31 | 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?

2018-12-03 王争

31 | 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?
朗读人：修阳　10′42″ | 9.81M

上一节我们讲了图的表示方法，讲到如何用有向图、无向图来表示一个社交网络。在社交网络中，有一个六度分割理论，具体是说，你与世界上的另一个人间隔的关系不会超过六度，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很好理解，就是他的好友，二度连接用户就是他好友的好友，三度连接用户就是他好友的好友的好友。在社交网络中，我们往往通过用户之间的连接关系，来实现推荐“可能认识的人”这么一个功能。今天的开篇问题就是，**给你一个用户，如何找出这个用户的所有三度（其中包含一度、二度和三度）好友关系?**

这就要用到今天要讲的深度优先和广度优先搜索算法。

什么是“搜索”算法?

我们知道，算法是作用于具体数据结构之上的，深度优先搜索算法和广度优先搜索算法都是基于“图”这种数据结构的。这是因为，图这种数据结构的表达能力很强，大部分涉及搜索的场景都可以抽象成“图”。

图上的搜索算法，最直接的理解就是，在图中找出从一个顶点出发，到另一个顶点的路径。具体方法有很多，比如今天要讲的两种最简单、最“暴力”的深度优先、广度优先搜索，还有 A*、IDA* 等启发式搜索算法。

我们上一节讲过，图有两种主要存储方法，邻接表和邻接矩阵。今天我会用邻接表来存储图。

我这里先给出图的代码实现。需要说明一下，深度优先搜索算法和广度优先搜索算法，既可以用在无向图，也可以用在有向图上。在今天的讲解中，我都针对无向图来讲解。

复制代码

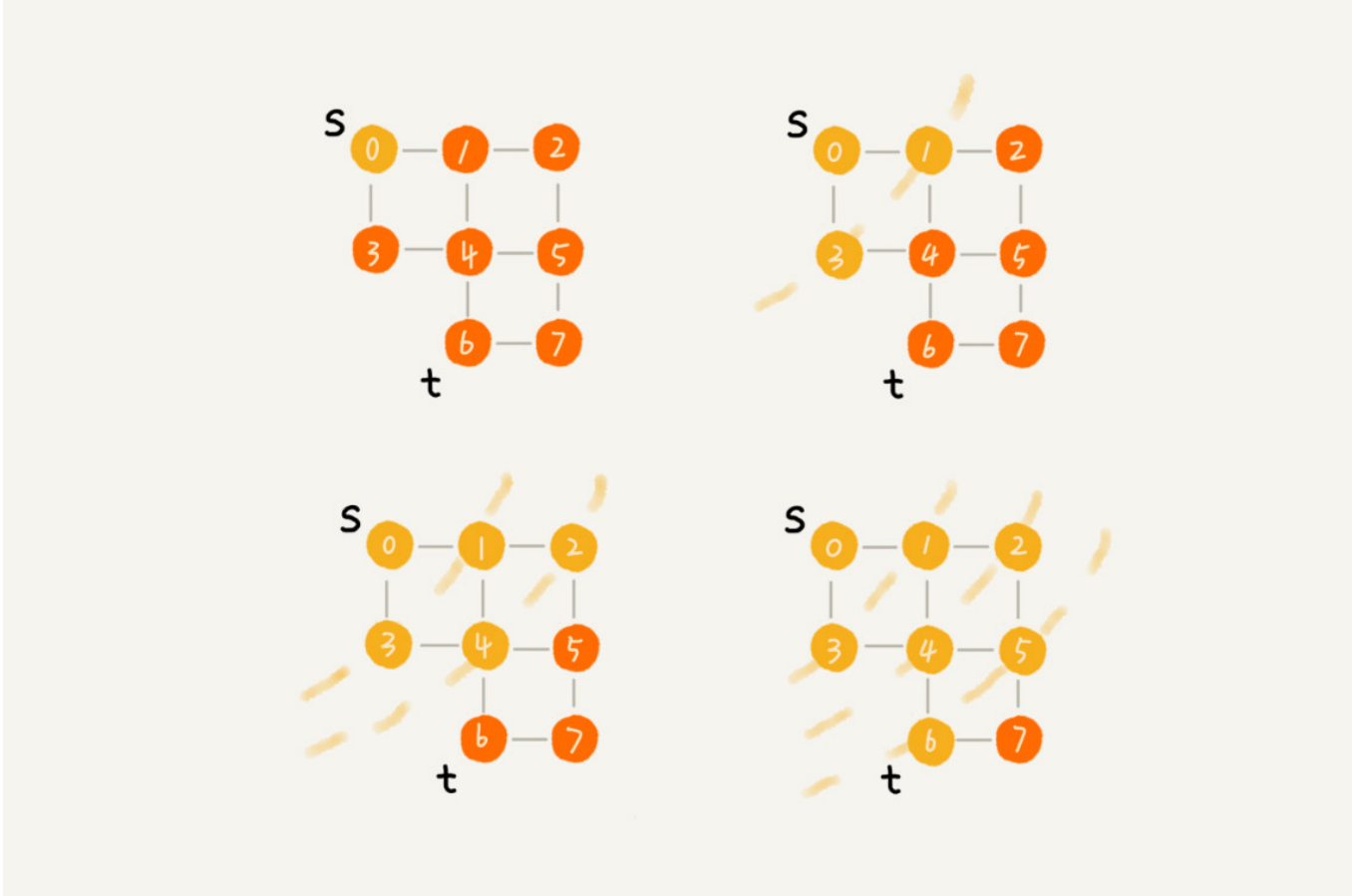
```
public class Graph { // 无向图
  private int v; // 顶点的个数
  private LinkedList<Integer> adj[]; // 邻接表

  public Graph(int v) {
    this.v = v;
    adj = new LinkedList[v];
    for (int i=0; i<v; ++i) {
      adj[i] = new LinkedList<>();
    }
  }

  public void addEdge(int s, int t) { // 无向图一条边存两次
    adj[s].add(t);
    adj[t].add(s);
  }
}
```

