Graphs - II

Breadth-first search → used to search in a graph

Given a graph $G=(V,E)$, and a source vertex S, BFS systematically explores the edges of G to "discover" every vertex that is reachable from S.

- ↳ It computes distance from s to each reachable vertex in G.
- ↳ It also produces a breadth-first tree with root S that contain all the reachable vertices.
- ↳ For any vertex v reachable from S, the simple path in the breadth-first tree from S to v corresponds to a shortest path from S to v in G. (i.e, a path containg least number of edges)
- ↳ works for both directed & undirected graphs

- BFS algorithm colors each vertex: WHITE, GRAY, and BLACK
  - ↳ Initially every vertex is WHITE
  - ↳ A discovered vertex becomes GRAY
  - ↳ A GRAY vertex whose edges are explored becomes BLACK.
- Initally all vertices are set to WHITE
- All GRAY vertices are kept in a FIFO queue
- BFS assumes that the given graph $G=(V,E)$ is represented by an Adjacency list, the queue is denoted by Q, and it contains the following attributes to v:
  - ↳ v.color : WHITE, GRAY, or BLACK
  - ↳ v.d holds the distance form s to v
  - ↳ $v.\pi$ is v's predecessor in the breadth-first tree. If v has no predecessor then $v.\pi = NIL$ (v beig source or undiscovered)

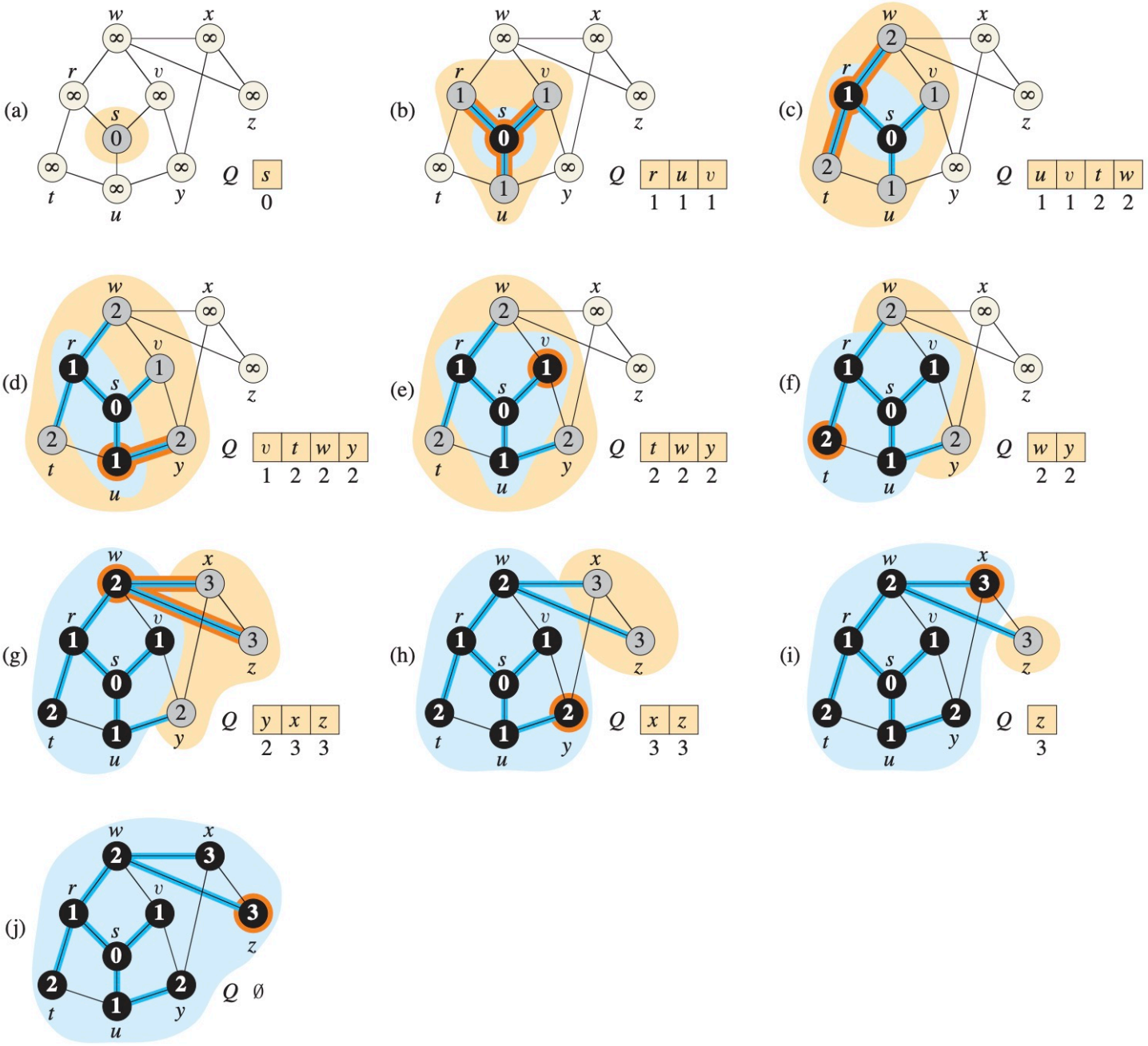
BFS (G, s)

```
for each vertex u ∈ G.V - {s}
    u.color = WHITE
    u.d     = ∞
    u.π     = NIL
s.color = GRAY
s.d     = 0
s.π     = NIL
Q = ∅
Enqueue (Q, s)
while Q ≠ ∅ ──→ NIL
    u = Dequeue (Q)
    for each vertex v in G.Adj[u]
        if v.color == WHITE
            v.color = GRAY
            v.d     = u.d + 1
            v.π     = u
            Enqueue (Q, v)
    u.color = BLACK
```

