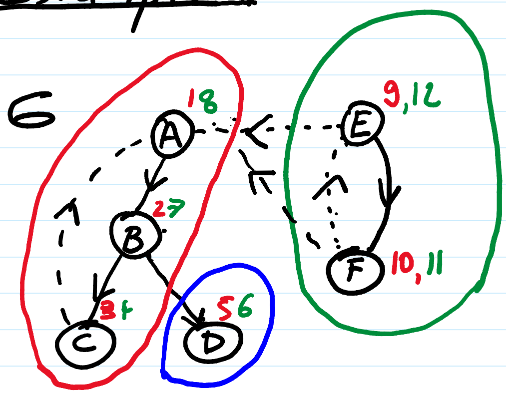
1) Run DFS on G^T, the graph with all edges reversed, and order vertices reversely by their post numbers
2) Run DFS on G, considering vertices in the above order. Whenever explore terminates, set SSC-count = SCC-count + 1, and restart explore(v) at the vertex with highest post value among the left vertices.

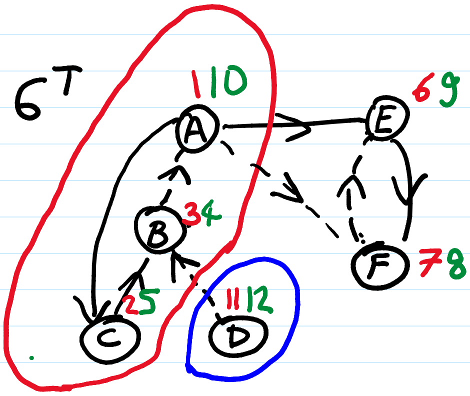
Correctness Proof uses

Property 1: If explore is started at v it will terminate when all vertices reachable from v have been discovered. Furthermore, pre(v) is the lowest, and post(v) the highest among all discovered vertices

Property 3: If we order all SCCs by their largest post number, then all edges point forward

Example 1

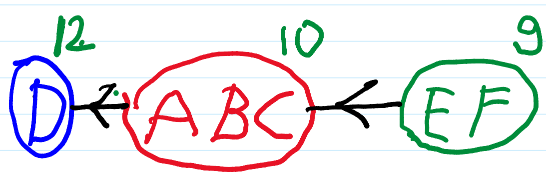




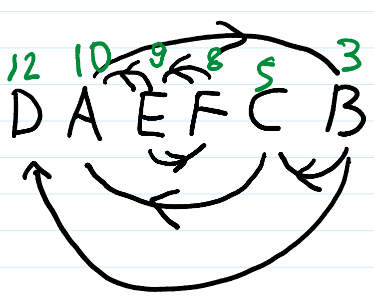
Order by post(·) in G^T

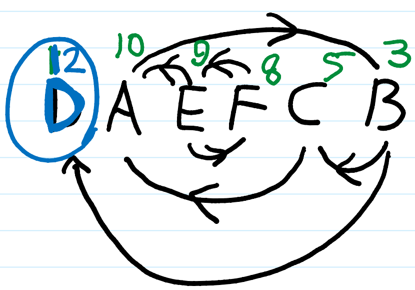
12 10 9 8 5 3
D A E F C B

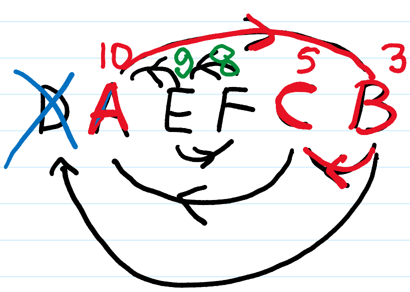
SCC, largest post, edges in G



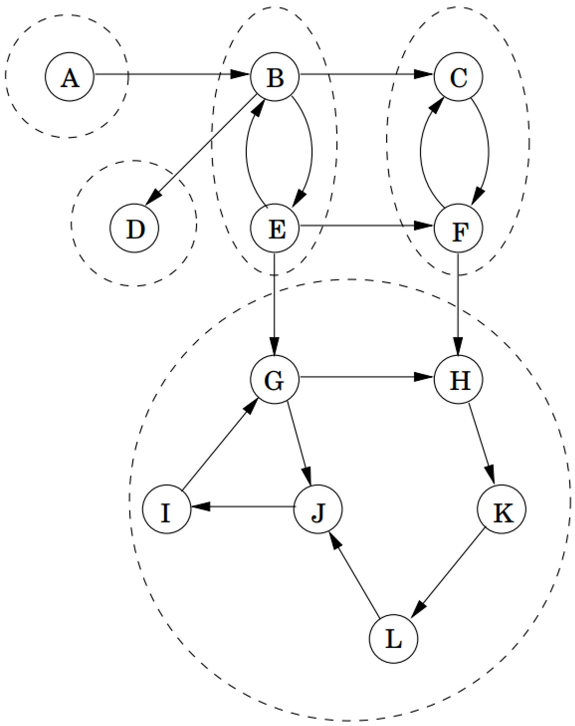
DFS in G:



