深度和广度优先搜索：如何找出社交网络中的三度好友关系

在社交网络中有一个六度分割理论，也就是说，你与世界上另一个人的间隔关系不会超过六度。也就是说平均只需要六步就可以联系到任何两个互不相识的人。

一个用户的一度连接用户就是他的好友，二度连接就是他好友的好友，三度连接用户就是他好友的好友。

开篇问题：给你一个用户如何找到这个用户的三度，包括一度，二度，三度好友呢?

这就需要搜索。

什么是"搜索"算法?

算法是作用于具体的数据结构的，深度优先搜索算法和广度优先搜索算法都是基于图的这种数据结构的。

图的搜索算法，最直接的理解就是，在图中找出一个顶点出发，到另一个顶点的路径。

具体的方法有很多，这里讲解的是最直接最暴力的，深度优先和广度优先搜索。

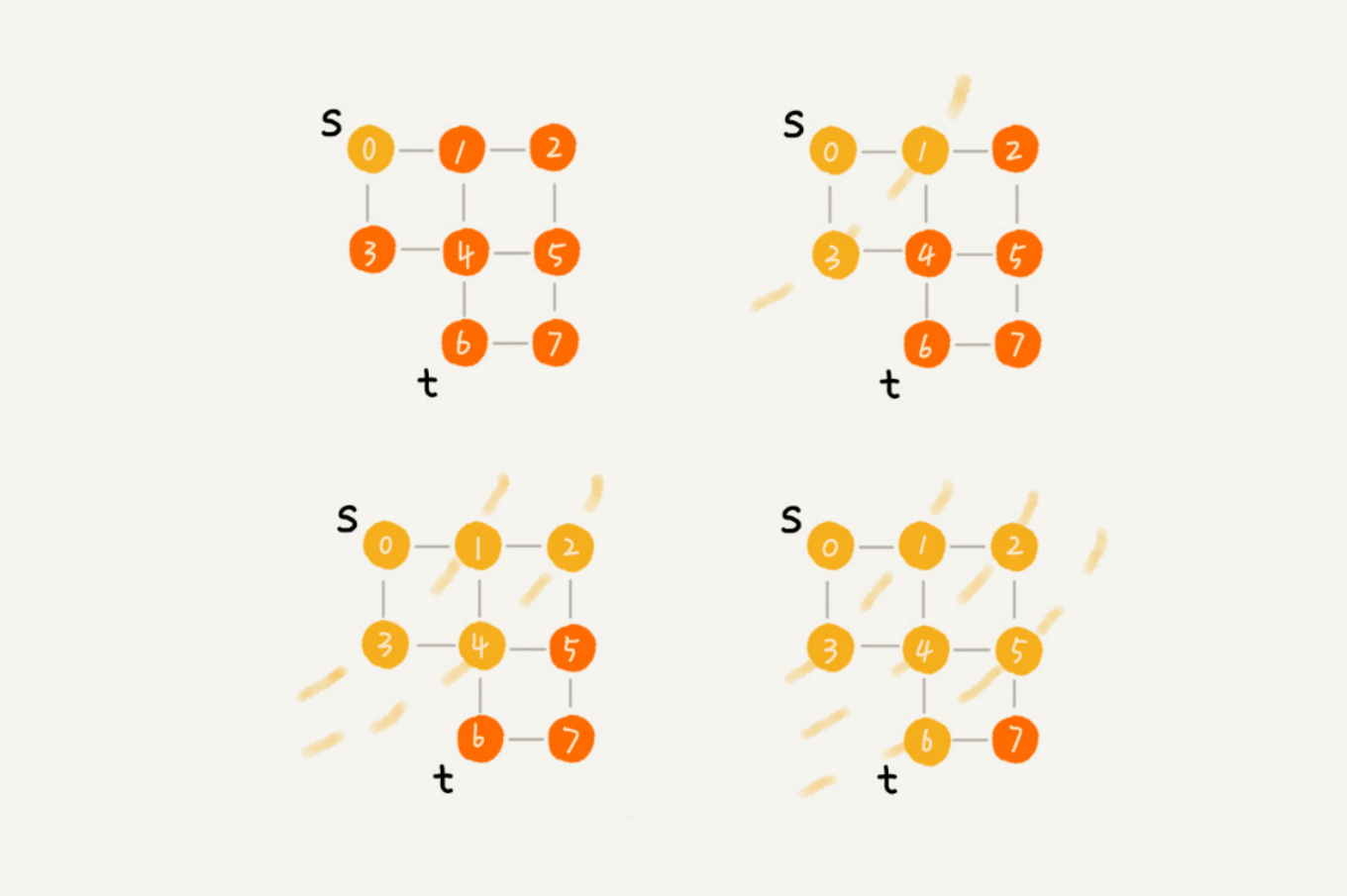
上一节中我们讲述了图的存储方法。邻接矩阵和邻接表。本节中以无向图两讲解； 那么我们先实现一个图的代码；

```
/**
*无向图
*/
public class Graph {
    private int v;
    private LinkedList<Integer> adj[];

    /**
    * 构造函数
    * @param v 无向图中顶点个数
    */
    public Graph(int v){
        this.v = v;
        adj = new LinkedList[v];
        for(int i = 0; i < v; i++){
            adj[i] = new LinkedList<>();
        }
    }

    /**
    * 无向图中增加一条边。需要存储两次
    * @param s 两个顶点之一
    * @param t 两个顶点之二
    */
    public void addEdge(int s,int t){
```

```
            adj[s].add(t);
            adj[t].add(s);
        }
    }
```

