## Breadth First Search

*Description:*

*Breadth First Search* is the simplest Graph Algorithm, and it is also an important prototype of many Graph Algorithm. For example, Prim Algorithm and Dijkstra Algorithm all use the Breadth First Search Algorithm.

For a given Graph G = (V, E), and source node s, Breadth First Search can be used to find all accessible nodes starting from source node s. The algorithm can be used to calculate the distance from source node s to each accessible nodes, and finally generate the *“Breadth First Search Tree”*.

The tree treats the Source Node s as Root Node, including all accessible nodes. For each accessible Node v starting from Source Node u, *in the Breadth First Search Tree, the simple routine from Node s to Node v is the Shortest Routine from Node s to Node v in Graph, which includes the least edge routine.*

*This algorithm can be used in Directed Graph and Undirected Graph.*

Breadth First Search Tree is so famous because the algorithm always enlarge the boundary between Known Node and Unknown Node. Enlarge Breadth First Search Tree through its breadth direction, which means that *the algorithm needs to finish finding all nodes away from node s by k, and then it can find all nodes away from node s by k + 1.*

*Procedure:*

In order to chase the phase of algorithm, Breadth First Search Tree needs to print each node with color white, gray, or black.

During the search process, the node which first met would mean “finding the node”, therefore the color of node changed. Therefore, the color black or gray means the node has been visited. But Breadth First Search Tree would differentiate between the black node and gray node.

If the edge ( u, v ) belongs to E and node u is black, then the node v would be gray or black. *The white node means that the node has not been visited before. The black node u means that all nodes that connected with node u have already been visited. The gray node v means that there still exist white node that connected with node u.*

During the procedure of Breadth First Search, *Adjacent Linked List Structure* has been used. Also we keep three extra information for each node, which are *the Parent Node of the current Node, the distance between the current Node and the source Node s, and the color of the current Node.*

*structure Node {*

*int distance; ( Distance stands for the shortest distance between node s and v. )*

*Node Parent Node; ( Parent Node of the current node v. )*

*string color; ( The color of the current node v. )*

*};*

*Rule:*

1. Starts from the source node s, and visit all nodes in the Adjacent Linked List. As long as the node has been visited and the color of the node equals to white, then we just add the node v and the edge ( u, v ) into the Breadth First Search Tree.

*The distance of the node v = s.distance + 1*

*The parent node of node v = s*

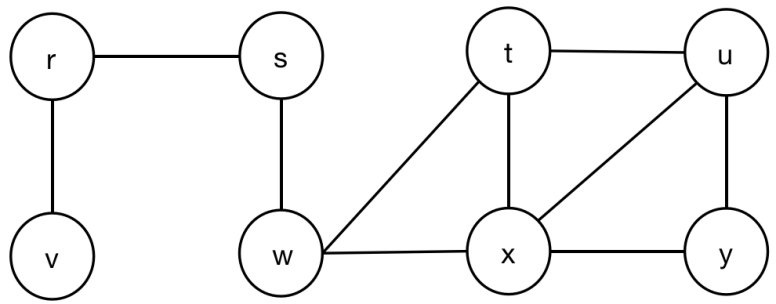
*The color of node v = gray*

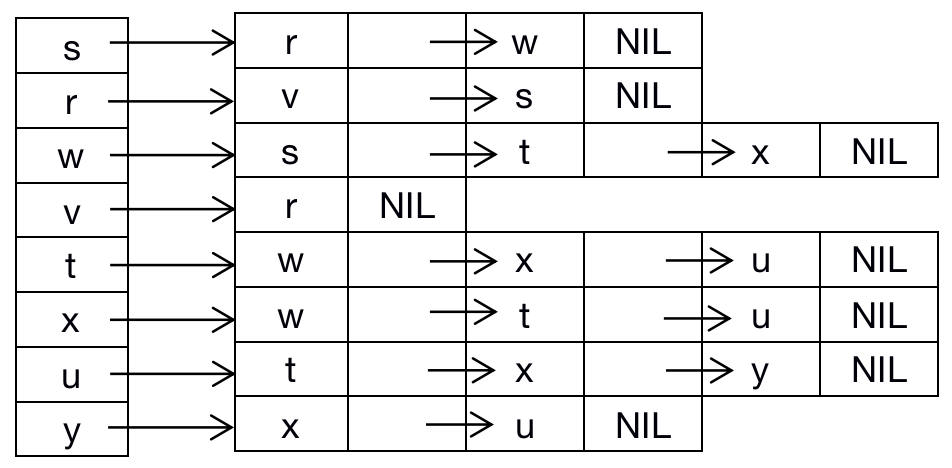
However, we do not really add the node v and the edge ( u, v ) into the Breadth First Search Tree. We just add the node into the queue and update its related information.