Analysis:

- Enque, Dequee ⟶ $O(1)$
- Total time for queue ops. ⟶ $O(V)$
- Since sum of all $|V|$ adjacency lists is $\theta(|E|)$, and overhead is $O(|V|)$, Total complexity = $O(|V|+|E|)$

Example:


(a)
(b)
(c)
(d)
(e)
(f)
(g)
(h)
(i)
(j)
Q
s
0
r u v
1 1 1
u v t w
1 1 2 2
v t w y
1 2 2 2
t w y
2 2 2
w y
2 2
y x z
2 3 3
x z
3 3
z
3
Ø

Depth First Search:

↳ Searches deeper in the graph whenever possible.

↳ Explores edges out of the most recently discovered vertex v that still has undiscovered edges leaving it.

↳ Once all of v's edges have been explored, the search back tracks to explore edges leaving the vertex from which v was discovered

Predecessor subgraph:

$G_\pi = (V, E_\pi)$, where

$$E_\pi = \{ (v.\pi, v) : v \in V \text{ and } v.\pi \neq \text{NIL} \}$$

↳ Depth-first search produces a depth-first forest comprising of several depth-first trees

↳ Edges in E_π are tree-edges

→ Depth-first search also timestamps each vertex

each vertex has two timestamps

① $v.d$ ⇒ when v is first discovered
↳ Grayed out

② $v.f$ ⇒ when search finishes examining v's adjacency list
↳ Blacked out

for every vertex u, $u.d < u.f$

- Before time $u.d$ ⟶ vertex u is WHITE
- Between $u.d$ and $u.f$ ⟶ vertex u is GRAY
- After time $u.f$ ⟶ vertex u is BLACK

```
DFS (G)
for each vertex u ∈ G.V
    u.color = WHITE
    u.π     = NIL
time = 0    // Global variable
for each vertex u ∈ G.V
    if u.color == WHITE
        DFS-VISIT (G,u)
```

```
DFS - VISIT (G,u)
time = time + 1
u.d = time    // vertex u is discovered
u.color = GRAY
for each vertex v in G.Adj [u]
    if v.color == WHITE
        v.π = u
        DFS-VISIT (G,v)
time = time + 1
u.f = time
u.color = BLACK
```

- Every call to DFS-VISIT(G, u), makes u the root of a new tree in the depth-first forest.

- Result of depth-first search may depend on the order of selecting vertex u for V & on the order of vertices in the adjacency list.

- Complexity : $\Theta(|V| + |E|)$