Breadth-First Search Algorithm: BFS

BFS:

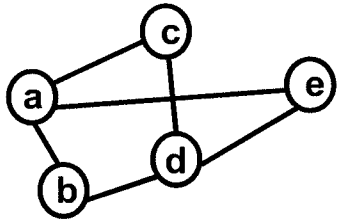
- assumes that $G = (V, E)$ is represented using an adjacency list.
- uses a first-in-first-out (FIFO) queue, Q, to manage the traversal of vertices.
- keeps track of the color of a vertex $u \in V$ using $color[u]$. If u is WHITE, it has not been discovered; if GRAY, it has been put on Q; if BLACK, its successors have been added to Q.
- keeps track of the distance between the source vertex s and $u \in V$ in $d[u]$.
- uses $\pi[u]$ to store the predecessor of u in order to construct a BFS-Tree. If there is no predecessor, the value of $\pi[u]$ is NIL.

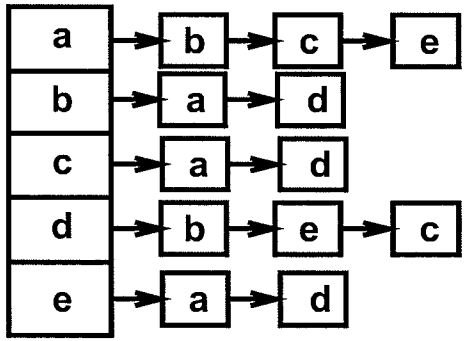
```
BFS(G, s)
1. for each vertex u ∈ V[G] − {s}
2.       do color[u] ← WHITE
3.          d[u] ← ∞
4.          π[u] ← NIL
```

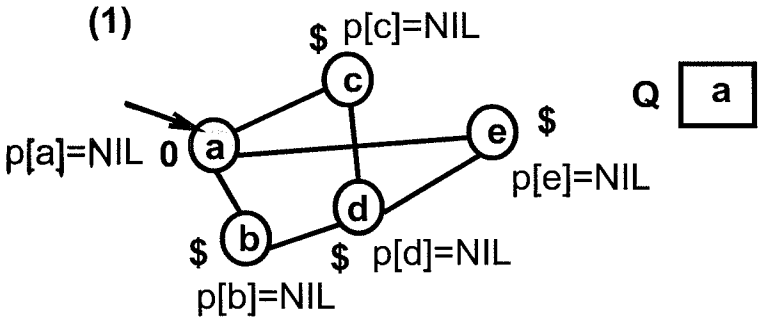
Breadth-First Search Algorithm: BFS *continued*

