AQA Computer Science A-Level
4.3.1 Graph-traversal
Intermediate Notes

<u>Specification:</u>

4.3.1.1 Simple graph-traversal algorithms

Be able to trace breadth-first and depth-first search algorithms and describe typical applications of both. Breadth-first: shortest path for an unweighted graph. Depth-first: Navigating a maze.

Graph-Traversal

Graph-traversal is the process of visiting each vertex in a graph. There are two algorithms in this section - depth-first and breadth-first graph-traversals. In a depth-first traversal, a branch is fully explored before backtracking, whereas in a breadth-first traversal a node is fully explored before venturing on to the next node.

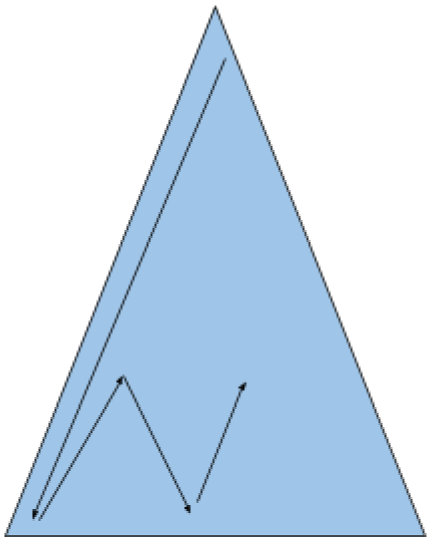
Synoptic Link

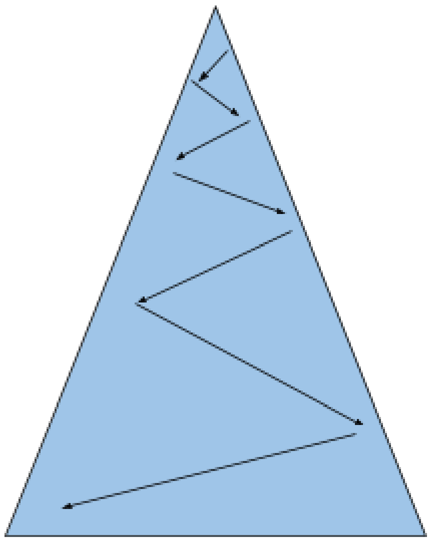
Graphs can be used as **visual representations** of **complex relationships**.

Graphs are covered in **Graphs** under **Fundamentals of Data Structures**.

Fully Explored

For the context of this resource, s node is **discovered** when it has been included in the result and a node is **completely/fully explored** when all of its adjacent nodes have been discovered.


Depth-First Traversal


Breadth-First Traversal

Synoptic Link

Stacks are abstract data types which use a LIFO (last in, first out) order of execution.

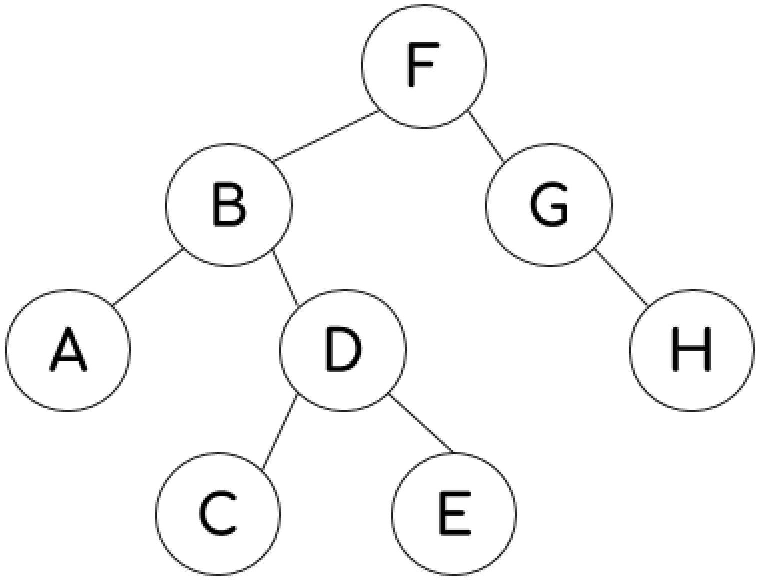
Stacks are covered in **Stacks** under **Fundamentals of Data Structures**.

Depth-First Search

Depth-first traversal uses a stack. Depth-first traversal is used for navigating a maze. The following example uses a tree, but a depth-first algorithm can be performed on any connected graph.

Example:

Here is a graph. This is a binary-tree.