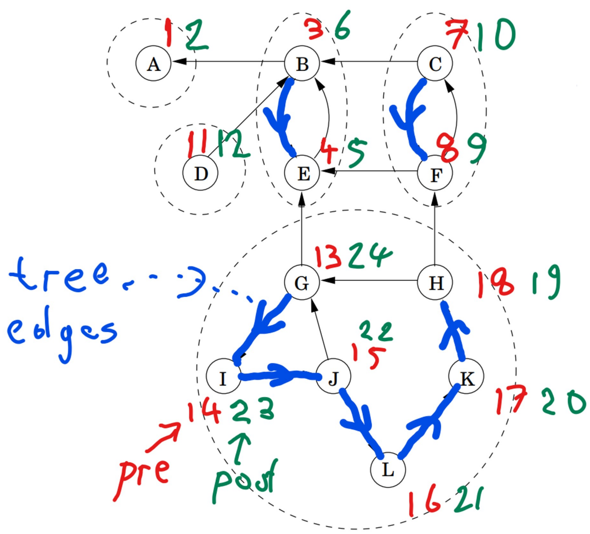


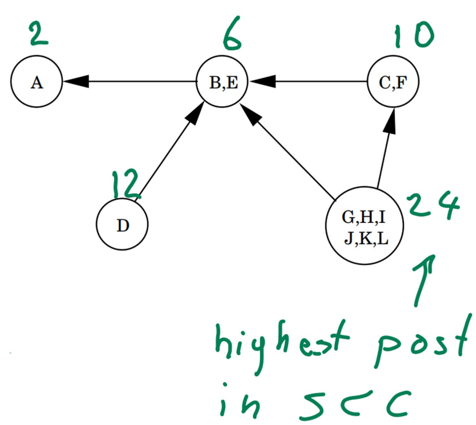


Example 2:



G

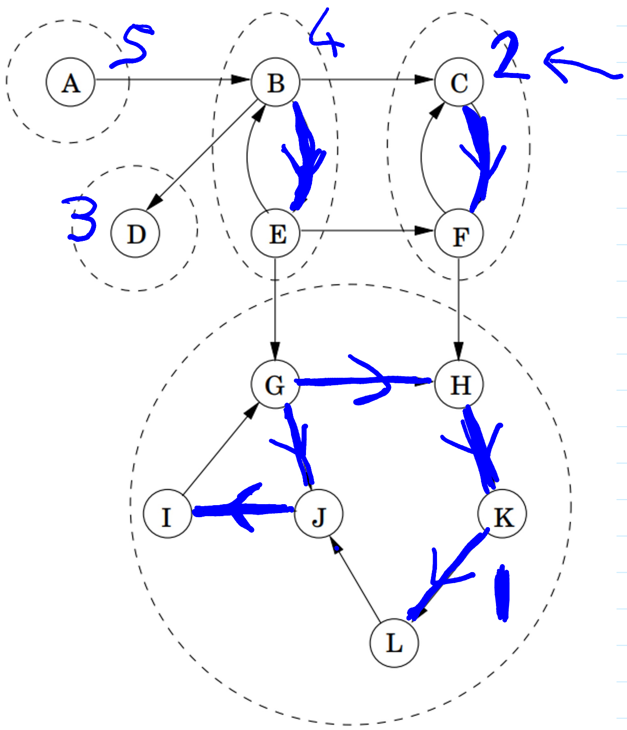




G^T SCC-DAG of G^T

Ordered by post in G^T

BFS in G

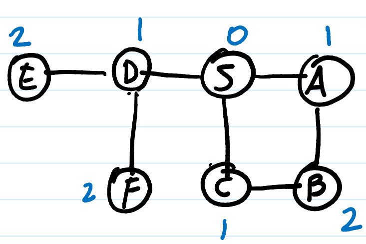


← scc-counter

Path in Graphs

Goal: Calculate distances

Def: dist(u, v) = length of shortest path between u & v



distances from S

How to find distances from a source s ?

