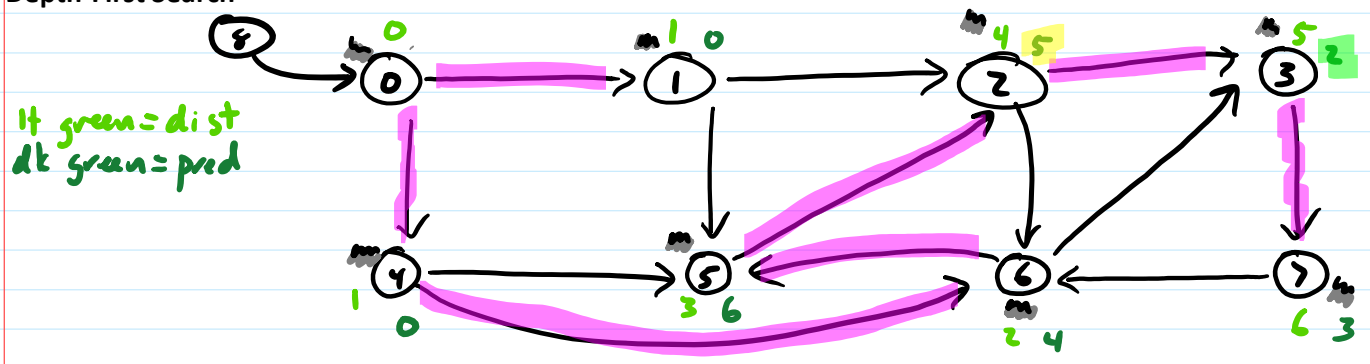
## Depth-First Search




Depth-first search: Keep following edges from current vertex
Backtrack when no edges to unvisited vertices

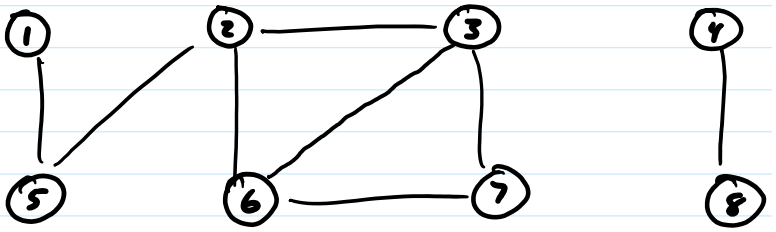
path 0 ⇝ 7 [0 4 6 5 2 3] 7

subpath 0⇝2, next-to-last is pred[7] = 5

subpath is path 0⇝3, next-to-last on that subpath is pred[3] = 2

```
DFS-VISIT(G,u)
```
← called at most once per vertex

```
  for each v adjacent to u
     if (color[v] = WHITE)
        pred[v] = u
        dist[v] = dist[u] + 1
        color[v] <- GRAY
        DFS-VISIT(G,v)

  color[u] <- BLACK
```