Note

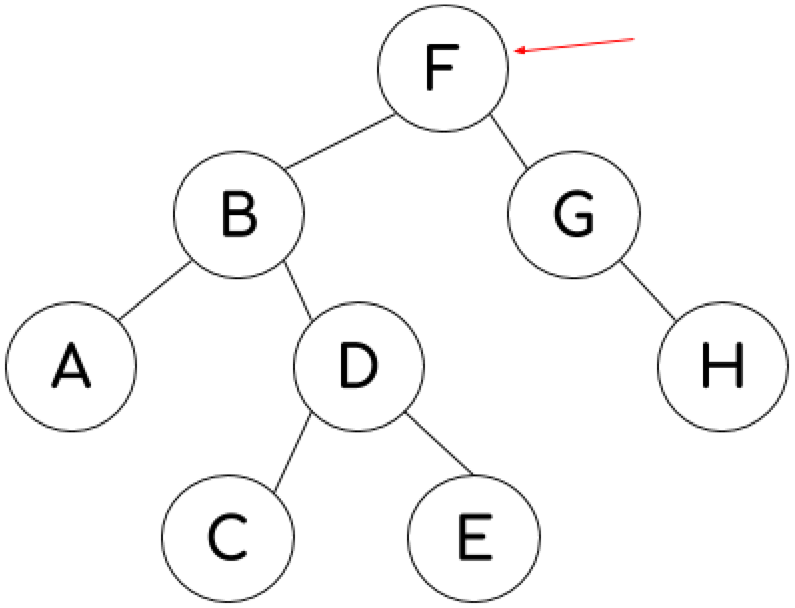
Whilst the depth-first algorithm can be used on a tree, it is not an example of tree-traversal as a depth-first traversal can be performed on any connected graph and tree-traversals are unique to trees.

Synoptic Link

A **tree** is a **connected acyclic graph**. A **binary tree** is a **rooted tree** where each node has at most **two** children. The **root node** has **no parent**.

Trees are covered in **Trees** under **Fundamentals of Data Structures**.

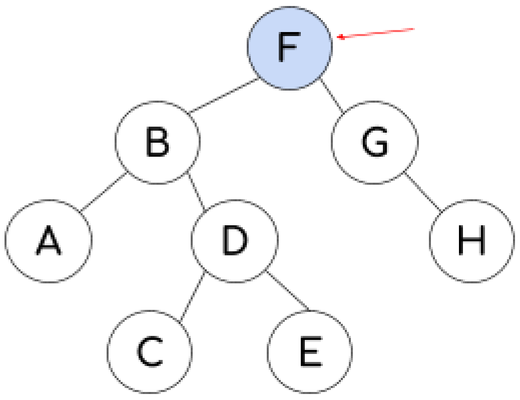
A graph traversal can start from any node, but for simplicity, the root node F will be chosen.

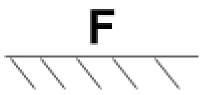


As F is a new node, it will be added to the result and to the stack. To show F has been discovered, it has been shaded blue.

Note

Node and **vertex** can be used interchangeably, as can **edge** and **arc**.



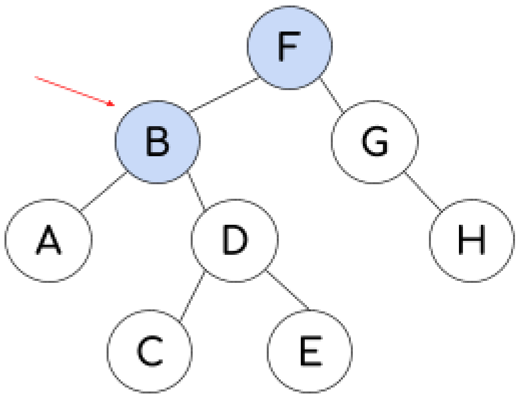


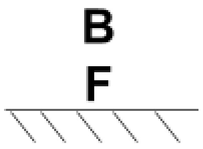
Result: **F**

Adjacent

When two **nodes** are **connected** to one another by a single **edge**, they can be said to be **adjacent**.

Next, the nodes adjacent to F are observed. These are B and G. B is higher alphabetically so B is discovered first.



