22.3 Depth First Search

Depth-First Search(DFS) always visits a neighbour of the most recently visited vertex with an unvisited neighbour.

This means the search moves “forward” when possible, and only “backtracks” when moving forward is impossible.

Sometimes DFS is also referred to as the “backtracking search”.

DFS can be written similarly to BFS, but using a stack rather than a queue. This stack can either by represented explicitly (by a stack data-type in our language) or implicitly when using recursive functions.

As with BFS, vertices are colored WHITE, GRAY, or BLACK during the search, with the same meanings.

22.3 Depth First Search (continued)

As well as $v.color$ and $v.\pi$, the search defines:

- *time* : a global clock with values $0, 1, 2, \ldots$
- $v.d$: discovery time (when v is first visited)
- $v.f$: finishing time (when all the neighbours of v have been examined)

DFS(G)

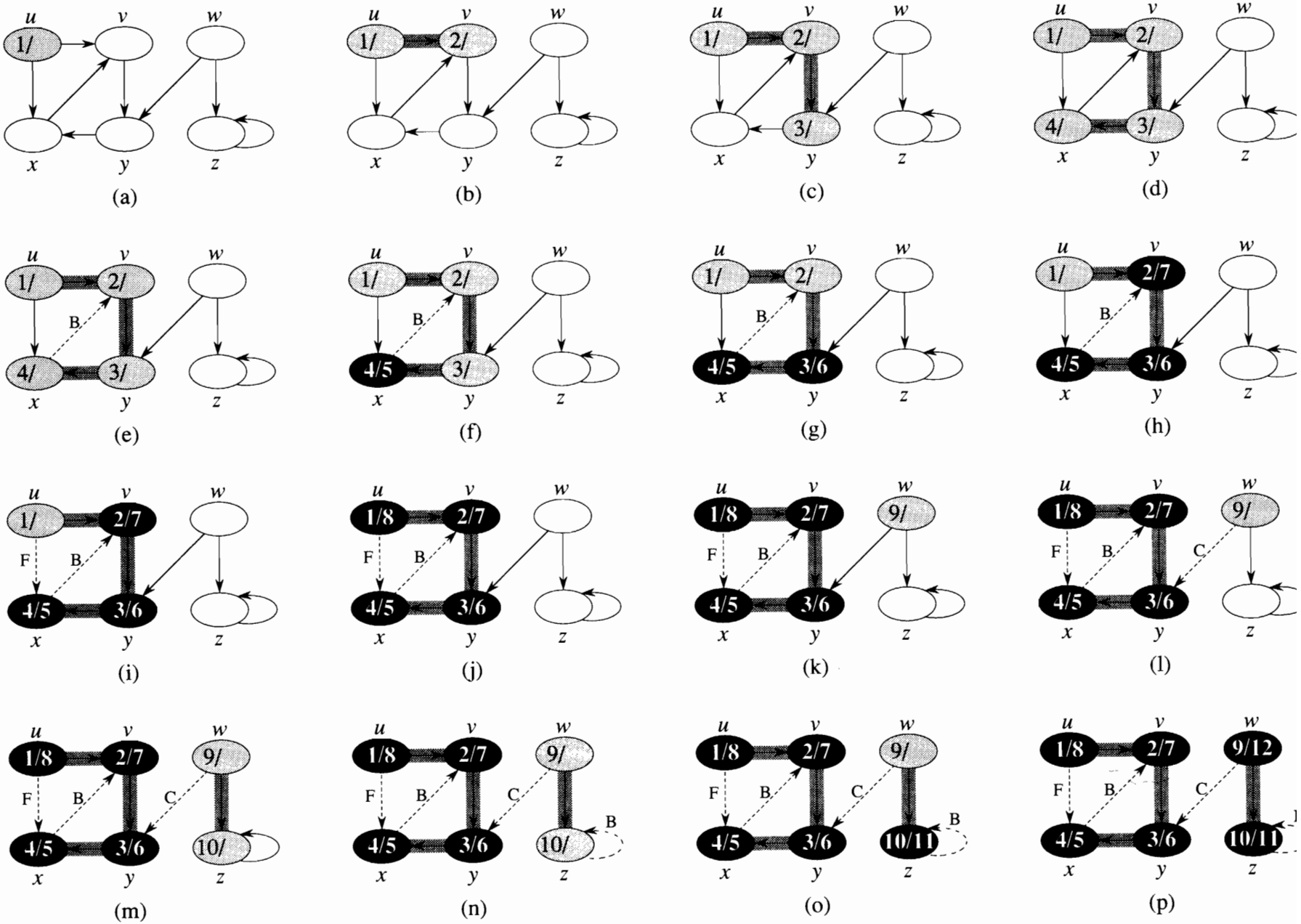
```
1       for each vertex u ∈ V[G]
2           u.color ← WHITE;
3           u.π ← NIL;
4       time ← 0;
5       for each vertex u ∈ V[G]
6           if u.color == WHITE
7               DFS_Visit(G, u);
```

