Breadth-First Search

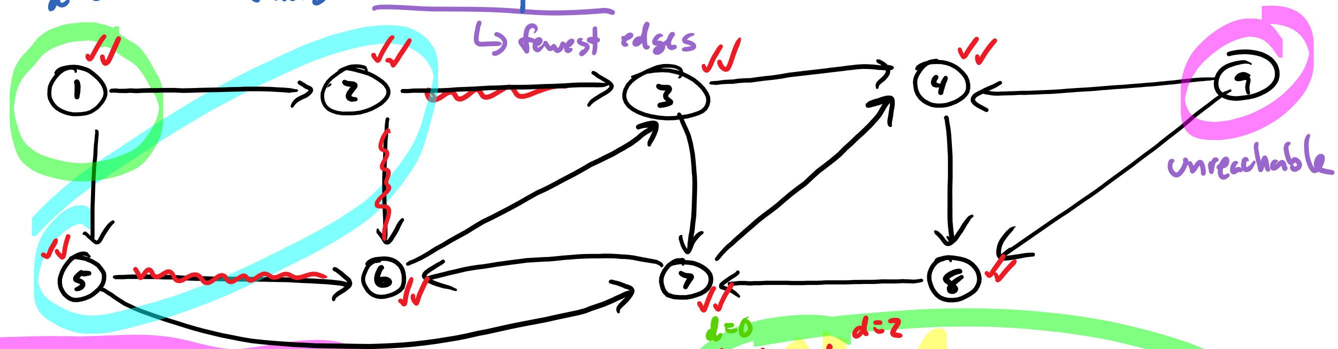
find which verts are reachable from start

finds shortest paths to reachable verts

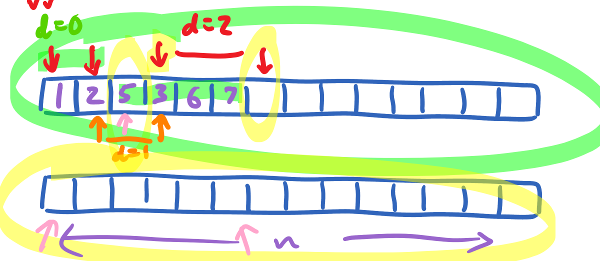
↳ fewest edges

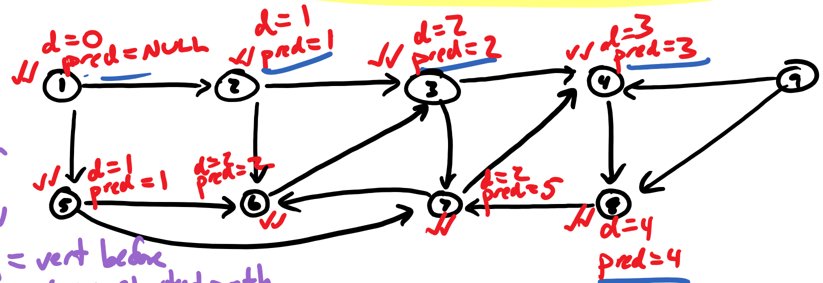
✓ : know shortest

✓✓ : done



0: 1
1: 2 5
2: 3 6 7
3: 4
4: 8
5:
6:
7:
8:






```
d[v] ← unseen for all v
Q ← [start]
color[start] ← in queue
d[start] ← 0
pred[start] ← NULL
while Q not empty
    u ← dequeue(Q)
    for each outneighbor v of u
        if color[v] == unseen
            enqueue(Q, v)
            color[v] ← in queue
            pred[v] ← u
            d[v] ← d[u] + 1
    color[u] ← done ✓✓
```

examine (u,v)