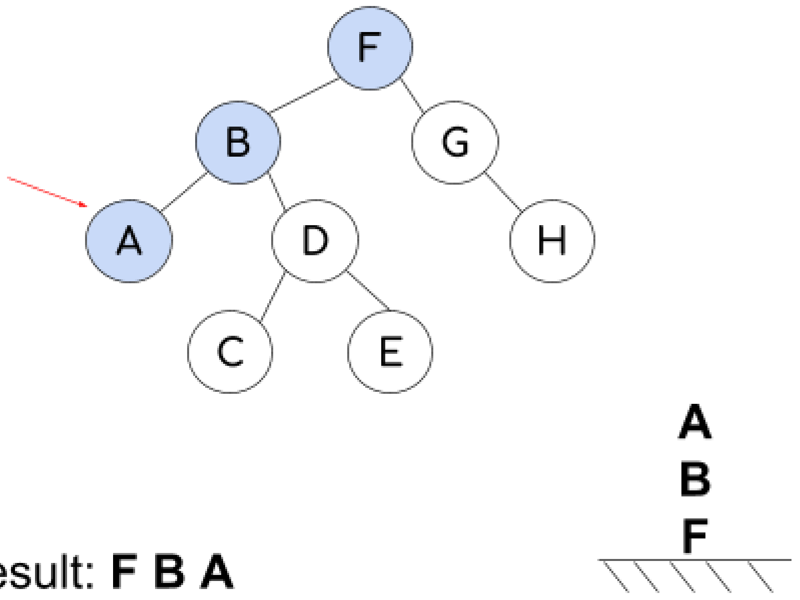


Result: **F B**

Note

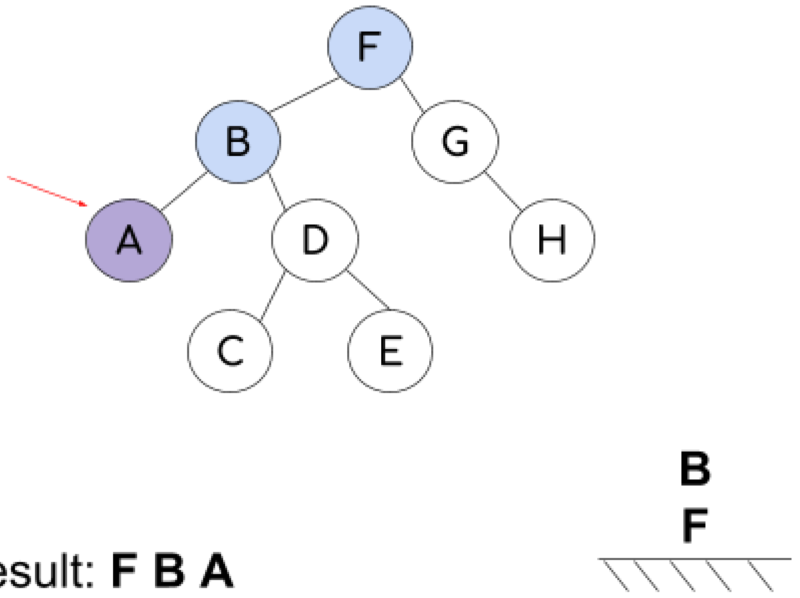
A **binary tree** may be made in the **reverse order**, in which case the higher item would be traversed first.

The undiscovered vertices adjacent to B are A and D; A is less than D so A is discovered first.



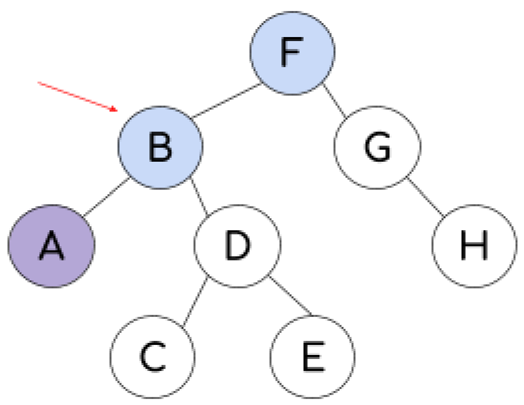
Result: **F B A**

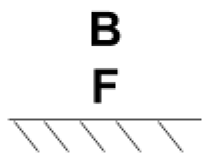
There are no undiscovered nodes adjacent to A. Therefore, A can be popped off the stack and labelled completely explored, visually indicated by the purple colour.



Result: **F B A**

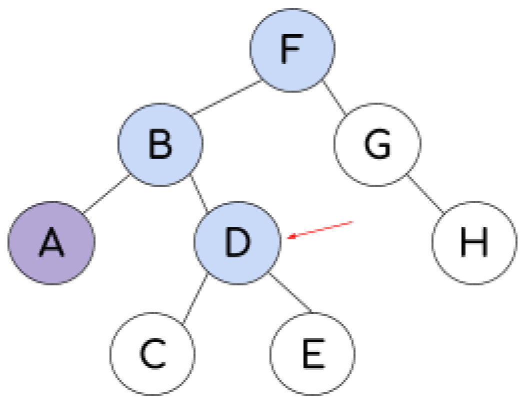
The next item in the stack is looked at - B.

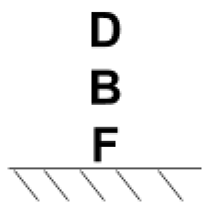




Result: **F B A**

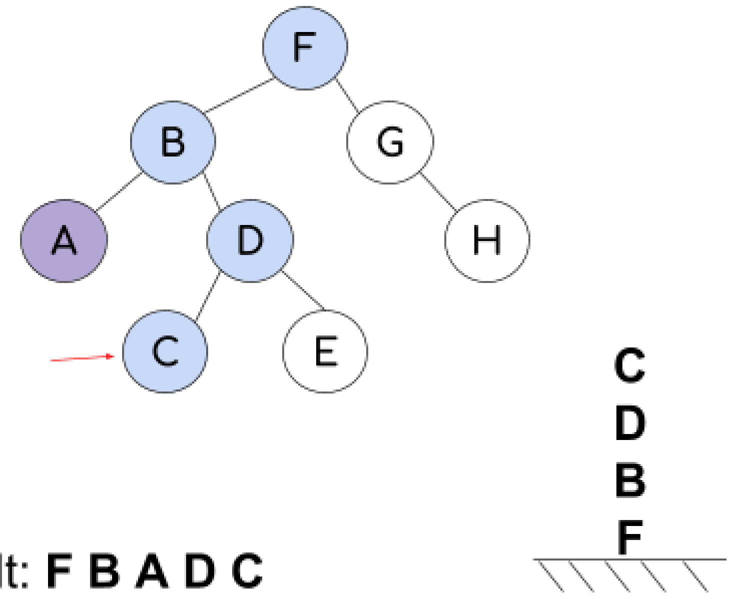
B has an adjacent undiscovered node, so D is visited.





