

九章算法班2021版直播第4章

性价比之王的宽度优先搜索

主讲：夏天

扫一扫 加我免费送课





今日看题 0 分钟

随机一题

我的题单 | 收藏题解 | 学习笔记

类型筛选 | 题目标签 | 综合排序

1905 · 字符删除

简单 · 55% 通过率 · 3 题解

1904 · 放小球

困难 · 51% 通过率 · 3 题解

1903 · 部门统计

简单 · 45% 通过率 · 6 题解

1902 · 寻找Google

简单 · 43% 通过率 · 3 题解

1901 · 有序数组的平方

简单 · 76% 通过率 · 5 题解

1900 · 基因相似度

困难 · 46% 通过率 · 3 题解

输入评论...

137 · 克隆图

题目 | 题解 | 笔记

137 · 克隆图

脸书 Depth-first Search 优步 谷歌 哈希表
哈希表 Breadth-first Search 哈希表 脸书 谷歌
Pocket Gems

中等 · 37% 通过率

添加题单

描述: study322
克隆一张无向图. 无向图的每个节点包含一个 label 和一个列表 neighbors. 保证每个节点的 label 互不相同.

你的程序需要返回一个经过深度拷贝的新图. 新图和原图具有同样的结构, 并且对新图的任何改动不会对原图造成任何影响.

说明
你需要返回与给定节点具有相同 label 的那个节点.

示例

样例1

```

    输入:
    {1,2,4#2,1,4#4,1,2}
    输出:
    {1,2,4#2,1,4#4,1,2}
    解释:
    1-----2
  
```

输入评论...

137 · 克隆图

题目 | 题解 | 笔记

137 · 克隆图

脸书 Depth-first Search 优步 谷歌 哈希表
哈希表 Breadth-first Search 哈希表 脸书 谷歌
Pocket Gems

中等 · 37% 通过率

添加题单

描述: study322
克隆一张无向图. 无向图的每个节点包含一个 label 和一个列表 neighbors. 保证每个节点的 label 互不相同.

你的程序需要返回一个经过深度拷贝的新图. 新图和原图具有同样的结构, 并且对新图的任何改动不会对原图造成任何影响.

说明
你需要返回与给定节点具有相同 label 的那个节点.

示例

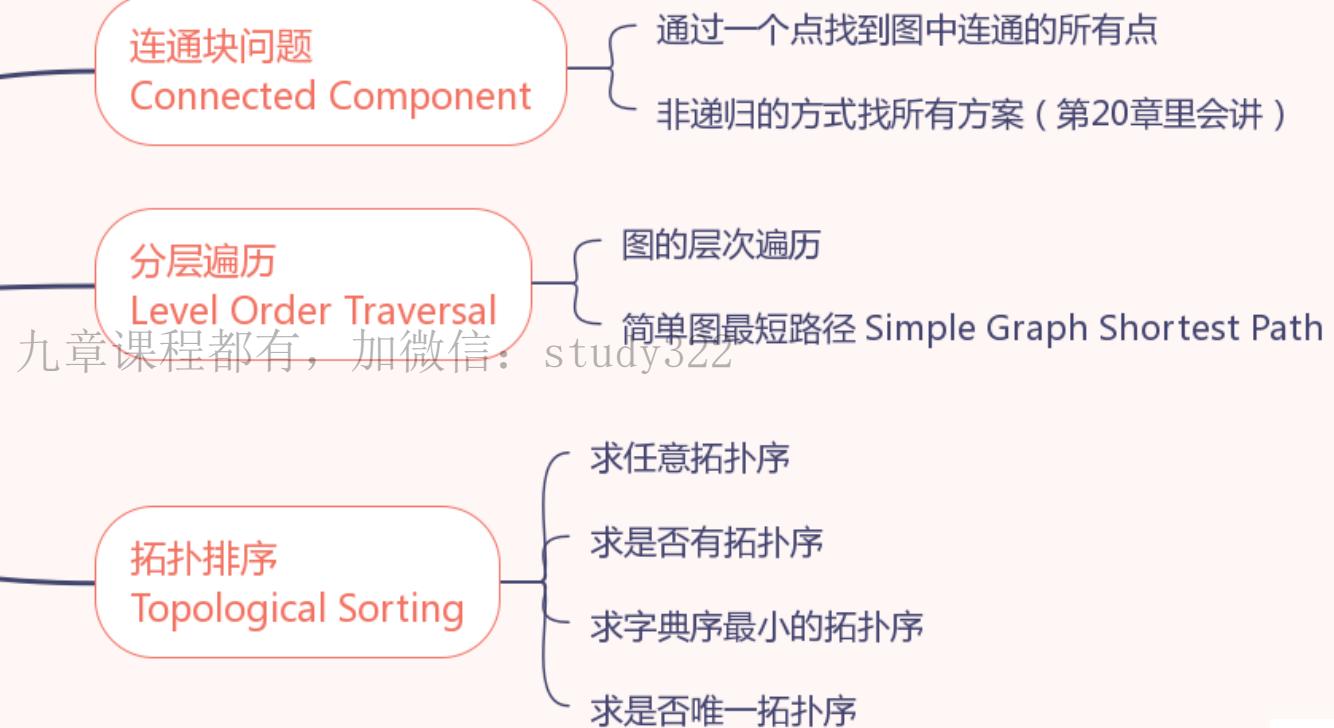
样例1

```

    输入:
    {1,2,4#2,1,4#4,1,2}
    输出:
    {1,2,4#2,1,4#4,1,2}
    解释:
    1-----2
  
```

输入评论...