广度优先搜索BFS

广度优先搜索，简称BFS，直观的讲就是一种地毯式的层层推进的搜索策略，即先查找距离起始顶点最近的，然后是次近的，依次向外搜索。



正如图中所所示，广度优先搜索，是一层一层依次向外进行搜索的一种搜索策略；

代码实现，其中s表示起始顶点，t表示终止顶点，我们搜索一条S到t的路径，实际上这样的路径就是从s到t的最短路径。

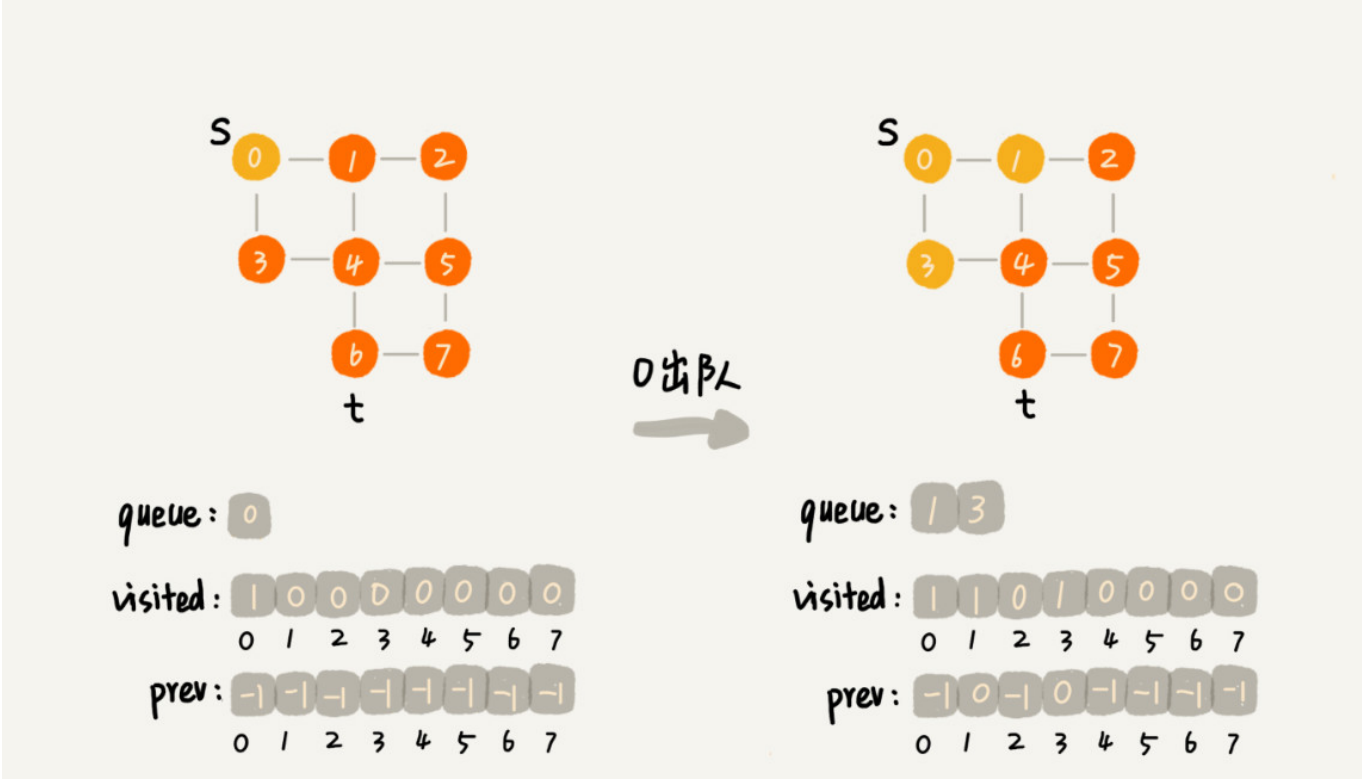
在代码中有三个辅助的变量visited，queue，prev。

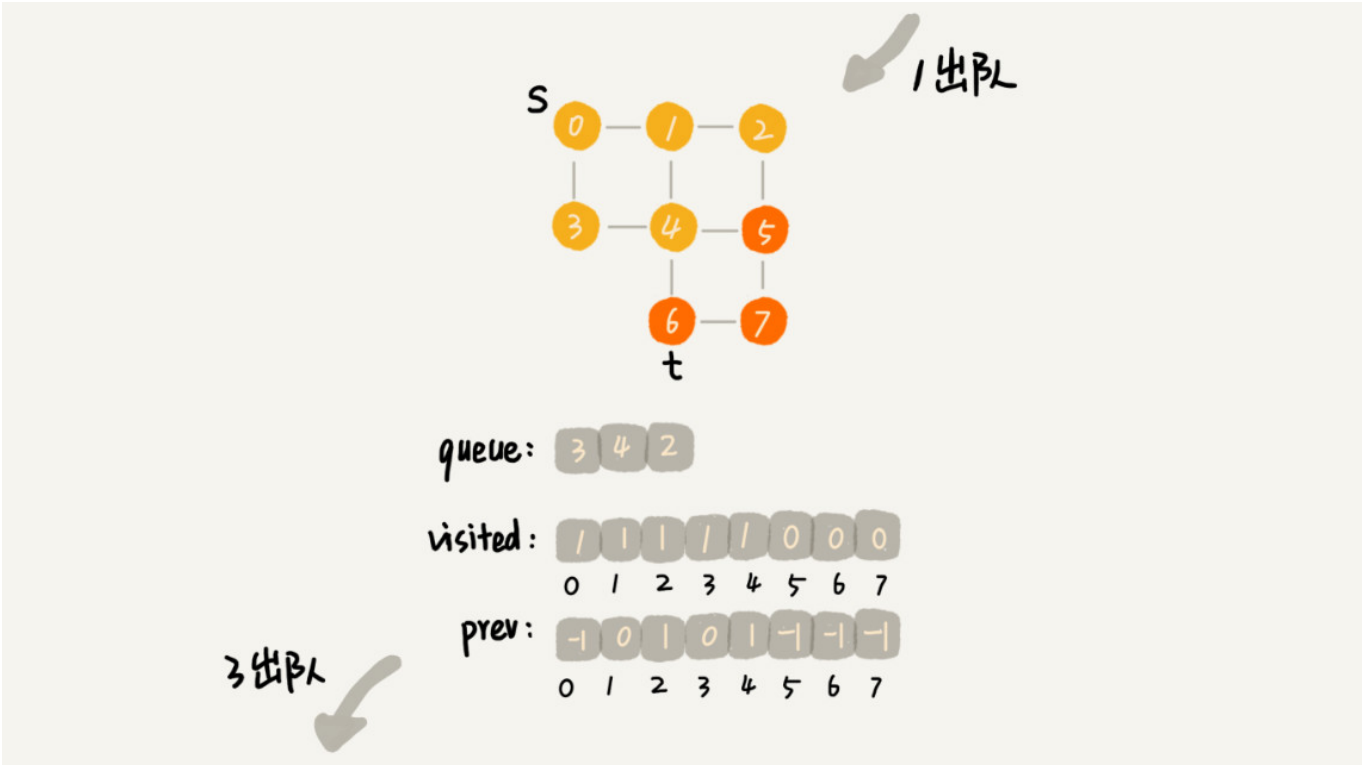
visited是用来记录已经被访问的顶点，用来避免顶点被重复访问。如果q被访问，那么对应的visited[q]将会被设置为true；