d[v] = # edges on shortest start ⇝ v

pred[v] = vert before v on shortest path

once per vertex (as u) → 1

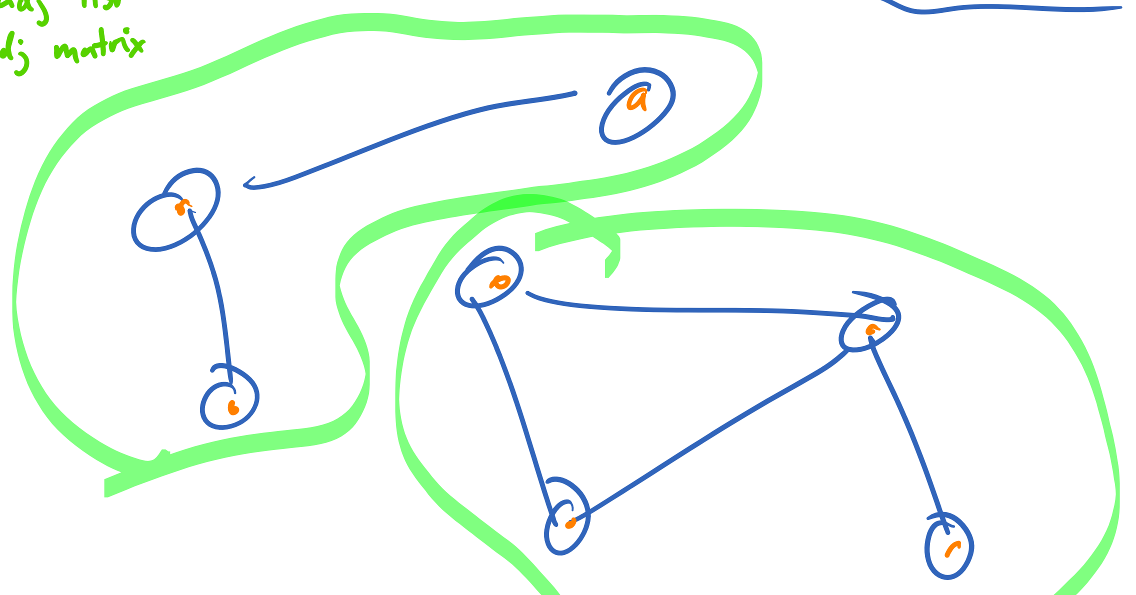
O(n+m) for adj list

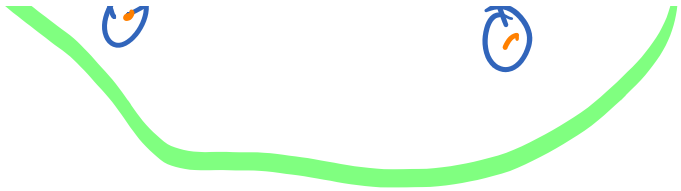
O(n²) for adj matrix

~~1~~ ~~2~~ ~~5~~ ~~3~~ ~~6~~ ~~7~~ ~~4~~ ~~8~~

d=0 d=1 d=2 d=3 d=4

1 2 3 4 8





Depth-First Search

finds all verts reachable from start
detects cycles

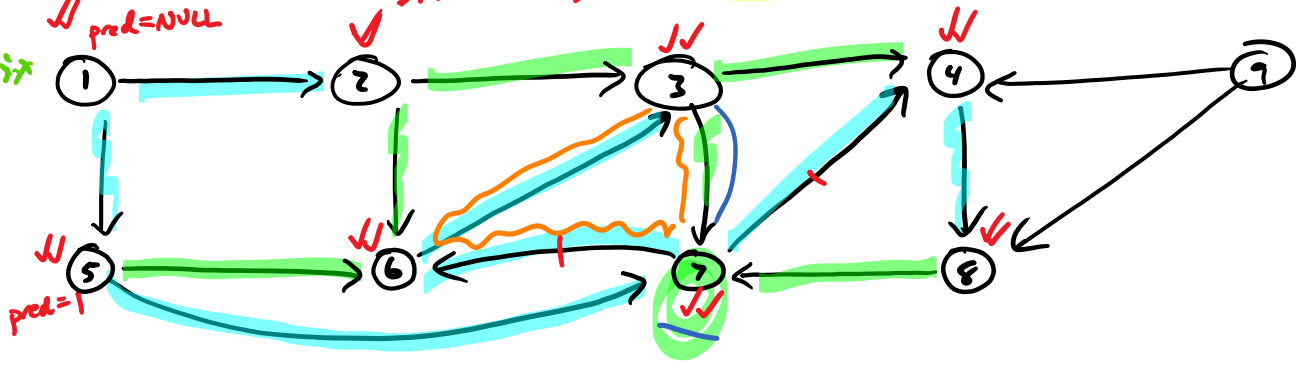