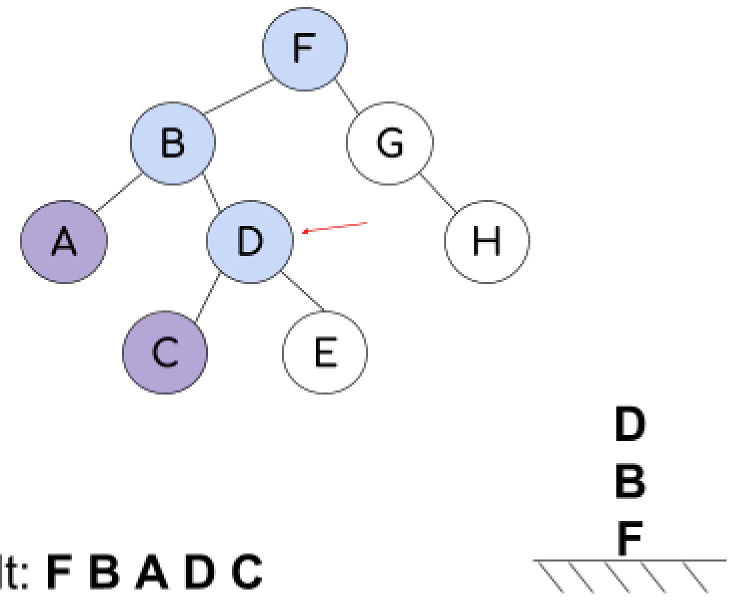
Result: **F B A D**

D has two adjacent undiscovered nodes, C and E. C is less than E so it is discovered first.



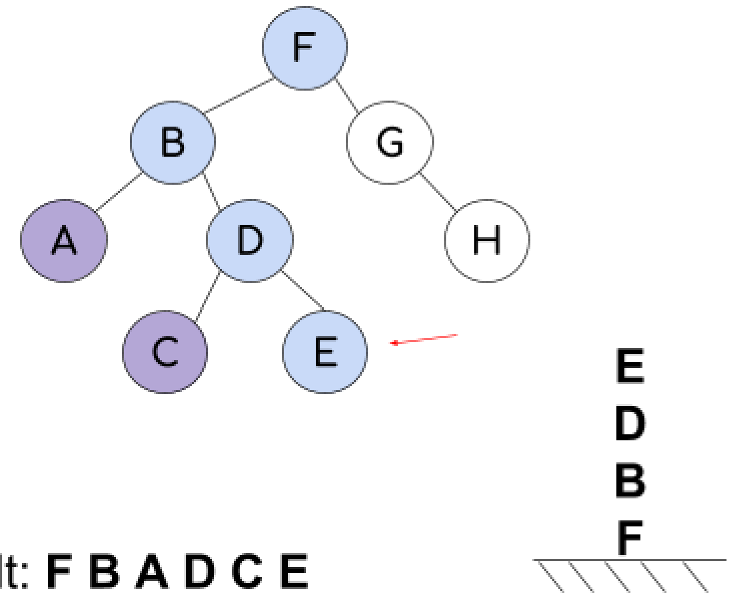
Result: **F B A D C**

C has no adjacent undiscovered nodes (it is completely explored) so it is popped off the stack, and the next item in the stack, D, is revisited.



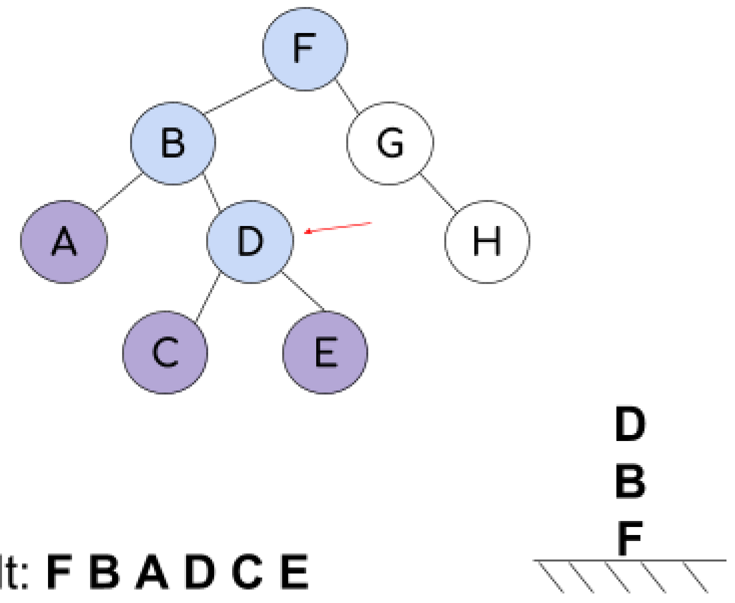
Result: **F B A D C**

D is adjacent to just one undiscovered node, E.