<u>Single Source Shortest Path Algor.</u>

Breath First Search BFS

Dijkstra's Algorithm

Bellman-Ford Algorithm

BFS

- Start from source s
- Find its neighbors → vertices at distance 1

  ⋮

- give set of vertices at distance d, find yet to be seen vertices at distance $d+1$

bfs(G, s)

<u>Input</u>: $G=(V,E)$ $s \in V$

<u>Output</u>: For all vertices reach. from u,
$\text{dist}(u) = \text{dist}(s,u)$

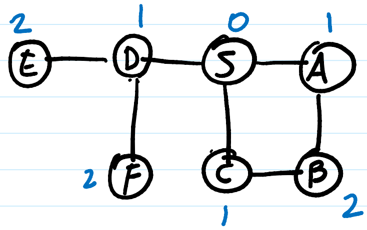
$\forall u \in V: \text{dist}(u) = \infty$

$\text{dist}(s) = 0$

$Q = [s]$ (queue contain. s)

```
While Q ≠ ∅
    u = eject(Q)
    For all edges (u,v)
        if dist(v) = ∞
            dist(v) = dist(u) + 1
            inject(Q,v)
```

Example



u	Q
	[S]
S	[A, C, D]
A	[C, D, B]
C	[D, B]
D	[B, E, F]
B	[E, F]
E	[F]
F	ϕ

dist 0: S

dist 1: A, C, D,

dist 2: B

dist 2 : E, F

Claim: This gives $dist(v) = dist(s, v)$

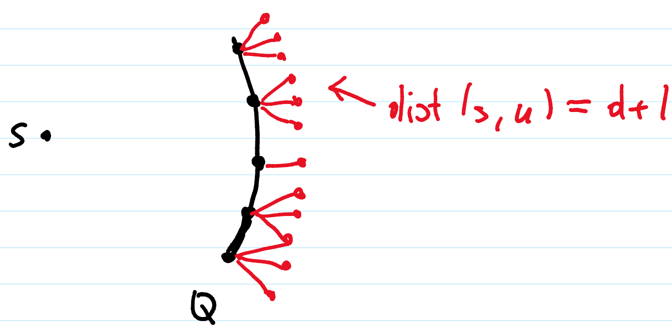
Pf: By induction

Assume at time t, we have

$dist(v) = dist(s,v)$ if $\le d$

$dist(v) = \infty$ otherwise

$Q = \{ v \in V : dist(s,v) = d \}$



Claim: Running Time $= O(|V| + |E|)$

Dijkstra's Algorithm

For many problems, edges have length:

street Networks,

time it take an infection to infect neighbors in contact network

Formal Setting

Graph $G = (V, E)$

edge lengths $\ell(u,v)$ for $(u,v) \in E$

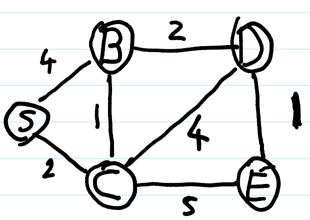
source s

<u>Goal</u>: $\forall w \in V$, find

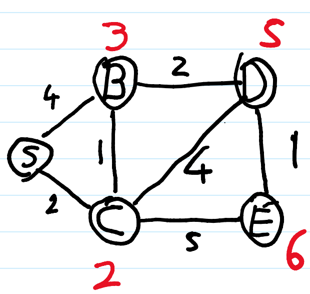
$$d(s,w) = \min_{\omega} \sum_{(u,v) \in \omega} \ell(u,v)$$

where the minimum goes over all paths ω from s to w

Example:



Try to solve systematically, finding nearest vertices first



1. $v_1 = S$ $\quad d_1 = 0$
2. $v_2 = C$ $\quad d_2 = 2$
3. $v_3 = B$ $\quad d_3 = 3$
4. $v_4 = D$ $\quad d_4 = 5$
5. $v_5 = E$ $\quad d_5 = 6$