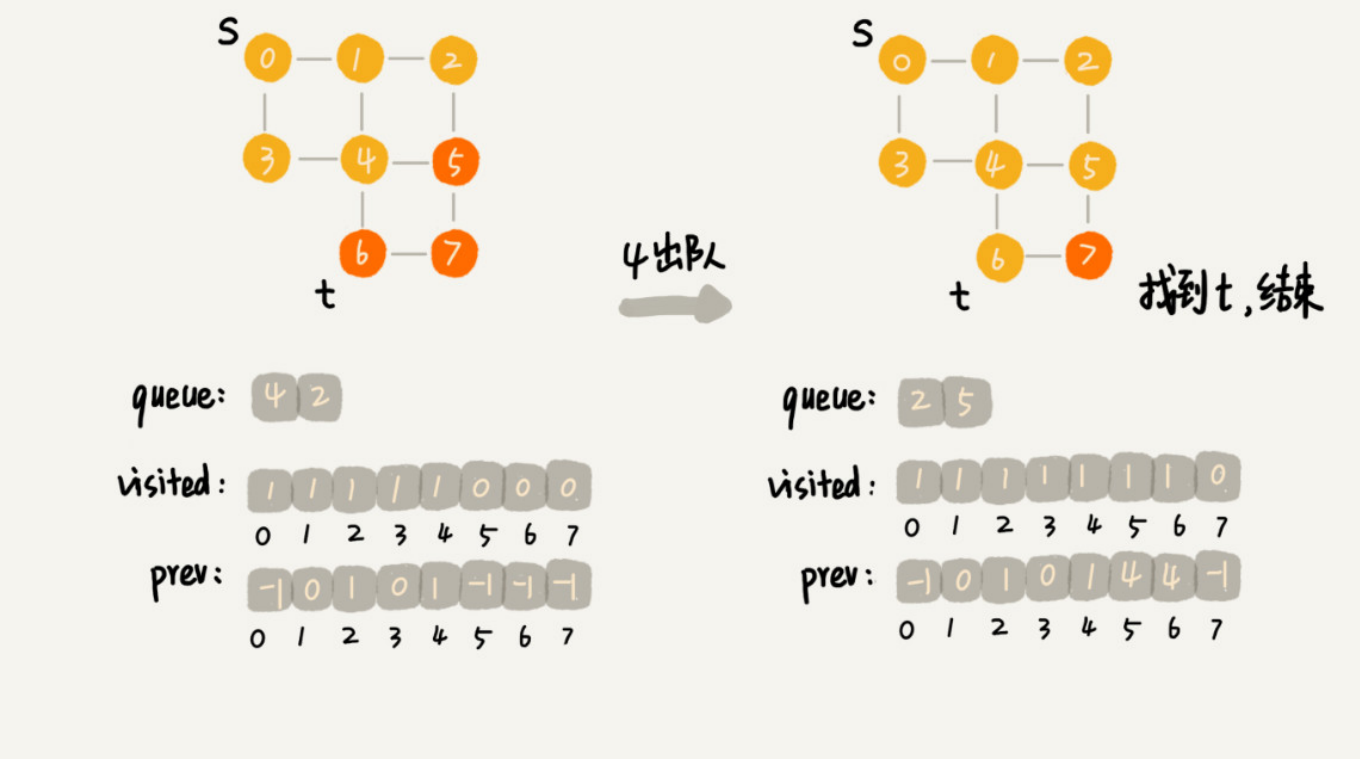
queue是一个队列用来存储已经被访问，但相连的顶点还没有被访问的顶点。因为广度优先搜索是逐层访问的，也就是说，我们需要在访问第k层的顶点的时候，将于第k层相连的k+1层的元素存储起来，才能实现功能。具体思想参照按层遍历树。

