a) Below is an adjacency matrix of the graph as a table

	A	B	C	D	E	F	G	H
A	0	0	0	0	1	0	0	1
B	1	0	0	0	0	0	0	0
C	0	0	0	0	0	1	1	0
D	1	0	0	0	1	0	0	0
E	0	0	1	0	0	0	0	0
F	0	0	0	1	1	0	0	0
G	0	1	0	0	1	0	0	0
H	0	0	0	1	0	0	0	0

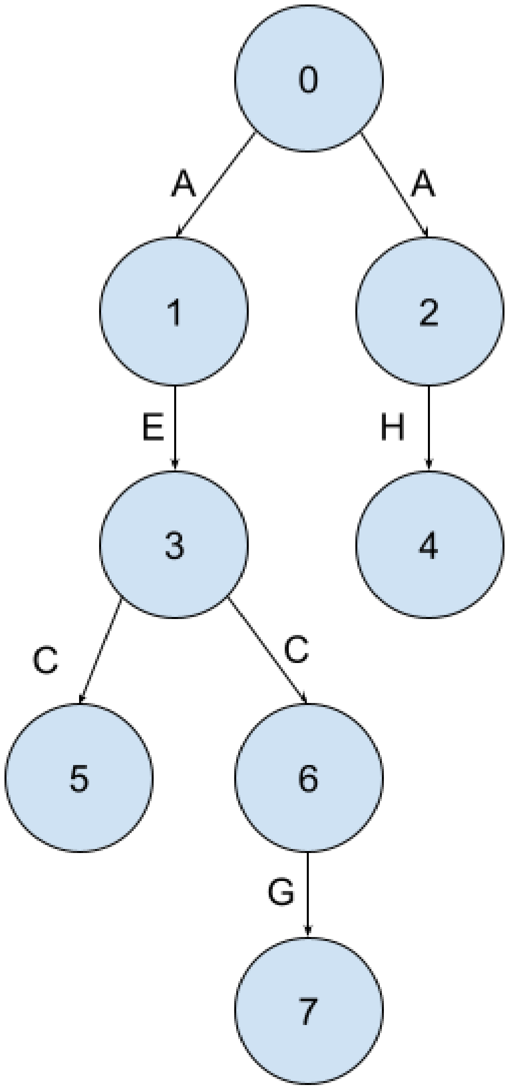
b) Below is the adjacency list of elements

A → [E, H]
B → [A]
C → [F, G]
D → [A, E]
E → [C]
F → [D, E]
G → [B, E]
H → [D]

c) Using the algorithm from class, the below steps are the breadth-first search of the graph starting with node A

Step Number	Description	Queue
1	Visit A	[A]
2	Queue A's Neighbors (E and H)	[A, E, H]
3	Dequeue A	[E, H]
4	Visit E and queue neighbors (C)	[E, H, C]
5	Dequeue E	[H, C]
6	Visit H and queue neighbors (D)	[H, C, D]
7	Dequeue H	[C, D]
8	Visit C and queue neighbors (F and G)	[C, D, F, G]
9	Dequeue C	[D, F, G]
10	Visit D and queue neighbors (A and E), but both have already been visited. No queuing happens.	[D, F, G]
11	Dequeue D	[F, G]
12	Visit F and queue neighbors (D and E), but both have already been visited. No queuing happens.	[F, G]
13	Dequeue F	[G]
14	Visit G and queue neighbors (B and E), but E has already been visited. Only queue B	[G, B]
15	Dequeue G	[B]
16	Visit B and queue neighbors (A), but A has already been visited. No queuing happens	[B]
17	Dequeue B	[ ]
18	Queue is empty	[ ]

The Tree for the above process would look like the following:



The overall processing order would be as follows:

[A, E, H, C, D, F, G, B]