广度优先搜索（BFS）

广度优先搜索（Breadth-First-Search），我们平常都把简称为 BFS。直观地讲，它其实就是一种“地毯式”层层推进的搜索策略，即先查找离起始顶点最近的，然后是次近的，依次往外搜索。理解起来并不难，所以我画了一张示意图，你可以看下。



尽管广度优先搜索的原理挺简单，但代码实现还是稍微有点复杂度。所以，我们重点讲一下它的代码实现。

这里面，bfs() 函数就是基于之前定义的，图的广度优先搜索的代码实现。其中 s 表示起始顶点，t 表示终止顶点。我们搜索一条从 s 到 t 的路径。实际上，这样求得的路径就是从 s 到 t 的最短路径。

复制代码

```
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);
      }
    }
  }
}

private void print(int[] prev, int s, int t) { // 递归打印 s->t 的路径
  if (prev[t] != -1 && t != s) {
    print(prev, s, prev[t]);
  }
  System.out.print(t + " ");
}
```

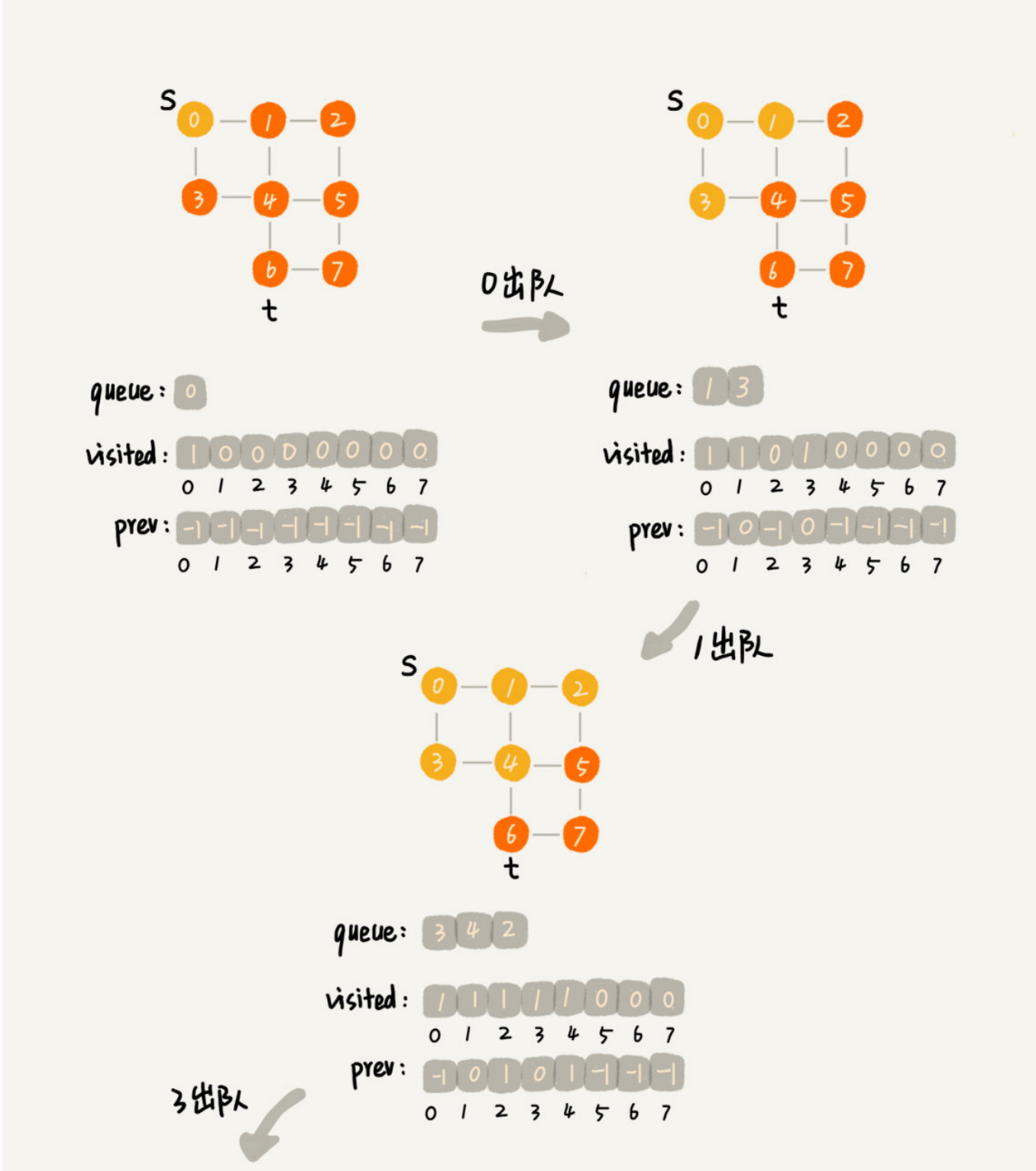
这段代码不是很好理解，里面有三个重要的辅助变量 visited、queue、prev。只要理解这三个变量，读懂这段代码估计就没什么问题了。

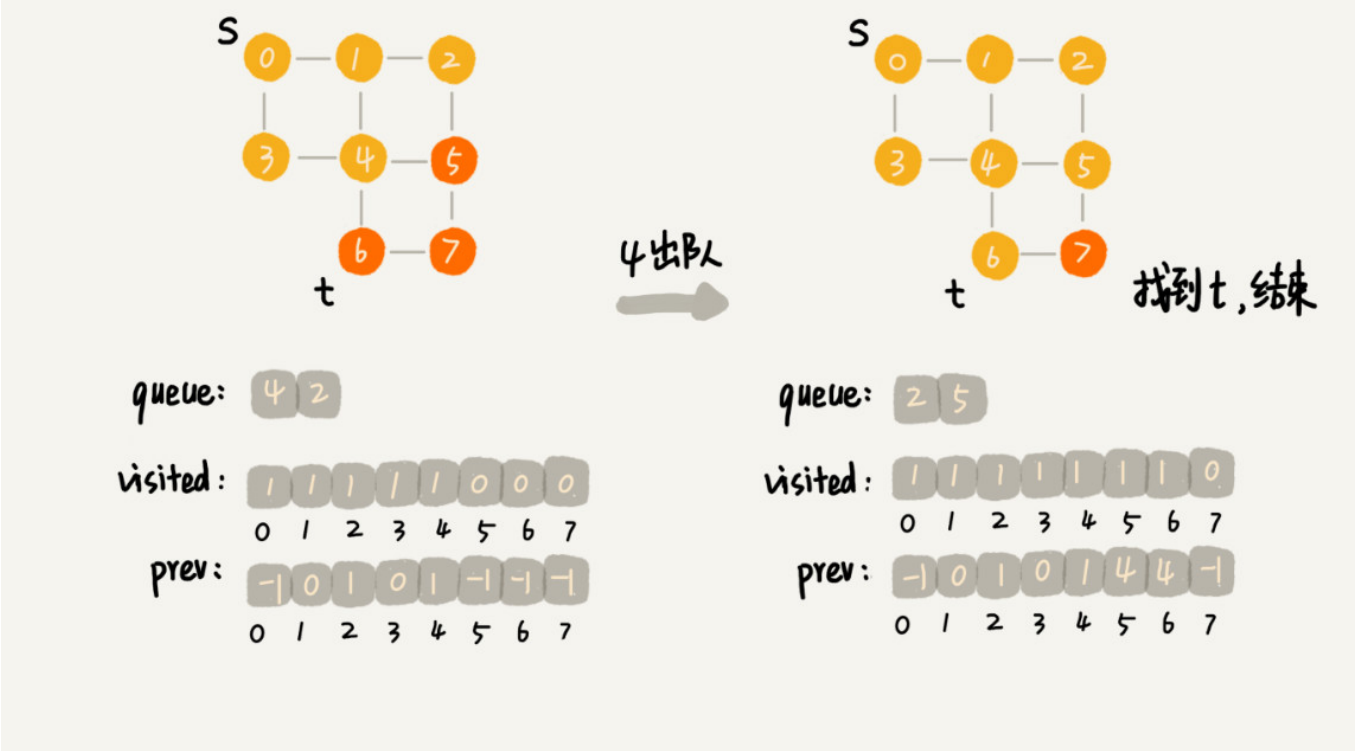
visited是用来记录已经被访问的顶点，用来避免顶点被重复访问。如果顶点 q 被访问，那相应的 visited[q] 会被设置为 true。

