BFS

- one of the **simplest** algorithms for searching a graph.
- It is the **archetype** for many important graph algorithms. **Eg:** Prim's minimum-spanning-tree algorithm and Dijkstra's single-source shortest-paths algorithm use ideas similar to those in breadth-first search.
- Given a graph G = (V, E) and a distinguished source vertex s, breadth-first search systematically explores the edges of G to "discover" every vertex that is **reachable** from s.

- Why is it named BFS? the algorithm discovers all vertices at distance k from s before discovering any vertices at distance k + 1. it expands the frontier between discovered and undiscovered vertices uniformly across the breadth of the frontier.
- To keep track of progress, breadth-first search colors each vertex white, gray, or black.
- All vertices start out white and may later become gray and then black.
- A vertex is discovered the first time it is encountered during the search, at which time it becomes nonwhite. Gray and black vertices, therefore, have been discovered, but breadth-first search distinguishes between them to ensure that the search proceeds in a breadth-first manner.
- If (u, v) ∈ E and vertex u is black, then vertex v is either gray or black; that is, all vertices adjacent to black vertices have been discovered. Gray vertices may have some adjacent white vertices; they represent the frontier between discovered and undiscovered vertices.
- Breadth-first search constructs a breadth-first tree, initially containing only its root, which is the source vertex *s*. Whenever a white vertex v is discovered in the course of scanning the adjacency list of an already discovered vertex u, the vertex v and the edge (u, v) are added to the tree. We say that u is the **predecessor** or parent of v in the breadth-first tree (Since a vertex is discovered at most once, it has at most one parent).

- The breadth-first-search procedure BFS below assumes that the input graph $G = (V, E)$ is represented **using adjacency lists**.
- The algorithm works on both directed and undirected graphs.

- It maintains several additional data structures with each vertex in the graph.

  The color of each vertex $u \in V$ is stored in the variable ***color*[*u*],** and the predecessor of u is stored in the variable $\boldsymbol{\pi}$**[*u*].** If u has no predecessor (for example, if u = s or u has not been discovered), then $\pi[u] = NIL$.
  The distance from the source s to vertex u computed by the algorithm is stored in **d[u].**

```
BFS(G, s)
 1  for each vertex u ∈ V[G] − {s}
 2       do color[u] ← WHITE
 3          d[u] ← ∞
 4          π[u] ← NIL
 5  color[s] ← GRAY
 6  d[s] ← 0
 7  π[s] ← NIL
 8  Q ← ∅
 9  ENQUEUE(Q, s)
10  while Q ≠ ∅
11       do u ← DEQUEUE(Q)
12          for each v ∈ Adj[u]
13               do if color[v] = WHITE
14                    then color[v] ← GRAY
15                         d[v] ← d[u] + 1
16                         π[v] ← u
17                         ENQUEUE(Q, v)
18          color[u] ← BLACK
```

Working:

Lines 1–4 paint every vertex white, set d[u] to be infinity for each vertex u, and set the parent of every vertex to be NIL. Line 5 paints the source vertex s gray, since it is considered to be discovered when the procedure begins. Line 6 initializes d[s] to 0, and line 7 sets the predecessor of the source to be NIL. Lines 8–9 initialize Q to the queue containing just the vertex s.

The while loop of lines 10–18 iterates as long as there remain gray vertices, which are discovered vertices that have not yet had their adjacency lists fully examined. This while loop maintains the following invariant:

"At the test in line 10, the queue *Q* consists of the set of gray vertices"

The for loop of lines 12–17 considers each vertex v in the adjacency list of u. If v is white, then it has not yet been discovered, and the algorithm discovers it by executing lines 14–17. It is first grayed, and its distance d[v] is set to d[u] + 1. Then, u is recorded as its parent. Finally, it is placed at the tail of the queue Q.
When all the vertices on u's adjacency list have been examined, u is blackened in line 18.

Note:
The results of breadth-first search may depend upon the order in which the neighbors of a given vertex are visited in line 12: the breadth-first tree may vary, but the distances d computed by the algorithm will not.

Running Time:

- After initialization, no vertex is ever whitened, and thus the test in line 13 ensures that each vertex is enqueued at most once, and hence dequeued at most once. The operations of enqueuing and dequeuing take O(1) time, so the total time devoted to queue operations is O(V).

- Because the adjacency list of each vertex is scanned only when the vertex is dequeued, each adjacency list is scanned at most once. Since the sum of the lengths of all the adjacency lists is $\Theta(E)$, the total time spent in scanning adjacency lists is O(E).

- The overhead for initialization is O(V), and thus the total running time of BFS is O (V + E). Thus, breadth-first search runs in time linear in the size of the adjacency-list representation of G.