More systematically

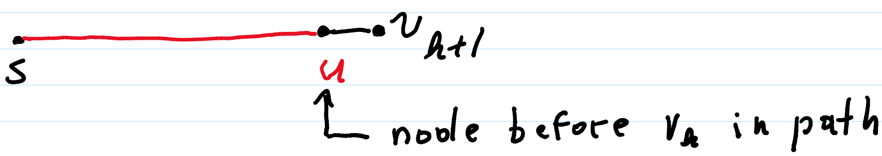
K = known nodes $= \{v_1, \cdots v_k\}$

U = unknown nodes

Want to keep $d(s, v_k)$ as small as possible

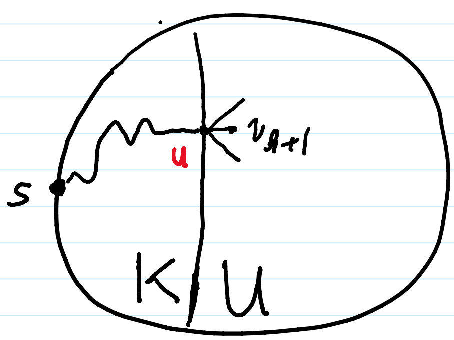
How to find v_{k+1}

v_{k+1} must lie on shortest path ω



- u must be in K
- path from s to u must be a shortest path

$\Rightarrow \quad \ell(\omega) = d(s, u) + \ell(u, v_{k+1})$



$d(s, v_{k+1}) =$

$$= \min_{u \in K} d(s,u) + \ell(u, v_{k+1})$$

$$= \min_{v \notin K} \min_{u \in K} \left(d(s,u) + \ell(u,v)\right)$$

$$\Rightarrow \boxed{v_{k+1} \text{ minimizes } \min_{u \in K} \{d(s,u) + \ell(u,v)\}}$$

Dijkstra does this inductively, keeping an array dist[v], $v \in V$ such that

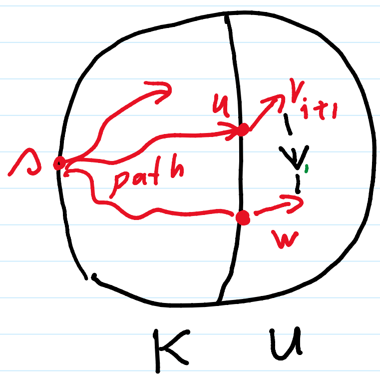
$$\text{dist}[v] = \begin{cases} d(s, v) & v \in K \\ \min_{u \in K} \left(\text{dist}(u) + \ell(u, v)\right) \end{cases}$$

so we can inductively choose v_{k+1} such that it minimizes dist[u] over $U = V \setminus K$

Question: After adding v_{k+1} to K, how do we update dist[w] ?

Answer: We want to maintain

$$\text{dist}[v] = \begin{cases} d(s,v) & v \in K \\ \min\limits_{u \in K}(\text{dist}(u) + \ell(u,v)) \end{cases}$$



new shortest path in K via v_{k+1}

$$\text{dist}[w] = \min \underbrace{\{\text{dist}[w], \text{dist}(v_{k+1}) + \ell(v_{k+1}, w)\}}_{\text{update}(v_{k+1},\, w)}$$

<u>dijkstra (G, ℓ, s)</u>

dist[s] = 0

$\forall\, v \neq s$ dist[v] = ∞

$U = V$

while $U \neq \emptyset$

choose $u \in U$ s.th. dist[u] is minimal

remove u from U

$\forall$ edges $(u,v) \in E$

dist[v] = min{dist[v], dist[u] + ℓ(u,v)}

Q: How do we implement this ?

Priority Queue

contains a set of
(element, key) pair

vertices   dist

Operations

- Insert (elem, key)
- Decrease (elem, key)
- Delete Min ← removes elem with lowest key

dijkstra (G, ℓ, s)

dist [s] = 0

$\forall$ v ≠ s   dist [v] = ∞

~~U = V~~ $\forall$u insert (u, dist [u])

while U ≠ ∅

choose u ∈ U s.th. dist [u] is minimal

~~remove u from~~ U   u = Delete Min

$\forall$ edges (u, v) ∈ E

dist [v] = min {dist [v], dist [u] + ℓ (u, v)}

Decrease Key (v, dist [v])

Running Time

|V| inserts & deletes

|E| decrease

Implementation	deletemin	insert / decreasekey	$\|V\| \times$ deletemin + $(\|V\|+\|E\|) \times$ insert
Aray	$O(n)$	$O(1)$	$O(n^2)$
Binary heap	$O(\log n)$	$O(\log n)$	$O((n+m)\log n)$
d-ary heap	$O\left(\frac{d \log n}{\log d}\right)$	$O\left(\frac{\log n}{\log d}\right)$	$O(nd + m\frac{\log n}{\log d}$
Fibonacy heap	$O(\log n)$	$O(1)$	$O(n\log n + m)$