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，我们只有把第 k 层的顶点都访问完成之后，才能访问第 k+1 层的顶点。当我们访问到第 k 层的顶点的时候，我们需要把第 k 层的顶点记录下来，稍后才能通过第 k 层的顶点来找第 k+1 层的顶点。所以，我们用这个队列来实现记录的功能。

prev用来记录搜索路径。当我们从顶点 s 开始，广度优先搜索到顶点 t 后，prev 数组中存储的就是搜索的路径。不过，这个路径是反向存储的。prev[w] 存储的是，顶点 w 是从哪个前驱顶点遍历过来的。比如，我们通过顶点 2 的邻接表访问到顶点 3，那 prev[3] 就等于 2。为了正向打印出路径，我们需要递归地来打印，你可以看下 print() 函数的实现方式。

为了方便你理解，我画了一个广度优先搜索的分解图，你可以结合着代码以及我的讲解一块儿看。


s
0
1
2
3
4
5
6
7
t
0出队
queue: 0
visited: 1 0 0 0 0 0 0 0
prev: -1 -1 -1 -1 -1 -1 -1 -1
queue: 1 3
visited: 1 1 0 1 0 0 0 0
prev: -1 0 -1 0 -1 -1 -1 -1
1出队
queue: 3 4 2
visited: 1 1 1 1 1 0 0 0
prev: -1 0 1 0 1 -1 -1 -1
3出队



掌握了广优先搜索算法的原理，我们来看下，广度优先搜索的时间、空间复杂度是多少呢?

最坏情况下，终止顶点 t 离起始顶点 s 很远，需要遍历完整个图才能找到。这个时候，每个顶点都要进出一遍队列，每个边也都会被访问一次，所以，广度优先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，也就是说一个图中的所有顶点都是连通的，E 肯定要大于等于 V-1，所以，广度优先搜索的时间复杂度也可以简写为 O(E)。

