```
        remove the first node from queue;
      }
      step = step + 1;
    }
    return -1;        // there is no path from root to target
}
```

1. As shown in the code, in each round, the nodes in the queue are the nodes which are waiting to be processed .
2. After each outer while loop, we are one step farther from the root node . The variable step indicates the distance from the root node and the current node we are visiting.

Template II

Sometimes, it is important to make sure that we never visit a node twice . Otherwise, we might get stuck in an infinite loop, *e.g.* in graph with cycle. If so, we can add a hash set to the code above to solve this problem. Here is the pseudocode after modification:

Java Copy

```java
/**
 * Return the length of the shortest path between root and target node.
 */
int BFS(Node root, Node target) {
    Queue<Node> queue;  // store all nodes which are waiting to be processed
    Set<Node> visited;  // store all the nodes that we've visited
    int step = 0;       // number of steps neeeded from root to current node
    // initialize
    add root to queue;
    add root to visited;
    // BFS
    while (queue is not empty) {
        // iterate the nodes which are already in the queue
        int size = queue.size();
        for (int i = 0; i < size; ++i) {
            Node cur = the first node in queue;
            return step if cur is target;
            for (Node next : the neighbors of cur) {
                if (next is not in visited) {
                    add next to queue;
                    add next to visited;
                }
            }
            remove the first node from queue;
        }
        step = step + 1;
    }
    return -1;        // there is no path from root to target
}
```

> There are some cases where one does not need keep the visited hash set:
>
> 1. You are absolutely sure there is no cycle, for example, in tree traversal;
> 2. You do want to add the node to the queue multiple times.