1. After finishing finding all nodes which have been connected with node s, then we need to mark the color of node s, which equals to black.
2. Then, in the queue, we need to pop out the next node from queue and finding all nodes which have been connected with it, just as the step 1, after finding all nodes v which connected with the node, then we need to update all related information related with the node, just as the same, the information including *the distance of the node v, the parent node of node v, and the color of node v.*
3. After finishing all these step, just as the same step, we need to update the color black of the node.

*Example:*

The Graph is as below, and it totally has 8 nodes and 10 edges. Through finding all nodes which have been connected with node s, and mark the color of node s, we can finish Breadth First Search Algorithm on the Graph.



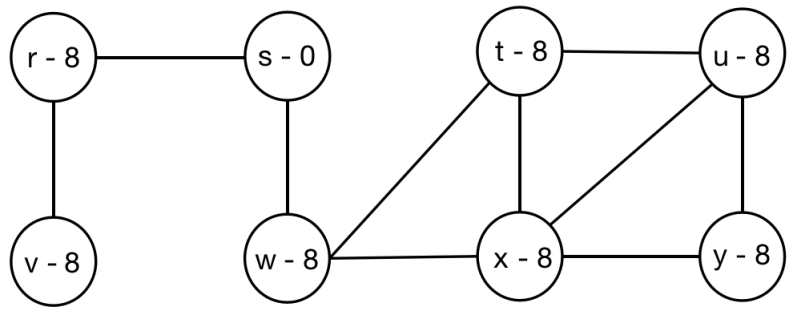


*Step 1:*

Initialize all nodes with extra information, including distance, parenting node, and color of node.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Node v | Node r | Node s | Node w | Node t | Node x | Node u | Node y |
| Dist | Infinite | Infinite | Infinite | Infinite | Infinite | Infinite | Infinite | Infinite |
| Parent | NIL | NIL | NIL | NIL | NIL | NIL | NIL | NIL |
| Color | White | White | White | White | White | White | White | White |

Here, we use 0 to stand for 0 distance between source node s and source node s. Use 8 to stand for Infinite distance between node and source node s.



*Step 2:*

Prepare the queue to store each node into the queue. Also, initialize the queue with the size equals to the number of nodes.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|  |  |  |  |  |  |  |  |

*First Round:*

1. Push the source node s into the queue.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| s |  |  |  |  |  |  |  |

1. Recursively run all below several steps:
2. Pop out the first node s.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|  |  |  |  |  |  |  |  |

1. Update the information table of source node s:

* Distance(s, s) = 0.
* Parent Node(s) = NIL.
* Color(s) = Gray.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Node v | Node r | Node s | Node w | Node t | Node x | Node u | Node y |
| Dist | Infinite | Infinite | 0 | Infinite | Infinite | Infinite | Infinite | Infinite |
| Parent | NIL | NIL | NIL | NIL | NIL | NIL | NIL | NIL |
| Color | White | White | Gray | White | White | White | White | White |

3) Recursively visit all adjacent nodes of source node s.

1. Visit the first adjacent node r, and update all related information of node s.

* Distance(s, r) = 1.
* Parent Node(r) = s.
* Color(s) = Gray.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Node v | Node r | Node s | Node w | Node t | Node x | Node u | Node y |
| Dist | Infinite | 1 | 0 | Infinite | Infinite | Infinite | Infinite | Infinite |
| Parent | NIL | s | NIL | NIL | NIL | NIL | NIL | NIL |
| Color | White | Gray | Gray | White | White | White | White | White |

1. Push the adjacent node r into queue.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| r |  |  |  |  |  |  |  |

1. Visit the second adjacent node w, and update all related information w.

* Distance(s, w) = 1.
* Parent Node(w) = s.
* Color(w) = Gray.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Node v | Node r | Node s | Node w | Node t | Node x | Node u | Node y |
| Dist | Infinite | 1 | 0 | 1 | Infinite | Infinite | Infinite | Infinite |
| Parent | NIL | s | NIL | s | NIL | NIL | NIL | NIL |
| Color | White | Gray | Gray | Gray | White | White | White | White |

1. Push the adjacent node w into queue.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| r | w |  |  |  |  |  |  |

1. Update the color of the first Node s, s.color = Black.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Node v | Node r | Node s | Node w | Node t | Node x | Node u | Node y |
| Dist | Infinite | 1 | 0 | 1 | Infinite | Infinite | Infinite | Infinite |
| Parent | NIL | s | NIL | s | NIL | NIL | NIL | NIL |
| Color | White | Gray | Black | Gray | White | White | White | White |

1. Pop the first node r out of queue.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| w |  |  |  |  |  |  |  |

*Second Round*

*Third Round*

*Forth Round*