广度优先搜索的空间消耗主要在几个辅助变量 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)。

深度优先搜索（DFS）

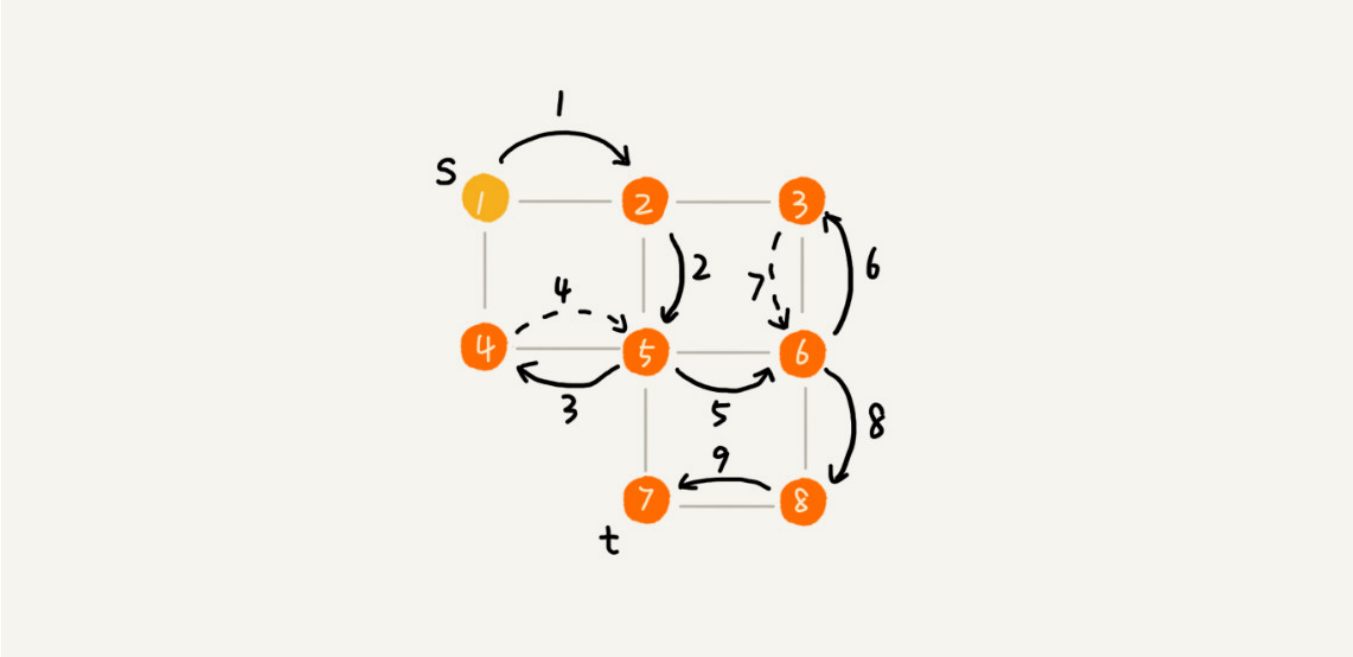
深度优先搜索（Depth-First-Search），简称 DFS。最直观的例子就是“走迷宫”。

假设你站在迷宫的某个岔路口，然后想找到出口。你随意选择一个岔路口来走，走着走着发现走不通的时候，你就回退到上一个岔路口，重新选择一条路继续走，直到最终找到出口。这种走法就是一种深度优先搜索策略。

走迷宫的例子很容易能看懂，我们现在再来看下，如何在图中应用深度优先搜索，来找某个顶点到另一个顶点的路径。

你可以看我画的这幅图。搜索的起始顶点是 s，终止顶点是 t，我们希望在图中寻找一条从顶点 s 到顶点 t 的路径。如果映射到迷宫那个例子，s 就是你起始所在的位置，t 就是出口。

我用深度递归算法，把整个搜索的路径标记出来了。这里面实线箭头表示遍历，虚线箭头表示回退。从图中我们可以看出，深度优先搜索找出来的路径，并不是顶点 s 到顶点 t 的最短路径。



实际上，深度优先搜索用的是一种比较著名的算法思想，回溯思想。这种思想解决问题的过程，非常适合用递归来实现。回溯思想我们后面会有专门的一节来讲，我们现在还回到深度优先搜索算法上。

我把上面的过程用递归来翻译出来，就是下面这个样子。我们发现，深度优先搜索代码实现也用到了 prev、visited 变量以及 print() 函数，它们跟广度优先搜索代码实现里的作用是一样的。不过，深度优先搜索代码实现里，有个比较特殊的变量 found，它的作用是，当我们已经找到终止顶点 t 之后，我们就不再递归地继续查找了。

复制代码

```
boolean found = false; // 全局变量或者类成员变量

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);
}

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]) {
```

```
24        prev[q] = w;
25        recurDfs(q, t, visited, prev);
26      }
27    }
28  }
```

理解了深度优先搜索算法之后，我们来看，深度度优先搜索的时、空间间复杂度是多少呢?

