Lecture 24

Depth-first search (DFS)

Traversal algorithms differ in the rules for choosing the next grey and next white node, with different rules leading to very different results.

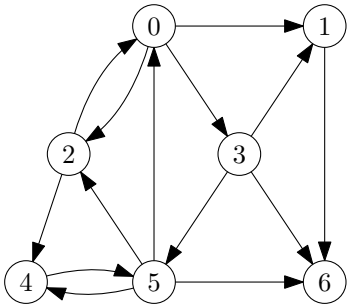
Definition 24.1. In ***depth-first search*** (DFS) the new grey node chosen is the one that has been grey for the **shortest** time.

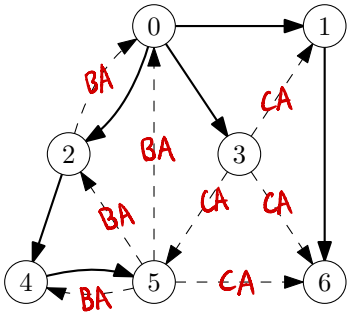
DFS takes us away from the root node as quickly as possible. If the first visited neighbour of the root has a neighbour, we immediately visit that neighbour. Then, if that has a neighbour, we visit that and so on, thus "deeply" searching as far away from the root as possible. We backtrack as little as possible before continuing away from the root again. The root is the last node to turn black.

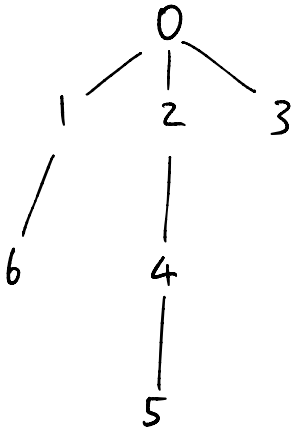
There still remains the choice of which white neighbour of the chosen grey node to visit. It does not matter what choice is made but our **convention is to choose the one with lowest index** (recall that nodes have indices $0, \ldots, n-1$).

These choices can be made in constant time so that the running time is $\Theta(n+m)$ (assuming adjacency lists) and DFS is linear in the size + order of a digraph.

Example 24.2. A digraph and its DFS search tree, rooted at node 0. The dashed arcs indicate the original arcs that are not part of the DFS search tree.

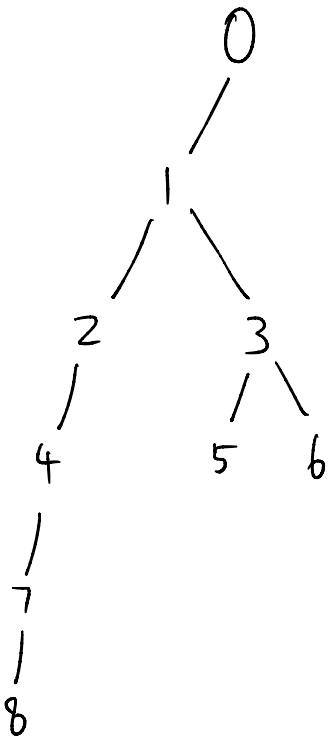






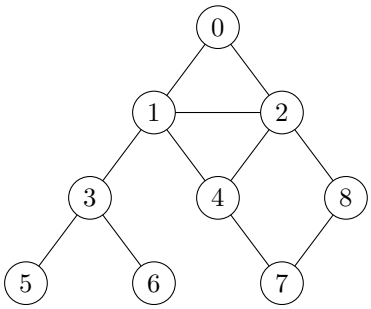
Find a tree arc, cross arc, forward arc and back arc (or say if that type of arc does not exist for that traversal).

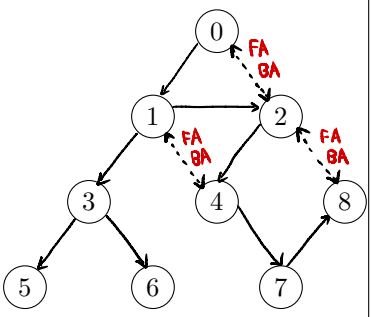
No forward arcs



Example 24.3. Use the nodes on the right to draw the search tree you obtain by running DFS on the graph on the left, starting at vertex 0. Use dashed edges to indicate edges that are not arcs in the search tree.

edges ∴ dashed edges can go both ways





Find a tree arc, cross arc, forward arc and back arc (or say if that type of arc does not exist for that traversal).

No cross arcs

For a general digraph, and unlike the examples given here, all nodes may not be reachable from the starting node. In this case, we get a search forest where the number of trees in the forest is the number of calls made to `DFSvisit`.

24.1 Pseudocode for depth-first search

The "last in, first out" order of choosing grey nodes in DFS is mimicked by a **stack** data structure. We thus store grey nodes in a stack as they are discovered.

