Representing Graphs

Undirected Graph

Directed Graph

Breath-First Search

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

Depth-First Search

DFS(G)
1 for each vertex u ∈ V[G]
2 do color[u] ← WHITE
3 π[u] ← NIL
4 time ← 0
5 for each vertex u ∈ V[G]
6 do if color(u) = WHITE
7 then DFS-Visit(u)

DFS-Visit(u)
1 color[u] ← GRAY  // White vertex u has just been discovered.
2 time ← time + 1
3 d[u] ← time
4 for each v ∈ Adj[u]  // Explore edge (u, v).
5 do if color[v] = WHITE
6 then π[v] ← u
7 DFS-Visit(v)
8 color[u] ← BLACK  // Blacken u; it is finished.
9 f[u] ← time ← time + 1
Properties of Depth-First Search

Topological Sort

A DAG for Topological Sorting

Strongly Connected Components

Strongly Connected Components

STRONGLY-CONNECTED-COMPONENTS(G)
1. call DFS(G) to compute finishing times f[u] for each vertex u
2. compute G^T
3. call DFS(G^T), but in the main loop of DFS, consider the vertices
   in order of decreasing f[u] (as computed in line 1)
4. output the vertices of each tree in the depth-first forest formed in line 3 as a separate strongly connected component

TOPOLOGICAL-SORT(G)
1. call DFS(G) to compute finishing times f[u] for each vertex u
2. as each vertex is finished, insert it onto the front of a linked list
3. return the linked list of vertices