从我前面画的图可以看出，每条边最多会被访问两次，一次是遍历，一次是回退。所以，图上的深度优先搜索算法的时间复杂度是 O(E)，E 表示边的个数。

深度优先搜索算法的消耗内存主要是 visited、prev 数组和递归调用栈。visited、prev 数组的大小跟顶点的个数 V 成正比，递归调用栈的最大深度不会超过顶点的个数，所以总的空间复杂度就是 O(V)。

解答开篇

了解了深度优先搜索和广度优先搜索的原理之后，开篇的问题是不是变得很简单了呢？我们现在来一起看下，如何找出社交网络中某个用户的三度好友关系?

上一节我们讲过，社交网络可以用图来表示。这个问题就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往外推进的。首先，遍历与起始顶点最近的一层顶点，也就是用户的一度好友，然后再遍历与用户距离的边数为 2 的顶点，也就是二度好友关系，以及与用户距离的边数为 3 的顶点，也就是三度好友关系。

我们只需要稍加改造一下广度优先搜索代码，用一个数组来记录每个顶点与起始顶点的距离，非常容易就可以找出三度好友关系。

内容小结

广度优先搜索和深度优先搜索是图上的两种最常用、最基本的搜索算法，比起其他高级的搜索算法，比如 A*、IDA* 等，要简单粗暴，没有什么优化，所以，也被叫作暴力搜索算法。所以，这两种搜索算法仅适用于状态空间不大，也就是说图不大的搜索。

广度优先搜索，通俗的理解就是，地毯式层层推进，从起始顶点开始，依次往外遍历。广度优先搜索需要借助队列来实现，遍历得到的路径就是，起始顶点到终止顶点的最短路径。深度优先搜索用的是回溯思想，非常适合用递归实现。换种说法，深度优先搜索是借助栈来实现的。在执行效率方面，深度优先和广度优先搜索的时间复杂度都是 O(E)，空间复杂度是 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?

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？

欢迎留言和我分享，也欢迎点击“请朋友读”，把今天的内容分享给你的好友，和他一起讨论、学习。

上一篇 30 | 图的表示：如何存储微博、微信等社交网络中的好友关系？

下一篇 32 | 字符串匹配基础（上）：如何借助哈希算法实现高效字符串匹配？

写留言

精选留言

P@tricK 6

思考题：

1. 可以。DFS递归时传多一个离初始节点的距离值，访问节点时，距离超过3的不再继续递归

2. 初始化两个顶点为迷宫起点和终点，从起点开始，遇到分叉点，为每个分支都新建一个节点，并和前一节点连接，递归每个分支直到终点

2018-12-03

五岳寻仙 4

课后思考题：

1. 深度优先用于寻找3度好友，可以设定搜索的深度，到3就向上回溯。正如文中提到的，可能不是最短路径，所以会涉及到更新结点度的问题。

2. 关于迷宫存储问题。类似于欧拉七桥问题，需要将迷宫抽象成图，每个分叉路口作为顶点，顶点之间连成边，构成一张无向图，可以存储在邻接矩阵或邻接表中。

2018-12-03

Jerry银银 3

朗读者原谅我的有强迫症：queue这个单词读错了，付上正确的音标如下［kju］

我觉得避免这种问题，有个方法就是，朗读之前，逐个查询一下单词的正确发音，一篇文章中的单词屈指可数，这个工作量按理说应该不大。

但是这个简单的举措，能大大提高听文章的体验，不然听起来总觉得很怪

往大了说影响，毕竟咱们极客时间做的是知识，做的是学问

2018-12-03

作者回复

2018-12-05

韩 3

我的想法:迷宫可以用带权无向图存储，每个多岔路口是一个顶点，相邻的多岔路口是一条边。带权是为了记录两个顶点间的距离。

但用上面这种方式，会丢失一些拐点信息。可以结合业务场景，如果需要这些信息，就把拐点也作为一个顶点存储。

2018-12-03

猫头鹰爱拿铁 1

我感觉用深度优先查找人脉的算法可能会把小于这个度数的数据查找出来吧，例如查找度数为3的顶点，用文中广度优先搜索的那个数据，使用深度优先的算法会把0，1，4，3也给查出来。。。