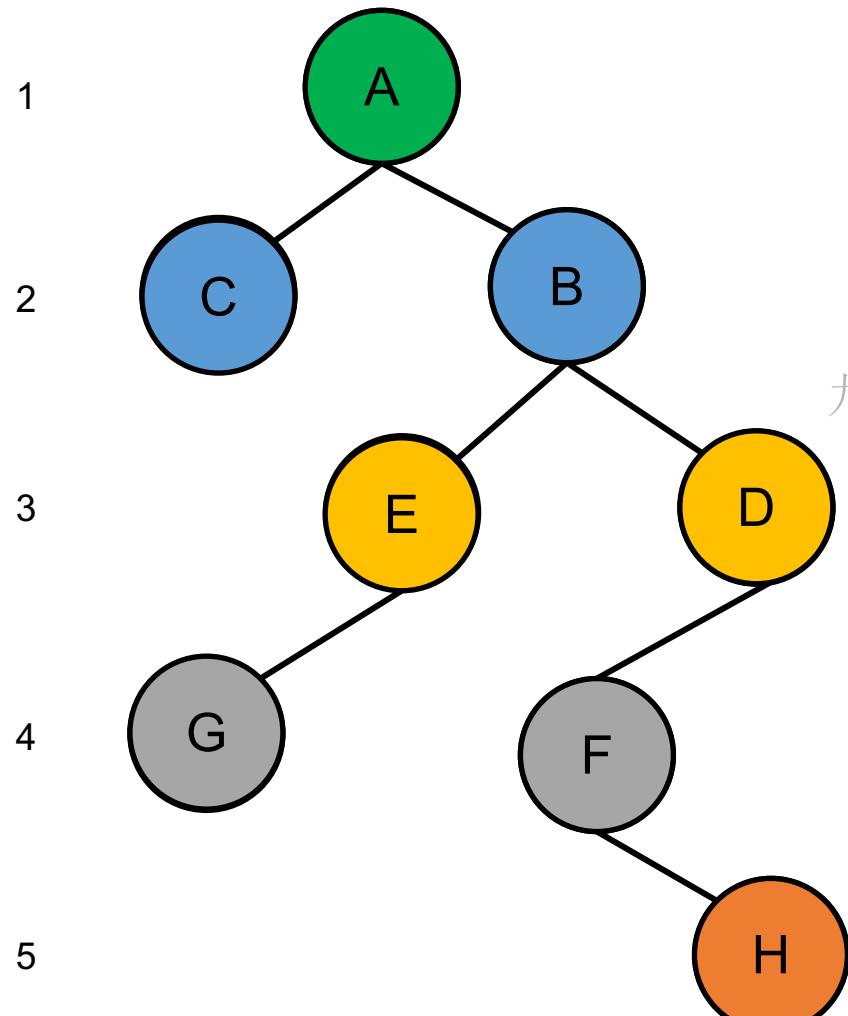
BFS使用场景



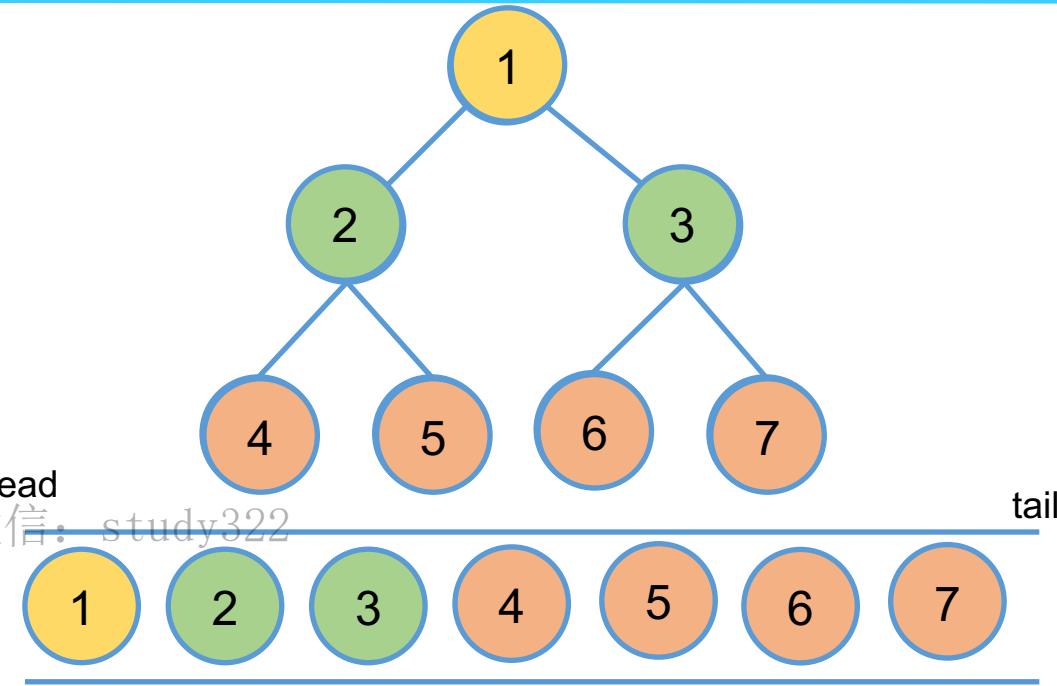
扫一扫 加我免费送课



树BFS，每种颜色代表一层



九章课程都有，加微信：study322

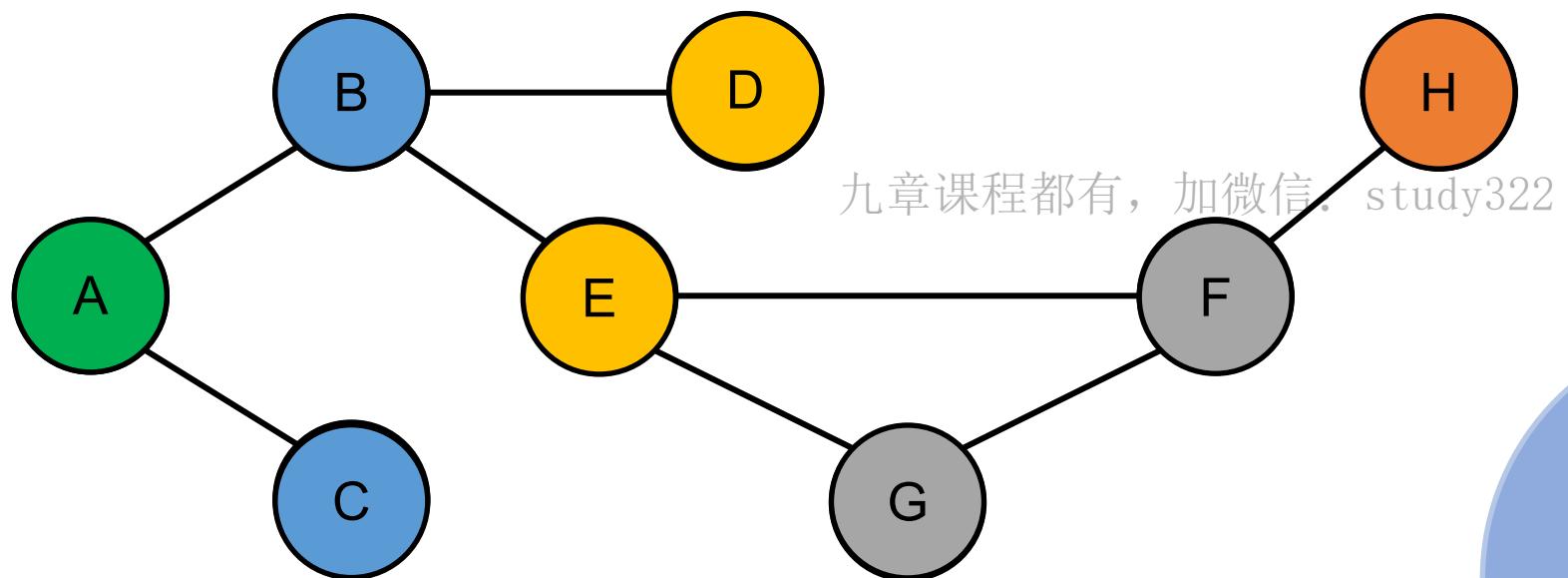


扫一扫 加我免费送课



图的BFS

图BFS，每种颜色代表一层



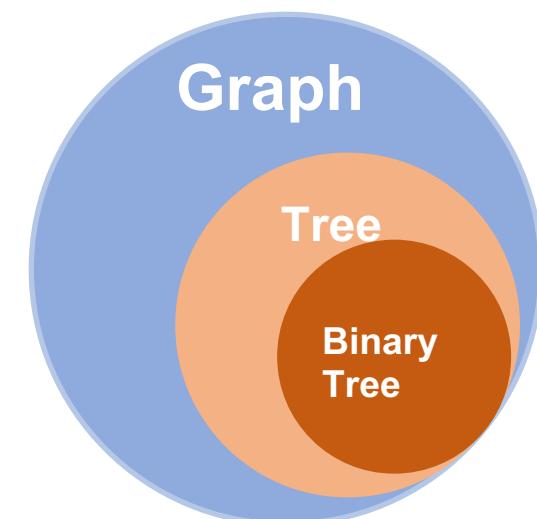
如果图中存在环，则同一个节点可能**重复**进入队列

BFS中，为什么同一个节点不需要重复进入队列？

- 对于联通块问题，不可能带来新的节点
- 对于最短路问题，不可能带来最短的路径

使用哈希表去重：

- Java: HashMap / HashSet
- Python: dict / set
- C++: unordered_map / unordered_set



