In graph theory, **breadth-first search (BFS)** is a graph search algorithm that begins at the root node and explores all the neighboring nodes. Then for each of those nearest nodes, it explores their unexplored neighbor nodes, and so on, until it finds the goal.

**How it works**

BFS is an uninformed search method that aims to expand and examine all nodes of a graph or combination of sequences by systematically searching through every solution. In other words, it exhaustively searches the entire graph or sequence without considering the goal until it finds it. It does not use a heuristic algorithm.

From the standpoint of the algorithm, all child nodes obtained by expanding a node are added to a FIFO (i.e., First In, First Out) queue. In typical implementations, nodes that have not yet been examined for their neighbors are placed in some container (such as a queue or linked list) called "open" and then once examined are placed in the container "closed".
Algorithm (informal)

1. Enqueue the root node.
2. Dequeue a node and examine it.
   - If the element sought is found in this node, quit the search and return a result.
   - Otherwise enqueue any successors (the direct child nodes) that have not yet been discovered.
3. If the queue is empty, every node on the graph has been examined — quit the search and return "not found".
4. If the queue is not empty, repeat from Step 2.

Note: Using a stack instead of a queue would turn this algorithm into a depth-first search.

Pseudocode

```
1 procedure BFS(Graph, source):
2     create a queue Q
3     enqueue source onto Q
4     mark source
5     while Q is not empty:
6         dequeue an item from Q into v
7         for each edge e incident on v in Graph:
8             let w be the other end of e
9             if w is not marked:
10                mark w
11                enqueue w onto Q
```

Source URL: http://www.en.wikipedia.org/wiki/Breadth-first_search
Saylor URL: http://www.saylor.org/courses/cs408

Attributed to [Wikipedia]
Features

Space complexity
Since all of the nodes of a level must be saved until their child nodes in the next level have been generated, the space complexity is proportional to the number of nodes at the deepest level. Given a branching factor $b$ and graph depth $d$ the asymptotic space complexity is the number of nodes at the deepest level, $O(b^d)$. When the number of vertices in the graph is known ahead of time, and additional data structures are used to determine which vertices have already been added to the queue, the space complexity can also be expressed as $O(|V|)$ where $|V|$ is the cardinality of the set of vertices. In the worst case the graph has a depth of 1 and all vertices must be stored. Since it is exponential in the depth of the graph, breadth-first search is often impractical for large problems on systems with bounded space.

Time complexity
Since in the worst case breadth-first search has to consider all paths to all possible nodes the time complexity of breadth-first search is $1 + b + b^2 + b^3 + \ldots + b^d$ which is $O(b^d)$. The time complexity can also be expressed as $O(|E| + |V|)$ since every vertex and every edge will be explored in the worst case.

Completeness
Breadth-first search is complete. This means that if there is a solution, breadth-first search will find it regardless of the kind of graph. However, if the graph is infinite and there is no solution breadth-first search will diverge.

Proof of completeness
If the shallowest goal node is at some finite depth say $d$, breadth-first search will eventually find it after expanding all shallower nodes (provided that the branching factor $b$ is finite).[1]

Optimality
For unit-step cost, breadth-first search is optimal. In general breadth-first search is not optimal since it always returns the result with the fewest edges between the start node and the goal node. If the graph is a weighted graph, and therefore has costs associated with each step, a goal next to the start does not have to be the cheapest goal available. This problem is solved by improving breadth-first search to uniform-cost search which considers the path costs. Nevertheless, if the graph is not weighted, and therefore all step costs are equal, breadth-first search will find the nearest and the best solution.

Bias towards nodes of high degree
It has been empirically observed (and analytically shown for random graphs) that incomplete breadth-first search is biased towards nodes of high degree. This makes a breadth-first search sample of a graph very difficult to interpret. For example, a breadth-first sample of 1 million nodes in Facebook (less than 1% of the entire graph) overestimates the average node degree by 240%. [2]

Applications
Breadth-first search can be used to solve many problems in graph theory, for example:

- Finding all nodes within one connected component
- Copying Collection, Cheney's algorithm
- Finding the shortest path between two nodes $u$ and $v$
- Testing a graph for bipartiteness
• (Reverse) Cuthill–McKee mesh numbering
• Ford–Fulkerson method for computing the maximum flow in a flow network
• Serialization/Deserialization of a binary tree vs serialization in sorted order, allows the tree to be re-constructed in an efficient manner.

Finding connected components
The set of nodes reached by a BFS (breadth-first search) form the connected component containing the starting node.

Testing bipartiteness
BFS can be used to test bipartiteness, by starting the search at any vertex and giving alternating labels to the vertices visited during the search. That is, give label 0 to the starting vertex, 1 to all its neighbours, 0 to those neighbours’ neighbours, and so on. If at any step a vertex has (visited) neighbours with the same label as itself, then the graph is not bipartite. If the search ends without such a situation occurring, then the graph is bipartite.

References

External links
• Breadth-First Explanation and Example (http://www.cse.ohio-state.edu/~gurari/course/cis680/cis680Ch14.html#QQ1-46-92)
Article Sources and Contributors

Breadth-first search  Source: http://en.wikipedia.org/w/index.php?oldid=446717394  Contributors: -

Image Sources, Licenses and Contributors


Image:MapGermanyGraph.svg  Source: http://en.wikipedia.org/w/index.php?title=File:MapGermanyGraph.svg  License: Public Domain  Contributors: -


License

Creative Commons Attribution-Share Alike 3.0 Unported
http://creativecommons.org/licenses/by-sa/3.0/

Attributed to [Wikipedia]