Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools
What links here
Related changes
Upload file
Special pages
Permanent link
Page information
Wikidata item
Cite this page

Print/export
Create a book
Download as PDF
Printable version

Languages
Azərbaycanca
Български
Català
Čeština
Deutsch
Eesti
Español
فارسی
Français
한국어
Italiano
עברית
Latviešu
Lietuvių
Македонски
Nederlands
日本語
Norsk bokmål
Polski
Português
Русский
Српски / srpski
Svenska
Tagalog
ไทย
Українська
Tiếng Việt
中文
Edit links

Breadth-first search

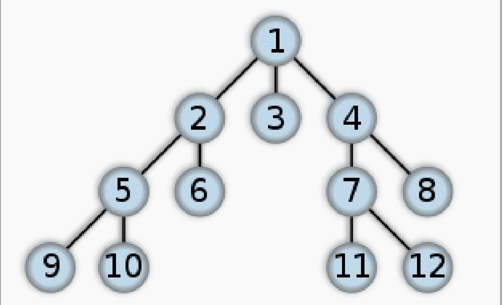
From Wikipedia, the free encyclopedia

This article **needs additional citations for verification**. Please help improve this article by adding citations to reliable sources. Unsourced material may be challenged and removed. *(April 2012)*

Breadth-first search (BFS) is an algorithm for traversing or searching tree or graph data structures. It starts at the tree root (or some arbitrary node of a graph, sometimes referred to as a `search key'[1]) and explores the neighbor nodes first, before moving to the next level neighbors. Compare BFS with the equivalent, but more memory-efficient iterative deepening depth-first search and contrast with depth-first search.

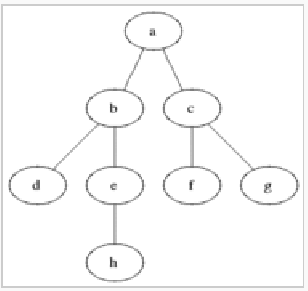
BFS was invented in the late 1950s by E. F. Moore, who used it to find the shortest path out of a maze,[2] and discovered independently by C. Y. Lee as a wire routing algorithm (published 1961).[3][4]

Breadth-first search



Order in which the nodes are expanded

Class	Search algorithm
Data structure	Graph
Worst case performance	$O(\|E\|) = O(b^d)$
Worst case space complexity	$O(\|V\|) = O(b^d)$



Animated example of a breadth-first search

Contents [hide]

Pseudocode [edit]

Input: A graph G and a *starting vertex* v of G

Output: All vertices reachable from v labeled as explored.

A non-recursive implementation of breadth-first search:

```
Breadth-First-Search(G, v):

    for each node n in G:
        n.distance = INFINITY
        n.parent = NIL

    create empty queue Q

    v.distance = 0
    Q.enqueue(v)

    while Q is not empty:

        u = Q.dequeue()

        for each node n that is adjacent to u:
            if n.distance == INFINITY:
```

```
18                  n.distance = u.distance + 1
19                  n.parent = u
20                  Q.enqueue(n)
```

Comments [edit]

Line 3: We iterate through all the nodes of the graph *G*, and we set their distances to *INFINITY*, which simply means that the nodes have not yet been reached, or that their distance to the starting node has not yet been determined. We set also the parent of each node to *NIL*, which means similarly that the parent (or predecessor) of all nodes has not yet been determined.

Line 9: Since *v* is the starting node, its distance from itself is clearly *0*. Note that the starting node *v* has no parent, and *v.parent* was already set to *NIL* in the loop of line 3.

Line 10: The first node to be enqueued is the starting node, because we start exploring its adjacent nodes.

Line 12: The algorithm keeps exploring the nodes until the queue is empty. Note that initially the queue *Q* contains only the starting node *v*, but its adjacent nodes are immediately added to the queue *Q* in the for loop starting at line 16.

Line 14: When we dequeue a node *u* from *Q*, it means that we are going to visit all the adjacent nodes of *u*.

Line 16: We want to visit and enqueue in *Q* all the adjacent nodes to *u* that have not been reached yet. Those are enqueued in line 20 for their future exploration.

More details [edit]

This non-recursive implementation is similar to the non-recursive implementation of depth-first search, but differs from it in two ways:

1. it uses a queue instead of a stack and
2. it checks whether a vertex has been discovered before enqueueing the vertex rather than delaying this check until the vertex is dequeued from the queue.

The *distance* attribute of each vertex (or node) is needed for example when searching for the shortest path between nodes in a graph. At the beginning of the algorithm, the distance of each vertex is set to *INFINITY*, which is just a word that represents the fact that a node has not been reached yet, and therefore it has no distance from the starting vertex. We could have used other symbols, such as *-1*, to represent this concept.

The *parent* attribute of each vertex can also be useful to access the nodes in a shortest path, for example by backtracking from the destination node up to the starting node, once the BFS has been run, and the predecessors nodes have been set.

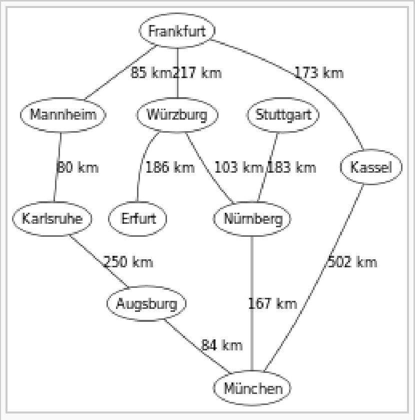
The *NIL* is just a symbol that represents the absence of something, in this case it represents the absence of a parent (or predecessor) node; sometimes instead of the word *NIL*, words such as *null*, *none* or *nothing* can also be used.

