DFS Algorithm - Pseudocode

☑DFS named *DEPTHFIRST search*, as it continues the search in the forward (deeper) direction.

☑The algorithm is shown in Algorithm DFS:

Algorithm 8.1 DFS
Input: A (directed or undirected) graph $G = (V, E)$.
Output: Preordering and postordering of the vertices in the corresponding depth-first search tree.

1. *predfn* $\leftarrow 0$; *postdfn* $\leftarrow 0$
2. **for** each vertex $v \in V$
3. mark v *unvisited*
4. **end for**
5. **for** each vertex $v \in V$
6. **if** v is marked *unvisited* **then** *dfs*(v)
7. **end for**

Procedure *dfs*(v)

1. mark v *visited*
2. *predfn* $\leftarrow$ *predfn* $+ 1$
3. **for** each edge $(v, w) \in E$
4. **if** w is marked *unvisited* **then** *dfs*(w)
5. **end for**
6. *postdfn* $\leftarrow$ *postdfn* $+ 1$

Breadth-First Search (BFS)- Pseudocode

Algorithm 8.4 BFS
Input: A directed or undirected graph $G = (V, E)$.
Output: Numbering of the vertices in breadth-first search order.

1. *bfn* $\leftarrow 0$
2. **for** each vertex $v \in V$
3. mark v *unvisited*
4. **end for**
5. **for** each vertex $v \in V$
6. **if** v is marked *unvisited* **then** *bfs*(v)
7. **end for**

Procedure *bfs*(v)

1. $Q \leftarrow \{v\}$
2. mark v *visited*
3. **while** $Q \neq \{\}$
4. $v \leftarrow Pop(Q)$
5. *bfn* $\leftarrow$ *bfn* $+ 1$
6. **for** each edge $(v, w) \in E$
7. **if** w is marked *unvisited* **then**
8. Push(w, Q)
9. mark w *visited*
10. **end if**
11. **end for**
12. **end while**

Algorithm 7.4 PRIM

Input: A weighted connected undirected graph $G = (V, E)$, where $V = \{1, 2, \ldots, n\}$.

Output: The set of edges T of a minimum cost spanning tree for G.

$T \leftarrow \{\}$; $X \leftarrow \{1\}$; $Y \leftarrow V - \{1\}$
for $y \leftarrow 2$ **to** n
    if y adjacent to 1 **then**
        $N[y] \leftarrow 1$
        $C[y] \leftarrow c[1, y]$
    else $C[y] \leftarrow \infty$
    end if
end for
for $j \leftarrow 1$ **to** $n - 1$ {find $n - 1$ edges}
    Let $y \in Y$ be such that $C[y]$ is minimum
    $T \leftarrow T \cup \{(y, N[y])\}$ {add edge $(y, N[y])$ to T}
    $X \leftarrow X \cup \{y\}$ {add vertex y to X}
    $Y \leftarrow Y - \{y\}$ {delete vertex y from Y}
    for each vertex $w \in Y$ that is adjacent to y
        if $c[y, w] < C[w]$ **then**
            $N[w] \leftarrow y$
            $C[w] \leftarrow c[y, w]$
        end if
    end for
end for

Bellman-Ford Algorithm-Pseudocode

```
function bellmanFord(G, S)
  for each vertex V in G
    distance[V] <- infinite
      previous[V] <- NULL
  distance[S] <- 0

  for each vertex V in G
    for each edge (U,V) in G
      tempDistance <- distance[U] + edge_weight(U, V)
      if tempDistance < distance[V]
        distance[V] <- tempDistance
        previous[V] <- U

  for each edge (U,V) in G
    If distance[U] + edge_weight(U, V) < distance[V}
      Error: Negative Cycle Exists

  return distance[], previous[]
```