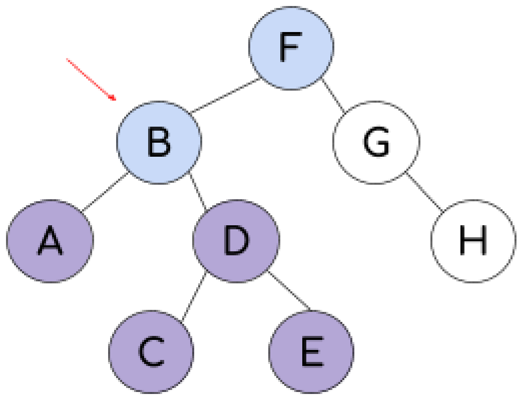
Result: **F B A D C E**

E has no undiscovered adjacent node so it is completely explored and can be removed from the stack. The next item on the stack, D, is revisited.

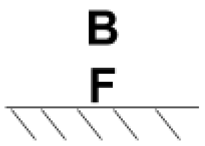


Result: **F B A D C E**

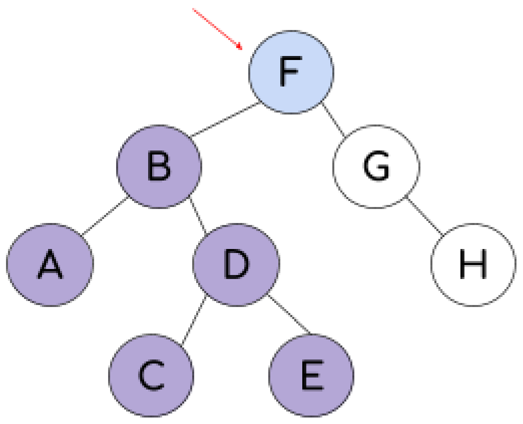
D is completely explored. It is popped off the stack and B is revisited.



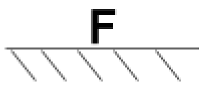
Result: **F B A D C E**



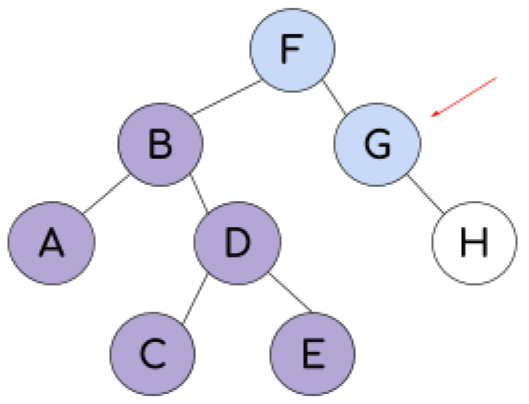
B is completely explored. B is popped off the stack and F is revisited.

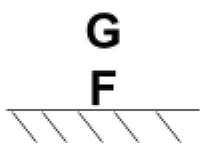


Result: **F B A D C E**



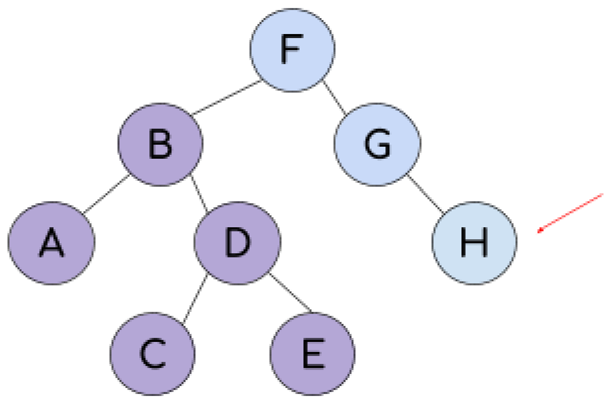
F has an adjacent undiscovered node. G is discovered, added to the stack and printed in the result.

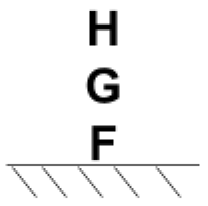




Result: **F B A D C E G**

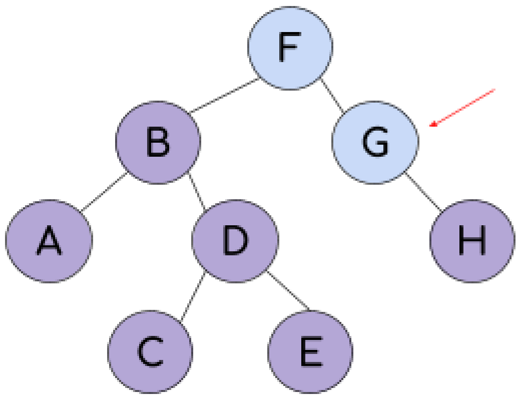
H is the only undiscovered node adjacent to G.