Note that the word *node* is usually interchangeable with the word *vertex*.

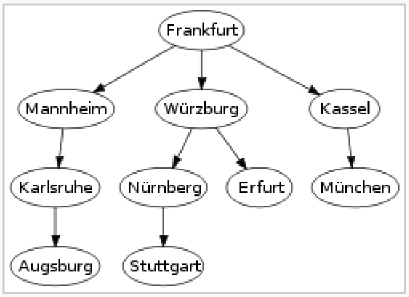
Breadth-first search produces a so-called *breadth first tree*. You can see how a *breadth first tree* looks in the following example.

Example [edit]

The following is an example where *BFS* is applied on map of Germany. You can also see the *breadth-first tree* obtained when running BFS on the same map, starting from *Frankfurt*:



An example map of Germany with some connections between cities



The breadth-first tree obtained when running BFS on the given map and starting in Frankfurt

Analysis [edit]

Time and space complexity [edit]

The time complexity can be expressed as $O(|V|+|E|)$,[5] since every vertex and every edge will be explored in the worst case. $|V|$ is the number of vertices and $|E|$ is the number of edges in the graph. Note that $O(|E|)$ may vary between $O(1)$ and $O(|V|^2)$, depending on how sparse the input graph is.

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 be expressed as $O(|V|)$, where $|V|$ is the cardinality of the set of vertices (as said before). If the graph is represented by an adjacency list it occupies $\Theta(|V|+|E|)$[6] space in memory, while an adjacency matrix representation occupies $\Theta(|V|^2)$.[7]

When working with graphs that are too large to store explicitly (or infinite), it is more practical to describe the complexity of breadth-first search in different terms: to find the nodes that are at distance d from the start node (measured in number of edge traversals), BFS takes $O(b^{d+1})$ time and memory, where b is the "branching factor" of the graph (the average out-degree).[8]:81

Completeness and optimality [edit]

In the analysis of algorithms, the input to breadth-first search is assumed to be a finite graph, represented explicitly as an adjacency list or similar representation. However, in the application of graph traversal methods in artificial intelligence the input may be an implicit representation of an infinite graph. In this context, a search method is described as being complete if it is guaranteed to find a goal state if one exists. Breadth-first search is complete, but depth-first search is not: when applied to infinite graphs represented implicitly, it may get lost in parts of the graph that have no goal state and never return.[9]

Applications [edit]

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

- Copying garbage collection, Cheney's algorithm
- Finding the shortest path between two nodes *u* and *v* (with path length measured by number of edges)
- 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.
- Construction of the *failure function* of the Aho-Corasick pattern matcher.

Testing bipartiteness [edit]

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 neighbors, 0 to those neighbors' neighbors, and so on. If at any step a vertex has (visited) neighbors 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.

Graph and tree

See also [edit]

- Depth-first search
- Iterative deepening depth-first search
- Level structure
- Lexicographic breadth-first search

search algorithms

α–β · A* · B* · Backtracking · Beam · Bellman–Ford · Best-first · Bidirectional · Borůvka · Branch & bound · **BFS** · British Museum · D* · DFS · Depth-limited · Dijkstra · Edmonds · Floyd–Warshall · Fringe search · Hill climbing · IDA* · Iterative deepening · Johnson · Jump point · Kruskal · Lexicographic BFS · Prim · SMA*

Listings

Graph algorithms · *Search algorithms* · *List of graph algorithms*

Related topics

Dynamic programming · Graph traversal · Tree traversal · Search games

v · t · e

References [edit]

1. ^ "Graph500 benchmark specification (for petascale-era supercomputer performance evaluation)". Graph500.org, 2010.
2. ^ Skiena, Steven (2008). *The Algorithm Design Manual*. Springer. p. 480. doi:10.1007/978-1-84800-070-4_4.
3. ^ Leiserson, Charles E.; Schardl, Tao B. (2010). *A Work-Efficient Parallel Breadth-First Search Algorithm (or How to Cope with the Nondeterminism of Reducers)* (PDF). ACM Symp. on Parallelism in Algorithms and Architectures.
4. ^ Lee, C. Y. (1961). "An Algorithm for Path Connections and Its Applications". *IRE Transactions on Electronic Computers*.
5. ^ Cormen, Thomas H., Charles E. Leiserson, and Ronald L. Rivest. p.597
6. ^ Cormen, Thomas H., Charles E. Leiserson, and Ronald L. Rivest. p.590
7. ^ Cormen, Thomas H., Charles E. Leiserson, and Ronald L. Rivest. p.591
8. ^ Russell, Stuart; Norvig, Peter (2003) [1995]. *Artificial Intelligence: A Modern Approach* (2nd ed.). Prentice Hall. ISBN 978-0137903955.
9. ^ Coppin, B. (2004). Artificial intelligence illuminated. Jones & Bartlett Learning. pp. 79–80.

- Knuth, Donald E. (1997), *The Art of Computer Programming Vol 1. 3rd ed.*, Boston: Addison-Wesley, ISBN 0-201-89683-4

External links [edit]

- Breadth-First Explanation and Example
- Open Data Structures - Section 12.3.1 - Breadth-First Search

Wikimedia Commons has media related to ***Breadth-first search***.

Categories: Graph algorithms | Search algorithms