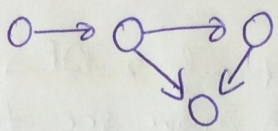
CS124 Lecture 5

Debugging order.
↳ topological [redacted] sort $B \to A$
↳ DFS (decr. postorder)

SCC: Strongly Connected Components



source ○ indeg 0

sink ○ outdeg 0

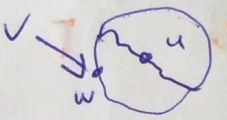
need source SCC or sink SCC ∈ DFS

lemma [redacted] if we start a DFS at a vertex in a sink SCC
↳ get sink SCC before jumping

problem: find a vertex in a sink SCC
– run DFS, smallest postorder → FALSE

lemma [redacted] largest postorder # in the source SCC

proof by contradiction: assume postorder(v) < postorder(u)



case 1 u is on stack before v
contradiction

case 2 v is on stack before u
there is a path from
v to u
contradiction

lemma reverse the graph ⇒
sinks become sources
sources become sinks

① take G, form G^R

② Do a DFS on G^R
find source vertex in G^R, sink vertex in G

③ Run DFS on sink vertex in G
and remove it

$O(2V) = O(V)$ DFS
$O(E)$ per DFS

get tighter bound:

$O(EV)$

but only 2 DFSs !

② remove scc, then get largest post order from remaining

③ go through each edge once

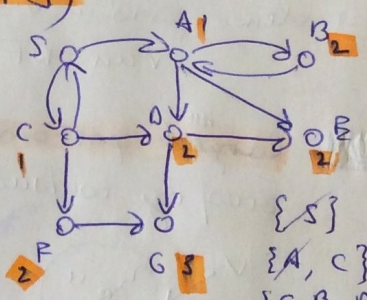
$O(E + V)$

Breadth First Search (BFS)

(notion of distance)

BFS ⇔ queue

```
Breadth First Search
    pick start vertex s
    dist[s] := 0
O(1)  inject (q, s)
    placed (s) := 1
O(V)  while (size (q) > 0)
        v := pop (q)
O(E)      for (v, w) ∈ E
total       if placed (w) = 0
              inject (q, w)
              placed (w) := 1
              dist[w] := dist[v] + 1
```



{S}
{A, C}
{C, B, D, E}
{B, D, E, F}
{E, F, G}

$O(E + V)$

shortest # of edges distance from s to any vertex that can be reached by s

By induction

O(1) pop

// # of edges distance → very basic

pop min <= only 2 level level by level

queue pop inject

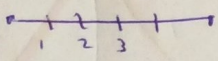
BFS invariants

- queue is sufficient to pop minimums
- each node in the queue has shortest paths

Add weights

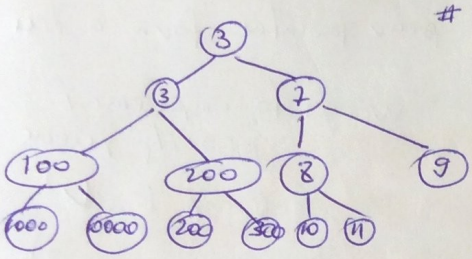
idea reduce to BFS

$O(\sum$ length all edges$)$



Priority queue: different implementations

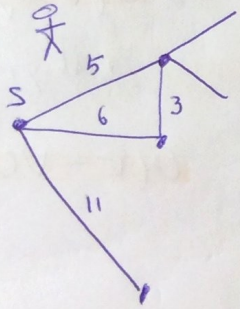
Heap min-heap (Binary trees to some level)
increases as go down any path



delete min – return object w/ smallest value and restructure

insert (x, y, h)
- insert item x with value y in the heap

change (x, y, h)
- if the current value for x is > y, change its value



Dijkstra's shortest [redacted] path alg

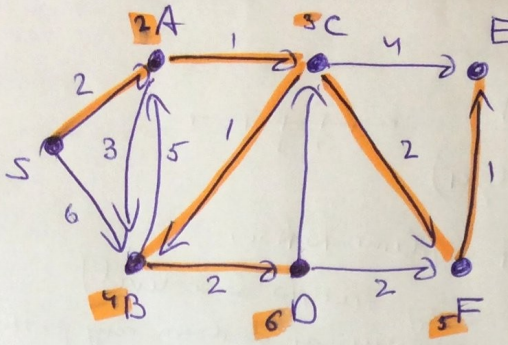
dist, prev. arrays, heap H (priority queue may be heap)

```
H := {s, 0}
for v ∈ V do
    dist[v] := ∞
    prev[v] := null
dist[s] := 0
while H ≠ ∅
    v := deletemin(H)
    for (v,w) ∈ E
        if dist[w] > dist[v] + length(v,w)
            dist[w] := dist[v] + length(v,w)
            prev[w] := v
            change / insert (w, dist[w], H)
```

Induction on popped vertices
→ each vertex deleted,
⇒ a shortest path to it is found. All positive distances, ~~the~~ paths to other fringe nodes are longer/as long

invariants:

[redacted] min node in priority queue has shortest paths [redacted]



S : 0

~~A : 2~~, ~~S : 6~~

~~C : 3~~, ~~A : 5~~

~~C : 4~~, ~~E : 7~~, ~~F : 5~~

~~E : 5~~, ~~D : 6~~, ~~B : 7~~

~~B : 6~~, ~~D : 6~~

prev pointer form a tree

$O(V \text{ deletemin} + E \text{ insert/change})$

ways to implement priority queue

Better if $E = V^2$

Unsorted list

$O(V)$

$O(1)$ & array of pointers to change

$\rightarrow O(V^2)$

Binary heap

$O(\log V)$

$O(\log V)$

$\rightarrow O(E \log V)$

d-ary heap

↑ sparse graph

Fibonacci heap

$O(\log V)$

$O(1)$ ~~amortized~~

↑ on average over the course of all ops

$O(E + V \log V)$