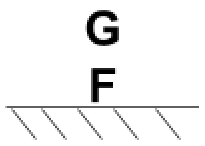




Result: **F B A D C E G H**

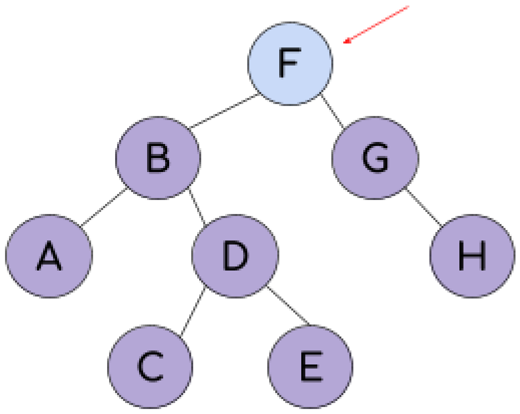
From a human's perspective, the procedure is complete as all nodes have been visited. However, a computer cannot know this until the algorithm has reached completion. H has no adjacent undiscovered nodes so it is completely explored.

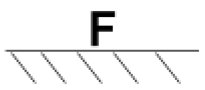




Result: **F B A D C E G H**

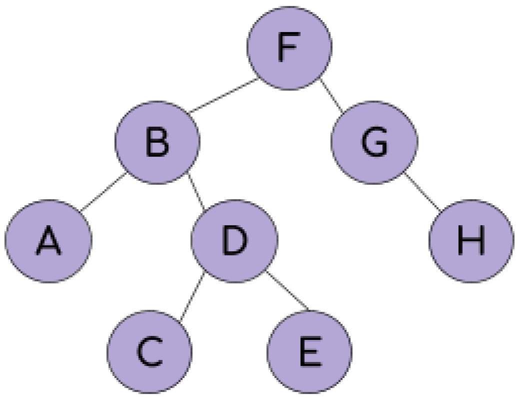
G is completely explored so it is popped from the stack.





Result: **F B A D C E G H**

Finally, F is completely explored.



Result: **F B A D C E G H**

There are no more items on the stack so the algorithm is complete.

Algorithm

An algorithm is a set of instructions which completes a task in a finite time and always terminates.

Breadth-First Search

Breadth-first traversal uses a queue. This algorithm will work on any connected graph. Breadth-first traversal is useful for determining the shortest path on an unweighted graph.

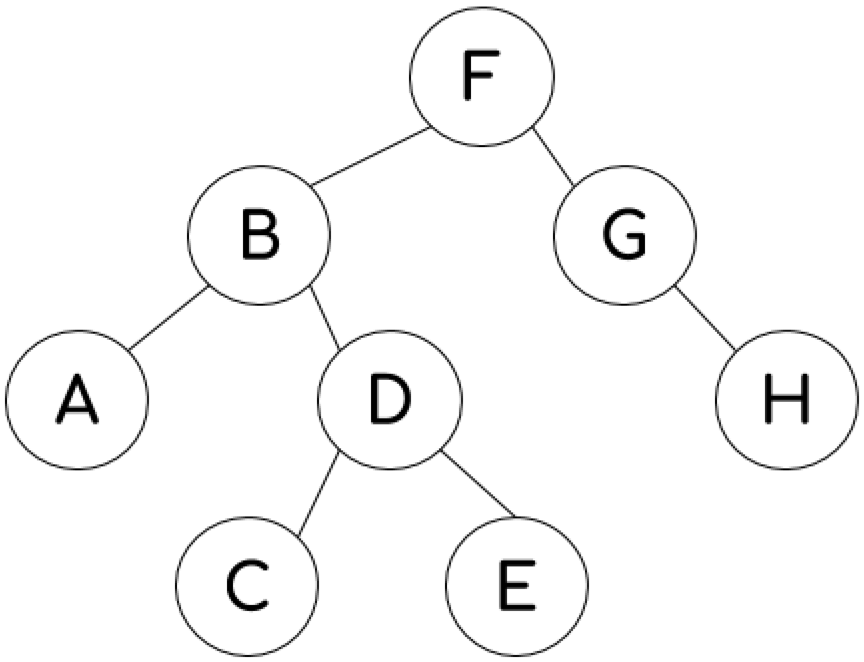
Synoptic Link

Queues are an abstract data type with a FIFO (first in, first out) order of execution.

Queues are covered in **Queues** under **Fundamentals of Data Structures**.

Example:

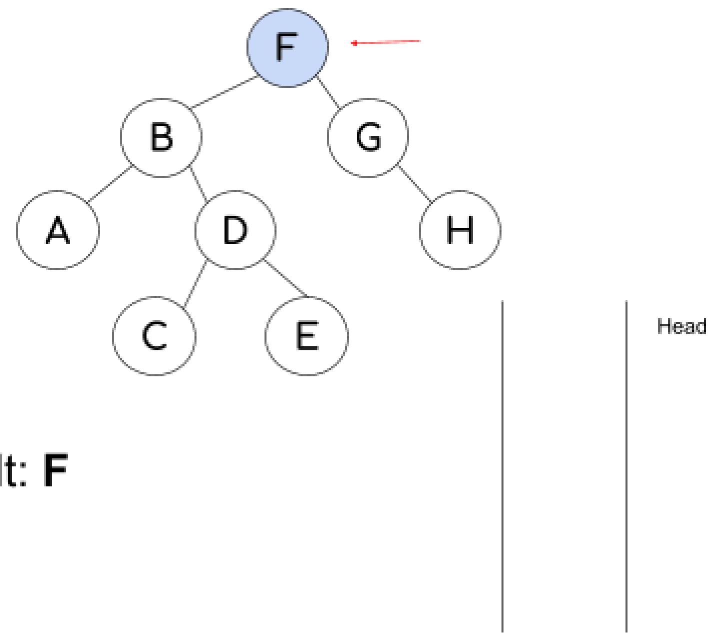
Here is a graph.



