BFS 宽度优先搜索

介绍

- BFS是迭代形式的遍历，优点是按层展开（找最短路径），缺电是内存占用大，需要一个缓存来存储待展开的节点

使用条件

- 拓扑排序，出现连通块的关键词，分层遍历，简单图最短路径，给定一个变换规则 从初始状态变到终止状态最少几步

复杂度

- Time Complexity: O(n + m)
  - n 是点数， m 是边数
- Space Complexity: O(n)

```
ReturnType bfs (Node startNode) {
  // BFS 必须要用队列 queue, 不要用 stack!
  Queue<Node> queue = new ArrayDeque<>();

  // Hashmap 有两个作用，一个是记录一个点是否被丢进队列，避免重复访问, 另一个是记录 startNode 到其他所有节点的最短距离
  // 如果只求连通性的话，则换成 HashSet 就行
  // node 作为 key 的时候比较的是内存

  Map<Node, Integer> distance = new HashMap<>();

  // 把起点放入队列与哈希表内，如果有多个起点，则都放进去

  queue.offer(startNode);

  distance.put(startNode, 0) // or 1 if neccessary

  // while 队列不空的情况下，不停地从队列中拿出一个点，拓展邻居节点放进队列中

  while (!queue.isEmpty()){

    Node node = queue.poll();

    // 如有明确的终点时，可以在这里加终点的判断
    if(node 是终点) {
      break or return something;
    }

    for(Node neighbor: node.getNeighbors()){
```

```
      if (distance.containsKey(neighbor)) continue;

      queue.offer(neighbor);

      distance.put(neighbor, distance.get(node) + 1);

    }
  }

  return distnace; // 如果需要返回所有点离起点的距离，就返回 HashMap

  return distance.keySet(); // 如果需要返回所有连通的节点，就 return HashMap 里的所有点

  return distance.get(endNode); // 如果需要返回离终点的最短距离

}
```

拓扑排序 BFS 模板

```
List<Node> topologicalSort(List<Node> nodes) {

  // 统计所有点的入度信息，放入 hashmap 里
  Map<Node, Integer> indegrees = getIndegrees(nodes);

  // 将所有入度为 0 的点放到队列中
  Queue<Node> queue = new ArrayDeque<>();

  for (Node node : nodes) {

    if (indegrees.get(node) == 0){
        queue.offer(node);
    }
  }

  List<Node> topoOrder = new ArrayList<>();

  while(!queue.isEmpty()) {

    Node node = queue.poll();

    topoOrder.add(node);

    for (Node neighbor : node.getNeighbors()) {

      // 入度减一
      indegrees.put(neighbor, indegrees.get(neighbor) - 1);

      // 入度减到0 说明不再依赖任何点，可以放入队列里
      if (indegrees.get(neighbor) == 0) {
        queue.offer(neighbor);
      }

    }
```

```
  }

  // 如果queue是空的时候，图里还没有被挖出来，说明存在坏的

  // 有环就没有拓扑序
  if (topoOrder.size() != node.size()) {
    return 没有拓扑序
  }

  Map<Node, Integer> getIndegrees(List<Node> nodes) {

    Map<Node, Integer> counter = new HashMap<>();

    for (Node node: nodes) {
      counter.put(node, 0);
    }

    for (Node node: nodes) {
      for (Node neighbor : node.getNeighbors()) {
        counter.put(neighbor, counter.get(neighbor) + 1);
      }
    }
    return counter;
  }
}
```

BFS模板

1. Initialize a Queue with all starting points, a HashSet to record visited nodes
2. While queue is not empty
   a. Retrieve current queue size as number of nodes in the current level
   b. for each node in current level
      i. Poll out one node
      ii. If this is the node we want, return it
      iii. Offer all its neighbor to the queue if not visited and valid
   c. Increase level

小技巧: 对于2D Matrix的图，matrix[i][j]的neighbors一般都是上下左右4个，所以预先存一个4 direction array可以帮助访问neighbors → directions = {{0, 1}, {1, 0}, {0, -1}, {-1, 0}}

Time Complexity O(V + E)