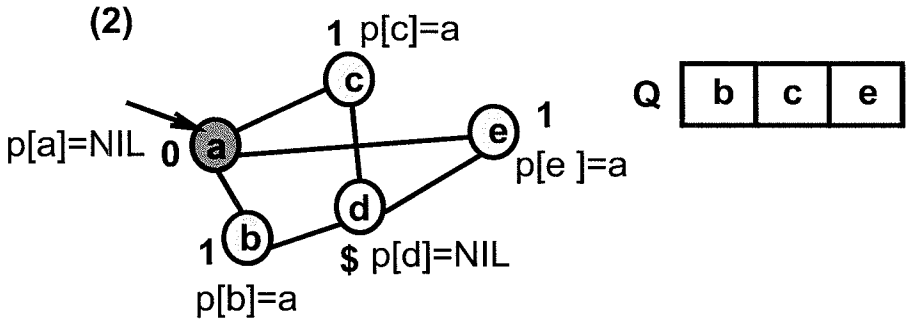
```
5.  color[s] ← GRAY
6.  d[s] ← 0
7.  π[s] ← NIL
8.  Q ← {s}
9.  while Q ≠ ∅
10.       do u ← head[Q]
11.          for each v ∈ Adj[u]
12.             do if color[v] = WHITE
13.                   then color[v] ← GRAY
14.                        d[v] ← d[u] + 1
15.                        π[v] ← u
16.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

Example: BFS(G, a)

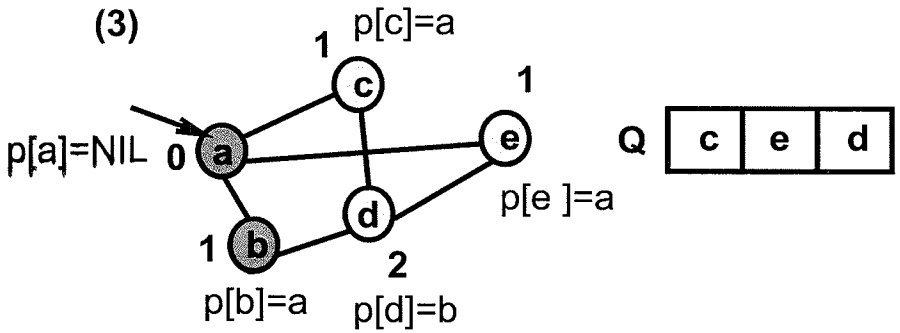


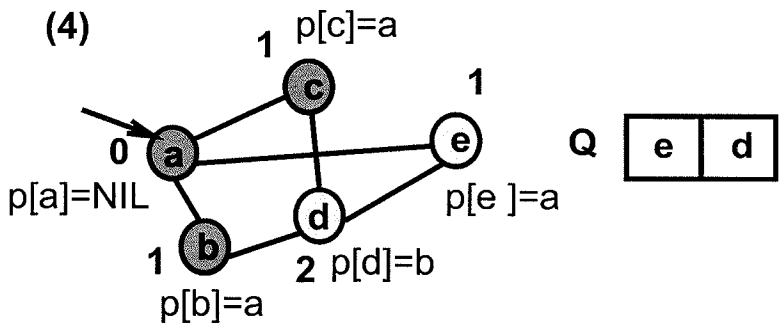


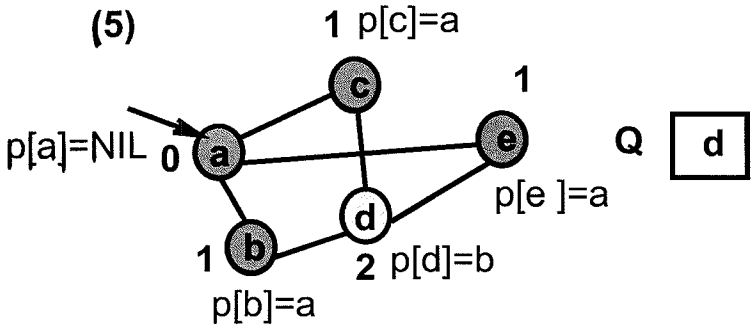


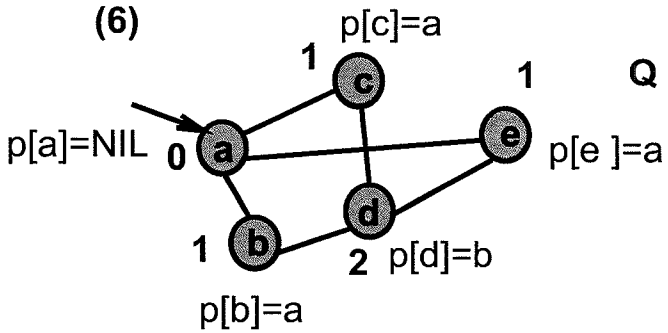


Example: $\mathrm{BFS}(G, a)$ *continued*









Complexity of BFS

- **Initialize:** $|V| - 1$ vertices are initialized to WHITE in $\Theta(V)$.

- **Queue Operations:** Because vertices when enqueued become GRAY and they are never reset to WHITE, vertices are enqueued at most once. Each enqueue and dequeue operation uses $O(1)$ time; hence, all queue operations take $O(V)$ time.

- **Scanning Adjacency Lists:** The lists are scanned at most once, right after dequeuing. The sum of their lengths is $\Theta(E)$. Hence, the time to scan the lists is $O(E)$.

- **Resulting Complexity:** $O(V + E)$

Prim's MST Algorithm

The algorithm uses a queue, Q, to select an edge to add to A. For each vertex v, $key[v]$ keeps the minimum weight of any edge connecting to v from vertices in the tree, and $\pi[v]$ points to the parent node in the tree. Initially, $key[v] = \infty$.