prev用来记录搜索路径，当我们从顶点s开始，搜索到顶点t后，prev数组中存储的是搜索的路径，但是这个存储路径是反向存储的，也就是在prev[w]中存储的是顶点w，上一层的也就是w的前驱顶点。所以最后需要递归的来打印路径。

具体的广度搜索过程如图所示：







具体代码过程：

```
/**
 * 广度优先搜索
 * @param s
 * @param t
 */
public void bfs(int s,int t){
    if(s == t) {
        return;
    }
    boolean[] visited = new boolean[v];
    visited[s] = true;
    Queue<Integer> queue = new LinkedList<>();
    queue.add(s);
    int[] prev = new int[v];
    for(int i = 0; i < v; ++i){
        prev[i] = -1;
    }
    while(queue.size() != 0){
        int w = queue.poll();
        for(int i = 0; i < adj[w].size();++i){
            int q = adj[w].get(i);
            if(!visited[q]){
                prev[q] = w;
                if(q == t){
                    print(prev,s,t);
                    return;
                }
                visited[q] = true;
```

```
                queue.add(q);
            }
        }
    }
}

/**
 * 打印路径
 * @param prev
 * @param s
 * @param t
 */
private void print(int[] prev,int s,int t){
    if(prev[t] != -1 && t != s){
        print(prev,s,prev[t]);
    }
    System.out.println(t + " ");
}
```

时间复杂度，空间复杂度 最坏情况下，终止顶点t离顶点s很远，需要遍历整个图才能找到；这个时候每个顶点都要进出一遍队列，每个边也要被访问一遍，所以广度搜索的时间复杂度是O(V+E),其中V表示顶点的个数，E表示边的个数，当然对于一个连通图来说，E大于等于V-1，所以广度优先搜索的时间复杂度可以简写成O(E);

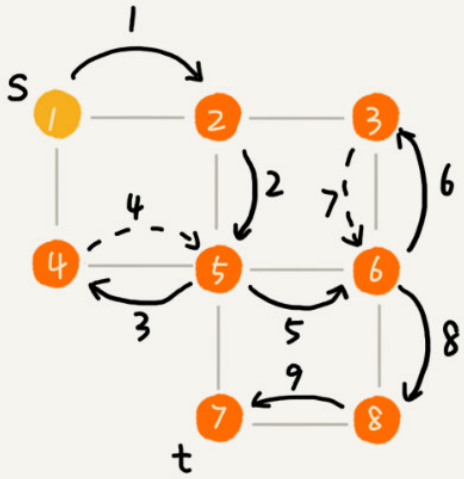
空间复杂度，在搜索过程中几个辅助变量visited数组，queue队列，prev数组上，这三个存储空间的大小都不会超过顶点的个数，所以空间复杂度是O(V);