This is an example of a binary tree, but a breadth-first traversal will work on any connected graph. Any node can be chosen as a starting position, but as this is a binary tree it makes logical sense to start from the root F. F is discovered.

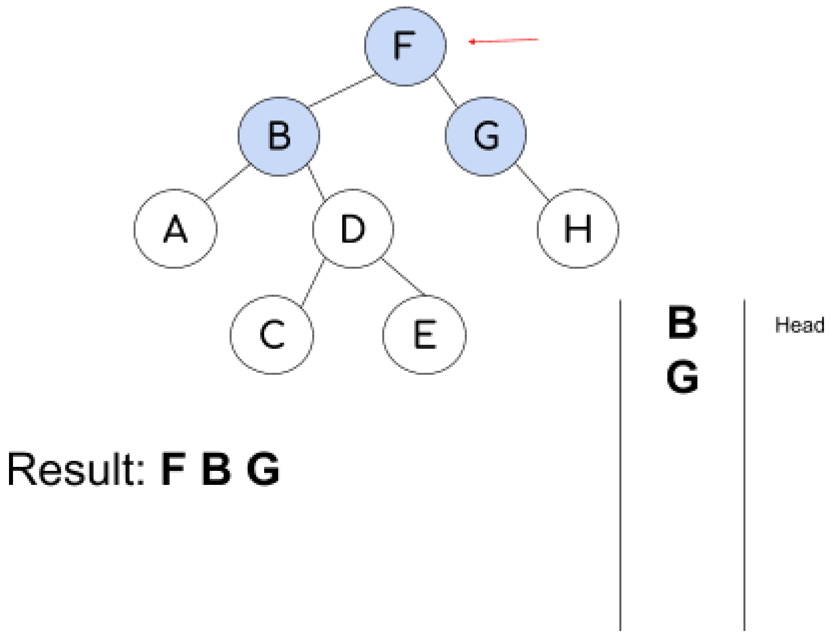
Connected Graph

In a **connected graph** there is a **path** between each pair of nodes; there are **no unreachable** nodes.