```
MST-PRIM(G, w, r)
1. Q ← V[G]
2. for each vertex u ∈ Q
3.       do key[u] ← ∞
4. key[r] ← 0
5. π[r] ← NIL
6. while Q ≠ ∅
7.          do u ← EXTRACT-MIN(Q)
8.              for each v ∈ Adj[u]
9.                  do if v ∈ Q and w(u, v) < key[v]
10.                       then π[v] ← u
11.                            key[v] ← w(u, v)
```

MST-PRIM **Discussion**

During the course of the algorithm:

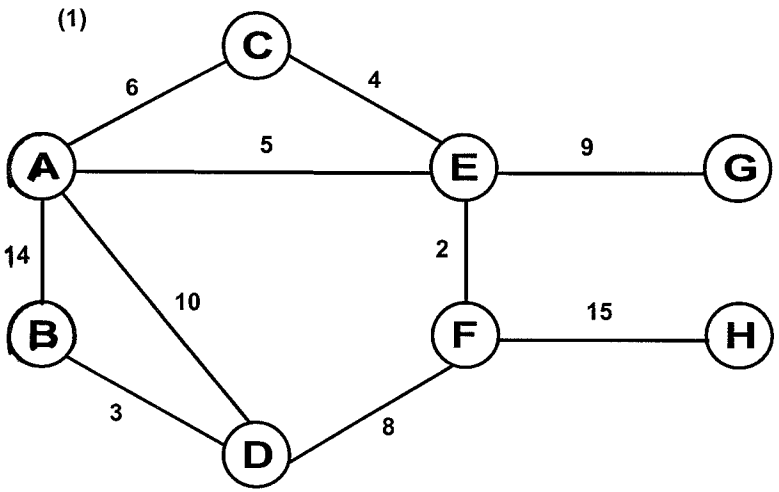
$$A = \{(v, \pi[v]) : v \in V - \{r\} - Q\}$$

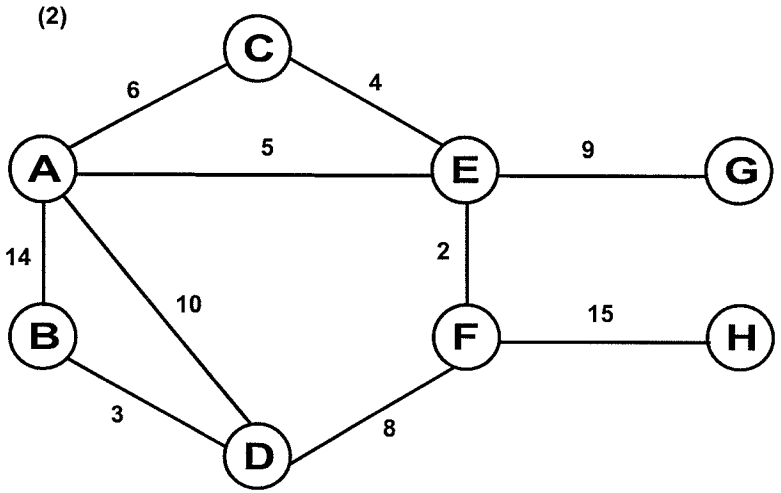
At the end:

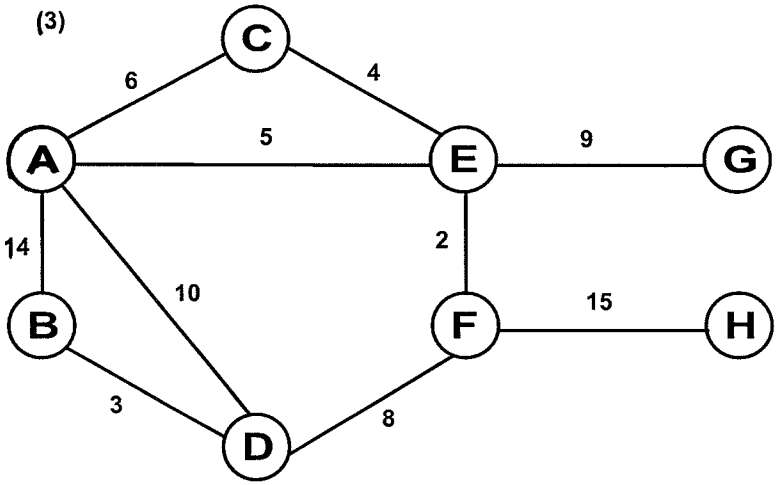
$$A = \{(v, \pi[v]) : v \in V - \{r\}\}$$

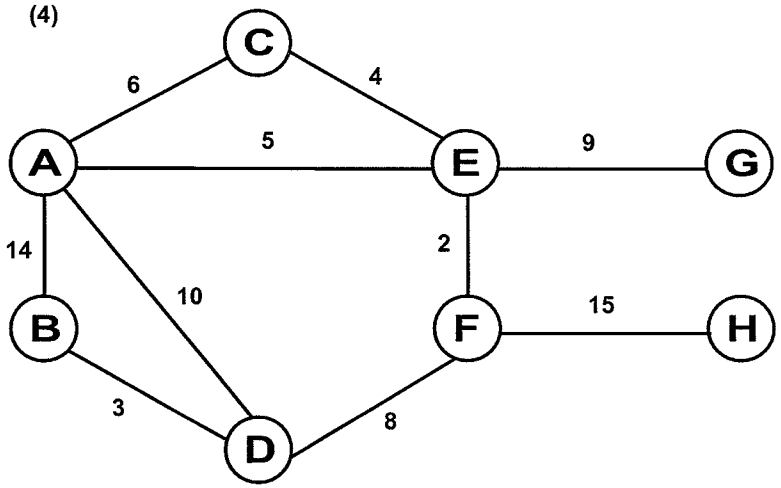
MST-PRIM starts with an arbitrary root r and it loops until it constructs a tree spanning the vertices of V. At each step, a light edge connecting a vertex in A to a vertex in $V - A$ is added to the tree. [redacted] The algorithm will add only edges that are safe for A, so the resulting tree will be an MST. The strategy is clearly a greedy one, since at each step, the tree is augmented with an edge that contributes a minimum amount to the tree's weight.

MST-PRIM **Example**









MST-PRIM **Example** *continued*