22.3 Depth First Search (continued)

DFS_Visit(G, u)

```
1    time ← time + 1;
2    u.d ← time;
3    u.color ← GRAY;
4    for each vertex v ∈ G.Adj[u]
5        if v.color == WHITE
6            v.π ← u;
7            DFS_Visit(G, v);
8    u.color ← BLACK;
9    time ← time + 1;
10   u.f ← time;
```

Theorem. The call to **DFS_Visit**(G, u) in line 7 of **DFS**(G) will visit each vertex reachable from the starting vertex u that has not previously been visited.


u v w
1/
x y z
(a)
1/ 2/
(b)
1/ 2/ 3/
(c)
1/ 2/ 4/ 3/
(d)
B
4/ 3/
(e)
4/5 3/
(f)
3/6
(g)
2/7
(h)
F
1/ 2/7 4/5 3/6
(i)
1/8 2/7 4/5 3/6
(j)
9/
(k)
C
(l)
10/
(m)
(n)
10/11
(o)
9/12
(p)

22.3 Running time of DFS

The loops on lines 1-3 and 5-7 of **DFS** take time $\Theta(V)$, plus the time to execute the calls to **DFS_Visit**.

DFS_Visit is called exactly once for each vertex v in the graph. During the execution of **DFS_Visit** for vertex v, the loop on lines 4-7 executes once for each vertex in the adjacency list of v.

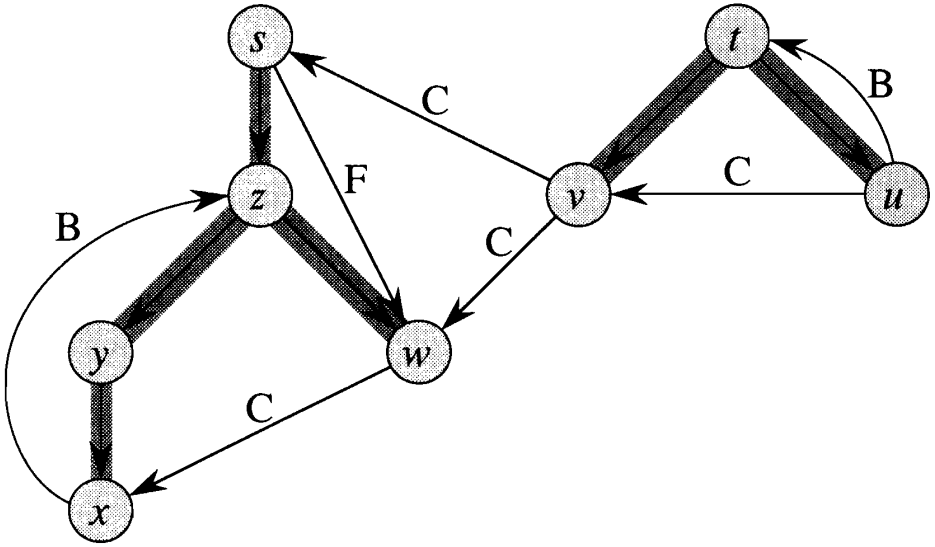
Thus, the total running time is

$$\begin{aligned} \Theta(|V|) + \Theta(\sum_{v \in V} |Adj[v]|) &= \Theta(|V|) + \Theta(|E|) \\ &= \Theta(|V| + |E|). \end{aligned} \tag{1}$$

22.3 Edge classification in DFS

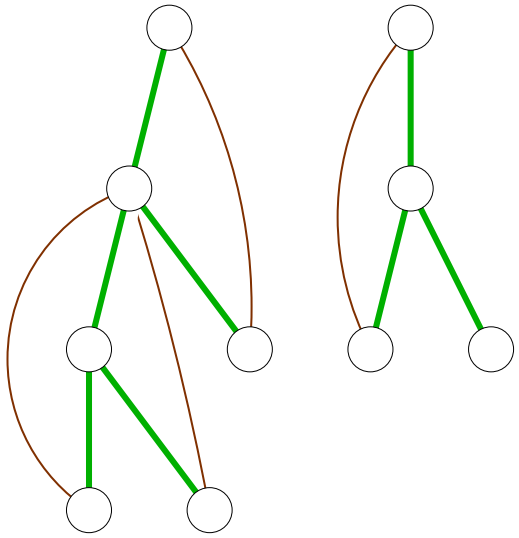
DFS classifies the edges in **a directed graph** into four types.