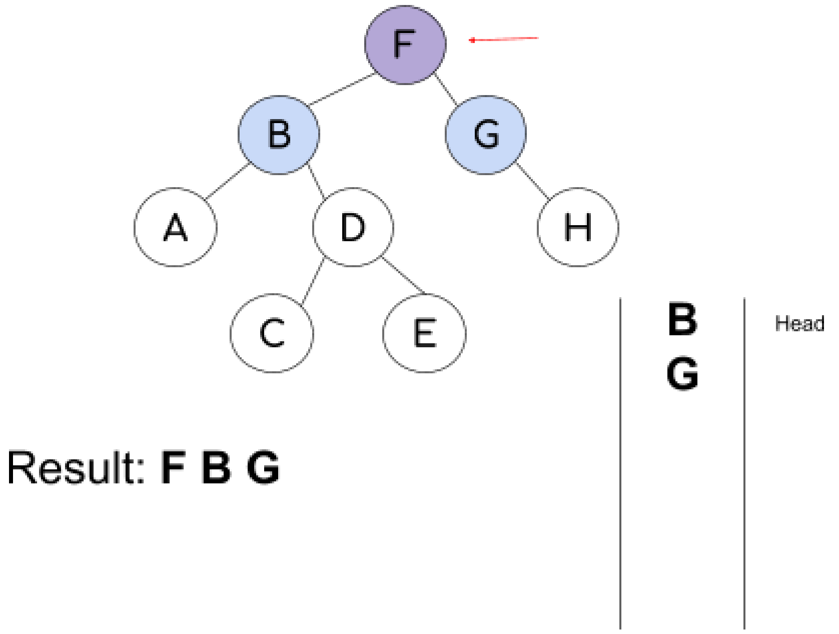


Result: **F**

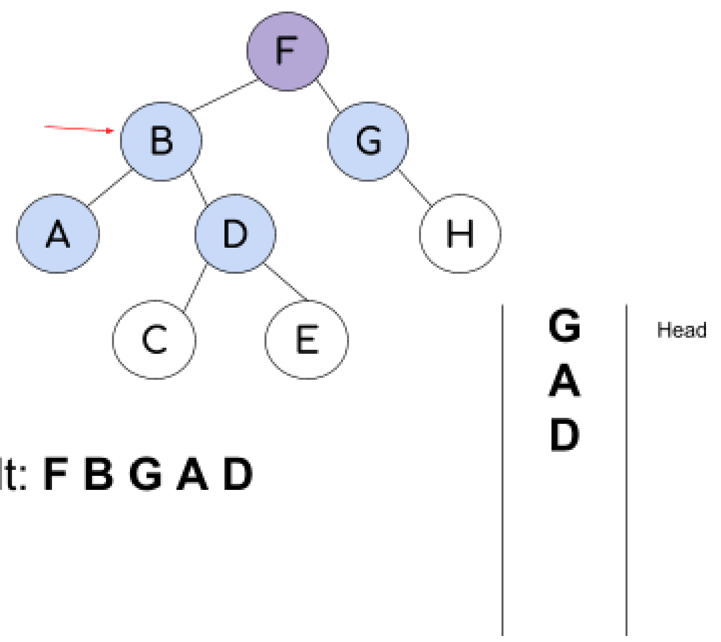
The undiscovered nodes adjacent to F are added to the queue and the result in alphabetical order.



Because all of it’s adjacent nodes are discovered, F can be said to be completely explored (represented by the purple colouring)

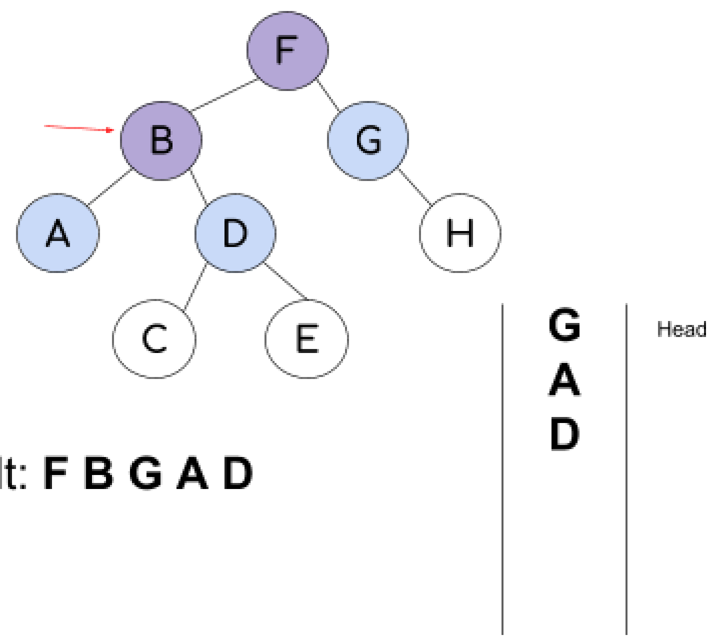


Now that F is completely explored, we can move on to the next node. To do this, we look at the first position of the queue. B is removed from the top of the queue, so this is the next node to be inspected. The undiscovered nodes adjacent to B are added to the queue and results - A and D have been discovered.



Result: **F B G A D**

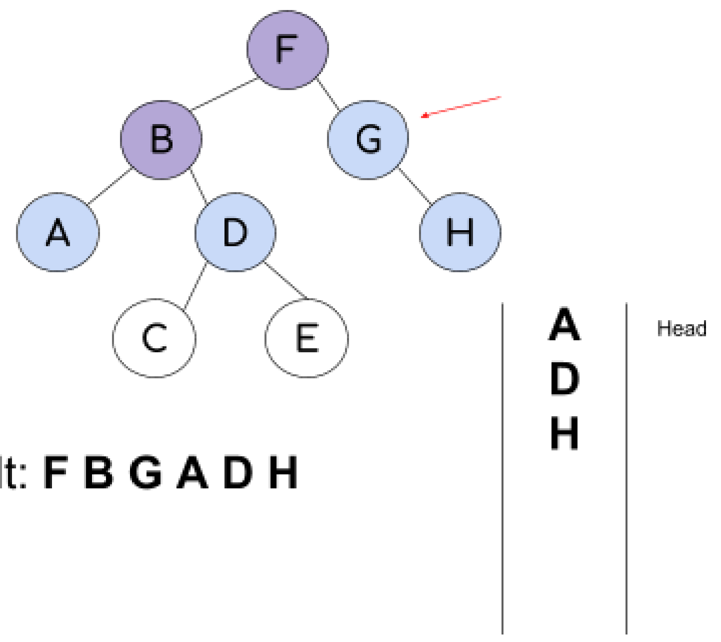
B is now completely discovered.



Result: **F B G A D**

The next item in the queue is removed and inspected.

G has one adjacent undiscovered node. H is added to the result and to the queue.



Result: **F B G A D H**

G is now completely explored.

A is next in the list. It is removed and inspected.

There are no undiscovered vertices adjacent to A, so it is completely explored.

D is the next item in the queue.

D has two adjacent undiscovered nodes which are put into the queue and the result in alphabetical order.

D is completely explored.

The next item in the queue is H.

H has no adjacent undiscovered nodes so it is completely explored.

C is inspected next.

C is completely explored.

Finally, E is at the top of the queue.

E is completely explored.

There are no more items in the queue, so the algorithm terminates and the result is printed.