Chapter 3

1. (a) The depth-first forest obtained is the path $ABCDHGFE$. Thus, the pre/post times for each vertex are given by:

A	B	C	D	H	G	F	E
1/16	2/15	3/14	4/13	5/12	6/11	7/10	8/9

   - The tree edges are the path edges, i.e., AB, BC, CD, DH, HG, GF and FE;
   - There are no cross edges because every vertex is either a descendant or ascendant of every other vertex;
   - The back edges are DB, ED, EG and FG; and
   - The forward edges are AF and BE.

   (b) The depth-first forest is composed of the path AGH, the path $BCDF$ and the isolated vertex E. The pre/post times are:

A	G	H	B	C	D	F	E
1/6	2/5	3/4	7/14	8/13	9/12	10/11	15/16

   - The tree edges are the path edges, nominally AG, GH, BC, CD and DF;
   - The cross edges are the ones between connected components of the forest: BA, BG, FG, ED and EF. This is because, among the connected components, which are paths, every vertex has an ascendant/descendant relationship with every other vertex;
   - The back edges are HA and FB; and
   - Only CF is a forward edge.

3. (a) As we run DFS, we identify the following edge types:

   - $AC \rightarrow$ tree edge
   - $BC \rightarrow$ cross edge
   - $BD \rightarrow$ tree edge
   - $DA \rightarrow$ cross edge
   - $DG \rightarrow$ tree edge
   - $GD \rightarrow$ back edge

   Since there is a back edge, we can stop and conclude that the graph is cyclic.

   (b) As we run DFS, we identify the following edge types:

   - $AC \rightarrow$ tree edge
   - $CE \rightarrow$ tree edge
   - $CF \rightarrow$ tree edge

- $FH \rightarrow$ tree edge
- $AH \rightarrow$ forward edge
- $BA \rightarrow$ cross edge
- $BD \rightarrow$ tree edge
- $DC \rightarrow$ cross edge
- $DG \rightarrow$ tree edge
- $GH \rightarrow$ cross edge

Since there is no back edge, the graph is acyclic.

5.

(a)

S	Q	$d(A)$	$d(B)$	$d(C)$	$d(D)$	$d(E)$	$d(F)$	$d(G)$	$d(H)$	Choice
$\{\}$	$\{A,B,C,D,E,F,G,H\}$	0	∞	∞	∞	∞	∞	∞	∞	A
$\{A\}$	$\{B,C,D,E,F,G,H\}$	0	1	∞	∞	4	8	∞	∞	B
$\{A,B\}$	$\{C,D,E,F,G,H\}$	0	1	3	∞	4	7	7	∞	C
$\{A,B,C\}$	$\{D,E,F,G,H\}$	0	1	3	4	4	7	5	∞	D
$\{A,B,C,D\}$	$\{E,F,G,H\}$	0	1	3	4	4	7	5	8	E
$\{A,B,C,D,E\}$	$\{F,G,H\}$	0	1	3	4	4	7	5	8	G
$\{A,B,C,D,E,G\}$	$\{F,H\}$	0	1	3	4	4	6	5	6	F
$\{A,B,C,D,E,F,G\}$	$\{H\}$	0	1	3	4	4	6	5	6	H
$\{A,B,C,D,E,F,G,H\}$	$\{\}$	0	1	3	4	4	6	5	6	

(b)

S	Q	$d(A)$	$d(B)$	$d(C)$	$d(D)$	$d(E)$	$d(F)$	$d(G)$	$d(H)$	Choice
$\{\}$	$\{A,B,C,D,E,F,G,H\}$	0	∞	∞	∞	∞	∞	∞	∞	A
$\{A\}$	$\{B,C,D,E,F,G,H\}$	0	∞	0.26	∞	0.38	∞	∞	∞	C
$\{A,C\}$	$\{B,D,E,F,G,H\}$	0	∞	0.26	∞	0.38	∞	∞	0.6	E
$\{A,C,E\}$	$\{B,D,F,G,H\}$	0	∞	0.26	∞	0.38	0.73	∞	0.6	H
$\{A,C,E,H\}$	$\{B,D,F,G\}$	0	∞	0.26	0.99	0.38	0.73	∞	0.6	F
$\{A,C,E,F,H\}$	$\{B,D,G\}$	0	1.05	0.26	0.99	0.38	0.73	∞	0.6	D
$\{A,C,D,E,F,H\}$	$\{B,G\}$	0	1.05	0.26	0.99	0.38	0.73	1.51	0.6	B
$\{A,B,C,D,E,F,H\}$	$\{G\}$	0	1.05	0.26	0.99	0.38	0.73	1.51	0.6	G
$\{A,B,C,D,E,F,G,H\}$	$\{\}$	0	1.05	0.26	0.99	0.38	0.73	1.51	0.6	

7. The following algorithm performs a DFS on the input graph and classifies all edges. If an undirected graph is passed in adjacency list representation (each edge is doubled), then the algorithm works without any modifications, but each edge will be reported twice (once as a back edge and once as either a forward or tree edge).