树是图的一种

扫一扫 加我免费送课

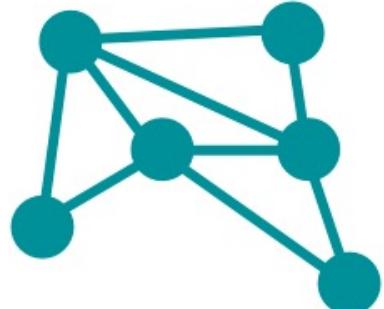


解决最短路径的算法有哪些？

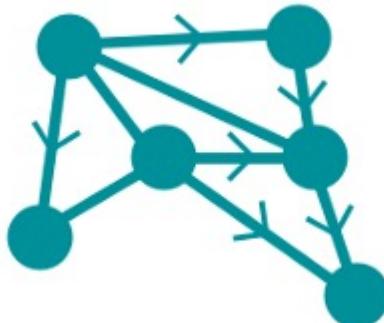
简单图	BFS
复杂图	SPFA(冲刺班), Floyd, Dijkstra, Bellman-ford 面试中一般不考复杂图最短路径问题

什么是简单图

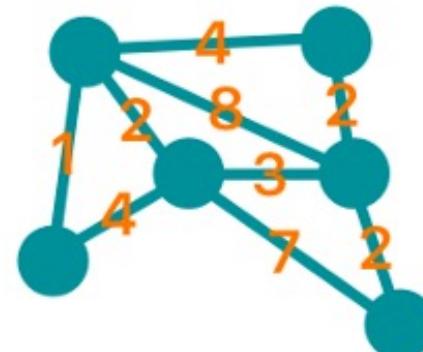
- 没有方向 (undirected)
- 没有权重 (unweighted)
- 两点之间最多只有一条边 (no multiple edges)
- 一个点没有一条边直接连着自己 (no graph loops , 这里的graph loop指的是自己直接指向自己的loop)



简单图



复杂图 (有向)



复杂图 (有权)



复杂图 (两点间有多条
有自己连自己的点)

最简洁的 BFS 算法的通用模板

Python 队列建议使用 ✓ `deque`, 不建议使用 ✗ `Queue` (涉及多线程加锁会更慢)

```
# step 1 初始化
# 把初始节点放到 deque 里, 如果有多个就都放进去
# 并标记初始节点的距离为0, 记录在 distance 的 dict 里
# distance 有两个作用, 一是判断是否已经访问过, 二是记录节点距离
queue = collections.deque([node])
distance = {node : 0}

# step 2: 不断访问队列
# while 循环 + 每次 pop 队列中的一个点出来
while queue:
    node = queue.popleft()
    # step 3: 拓展相邻节点
    # pop 出的节点的相邻节点, 加入队列并在 distance 中存储距离
    for neighbor in node.get_neighbors():
        if neighbor in distance:
            continue
        distance[neighbor] = distance[node] + 1
        queue.append(neighbor)
```

九章课程都有, 加微信: study326

Java 队列建议使用 ✓ `ArrayDeque`, 不建议使用 ✗ `new LinkedList` (链表比数组慢)

```
Queue<Node> queue = new ArrayDeque<>();
HashMap<Node, Integer> distance = new HashMap<>();

// step 1: 初始化
// 把初始节点放到 queue 里, 如果有多个就都放进去
// 并标记初始节点的距离为0, 记录在 distance 的 hashmap 里
// distance 有两个作用, 一是判断是否已经访问过, 二是记录离起点的距离
queue.offer(node);
distance.put(node, 0);

// step 2: 不断访问队列
// while 循环 + 每次 pop 队列中的一个点出来
while (!queue.isEmpty()) {
    Node node = queue.poll();
    // step 3: 拓展相邻节点
    // pop 出的节点的相邻节点, 加入队列并在 distance 中存储距离
    for (Node neighbor : node.getNeighbors()) {
        if (distance.containsKey(neighbor)) {
            continue;
        }
        distance.put(neighbor, distance.get(node) + 1);
        queue.offer(neighbor);
    }
}
```

扫一扫 加我免费送课

N个点, M条边, 图上BFS时间复杂度 = $O(N + M)$, 说是 $O(M)$ 问题也不大, 因为M一般都比N大

M最大是 $O(N^2)$ 的级别 (任意两个点之间都有边), 所以最坏情况可能是 $O(N^2)$



137 Clone Graph 克隆图

Clone an undirected graph. Each node in the graph contains a label and a list of its neighbors. **Nodes are labeled uniquely.**

You need to return a deep copied graph, which has the same structure as the original graph, and any changes to the new graph will not have any effect on the original graph.

You need return the node with the same label as the input node.

克隆一张无向图. 无向图的每个节点包含一个 label 和一个列表 neighbors. **保证每个节点的 label 互不相同.**

你的程序需要返回一个经过深度拷贝的新图. 新图和原图具有同样的结构, 并且对新图的任何改动不会对原图造成任何影响.

你需要返回与给定节点具有相同 label 的那个节点.

九章课程都有, 加微信: study322

输入:

{1,2,4#2,1,4#4,1,2}

输出:

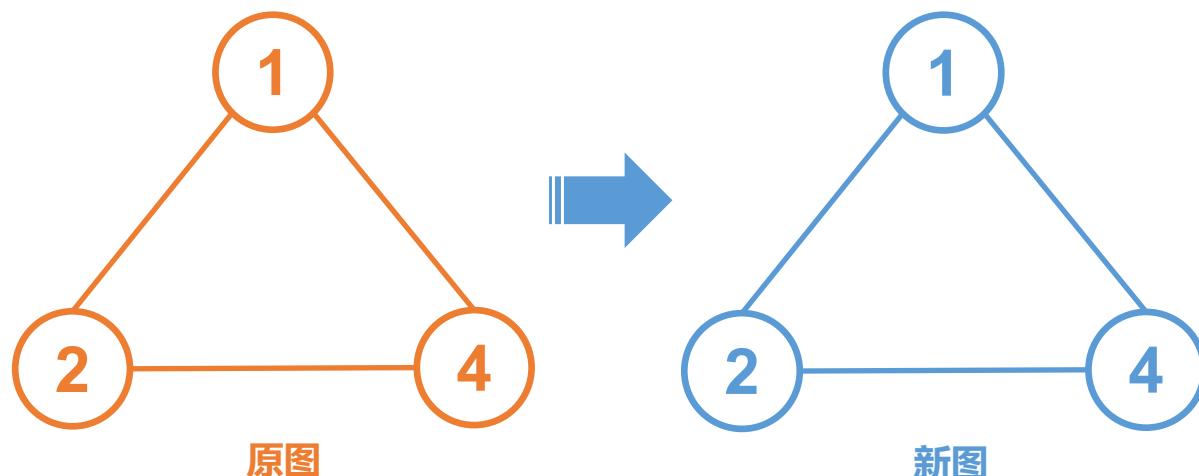
{1,2,4#2,1,4#4,1,2}

解释:

