Here are corrected versions of Figures 5.7, 5.9, 5.10 of the textbook. There was an error caused by cutting and pasting, and forgetting to edit. In fact Figure 5.9 has not changed at all — in the case of BFS we can get away with colouring all white neighbours grey in one step, but we clearly cannot in the other cases.

An important issue not properly discussed in the book is: how do we check whether a node u has a white neighbour v? It seems that in the worst case we might need to scan all neighbours of u until we find a white one. This would ruin the linear running time of DFS, for example. An easy way to fix this is, each time we colour v grey, to delete v from the adjacency list of u.

```
algorithm dfs
    Input: digraph G
begin
    stack S
    array colour[n], pred[n], seen[n], done[n]
    for u ∈ V(G) do
        colour[u] ← WHITE; pred[u] ← NULL
    end for
    time ← 0
    for s ∈ V(G) do
        if colour[s] = WHITE then
            dfsvisit(s)
        end if
    end for
    return pred, seen, done
end

algorithm dfsvisit
    Input: node s
begin
    colour[s] ← GREY
    seen[u] ← time; time ← time + 1
    S.insert(s)
    while not S.isempty() do
        u ← S.get_top()
        if there is a neighbour v with colour[v] = WHITE then
            colour[v] ← GREY; pred[v] ← u
            seen[v] ← time;
            S.insert(v)
        else
            S.delete()
            colour[u] ← BLACK
            done[u] ← time; time ← time + 1
        end if
    end while
end
```

Figure 1: Depth-first search algorithm.

```
algorithm bfs
    Input:  digraph G
begin
    queue Q
    array colour[n], pred[n], d[n]
    for u ∈ V(G) do
        colour[u] ← WHITE; pred[u] ← NULL
    end for
    for s ∈ V(G) do
        if colour[s] = WHITE then
            bfsvisit(s)
        end if
    end for
    return pred, d
end

algorithm bfsvisit
    Input:  node s
begin
    colour[s] ← GREY; d[s] ← 0
    Q.insert(s)
    while not Q.isempty() do
        u ← Q.get_head()
        for each v adjacent to u do
            if colour[v] = WHITE then
                colour[v] ← GREY; pred[v] ← u; d[v] ← d[u] + 1
                Q.insert(v)
            end if
        end for
        Q.delete()
        colour[u] ← BLACK
    end while
end
```

Figure 2: Breadth-first search algorithm.

```
algorithm pfs
    Input: digraph G
begin
    priority queue Q
    array colour[n], pred[n]
    for u ∈ V(G) do
        colour[u] ← WHITE; pred[u] ← NULL
    end for
    for s ∈ V(G) do
        if colour[s] = WHITE then
            pfsvisit(s)
        end if
    end for
    return pred
end

algorithm pfsvisit
    Input: node s
begin
    colour[s] ← GREY
    Q.insert(s, setkey (s))
    while not Q.isempty() do
        u ← Q.get_min()
        if u has a neighbour v with colour[v] = WHITE then
            colour[v] ← GREY
            Q.insert(v, setkey (v))
        else
            Q.delete_min()
            colour[u] ← BLACK
        end if
    end while
end
```

Figure 3: Priority-first search algorithm (first kind).