The pseudocode for `DFS` and `DFSvisit` is in Algorithms 3 and 4.

- We loop through the nodes adjacent to the chosen grey node u, and as soon as we find a white one, we add it to the stack.

- We also introduce a variable `time` and keep track of the time a node changes colour.
- The array `seen` records the time a node turns grey.
- The array `done` records the time a node turns black.

Algorithm 3 Depth-first search algorithm.

```
1:  function DFS(digraph G)
2:      stack S
3:      array colour[0..n−1], pred[0..n−1], seen[0..n−1], done[0..n−1]
4:      for u ∈ V(G) do
5:          colour[u] ← WHITE; pred[u] ← null        ▷ initialise arrays
6:      time ← 0    ▷ time will increment at every node colour change
7:      for s ∈ V(G) do
8:          if colour[s] = WHITE then                ▷ find a WHITE node
9:              DFSvisit(s)                          ▷ start traversal from s
10:     return pred, seen, done
```

Algorithm 4 Depth-first visit algorithm.

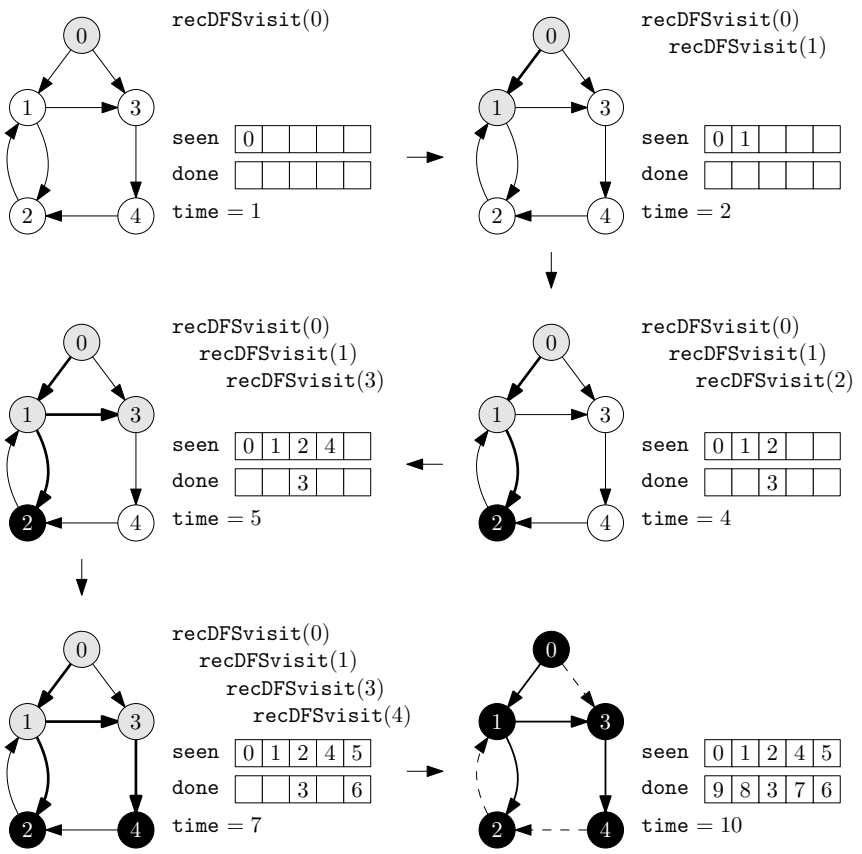
```
1:  function DFSVISIT(node s)
2:      colour[s] ← GREY
3:      seen[s] ← time; time ← time +1
4:      S.insert(s)                                   ▷ put s in stack
5:      while not S.isEmpty() do
6:          u ← S.peek()                              ▷ get node at front of stack
7:          if there is a neighbour v with colour[v] = WHITE then
8:              colour[v] ← GREY; pred[v] ← u         ▷ visit neighbour,
    update tree
9:              seen[v] ← time; time ← time +1        ▷ record and
    increment time
10:             S.insert(v)                           ▷ put neighbour in stack
11:         else
12:             S.delete()                            ▷ delete node from stack
13:             colour[u] ← BLACK                     ▷ colour it done
14:             done[u] ← time; time ← time +1        ▷ record and
    increment time
```

Given the relationship between stacks and recursion, we can replace the `DFSvisit` procedure by a recursive version `recursiveDFSvisit`:

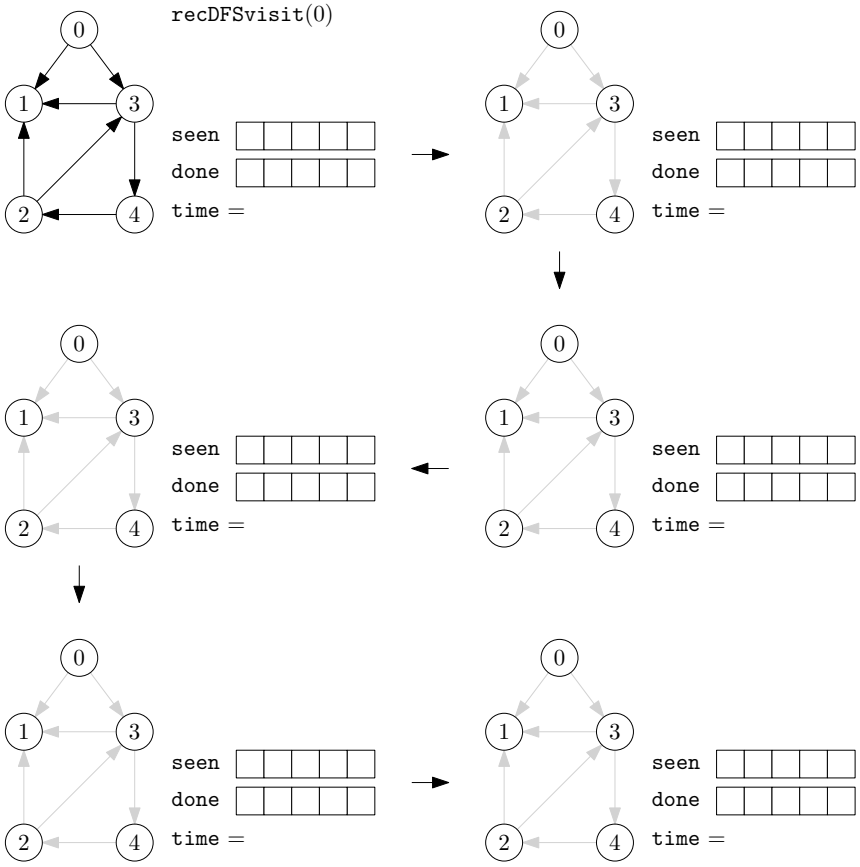
Algorithm 5 Recursive DFS visit algorithm.

```
1:  function RECURSIVEDFSVISIT(node s)
2:      colour[s] ← GREY
3:      seen[s] ← time; time ← time +1
4:      for each v adjacent to s do
5:          if colour[v] = WHITE then                 ▷ each unvisited neighbour
6:              pred[v] ← s                           ▷ update tree
7:              recursiveDFSvisit(v)                  ▷ now visit neighbour
8:      colour[s] ← BLACK          ▷ visted all white neighbours so done
9:      done[s] ← time; time ← time +1
```

Example 24.4. DFS with `recursiveDFSvisit`.



Example 24.5. Execute DFS with `recursiveDFSvisit` and following Example 24.4 fill out the values for `seen` and `done`, give the current `time`, and highlight the DFS search tree after each step.



24.2 DFS useful results and facts

Theorem 24.6. Suppose that we have performed DFS on a digraph G, resulting in a search forest F. Let $v, w \in V(G)$ and suppose that $\texttt{seen}[v] < \texttt{seen}[w]$.

- If v is an ancestor of w in F, then

$$\texttt{seen}[v] < \texttt{seen}[w] < \texttt{done}[w] < \texttt{done}[v].$$

- If v is not an ancestor of w in F, then

$$\texttt{seen}[v] < \texttt{done}[v] < \texttt{seen}[w] < \texttt{done}[w].$$

Note that this result rules out the timestamps $\texttt{seen}[v] < \texttt{seen}[w] < \texttt{done}[v] < \texttt{done}[w]$.

All four types of arcs in our search forest classification can arise with DFS. The different types of arcs can be easily distinguished while the algorithm is running or by looking at the timestamps `seen` and `done`.

Example 24.7. Explain how to determine, at the time when an arc is first explored by `DFS`, whether it is a tree-, back-, forward- or cross-arc.

tree-arc: when w is white and turn grey as a result of crossing this arc.

back-arc: w is ancestor of v, so w is still grey

forward-arc: seen[v] < seen[w] < done[w]

cross-arc: no ancestral relationship so done[w] < seen[v] before seeing v, already done with w.

Example 24.8. Suppose that we have performed DFS on a digraph G. Let $(v,w) \in E$. The following statements are true. Prove the first statement.

- (v,w) is a tree or forward arc if and only if

$$\texttt{seen}[v] < \texttt{seen}[w] < \texttt{done}[w] < \texttt{done}[v];$$

- (v,w) is a back arc if and only if

$$\texttt{seen}[w] < \texttt{seen}[v] < \texttt{done}[v] < \texttt{done}[w],$$

- (v,w) is a cross arc if and only if

$$\texttt{seen}[w] < \texttt{done}[w] < \texttt{seen}[v] < \texttt{done}[v].$$

If (v,w) is a forward arc, v is an ancestor of w in the same search forest

So result is immediate from theorem.