DFS-VISIT(u)
mark u as processing ✓

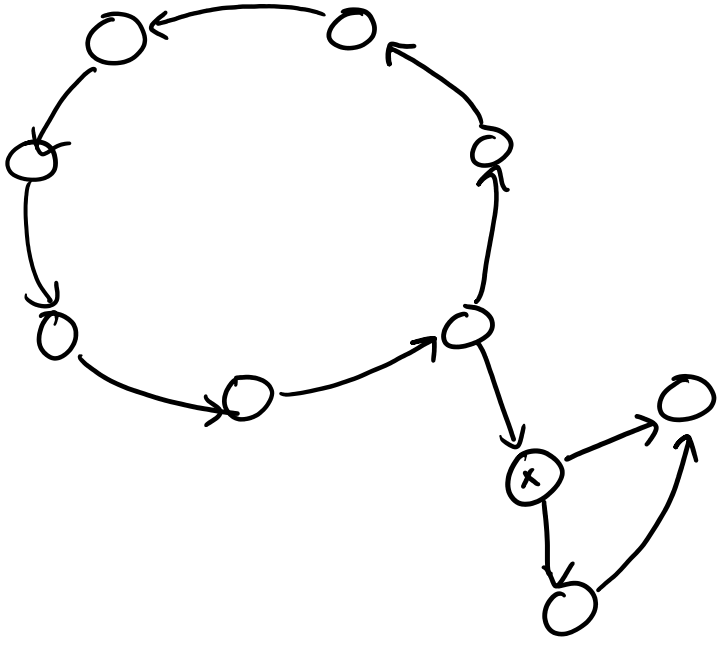
- for each neighbors v of the current vertex u
  - if v still unseen then DFS-VISIT(v)
  - if v processing then cycle!
- mark u as finished

at most one call per vertex

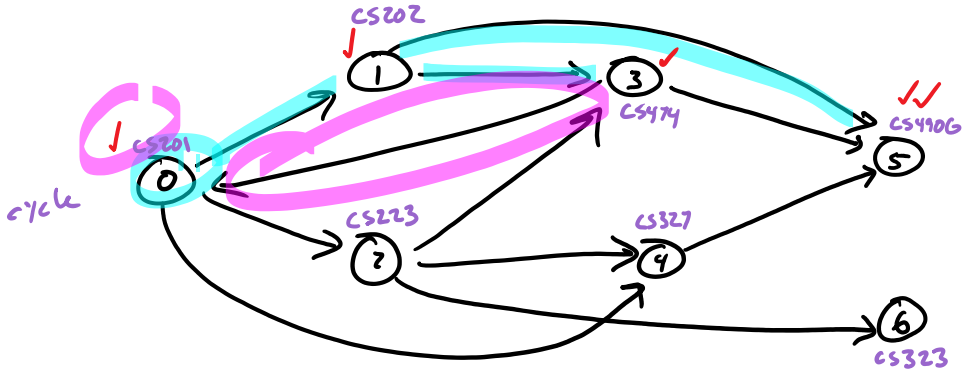
DFS-VISIT(1)
DFS-VISIT(2)
DFS-VISIT(5)
DFS-VISIT(7)
DFS-VISIT(6)
DFS-VISIT(4)
DFS-VISIT(8)
DFS-VISIT(2)