2018-12-03

Allen Zou 1

老师，深度优先代码中recurseDfs函数中循环的每次迭代加found 判断可以少一些函数调用

2018-12-03

。。。 0

谢谢老师 以前从来没有了解过图，老师通俗的讲解过后发现原来是那么的简单。这个专栏买的是真的超值。数据机构和算法入门最佳专栏啊

2018-12-06

梅坊帝卿 0

从我前面画的图可以看出，每条边最多会被访问两次，一次是遍历，一次是回退。所以，图上的深度优先搜索算法的时间复杂度是 O(E)，E 表示边的个数。

这里不太理解，从代码中看来 假设当前处理N 那么遍历它的孩子 递归调用之 这里递归调用后返回算是回退？从N的视角 它只是依次处理这些孩子
如果用栈来实现非递归 那么会先压栈所有孩子 再去最顶的孩子继续处理 有一个压栈和出栈 这就匹配了
所以在递归调用中 函数返回了 也可以等同于回退一条边

2018-12-05

+ + 0

广度优先 有点像树中的层序遍历啊 也是借助一个队列来实现的

2018-12-05

+ + 0

4出队时 当找到t顶点时

Queue: 2, 5
Visited: 1,1,1,1,1,1,1,0
Prev: -1,0,1,0,1,4,4,-1

感觉下标为6时 visited应该是0 即没有访问过
因为已经return了
所以visited[q] = true; 这句代码没有被执行

2018-12-05

jon 0

广度搜索用的是一个队列存储已经比较过的一层点点，深度搜索用的是递归也就是栈来存储搜索过的顶点

2018-12-05

他在她城断了弦 0

可不可以这样理解，深度优先用递归思想，广度优先用迭代思想？

2018-12-04

作者回复

不大行 因为深度优先也可以用非递归的方式实现

2018-12-05

Sharry 0

1. 设置搜索的深度为 3，超过了 3 则强行回溯
2. 将迷宫的分叉路口映射成为图中的顶点

2018-12-04

meng 0

提个小建议，代码里有些变量名比如 q, v, t，总是不太清楚它们的意思，能否起个更有意义的名字？

2018-12-04

作者回复

是不大符合命名规范 但为了代码简洁 故意这么写的

2018-12-05

許敲敲 0

想看看python的代码实现

2018-12-04

MIAN-勉 0

思考題2 本質上是構建圖的過程。而且衹知道一個起點節點，如何將遇到的分叉口抽象為一個節點是難點，看了所有的留言答案，感覺大家都避重就輕，沒答到點上。希望老師釋惑。

2018-12-03

Jerry银银 0

留言笔记不小心被自己删了（问了客服，也不能找回了），补上：

1. 可以用深度遍历，每次遍历到三度人脉，再回溯到上层节点，直到所有的三度人脉都找完。

2. 将迷宫的每个岔口记为"顶点"，岔口之间的路径记为"边"，可以用邻接表存储，也可以用邻接矩阵存储。但是个人感觉，像那种标准的方格迷宫，适合用邻接矩阵存储，因为稠密度比较高。

2018-12-03

nothing 0

老师想问问你，您觉得抽象数据类型重要吗，算法实现采用ADT来实现是否重要呢，因为我看到网上的还是您写的一些代码很多都是直接面向物理存储结构，比如数组链表。 但是我们老师在课上都要求我们一定要用ADT，但是我对这个不是很理解。

2018-12-03

作者回复

这个没有非得说要不要用adt吧 但是如果场景非常符合adt的特性 建议还是用adt。可以参考栈那一节的说明

2018-12-04

Smallfly 0

老师把广度和深度优先搜索讲的真的是通俗易懂。我有个问题是，如果图中存在相同的节点，两种算法是不是就不能工作了？

2018-12-03

作者回复

可以的。看你用什么信息去查找 以及怎么定义找到。所谓相同节点 这个说法有点笼统了 怎样才算两个节点相同呢

2018-12-05

Smallfly 0

广度搜索图， 4 出队列的时候 queue 应该是 2 5 6 吧

2018-12-03

作者回复

因为6已经是终止节点了 所以不用再入队列了

2018-12-04

🐱 您的好友William🐱 0

走迷宫就是现在地图类APP干的事情吧，我知道他们用A*和A*优化版。我自己想的是把每个路口设置成vertex然后按vertex间的路程时间长短构建有向带权图，然后在这个图里面找最短路径，可以使用狄克斯特拉求解，但是缺点是图中不能是无向的，也不能有环。最正规的解法应该是A*，我之前专门学了一下，没学明白。。。。

用深度优先搜索好友我们只要控制好每次深度不超过3，达到3就回溯应该就可以了。

2018-12-03

NeverMore 0

深度与广度应该都是可以互相实现的，主要复杂度问题。深度主要使用栈，广度主要使用堆。

2018-12-03