1 => 2, 4

2 => 1, 4

4 => 1, 2



原图和新图的点和边一模一样

题目已经给出图中的点的定义

```
2 class UndirectedGraphNode:
3     def __init__(self, x):
4         self.label = x
5         self.neighbors = []
```

```
2 class UndirectedGraphNode {
3     int label;
4     List<UndirectedGraphNode> neighbors;
5     UndirectedGraphNode(int x) {
6         label = x;
7         neighbors = new ArrayList<UndirectedGraphNode>();
8     }
9 }
```



这份代码有什么问题

```

def cloneGraph(self, node):
    if not node:
        return None
    queue = [node]
    start = 0
    mapping = {}
    while start < len(queue):
        curt_node = queue[start]
        start += 1
        if curt_node in mapping:
            new_node = mapping[curt_node]
        else:
            new_node = UndirectedGraphNode(curt_node.label)
            mapping[node] = new_node
        for neighbor in curt_node.neighbors:
            if neighbor in mapping:
                new_neighbor = mapping[neighbor]
            else:
                new_neighbor = UndirectedGraphNode(neighbor.label)
                mapping[neighbor] = new_neighbor
                queue.append(neighbor)
            new_node.neighbors.append(new_neighbor)
    return mapping[node]

```

高耦合的代码：寻点，复制点，复制边交错在一起

```

public class Solution {
    public UndirectedGraphNode cloneGraph(UndirectedGraphNode node) {
        if (node == null) {
            return null;
        }
        Queue<UndirectedGraphNode> queue = new ArrayDeque<>();
        int start = 0;
        queue.offer(node);
        Map<UndirectedGraphNode, UndirectedGraphNode> mapping = new HashMap<>();
        while (!queue.isEmpty()) {
            UndirectedGraphNode curtNode = queue.poll();
            UndirectedGraphNode newNode;
            if (mapping.containsKey(curtNode)) {
                newNode = mapping.get(curtNode);
            } else {
                newNode = new UndirectedGraphNode(curtNode.label);
                mapping.put(node, newNode);
            }
            UndirectedGraphNode newNeighbor;
            for (UndirectedGraphNode neighbor : curtNode.neighbors) {
                if (mapping.containsKey(neighbor)) {
                    newNeighbor = mapping.get(neighbor);
                } else {
                    newNeighbor = new UndirectedGraphNode(neighbor.label);
                    mapping.put(neighbor, newNeighbor);
                    queue.offer(neighbor);
                }
                newNode.neighbors.add(newNeighbor);
            }
        }
        return mapping.get(node);
    }
}

```

九章课程都有，加微信：

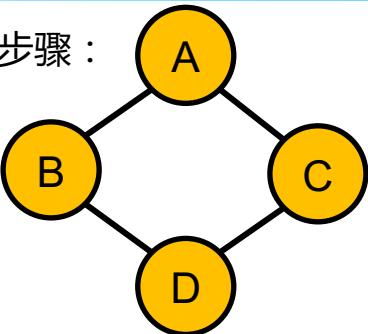
扫一扫 加我免费送课



代码分析 —— 低耦合的清晰代码 (劝分不劝和)

将整个算法分解为三个步骤：

1. 找到所有点
2. 复制所有点
3. 复制所有边



A => B, C
 B => A, D
 C => A, D
 D => B, C

```

2   class UndirectedGraphNode:
3       def __init__(self, x):
4           self.label = x
5           self.neighbors = []
6
7
8   class Solution:
9       def cloneGraph(self, node):
10          # 特殊情况处理
11          if not node:
12              return None
13
14          # 第一步：找到所有点
15          nodes = self.find_nodes_by_bfs(node)
16          # 第二步：复制所有点
17          mapping = self.copy_nodes(nodes)
18          # 第三步：复制所有边
19          self.copy_edges(nodes, mapping)
20
21          # 返回与给定节点具有相同 label 的那个节点
22          return mapping[node]
23
  
```

九章课程都有，加微信 study322

```

24      # 第一步：BFS找到所有点
25      def find_nodes_by_bfs(self, node):
26          queue = collections.deque([node])
27          # 用于保存所有点，不重不漏
28          visited = set([node])
29
30          while queue:
31              curt_node = queue.popleft()
32              for neighbor in curt_node.neighbors:
33                  # 如果之前已经找到了这个点，无需再次BFS，否则死循环
34                  if neighbor in visited:
35                      continue
36                  queue.append(neighbor)
37                  visited.add(neighbor)
38
39
40      # 第二步：复制所有点
41      def copy_nodes(self, nodes):
42          # (旧点->新点)的映射
43          mapping = {}
44          for node in nodes:
45              mapping[node] = UndirectedGraphNode(node.label)
46
47          return mapping
48
49      # 第三步：复制所有边
50      def copy_edges(self, nodes, mapping):
51          for node in nodes:
52              new_node = mapping[node]
53              # 旧点有哪些旧邻居，新点就有哪些新邻居
54              for neighbor in node.neighbors:
55                  new_neighbor = mapping[neighbor]
56                  new_node.neighbors.append(new_ne
  
```

[A, B, C, D]
 A => A'
 B => B'
 C => C'
 D => D'
 A' B' C' D'

扫一扫 加我免费送课



代码分析 —— 低耦合的清晰代码

```

18  public UndirectedGraphNode cloneGraph(UndirectedGraphNode node) {
19      // 特殊情况处理
20      if (node == null) {
21          return null;
22      }
23      // 第一步：找到所有点
24      List<UndirectedGraphNode> nodes = findNodesByBFS(node);
25      // 第二步：复制所有点
26      Map<UndirectedGraphNode, UndirectedGraphNode> mapping = copyNodes(nodes);
27      // 第三步：复制所有边
28      copyEdges(nodes, mapping);
29
30      // 返回与给定节点具有相同 label 的那个节点
31      return mapping.get(node);
32  }
33
34  // 第一步：BFS找到所有点
35  private List<UndirectedGraphNode> findNodesByBFS(UndirectedGraphNode node) {
36      Queue<UndirectedGraphNode> queue = new LinkedList<>();
37      // 用于保存所有点，不重不漏
38      Set<UndirectedGraphNode> visited = new HashSet<>();
39      queue.offer(node);
40      visited.add(node);
41      while (!queue.isEmpty()) {
42          UndirectedGraphNode curNode = queue.poll();
43          for (UndirectedGraphNode neighbor: curNode.neighbors) {
44              // 如果之前已经找到了这个点，无需再次BFS，否则死循环
45              if (visited.contains(neighbor)) {
46                  continue;
47              }
48              visited.add(neighbor);
49              queue.offer(neighbor);
50          }
51      }
52
53      return new LinkedList<>(visited);
54  }