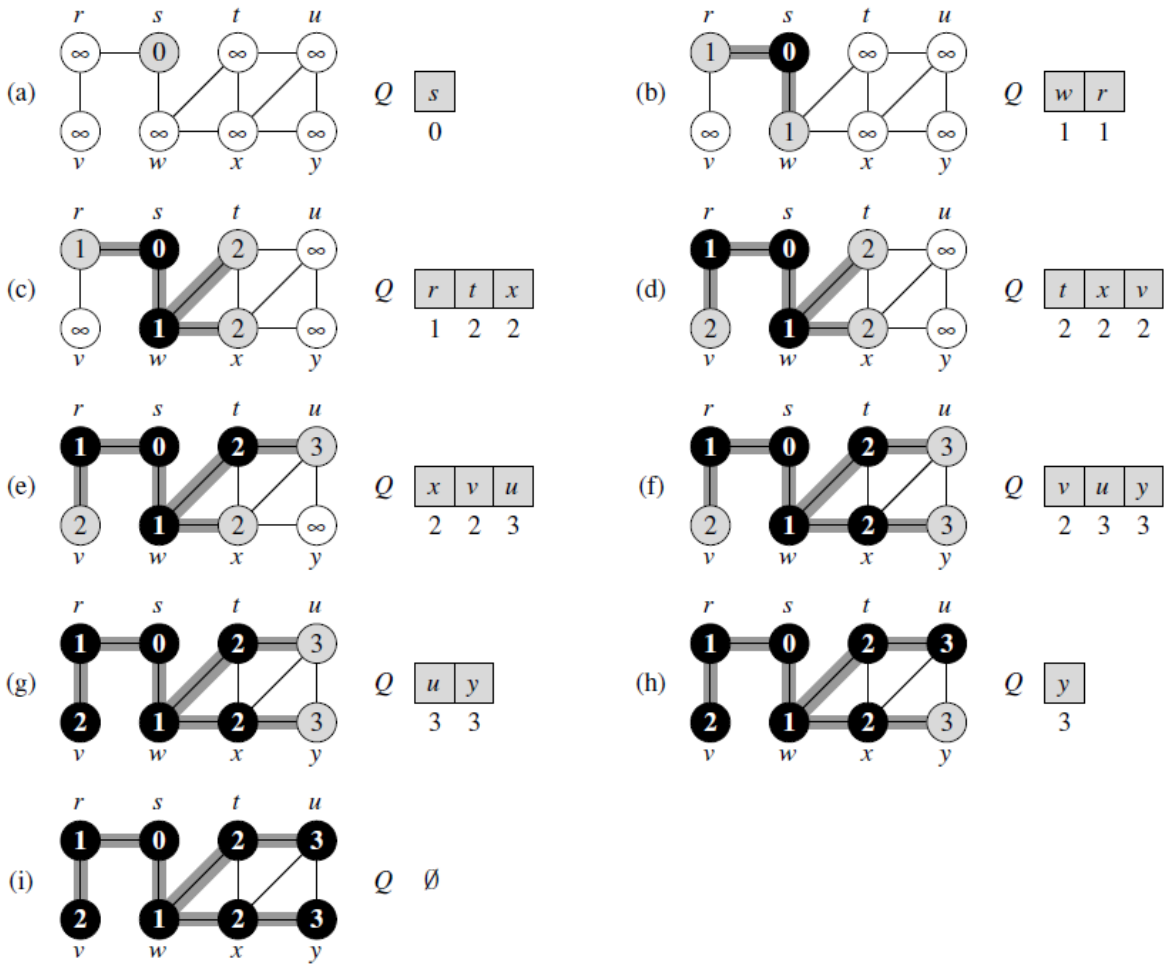
shortest-path distance:

- the shortest-path distance $\delta(s, v)$ from s to v as the minimum number of edges in any path from vertex s to vertex v; if there is no path from s to v, then $\delta(s, v)=\infty$.

- A path of length $\delta(s, v)$ from s to v is said to be a shortest path from s to v.

- It should be noted that the breadth-first search computes shortest-path distances.

Example:



Theorems & Lemmas:

1. Let $G = (V, E)$ be a directed or undirected graph, and let $s \in V$ be an arbitrary vertex. Then, for any edge $(u, v) \in E$,

$$\delta(s, v) \leq \delta(s, u) + 1$$

(property of shortest-path distances)

2. Let G = (V, E) be a directed or undirected graph, and suppose that BFS is run on G from a given source vertex s ∈ V. Then upon termination, for each vertex v ∈ V, the value d[v] computed by BFS satisfies $d[v] \geq \delta(s, v)$.

(d[v] bounds δ(s, v) from above.)

3. Suppose that during the execution of BFS on a graph G = (V, E), the queue Q contains the vertices $< v_1, v_2, \ldots, v_r >$, where v_1 is the head of Q and v_r is the tail.
Then, $d[v_r] \leq d[v_1] + 1$ and $d[v_i] \leq d[v_{i+1}]$ for $i = 1, 2, \ldots, r - 1$.

(at all times, there are at most two distinct d values in the queue.) ($d[v] = \delta(s, v)$)

4. Suppose that vertices *vi* and *vj* are enqueued during the execution of BFS, and that *vi* is enqueued before *vj*. Then $d[vi] \leq d[vj]$ at the time that *vj* is enqueued.

(It shows that the *d* values at the time that vertices are enqueued are monotonically increasing over time)

the *predecessor subgraph*

Formally, for a graph $G = (V, E)$ with source *s*, we define the ***predecessor subgraph*** of *G* as $G\pi = (V\pi, E\pi)$, where
$V\pi = \{v \in V : \pi[v] \neq \text{NIL}\} \cup \{s\}$ and
$E\pi = \{(\pi[v], v) : v \in V\pi - \{s\}\}$

The predecessor subgraph $G\pi$ is a ***breadth-first tree*** if $V\pi$ consists of the vertices reachable from *s* and, for all $v \in V\pi$, there is a unique simple path from *s* to *v* in $G\pi$ that is also a shortest path from *s* to *v* in *G*. The edges in $E\pi$ are called ***tree edges***.

5. When applied to a directed or undirected graph G = (V, E), procedure BFS constructs π so that the predecessor subgraph $G\pi = (V\pi, E\pi)$ is a breadth-first tree.