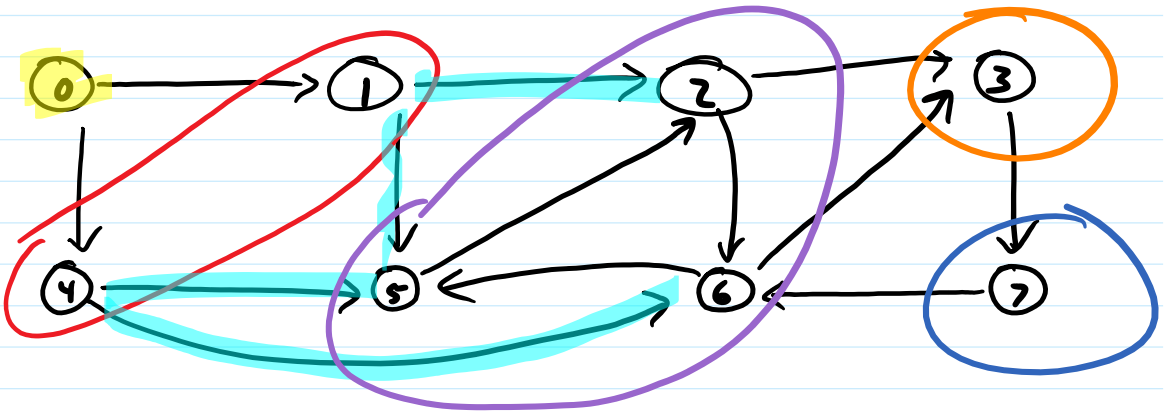



so just like for each vertex u
for each edge (u,v)
do something

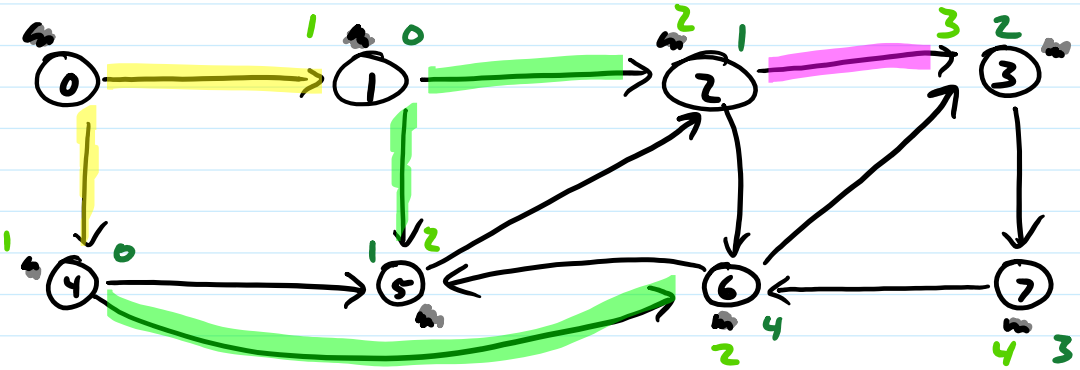
$O(n+m)$

```
DFS(G)
   for each u in G.V
      color[u] <- WHITE

   time <- 0

   for each u in G.V
      if color[u] = WHITE
         DFS-VISIT(G,u)
```

**Breadth-First Search**




starting from s
find verts 1 edge from s
find verts 2 edges from s
3 edges
4




FIFO queue →

dist 0 | dist 1 | dist 2 | dist 3 | dist 4

0 1 4 2 5

```
BFS(V,E,s)

for each vertex u in V
   color[u] <- WHITE
   d[u] <- infinity
   pred[u] <- NULL

F <- []

color[s] <- GRAY
d[s] <- 0
pred[s] <- NULL
Q <- [s]

while not Q.isEmpty()
   u <- Q.dequeue()
   for each v adjacent to u
        pred[v] <- u
        d[v] <- d[u] + 1
        color[v] <- GRAY
        Q.enqueue(v)
   color[u] = BLACK
   F = F + u
```

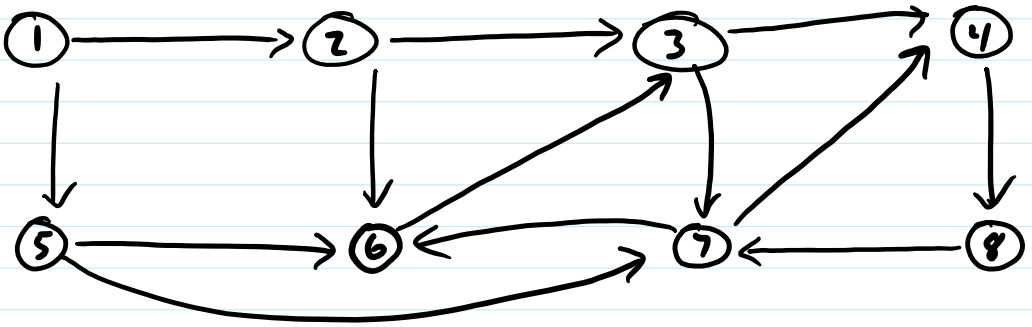
Θ(1)

Θ(1)

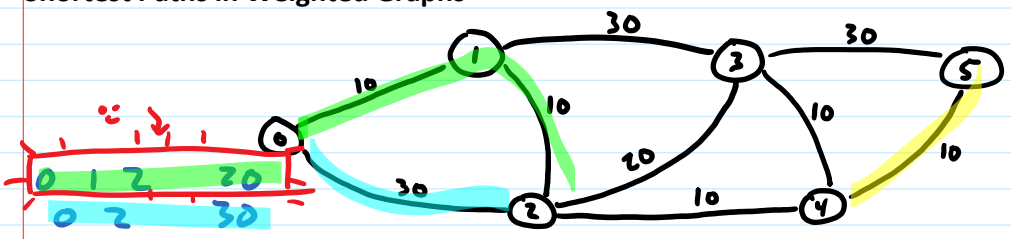
← iterates at most once for each vertex

so just like for each vertex u
for each edge (u,v)
do something

O(n+m)



## Shortest Paths in Weighted Graphs




adjacency list

0 : (1, 10), (2, 30)
1 : (0, 10), (2, 10), (3, 30)
2 : (0, 30), (1, 10), (4, 10), (3, 20)
3 : (1, 30), (5, 30), (4, 10), (2, 70)
4 : (3, 10), (2, 10), (5, 10)
5 : (3, 30), (4, 10)

Shortest Path 0 → 5
↳ least total weight

| | | | |
|---|---|---|---|
| | 0:00 | M → 1 (0:10) | J → 2 (0:30) |
| M arrives 1 | 0:10 | A → 3 (0:40), | R → 2 (0:20) ~~→ 0~~ |
| R arrives 2 | 0:20 | S → 3 (0:40), | L → 4 (0:50) |

times go with cities
↓
for each city, remember time latest messenger arrives