```

[A, B, C, D]

```

56      // 第一步：复制所有点
57  private Map<UndirectedGraphNode, UndirectedGraphNode> copyNodes(List<UndirectedGraphNode> nodes) {
58      // (旧点->新点)的映射
59      Map<UndirectedGraphNode, UndirectedGraphNode> mapping = new HashMap<>();
60      for (UndirectedGraphNode node: nodes) {
61          mapping.put(node, new UndirectedGraphNode(node.label));
62      }
63
64      return mapping;
65  }
66
67  // 第二步：复制所有边
68  private void copyEdges(List<UndirectedGraphNode> nodes, Map<UndirectedGraphNode,
69                      UndirectedGraphNode> mapping) {
70      for (UndirectedGraphNode node: nodes) {
71          UndirectedGraphNode newNode = mapping.get(node);
72          // 旧点有哪些旧邻居，新点就有哪些新邻居
73          for (UndirectedGraphNode neighbor: node.neighbors) {
74              UndirectedGraphNode newNeighbor = mapping.get(neighbor);
75              newNode.neighbors.add(newNeighbor);
76          }
77      }
78  }
79
80  class UndirectedGraphNode {
81      int label;
82      List<UndirectedGraphNode> neighbors;
83      UndirectedGraphNode(int x) {
84          label = x;
85          neighbors = new ArrayList<UndirectedGraphNode>();
86      }
87  }
88
89

```

A => A'
B => B'
C => C'
D => D'

A' => B', C'
B' => A', D'
C' => A', D'
D' => B', C'

扫一扫 加我免费送课



80% 的人都可能会写错的 BFS 算法

入队时标记为被访问

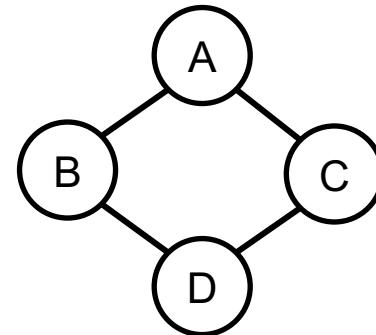
```
queue = collections.deque([node])
visited = set([node])
while queue:
    curt_node = queue.popleft()
    for neighbor in curt_node.neighbors:
        if neighbor in visited:
            continue
        visited.add(neighbor)
        queue.append(neighbor)
return list(visited)
```

A

出队时标记为被访问

```
queue = collections.deque([node])
# visited = set([node])
while queue:
    curt_node = queue.popleft()
    visited.add(curt_node)
    for neighbor in curt_node.neighbors:
        if neighbor in visited:
            continue
        # visited.add(neighbor)
        queue.append(neighbor)
return list(visited)
```

B



九章算法 https://study322.com

```
queue.offer(node);
set.add(node);
while (!queue.isEmpty()) {
    UndirectedGraphNode head = queue.poll();
    for (UndirectedGraphNode neighbor : head.neighbors) {
        if (!set.contains(neighbor)) {
            set.add(neighbor);
            queue.offer(neighbor);
        }
    }
}
```

A

```
queue.offer(node);
// set.add(node);
while (!queue.isEmpty()) {
    UndirectedGraphNode head = queue.poll();
    set.add(head);
    for (UndirectedGraphNode neighbor : head.neighbors) {
        if (!set.contains(neighbor)) {
            // set.add(neighbor);
            queue.offer(neighbor);
        }
    }
}
```

B

入队时标记为被访问，否则，会有元素重复入队

版权归属于九章算法（杭州）科技有限公司，贩卖和传播盗版将被追究刑事责任

扫一扫 加我免费送课



BFS分层 vs 不分层

不分层

```
def find_nodes_by_bfs(self, node):
    queue = collections.deque([node])
    visited = set([node])
    while queue:
        curt_node = queue.popleft()
        for neighbor in curt_node.neighbors:
            if neighbor in visited:
                continue
            visited.add(neighbor)
            queue.append(neighbor)
    return list(visited)
```

```
private ArrayList<UndirectedGraphNode> getNodes(UndirectedGraphNode node) {
    Queue<UndirectedGraphNode> queue = new ArrayDeque<UndirectedGraphNode>();
    HashSet<UndirectedGraphNode> set = new HashSet<>();

    queue.offer(node);
    set.add(node);
    while (!queue.isEmpty()) {
        UndirectedGraphNode head = queue.poll();
        for (UndirectedGraphNode neighbor : head.neighbors) {
            if (!set.contains(neighbor)) {
                set.add(neighbor);
                queue.offer(neighbor);
            }
        }
    }
    return new ArrayList<UndirectedGraphNode>(set);
}
```

分层

```
def find_nodes_by_bfs(self, node):
    queue = collections.deque([node])
    visited = set([node])
    while queue:
        for _ in range(len(queue)):
            curt_node = queue.popleft()
            for neighbor in curt_node.neighbors:
                if neighbor in visited:
                    continue
                visited.add(neighbor)
                queue.append(neighbor)
    return list(visited)
```

是否可以写成 while(len(queue) > 0) ?

九章课程讲解，加我微信：study322

```
private ArrayList<UndirectedGraphNode> getNodes(UndirectedGraphNode node) {
    Queue<UndirectedGraphNode> queue = new ArrayDeque<UndirectedGraphNode>();
    HashSet<UndirectedGraphNode> set = new HashSet<>();

    queue.offer(node);
    set.add(node);
    while (!queue.isEmpty()) {
        int size = queue.size();
        for (int i = 0; i < size; i++) {
            UndirectedGraphNode head = queue.poll();
            for (UndirectedGraphNode neighbor : head.neighbors) {
                if (!set.contains(neighbor)) {
                    set.add(neighbor);
                    queue.offer(neighbor);
                }
            }
        }
    }
    return new ArrayList<UndirectedGraphNode>(set);
}
```

是否可以写成 while(!queue.isEmpty()) ?

扫一扫 加我免费送课



120 Word Ladder 单词接龙

Given two words (*start* and *end*), and a dictionary, find the shortest transformation sequence from *start* to *end*, output the length of the sequence.

Transformation rule such that:

Only one letter can be changed at a time

Each intermediate word must exist in the dictionary. (**Start and end words do not need to appear in the dictionary**)

给出两个单词 (*start*和*end*) 和一个字典，找出从*start*到*end*的最短转换序列，输出**最短序列**的长度。

变换规则如下：

每次只能改变一个字母。

九章课程都有，加微信：study322

变换过程中的中间单词必须在字典中出现。(**起始单词和结束单词可以不出现在字典中**)

如果不存在这样的转换序列，返回 0。

所有单词具有相同的长度。

所有单词只由小写字母组成。

字典中不存在重复的单词。

你可以假设 *beginWord* 和 *endWord* 是非空的，且二者不相同。

输入：

start = "hit", *end* = "cog"

dict = ["hot", "dot", "dog", "lot", "log"]

输出：

5

解释：

"hit"->"hot"->"dot"->"dog"->"cog"

输入：

start = "a", *end* = "c"

dict = ["a", "b", "c"]