DFS(G)

for $v \in V$ **do**
  pre(v) $\leftarrow \infty$
  post(v) $\leftarrow \infty$
end for
time $\leftarrow 0$
for $v \in V$ **do**
  if pre(v) $= \infty$ **then**
    explore(v)
  end if
end for

```
explore(u)
  pre(u) ← time
  time ← time + 1
  for all edges e from u to a vertex v do
    if pre(v) = ∞ then
      Report e as a tree edge.
      explore(v)
    else if post(v) < pre(u) then
      Report e as a cross edge.
    else if pre(v) ≤ pre(u) then
      Report e as a back edge.
    else
      Report e as a forward edge.
    end if
  end for
  post(u) ← time
  time ← time + 1
```

9. Consider the DFS-forest. For each tree in this forest, label its vertices:

   - root: label L
   - child of an L-node: label R
   - child of an R-node: label L

   Test if the two vertices of each back edge have different labels.

   - If yes, the graph is bipartite.
   - If no, the graph is not bipartite.

   Here is how you need to modify DFS to get this algorithm.

```
Bipartite(G)
  for all v ∈ V do
    visited(v) = False
  end for
  for all v ∈ V do
    if visited(v) = False then
      Explore(v, L)
    end if
  end for
```

```
Explore(v, symbol)
  visited(v) = True
  label(v) = symbol
  for all edge {u, v} ∈ E do
    if visited(u) = False then
      if symbol = L then
        explore(u, R)
      else
        explore(u, L)
      end if
    end if
  end for
```

So you run Bipartite(G) and then you scan the edges of G to see if the two vertices of each back edge have different labels.

Since DFS takes $O(|V|+|E|)$ time, Bipartite(G) also takes $O(|V|+|E|)$ time. Scanning all edges of G to look at their endpoints also takes $O(|V| + |E|)$ time, for a total of $O(|V| + |E|)$.

11. This approach is limited to positive integer edge weights. Not only that, the size of the graph may increase. More precisely, if W is the total edge weight, $W - |E|$ new vertices and edges are created, for a total increase in running time of $\Theta(W)$.

    Generally speaking, this method is not very useful as W can be extremely large.

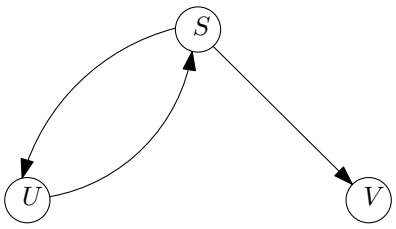
13. We keep a variable $\pi(v)$ for each vertex $v \in V$ which will, at the end of the algorithm, store the father of v in the shortest path tree. Initially, $\pi(v)$ is set to "none" for all vertices v, but whenever the value $d(v)$ is reduced using an edge from a vertex u with weight w because $d(u) + w < d(v)$, then we set $\pi(v)$ to u since a current shortest path to v comes from u.

    The running time of the algorithm is not asymptotically changed as we only change $\pi(v)$ when we also change $d(v)$. So, for a version of the algorithm with standard heaps, the running time is $O\big((|V| + |E|) \log |V|\big)$.

15. If all neighbours of u have already been discovered from other vertices, while no vertices from which u is reachable have been discovered. Thus, a new DFS tree may be started at u and will only contain u.

17. False.

    Run DFS starting at S on the following graph (using alphabetical order)



and you will see.

19. This is true. Here is a direct proof.

    Suppose that $|V| > 1$ to avoid having to talk about the null graph.

    Run DFS on G and consider a leaf v of the DFS-tree. If v does not have any back edge, then v has degree 1 in G. So we can remove it and G stays connected.

    Suppose that v has at least one back edge. Let G' be the graph we obtain by removing all back edges from v. If we remove the back edges from v, this does not modify the DFS-tree. So the DFS-tree is still connected and so is G'. Now, in G', v has degree 1. So we can remove it and the graph we obtain is connected (and it corresponds to G minus v.)

21. All shortest paths from s to t have the following shape: start at s and choose to go to x_1 or y_1. Then, from any vertex x_i or y_i (where $1 \leq i \leq n-1$), choose to go to x_{i+1} or y_{i+1}. Then, when you are at x_n or at y_n, go to t.

    Therefore, wherever you stand, you always have to choose between two options. Therefore, there are 2^n different shortest paths from s to t.

23. We prove by induction on n that any undirected and connected graph with n vertices and $n-1$ edges is a tree.

    For the base case, we have $n = 1$. There is only one possible graph with one vertex. It has no edge and it is a tree.

    For the induction hypothesis, let $n > 1$ and assume that any undirected and connected graph with $n-1$ vertices and $n-2$ edges is a tree.

    Note that no vertex can have degree 0, as then the graph would not be connected. Furthermore, if all vertices had degree at least two, the sum of the degrees would be at least $2n$ and, by the handshaking lemma, the number of edges would be at least $2n/2 = n$, which is a contradiction. So there must be a vertex v of degree one.

    Removing v does not disconnect the graph, and it decreases the number of vertices and edges of the graph by one. Thus, we can apply the induction hypothesis and we get a tree. Putting v back into this tree just adds a leaf to it. Hence, the original graph is also a tree and the induction is complete.