Recap:

Breadth-First Search (BFS) (a graph exploration problem)

Idea: - look at vertices reachable in 0 steps,
1 steps,
2 steps
....,

- Careful not repeatedly visit same vertices
- Work for both directed/undirected
- Slime mold doing kinda like BFS:

https://youtu.be/q8ST4lqtUzQ

Queue Q: $\boxed{a} \leftarrow \boxed{b} \leftarrow \boxed{c}$, First-In-First-Out (FIFO)

Q.push(d) : $\boxed{a} \leftarrow \boxed{b} \leftarrow \boxed{c} \leftarrow \boxed{d}$ $O(1)$

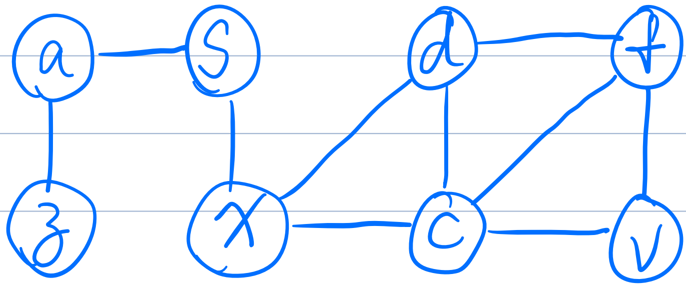
Q.pop( ) : $\boxed{a}$ $\boxed{b} \leftarrow \boxed{c} \leftarrow \boxed{d}$ $O(1)$

1. BFS (cont'd)

```
BFS(G,s):
    For all v ∈ V:
        dist(v) ← ∞                           O(|V|)
        parent(v) ← ⊥
        color(v) ← WHITE
    dist(s) = 0
    Q.push(s)                                 O(1)
    Color(s) = YELLOW
    While Q is not empty:
        u ← Q.pop()
            For all v ∈ Adj[u]
                If color(v) = WHITE:                    O(|E|)
                    dist(v) = dist(u) + 1
                    parent(v) = u             O(1)
                    color(v) = YELLOW
                    Q.push(v)
            color(u) = GREEN
```

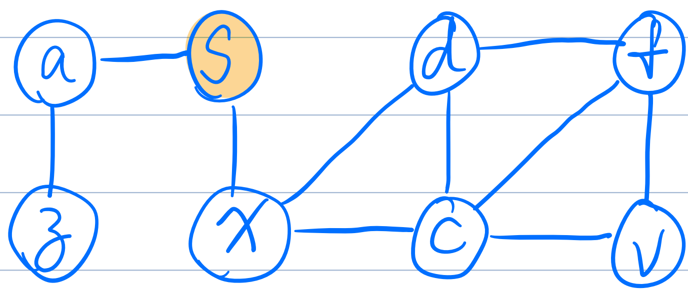
dist(v) gives the shortest distance from s to v

The path? Reverse (v, parent(v), parent(parent(v)), ...)



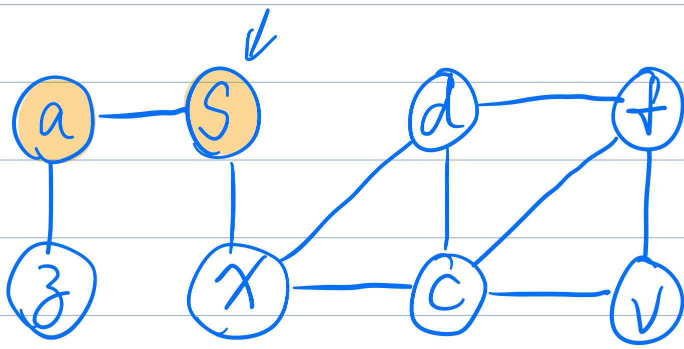
	dist	parent
a	∞	$\perp$
c	∞	$\perp$
d	∞	$\perp$
f	∞	$\perp$
s	∞	$\perp$
v	∞	$\perp$
x	∞	$\perp$
z	∞	$\perp$

Q:



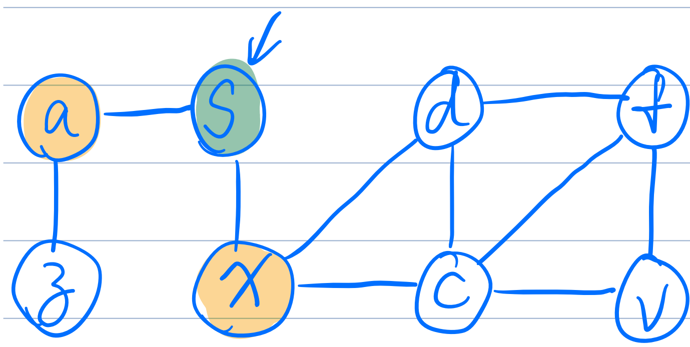
	dist	parent
a	∞	$\perp$
c	∞	$\perp$
d	∞	$\perp$
f	∞	$\perp$
s	0	$\perp$
v	∞	$\perp$
x	∞	$\perp$
z	∞	$\perp$

Q: s



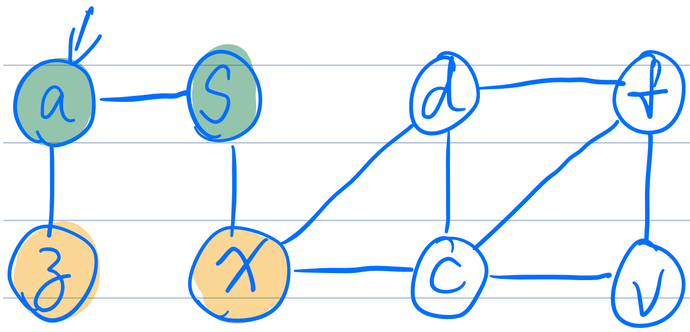
	dist	parent
a	1	S
c	∞	$\perp$
d	∞	$\perp$
f	∞	$\perp$
S	0	$\perp$
v	∞	$\perp$
x	∞	$\perp$
z	∞	$\perp$

Q: a



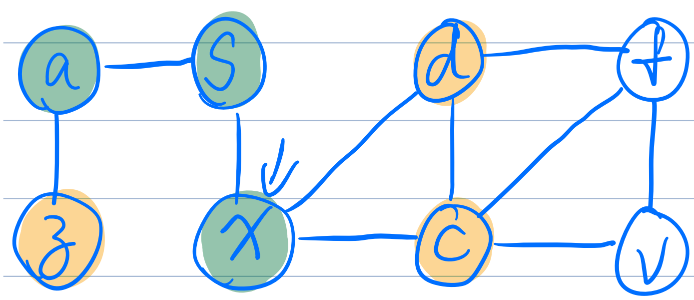
	dist	parent
a	1	S
c	∞	$\perp$
d	∞	$\perp$
f	∞	$\perp$
S	0	$\perp$
v	∞	$\perp$
x	1	S
z	∞	$\perp$

Q: a, x



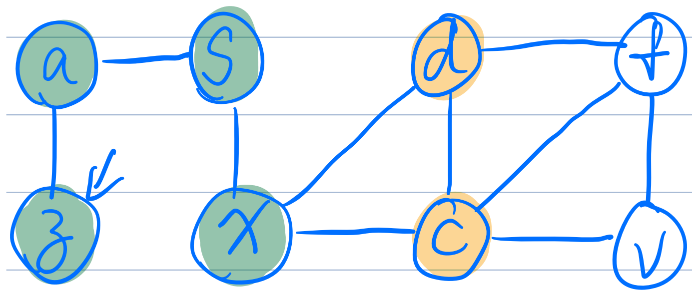
	dist	parent
a	1	S
c	∞	$\perp$
d	∞	$\perp$
f	∞	$\perp$
S	0	$\perp$
v	∞	$\perp$
x	1	S
z	2	a

Q: x, z



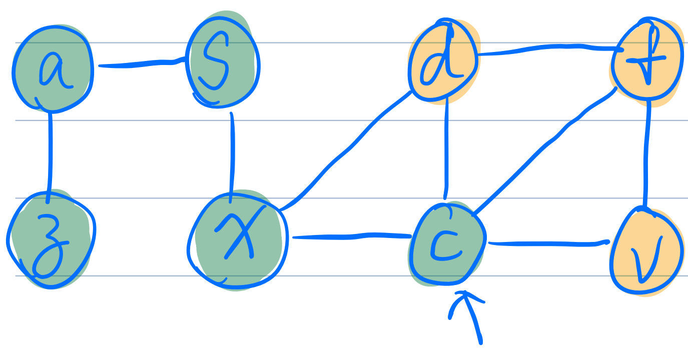
	dist	parent
a	1	S
c	2	x
d	2	x
f	∞	$\perp$
S	0	$\perp$
v	∞	$\perp$
x	1	S
z	2	a