输出： **扫一扫 加我免费送课**

2

解释：

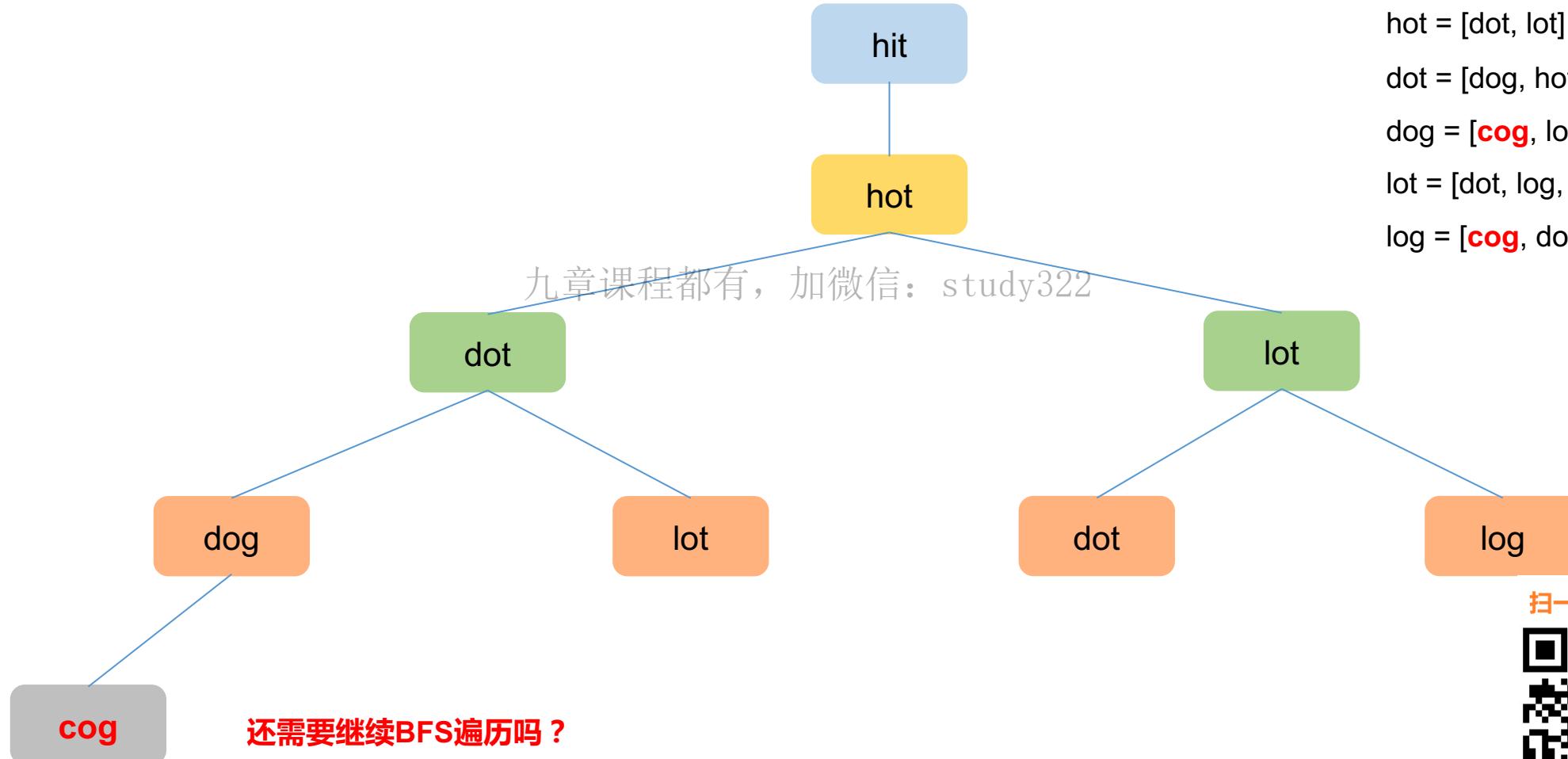
"a"->"c"



BFS 宽度优先搜索

输入 : start = "hit" , end = "cog" , dict =["hot", "dot", "dog", "lot", "log"]

从起点开始，逐层向外扩张，不走回头路



代码解析

```

3  def ladderLength(self, start, end, dict):
4      # 假设dict不为null。
5      # 假设 beginWord 和 endWord 是非空的，且二者不相同。
6
7      # 必须加入end。可以加入start。
8      dict.add(end)
9      queue = collections.deque([start])
10     visited = set([start])
11
12     distance = 0
13     while queue:
14         # 到当前层的长度
15         distance += 1
16         # 当前层有size个元素
17         size = len(queue)
18         for i in range(size):
19             word = queue.popleft()
20             # 如果当前层的词为尾词，直接返回当前层的长度
21             if word == end:
22                 return distance
23             # 得到下一步的单词
24             for next_word in self.get_next_words(word, dict):
25                 if next_word in visited:
26                     continue
27                 queue.append(next_word)
28                 visited.add(next_word)
29
30     return 0
31

```

九章课程都有，加微信 study324

```

2  public int ladderLength(String start, String end, Set<String> dict) {
3      // 假设dict不为null。
4      // 假设 beginWord 和 endWord 是非空的，且二者不相同。
5
6      // 必须加入end。可以加入start。
7      dict.add(end);
8
9      HashSet<String> visited = new HashSet<String>();
10     Queue<String> queue = new LinkedList<String>();
11     queue.offer(start);
12     visited.add(start);
13
14     // 记录最短路线长度，起始长度为1
15     int length = 1;
16     while(!queue.isEmpty()) {
17         // 到下一层（不是当前层）的长度
18         length++;
19         // 当前层有size个元素
20         int size = queue.size();
21         for (int i = 0; i < size; i++) {
22             String word = queue.poll();
23             // 得到下一步的单词
24             for (String nextWord: getNextWords(word, dict)) {
25                 if (visited.contains(nextWord)) {
26                     continue;
27                 }
28                 // 如果下一层的词为尾词，直接返回当前到下一层（不是当前层）的长度
29                 if (nextWord.equals(end)) {
30                     return length;
31                 }
32                 // 加入下一层，为后面BFS做准备
33                 visited.add(nextWord);
34                 queue.offer(nextWord);
35             }
36         }
37
38     }
39
40     // 不能实现首尾接龙，返回0
41     return 0;

```

扫一扫 加我免费送课



代码解析

O(N * L)

N为字典中词的个数

L为每个词的长度

O(26 * L * L)

L为每个词的长度

```

49     def get_next_words(self, word, dict):
50         next_words = []
51         # 枚举字典中的每个词
52         for next_word in dict:
53             has_one_diff = False
54             # 判断两个词是否只相差一个字母。如果是，则可以接龙
55             for i in range(len(word)):
56                 if (next_word[i] != word[i]):
57                     if has_one_diff:
58                         has_one_diff = False
59                         break
60                     has_one_diff = True
61             if has_one_diff:
62                 next_words.append(next_word)
63         return next_words

```

```

33     # 找到可以和word接龙的所有单词
34     # 比如 word = 'hot', dict = {'hot', 'hit', 'hog'}, return ['hit', 'hog']
35     def get_next_words(self, word, dict):
36         next_words = []
37         for i in range(len(word)):
38             left, right = word[:i], word[i + 1:]
39             for char in 'abcdefghijklmnopqrstuvwxyz':
40                 if word[i] == char:
41                     continue
42                     # 在s中，把位置index的字母替换成c，返回替换后的字符串
43                     new_word = left + char + right
44                     # 如果字母替换后的单词存在于dict，加入next_words
45                     if new_word in dict:
46                         next_words.append(new_word)
47         return next_words