深度优先搜素DFS

深度优先搜素，简称DFS，最直观的例子就是走迷宫；

假设你站在迷宫的某个岔路口，然后想找到出口，你随意找一个岔路口来走，走着走着发现走不通的时候，你就要回退到上一个岔路口，重现选一条路继续走，知道最终找到出口。这种走法是一种深度优先搜索策略。

如下图所示，起始顶点是s，终止顶点是t，我们希望从图中寻找一条从顶点s到顶点t的路径，如果映射到迷宫的例子，s就是入口，t就是出口。



深度优先搜索用到的是一种比较注明的思想，这种思想解决问题非常适合用递归来实现。

将上面的过程使用递归翻译出来就是下面的代码，深度优先搜索代码实现也用到了prev，visited变量以及print()函数。但是在深度优先搜索中，有一个特殊的变量found，他的作用是当找到终止断点之后，就不再寻找。

代码实现：

```
/**
 * 深度优先搜索
 * @param s
 * @param t
 */
public void dfs(int s,int t){
    found = false;
    boolean[] visited = new boolean[v];
    int[] prev = new int[v];
    for(int i = 0; i < v; i++){
        prev[i] = -1;
    }
    recurDfs(s,t,visited,prev);
    print(prev,s,t);
}

/**
 * 递归搜索该顶点的所有下层顶点，直到搜索到目标节点。
 * @param w
 * @param t
 * @param visited
 * @param prev
 */
private void recurDfs(int w,int t,boolean[] visited,int[] prev){
    if(found == true){
        return;
```

```
        }
        visited[w] = true;
        if(w == t){
            found = true;
            return;
        }
        for(int i = 0; i < adj[w].size();++i){
            int q = adj[w].get(i);
            if(!visited[q]){
                prev[q] = w;
                recurDfs(q,t,visited,prev);
            }
        }
    }
```

时间复杂度，空间复杂度 在前面的图中看到，每条边最多被遍历两次，一次是遍历，一次是回退，所以，图中的深度优先搜索算法复杂度是O(E),E表示的是边的个数。

深度优先搜索算法消耗的内存主要是visited，prev数组和递归调用栈，visited和prev数组的大小和顶点的个数V成正比，递归调用栈最大深度不超过V，所以总的空间复杂度是O(V);

解答开篇

了解了上面两个搜索原理之后，开篇的问题就变的很简单了，开篇需要寻找三度好友，这个问题非常适合用图的广度优先搜索算法来解决，因为广度优先搜索是层层向外推进的。首先遍历与起始点最近的一层顶点，也就是用户的一度好友，然后在遍历用户距离为2的顶点，也就是二度好友，以及用户距离为3的顶点，也就是三度好友。

内容小结

深度和广度是最常用最基本的搜索算法，比起其他高级的搜索算法，也叫做暴力搜索算法。

深度和广度有向搜索的时间复杂度都是O(E),空间复杂度都是O(V);

课后思考

1. 我们通过广度优先搜索算法解决了开篇的问题，你可以思考一下，能否用深度优先搜索来解决呢？ 答：可以在深度优先搜索的递归函数上多传一个参数，就是顶点与初始顶点的距离，这个距离通过每次的递归进行自增，当距离大于规定距离之后就不在进行递归。
2. 学习数据结构最难的不是理解和掌握原理，而是能灵活地将各种场景和问题抽象成对应的数据结构和算法。今天的内容中提到，迷宫可以抽象成图，走迷宫可以抽象成搜索算法，你能具体讲讲，如何将迷宫抽象成一个图吗？或者换个说法，如何在计算机中存储一个迷宫？ 答：将迷宫岔口记为顶点，将两个顶点之间的路径记为边，就可以用邻接矩阵或者邻接表存储在迷宫中了。