Q: z, c, d



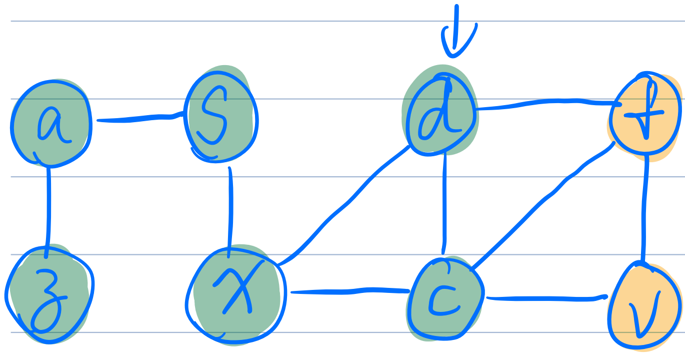
	dist	parent
a	1	s
c	2	x
d	2	x
f	∞	$\perp$
s	0	$\perp$
v	∞	$\perp$
x	1	s
z	2	a

Q: c, d



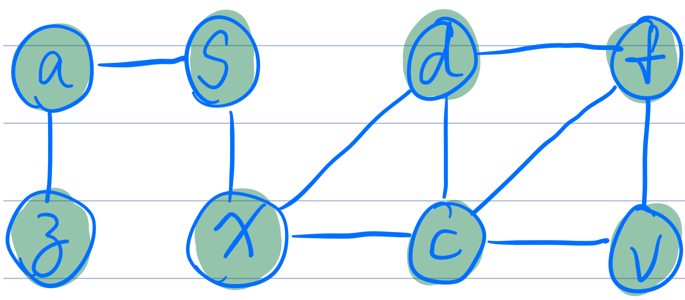
	dist	parent
a	1	s
c	2	x
d	2	x
f	3	c
s	0	$\perp$
v	3	c
x	1	s
z	2	a

Q: d, f, v



Q: f, v

	dist	parent
a	1	s
c	2	x
d	2	x
f	3	c
s	0	$\perp$
v	3	c
x	1	s
z	2	a



Q:

	dist	parent
a	1	s
c	2	x
d	2	x
f	3	c
s	0	$\perp$
v	3	c
x	1	s
z	2	a

○ : vertices not added to Q yet; "undiscovered"

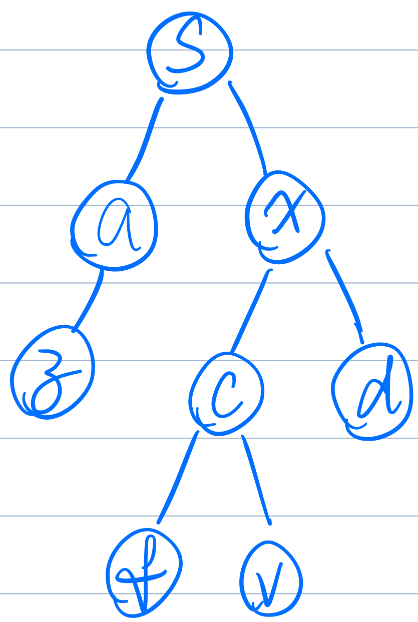
○ : vertices currently in Q ; "unexplored"

○ : vertices out of Q already; "finished"

Coloring kinda overkill here, could use a bool "visited".
But will be useful in the future...

* Sometimes also colored as ○ , ● , ●

This gives us a <u>BFS Tree</u> with root s.



<u>Runtime</u>: $\Theta(|V|+|E|)$

Correctness:

Key Lemma: For any u, v, if u added to Q before v, $\text{dist}(u) \leq \text{dist}(v)$.

Proof: Loop Invariant:

① For any two vertices v_1, v_2 currently in Q,

$$\text{dist}(v_1) \leq \text{dist}(v_2) + 1$$

② For any "discovered" (○ / ○) v_1, v_2,

if v_1 pushed into Q before v_2,

($\Leftrightarrow$ v_1 popped out before v_2.

$\Leftrightarrow$ v_1 "finished" before v_2),

then $\text{dist}(v_1) \leq \text{dist}(v_2)$

Init: ✓

Maintenance:

① By invariant ②, all vertices in Q are in increasing order of their dist $\Rightarrow$ their dist is at least $\text{dist}(u)$, but at most $\text{dist}(u)+1$. The new v's we add into Q have dist of $\text{dist}(u)+1$. So everything in Q has $\text{dist}(u)$ or $\text{dist}(u)+1$.