```

```

73     private ArrayList<String> getNextWords(String word, Set<String> dict) {
74         ArrayList<String> nextWords = new ArrayList<String>();
75         // 枚举字典中的每个词
76         for (String nextWord: dict) {
77             boolean hasOneDiff = false;
78             for (int i = 0; i < word.length(); i++) {
79                 // 判断两个词是否只相差一个字母。如果是，则可以接龙
80                 if (nextWord.charAt(i) != word.charAt(i)) {
81                     if (hasOneDiff) {
82                         hasOneDiff = false;
83                         break;
84                     }
85                     hasOneDiff = true;
86                 }
87             }
88             if (hasOneDiff) {
89                 nextWords.add(nextWord);
90             }
91         }
92         return nextWords;
93     }

```

```

45     // 在s中，把位置index的字母替换成c，返回替换后的字符串
46     private String replace(String s, int index, char c) {
47         char[] chars = s.toCharArray();
48         chars[index] = c;
49         return new String(chars);
50     }
51
52     // 找到可以和word接龙的所有单词
53     // 比如 word = 'hot', dict = {'hot', 'hit', 'hog'}, return ['hit', 'hog']
54     private ArrayList<String> getNextWords(String word, Set<String> dict) {
55         ArrayList<String> nextWords = new ArrayList<String>();
56         // 枚举当前替换字母
57         for (char c = 'a'; c <= 'z'; c++) {
58             // 枚举替换位置
59             for (int i = 0; i < word.length();
60                 if (c == word.charAt(i)) {
61                     continue;
62                 }
63                 String nextWord = replace(word,
64                     // 如果字母替换后的单词存在于dict，
65                     if (dict.contains(nextWord)) {
66                         nextWords.add(nextWord);
67                     }
68                 }
69             }
70         }
71         return nextWords;

```

扫一扫 加我免费送课



快扶我起来
我还能学

九章课程都有，加微信：study322



扫一扫 加我免费送课



矩阵中的宽度优先搜索

九章课程都有，加微信：study322
BFS in Matrix

扫一扫 加我免费送课



433. Number of Islands

Given a boolean 2D matrix, 0 is represented as the sea, 1 is represented as the island. If two 1 is adjacent, we consider them in the same island. We only consider up/down/left/right adjacent.

Find the number of islands.

给一个 01 矩阵，求不同的岛屿的个数。

0 代表海，1 代表岛，如果两个 1 相邻，那么这两个 1 属于同一个岛。我们只考虑上下左右为相邻。

九章课程都有，加微信：study322

输入：

[1,1,0,0,0],
[0,1,0,0,1],
[0,0,0,1,1],
[0,0,0,0,1],
[0,0,0,0,1]

01Matrix + 联通块的个数 → BFS/DFS

输出： 2

扫一扫 加我免费送课



1. 逐行逐列进行遍历
2. 如果找到一个1，岛屿数量增1
3. 把所有跟这个1相连的1都找出来。所有这些相连的1代表一个岛。
4. 回到步骤1继续遍历

岛屿数量 : 2

九章课程都有，加微信：study322

1	1	0	0	0
0	1	0	0	1
0	0	0	1	1
0	0	0	0	1
0	0	0	0	

扫一扫 加我免费送课



```

1  from collections import deque
2
3  # 四个方向的偏移量
4  DIRECTIONS = [(1, 0), (0, -1), (-1, 0), (0, 1)]
5
6  class Solution:
7      def numIslands(self, grid):
8          # 特殊情况处理
9          if not grid or not grid[0]:
10             return 0
11
12         islands = 0
13         # 记录某点是否被BFS过, 如果之前已经被BFS过, 不应再次被BFS
14         visited = set()
15
16         # 遍历矩阵中的每一个点
17         for i in range(len(grid)):
18             for j in range(len(grid[0])):
19                 # 如果为海洋, 无需BFS
20                 # 如果该点已经被visited, 无需做冗余遍历, 重复计算
21                 if grid[i][j] and (i, j) not in visited:
22                     self.bfs(grid, i, j, visited)
23                     islands += 1
24
25         return islands
  
```

九章课程都有，加微信 study323

```

1  // 定义一个class, 表示坐标系中的一点
2  class Coordinate {
3      int x, y;
4      public Coordinate(int x, int y) {
5          this.x = x;
6          this.y = y;
7      }
8  }
9
10 public class Solution {
11     // 四个方向的偏移量
12     int[] deltaX = {0, 1, -1, 0};
13     int[] deltaY = {1, 0, 0, -1};
14
15     public int numIslands(boolean[][] grid) {
16         // 特殊情况处理
17         if (grid == null || grid.length == 0 || grid[0] == null || grid[0].length == 0) {
18             return 0;
19         }
20
21         int islands = 0;
22         int row = grid.length, col = grid[0].length;
23         // 记录某点是否被BFS过, 如果之前已经被BFS过, 不应再次被BFS
24         boolean[][] visited = new boolean[row][col];
25
26         // 遍历矩阵中的每一个点
27         for (int i = 0; i < row; i++) {
28             for (int j = 0; j < col; j++) {
29                 // 如果为海洋, 无需BFS
30                 // 如果该点已经被visited, 无需做冗余遍历, 重复计算
31                 if (grid[i][j] && !visited[i][j]) {
32                     bfs(grid, i, j, visited);
33                     islands++;
34                 }
35             }
36         }
37     }
38
39     return islands;
40 }
  
```

扫一扫 加我免费送课



```

27   # 从一块土地出发，通过BFS，遍历整个岛屿
28   def bfs(self, grid, x, y, visited):
29       queue = deque([(x, y)])
30       visited.add((x, y))
31       while queue:
32           x, y = queue.popleft()
33           # 遍历上下左右四个方向
34           for delta_x, delta_y in DIRECTIONS:
35               next_x = x + delta_x
36               next_y = y + delta_y
37               if not self.is_valid(grid, next_x, next_y, visited):
38                   continue
39               queue.append((next_x, next_y))
40               # 一旦入队，标记此点已经参与过BFS
41               visited.add((next_x, next_y))
42
43   # 判断一个点是否可以被BFS
44   def is_valid(self, grid, x, y, visited):
45       n, m = len(grid), len(grid[0])
46       # 如果出界，返回false
47       if not (0 <= x < n and 0 <= y < m):
48           return False
49       # 如果已经BFS过，不要再次BFS，避免：1. 死循环 2. 冗余的BFS变量
50       if (x, y) in visited:
51           return False
52       # 如果是1，则为true；如果是0，则为false
53       return grid[x][y]

