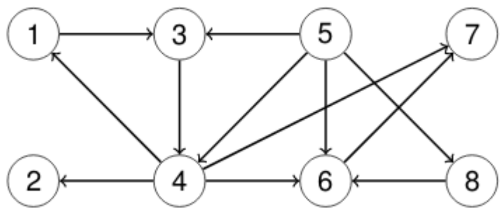
Depth First Search

1 Depth First Search

- Like BFS, DFS explores the graph (but does not find distances to the source)
- In contrast to BFS, when a vertex is discovered it is immediately explored
- Two timestamps are recorded for each vertex, d and f; the discovery and finish times. We can also record predecessors again
- Again colours are used: white for undiscovered, grey for discovered but not finished, black for finished

2 Example



- Initialize: source vertex grey, others white, source discovered at time 1
- Repeat
  - Increment the time
  - If there is a white neighbour of the current vertex, then it is coloured grey and its discovery time noted and it becomes current
  - Else colour the current vertex black, note its finish time and return to its predecessor or jump to an undiscovered vertex, or stop

3 Depth First Search

Listing 1: DFS(G)

```
for each vertex u ∈ V[G]
    do colour[u]←WHITE
        π[u] ← NIL
time ← 0
for each vertex u ∈ V[G]
    do if colour[u]=WHITE
        then DFS-VISIT(u)
```

Listing 2: DFS-VISIT(u)

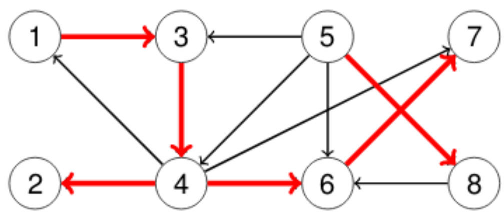
```
colour[u]← GREY                                  [vertex u has just been discovered]
time←time+1
d[u]←time
for each vertex v ∈ Adj[u]                                      [explore edge (u,v)]
    do if colour[v]=WHITE
        then π[v] ← u
            DFS-VISIT(v)
colour[u]←BLACK                                             [u has been processed]
f[u]←time←time+1
```

4 Analysis

- Initialisation takes time $O(V)$
- Time $O(V)$ is spent on incrementing time, colouring vertices and updating d and f
- Each vertex in each adjacency list is considered at most once. This takes time $O(E)$
- Total time is $O(V + E)$

The edges used for discovering new vertices from the depth first tree (or forest). Again we can find this with a predecessor array

5 Example



Once we have run DFS on a graph we can construct the predecessor subgraph. This has the same vertex set as the graph, and for each vertex v there is an edge from the predecessor of v to v

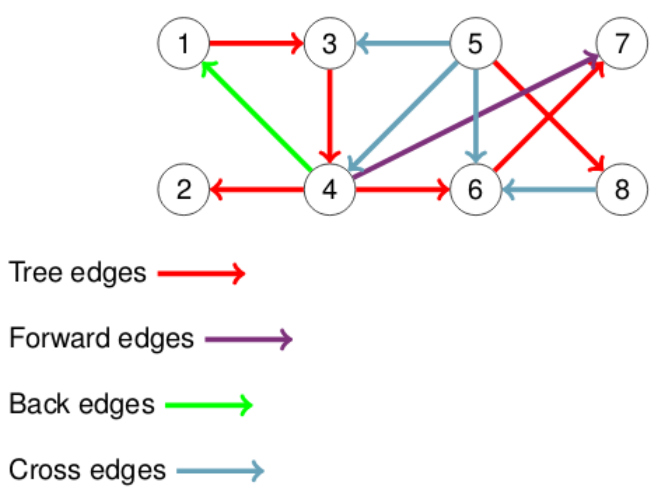
The predecessor subgraph is a depth first forest

6 Classification of the edges

Once we have obtained a DFS-Forest for the graph G, we can classify the edges of G

- Tree edges are those edges in the DFS-Forest
- Back edges are edges that join a vertex to an ancestor
- Forward edges are edges not in the tree that join a vertex to its descendant
- Cross edges: all other edges

7 Example



8 Classification of the edges

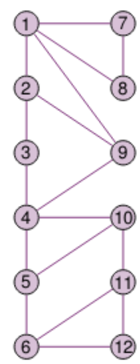
The classification is ambiguous for undirected graphs (back edges and forward edges are the same thing)

- e is a forward edge if DFS first consideres e from u
- e is a back edge if DFS first considers e from v

8.1 Theorem

In an undirected graph, every edge is a tree edge or a back edge

9 Using DFS



- Every edge in an undirected graph is either a tree edge or a black edge
- A graph is connected if each pair of vertices is joined by a path
- A cycle is a sequence of edges that start and end at the same vertex
- An articulation point is a vertex whose removal disconnects the graph

Can we adapt DFS to obtain algorithms that

9.1 Check whether a graph is connected

$O(V + E)$
Amend DFS to prevent jumping to undiscovered vertices (don't jump to undiscovered vertex if no more connected ones available)
Run DFS with an arbitrary source
The graph is connected $\Leftrightarrow$ DFS finds all vertices

9.2 Discover a cycle in a graph

$O(V + E)$
Run DFS with an arbitrary source
The graph contains a cycle $\Leftrightarrow$ a back edge is discovered during DFS

9.3 Find all the articulation points in a graph

$O((V + E)V) = O(V^3)$
For each vertex u
Remove u from the graph
Run DFS on the new graph from any source
u is an articulation point $\Leftrightarrow$ the new graph is not connected

9.3.1 Alternate method

Run DFS once
Can we recognise the articulation points?
The **source** is an articulation $\Leftrightarrow$ the source has more than one child in the depth first tree

Leaves don't need checking as they are not connected

Other vertices: a vector u is an articulation point unless there is a back edge from every child subtree to the parent subtree

9.3.2 One run method

Remember back edges - which ones most useful/important
Back edges in a chain of nodes mean that removing the nodes below that edge doesn't disconnect a graph
Create an array that records, for each vertex v, the most distant ancestor to which there is a back edge
In fact, we want the most distant ancestor from which there is a back edge from either v or one of its descendants

Create an empty array N
Let N[v]=v
Run DFS and update N to record the most distant ancestor connected by a back edge to v or its descendants.