② If not, that means when we're discovering v (adding it to Q), there is a vertex v' already discovered with $dist(v') > dist(v) = dist(u)+1$.

i) if v' added to Q before u,

$dist(v') \leqslant dist(u)$, contradiction!

ii) if v' added after u,

that means v' still in Q! By ①,

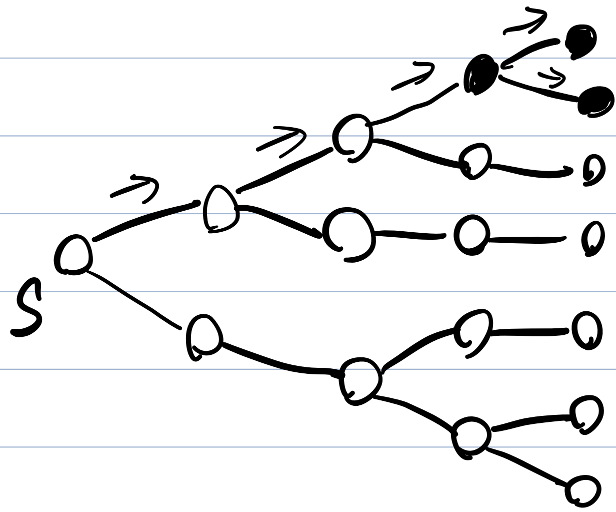
$dist(u) \leq dist(v') \leq dist(u)+1$, contradiction!

Termination: For any u, v, if u added to Q before v, $dist(u) \leqslant dist(v)$.

For full proof of correctness, see CLRS §20.2. (optional)

2. Depth First Search (DFS)

Idea: Pick an edge, explore everything in that subtree, then move on to the next edge.



Solving a maze using DFS:

https://youtu.be/RaQDROsFQoY

- When at a crossroad, put down a marker to mark the route you came from
- Explore the left most route (recursively)
- Explore the second left route...
- ...
- After exploring all routes, go back down the route you came from (backtrack)

Pseudo-Code (Ignore the red lines for now)

```
DFS-Visit(G, u)
    time += 1
    disc(u) = time
    color(u) = YELLOW
    For all v ∈ Adj[u]:
        If color(v) = WHITE:
            parent(v) = u
            DFS-Visit(G, v)
    color(u) = GREEN
    time += 1
    finish(u) = time
```

```
DFS(G)
    time = 0
    For all u ∈ V:
        color(u) = WHITE
        parent(u) = ⊥
    For all u ∈ V:
        If color(u) = WHITE:
            DFS-Visit(G, u)
```

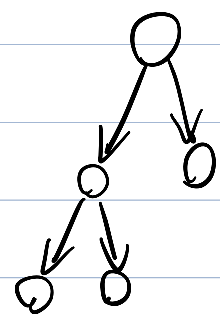
* We usually use DFS to learn something on the entire graph, so we run it on every vertex

O: Not discovered

O: Discovered, but not fully explored

O: Fully explored

Gives a <u>DFS tree</u>.



Runtime: - DFS: $\Theta(|V|)$ excluding calls to DFS-Visit

- One DFS-Visit call on each vertex, and it takes time $|Adj[u]|$. Total?

$$\sum_{u \in V} |Adj[u]| = \Theta(|E|)$$

$$\Rightarrow \Theta(|V| + |E|) \text{ runtime}$$

Correctness: Trivial to see we indeed visit every node.

Depth-First: ✓ line 4-7 in DFS-Visit

Show other properties later.

Magic Time!!

~~DBFS~~ DFS(G, s):
1. For all $v \in V$:
2. visited(v) = FALSE
3. parent$(v) = \perp$
4. $\mathcal{L}$ = ~~Queue~~ Stack. Init( )
5. explore$(G, \mathcal{L}, s)$
6. While $\mathcal{L}$ is not empty:
7. ① $(u, v) = \mathcal{L}$.pop( )
8. If !visited(v):
9. parent$(v) = u$
10. ② explore(v)

explore$(G, \mathcal{L}, v)$:
1. ② visited[v] = TRUE
2. For $w \in$ Adj[v]:
3. $\mathcal{L}$.push((v, w))

Differences from BFS above:

① Queue contains edges instead of vertices

② Only mark as visited when we add all its neighbors

Magic Trick: change BFS to DFS

Just use Stack (FILO)

instead of Queue (FIFO)

- Essentially, the stack simulates the recursion
- This trick does NOT work on all BFS implementations
  * Alg. needs to be carefully crafted (hence the red changes)
  * e.g. replacing Q with S on the old BFS does not work