Algoritmo DFS, Depth-First Search (búsqueda en profundidad).
Búsqueda de caminos en profundidad.

Algorithm 2.3: Graph depth-first search with a stack.

```
StackDFS(G, node) → visited
    Input: G = (V, E), a graph
           node, the starting vertex in G
    Output: visited, an array of size |V| such that visited[i] is TRUE if we
            have visited node i, FALSE otherwise
1   S ← CreateStack()
2   visited ← CreateArray(|V|)
3   for i ← 0 to |V| do
4       visited[i] ← FALSE
5   Push(S, node)
6   while not IsStackEmpty(S) do
7       c ← Pop(s)
8       visited[c] ← TRUE
9       foreach v in AdjacencyList(G, c) do
10          if not visited[v] then
11              Push(S, v)
12  return visited
```

Algoritmo BFS, Breadth-First Search (búsqueda en amplitud).
Camino más cortos en grafos con aristas sin peso

Algorithm 2.5: Graph breadth-first search.

```
BFS(G, node) → visited
    Input: G = (V, E), a graph
           node, the starting vertex in G
    Output: visited, an array of size |V| such that visited[i] is TRUE if we
            have visited node i, FALSE otherwise
1   Q ← CreateQueue()
2   visited ← CreateArray(|V|)
3   inqueue ← CreateArray(|V|)
4   for i ← 0 to |V| do
5       visited[i] ← FALSE
6       inqueue[i] ← FALSE
7   Enqueue(Q, node)
8   inqueue[node] ← TRUE
9   while not IsQueueEmpty(Q) do
10      c ← Dequeue(Q)
11      inqueue[c] ← FALSE
12      visited[c] ← TRUE
13      foreach v in AdjacencyList(G, c) do
14          if not visited[v] and not inqueue[v] then
15              Enqueue(Q, v)
16              inqueue[v] ← TRUE
17  return visited
```

Algoritmo de Dijkstra.
Camino más cortos en grafo con aristas con pesos positivos.

Algorithm 7.1: Dijkstra's algorithm.

```
Dijkstra(G, s) → (pred, dist)
    Input: G = (V, E), a graph
           s, the starting node
    Output: pred, an array of size |V| such that pred[i] is the predecessor
            of node i in the shortest path from s
            dist, an array of size |V| such that dist[i] is the length of the
            shortest path calculated from node s to i
1   pred ← CreateArray(|V|)
2   dist ← CreateArray(|V|)
3   pq ← CreatePQ()
4   foreach v in V do
5       pred[v] ← −1
6       if v ≠ s then
7           dist[v] ← ∞
8       else
9           dist[v] ← 0
10      InsertInPQ(pq, v, dist[v])
11  while SizePQ(pq) ≠ 0 do
12      u ← ExtractMinFromPQ(pq)
13      foreach v in AdjacencyList(G, u) do
14          if dist[v] > dist[u] + Weight(G, u, v) then
15              dist[v] ← dist[u] + Weight(G, u, v)
16              pred[v] ← u
17              UpdatePQ(pq, v, dist[v])
18  return (pred, dist)
```

Algoritmo de Bellman-Ford.
Camino más cortos en grafo con aristas con pesos (puede haber pesos negativos).

Algorithm 8.1: Bellman-Ford.

```
BellmanFord(G, s) → (pred, dist)
    Input: G = (V, E), a graph
           s, the starting node
    Output: pred, an array of size |V| such that pred[i] is the predecessor
            of node i in the shortest path from s
            dist, an array of size |V| such that dist[i] is the length of the
            shortest path calculated from node s to i
1   pred ← CreateArray(|V|)
2   dist ← CreateArray(|V|)
3   foreach v in V do
4       pred[v] ← −1
5       if v ≠ s then
6           dist[v] ← ∞
7       else
8           dist[v] ← 0
9   for i ← 0 to |V| do
10      foreach (u, v) in E do
11          if dist[v] > dist[u] + Weight(G, u, v) then
12              dist[v] ← dist[u] + Weight(G, u, v)
13              pred[v] ← u
14  return (pred, dist)
```