- Tree edges (edges along which a vertex is first discovered)
- Back edges (edges from a descendant to an ancestor)
- Forward edges (non-tree edges from an ancestor to a descendant)
- Cross edges (all other edges)



22.3 Edge classification in DFS (continued)

In an undirected graph, edges (u, v) and (v, u) are the same, so we cannot distinguish between forward edges and back edges. For such a graph, a non-tree edge between a pair of vertices that one is an ancestor of the other is referred to as a back edge.

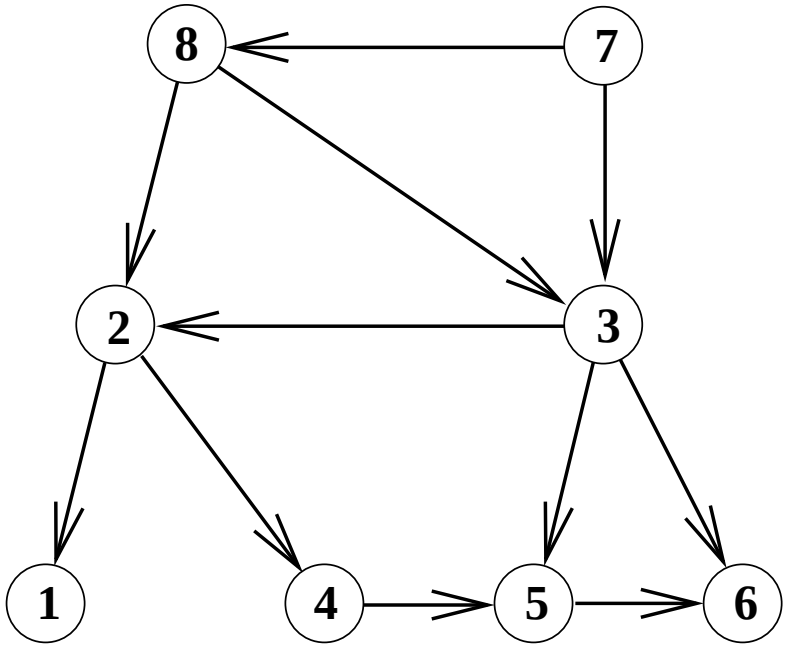


Theorem. In a depth-first search of an *undirected* graph G, each edge of G is either a tree edge or a back edge.

22.4 Topological Sorting

A directed graph $G(V, E)$ is called acyclic if it does not contain a directed cycle. Such a graph is also referred to as a Directed Acyclic Graph (DAG).

A topological ordering of a DAG is a linear ordering of the vertices such that the two endpoints of each edge in the ordered sequence are directed from left to right.



7−8−3−2−4−5−6−1

22.4 Topological Sorting (continued)

Theorem. A directed graph G has a directed cycle if and only if a depth-first search of G yields a back edge.

Proof:

($\Leftarrow$) If there is a back edge (u, v), then v is an ancestor of u (by the definition of back edges). The path of tree edges from v to u, together with edge (u, v), forms a directed cycle.

($\Rightarrow$) Let $v_1, v_2, \ldots, v_k, v_1$ be a directed cycle in G, where v_1 is the first vertex discovered during the DFS. By induction, vertices $v_2, \ldots, v_k$ are descendants of v_1. Therefore (v_k, v_1) is a back edge.

22.4 Topological Sorting (continued)

Theorem. Perform DFS on a directed acyclic graph G. Then, the vertex sequence listed by the decreasing order of their finishing time forms a topological ordering for G.

Proof:

Let (u,v) be an edge of G from vertex u to vertex v. We need to show that $v.f < u.f$.

Note that the finish time of a vertex x is when the call **DFS_Visit**(G,x) returns. The recursive structure of **DFS_Visit** implies that descendants always finish before their ancestors. Therefore, if (u,v) is a tree edge or forward edge, $v.f < u.f$.

Suppose that (u,v) is a cross edge. Vertex u cannot be discovered before vertex v, since then v would be a descendant of u. Therefore, v is discovered before u. Since u is not a descendant of v, it follows that v must be finished before u is discovered. Thus, $v.f < u.f$.

22.4 Topological Sorting (continued)

Exercise 1: A DAG may have more than one topological order. Show that for any topological order $v_1 - v_2 - \ldots - v_n$ of a DAG, there is a depth-first search corresponding to this topological order of the DAG.
That is, there is a DFS whose finishing times satisfy $v_1.f > v_2.f > \ldots > v_n.f$.

Exercise 2: Give a necessary and sufficient condition for the existence of a unique topological order for a DAG.