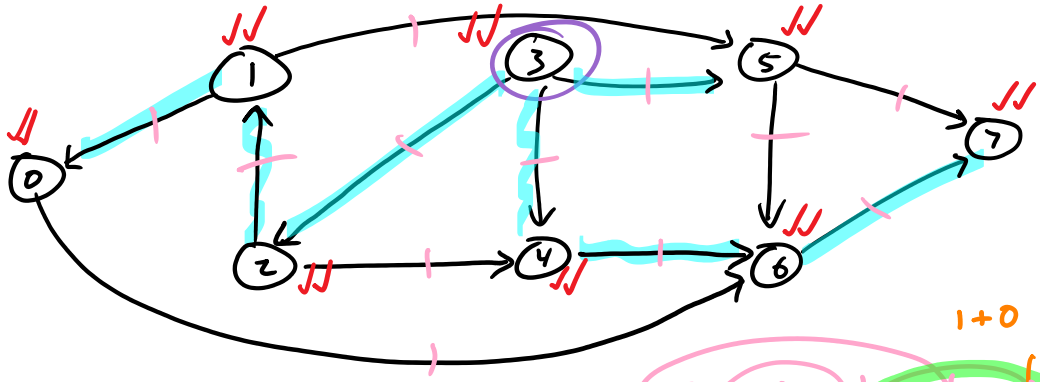
O(n+m) for adj list
O(n²) for adj matrix



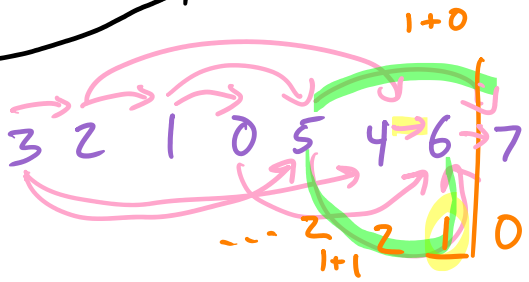


Topological Sort





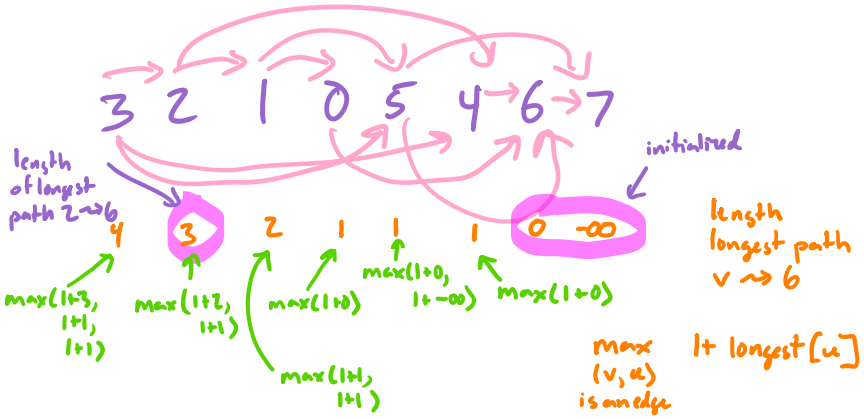
topological sort
(run DFS, record vertices in reverse finish)



longest path starting @ vertex

for each v in reverse order of topo sort
long[v] ← max (v,u) is an edge 1 + long[u]

max = 0 if no edges


3 2 1 0 5 4 6 7
length of longest path 2 ⇝ 6
initialized
length longest path v ⇝ 6
4 3 2 1 1 1 0 -∞
max(1+3, 1+1, 1+1)
max(1+2, 1+1)
max(1+1, 1+1)
max(1+0)
max(1+0, 1+-∞)
max(1+0)
max (v, u) is an edge
1+ longest[u]