```

九章课程都有，加微信：study922

```

42   // 从一块土地出发，通过BFS，遍历整个岛屿
43   private void bfs(boolean[][] grid, int x, int y, boolean[][] visited) {
44       Queue<Coordinate> queue = new ArrayDeque<>();
45       queue.offer(new Coordinate(x, y));
46       visited[x][y] = true;
47
48       while (!queue.isEmpty()) {
49           Coordinate coor = queue.poll();
50           // 遍历上下左右四个方向
51           for (int direction = 0; direction < 4; direction++) {
52               int newX = coor.x + deltaX[direction];
53               int newY = coor.y + deltaY[direction];
54               if (!isValid(grid, newX, newY, visited)) {
55                   continue;
56               }
57               queue.offer(new Coordinate(newX, newY));
58               // 一旦入队，标记此点已经参与过BFS
59               visited[newX][newY] = true;
60           }
61       }
62   }
63
64   // 判断一个点是否可以被BFS
65   private boolean isValid(boolean[][] grid, int x, int y, boolean[][] visited) {
66       int n = grid.length, m = grid[0].length;
67       // 如果出界，返回false
68       if (x < 0 || x >= n || y < 0 || y >= m) {
69           return false;
70       }
71       // 如果已经BFS过，不要再次BFS，避免：1. 死循环 2. 冗余的BFS变量
72       if (visited[x][y]) {
73           return false;
74       }
75       // 如果是1，则为true；如果是0，则为false
76       return grid[x][y];
77   }
78 }

```

可以不用visit来标记已经被访问，用其他的方式记录被访问过的点吗？

可以用DFS取代BFS来遍历同一个岛屿里面的点吗？

版权归属于九章算法（杭州）科技有限公司，贩卖和传播盗版将被追究刑事责任

扫一扫 加我免费送课

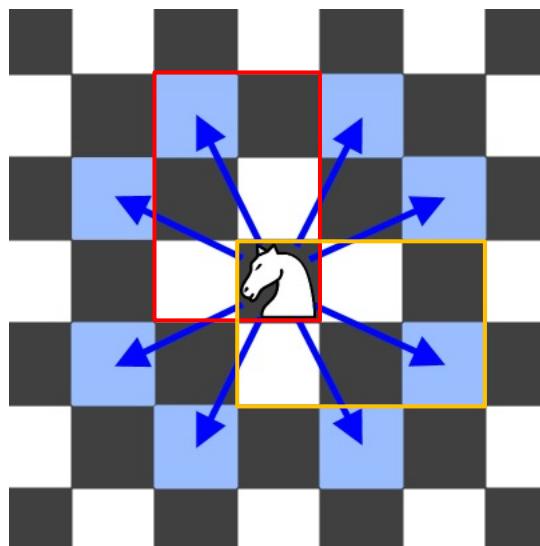


611 · Knight Shortest Path

Given a knight in a chessboard (a binary matrix with 0 as empty and 1 as barrier) with a source position, find the shortest path to a destination position, return the length of the route.

Return -1 if destination cannot be reached.

给定骑士在棋盘上的 初始位置(一个2进制矩阵 0 表示空 1 表示有障碍物) , 找到到达终点的最短路线 , 返回路线的长度。如果骑士不能到达则返回 -1 。



输入:

九章课程都有，加微信: study322

`[[0,0,0], [0,0,0], [0,0,0]]`

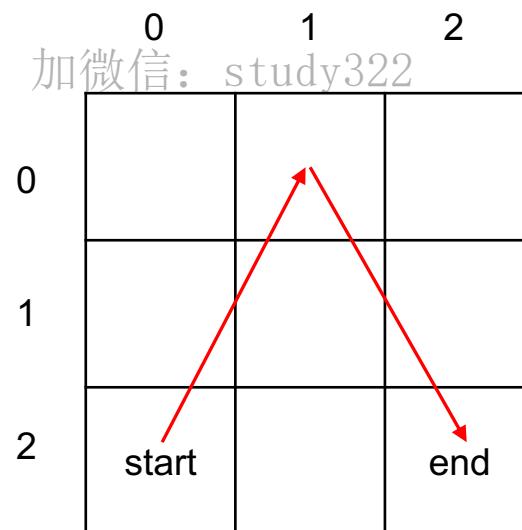
`source = [2, 0]`

`destination = [2, 2]`

输出: 2

解释:

`[2,0]->[0,1]->[2,2]`



扫一扫 加我免费送课



代码解析

```

10    OFFSETS = [
11        (-2, -1), (-2, 1), (-1, 2), (1, 2),
12        (2, 1), (2, -1), (1, -2), (-1, -2),
13    ]
14
15    class Solution:
16        def shortestPath(self, grid, source, destination):
17            queue = collections.deque([(source.x, source.y)])
18            # 记录从起点到(x, y)的最短距离, (x, y) -> shortest distance from start
19            # 在python中, 可以使用tuple存储横纵坐标, 存入dict
20            cell_to_dis_dict = {(source.x, source.y): 0}
21
22            while queue:
23                x, y = queue.popleft()
24                if (x, y) == (destination.x, destination.y):
25                    return cell_to_dis_dict[(x, y)]
26                # 遍历8个不同的方向
27                for dx, dy in OFFSETS:
28                    # 新点坐标
29                    next_x, next_y = x + dx, y + dy
30                    if not self.is_valid(next_x, next_y, grid):
31                        continue
32                    # 如果之前已经到达过此点, 不可能通过再次BFS此点找到最短路, 还会造成死循环
33                    if (next_x, next_y) in cell_to_dis_dict:
34                        continue
35                    queue.append((next_x, next_y))
36                    cell_to_dis_dict[(next_x, next_y)] = \
37                        cell_to_dis_dict[(x, y)] + 1
38
39            # 不能达到, 返回-1
40            return -1
41
42        # 如果一个点越界或者为障碍物, 返回false; 否则返回true
43        def is_valid(self, x, y, grid):
44            n, m = len(grid), len(grid[0])
45            if x < 0 or x >= n or y < 0 or y >= m:
46                return False
47            # 值1表示障碍物, 返回false
48            return not grid[x][y]

```

```

16    public int shortestPath(boolean[][] grid, Point source, Point destination) {
17        if (grid == null || grid.length == 0
18            || grid[0] == null || grid[0].length == 0) {
19            return -1;
20        }
21
22        Queue<Point> queue = new ArrayDeque();
23        // 记录从起点到(x, y)的最短距离, (x, y) -> shortest distance from start
24        Map<Integer, Integer> cellToDisMap = new HashMap();
25
26        int colCnt = grid[0].length;
27        queue.offer(source);
28        cellToDisMap.put(source.x * colCnt + source.y, 0);
29
30        while (!queue.isEmpty()) {
31            Point point = queue.poll();
32            int currPointKey = point.x * colCnt + point.y;
33            if (point.x == destination.x && point.y == destination.y) {
34                return cellToDisMap.get(currPointKey);
35            }
36
37            // 遍历8个不同的方向
38            for (int i = 0; i < 8; i++) {
39                // 新点坐标
40                int adjX = point.x + X_OFFSET[i];
41                int adjY = point.y + Y_OFFSET[i];
42                if (!isValid(adjX, adjY, grid)) {
43                    continue;
44                }
45                // 小套路: 把二维坐标转换为一维坐标
46                int newPointKey = adjX * colCnt + adjY;
47                // 如果之前已经到达过此点, 不可能通过再次BFS此点找到最短路, 还会造成死循环
48                if (cellToDisMap.containsKey(newPointKey)) {
49                    continue;
50                }
51                queue.offer(new Point(adjX, adjY));
52                cellToDisMap.put(newPointKey,
53                                cellToDisMap.get(currPointKey) + 1);
54            }
55        }
56    }
57    // 不能达到, 返回-1
58    return -1;
59 }

```

九章课程都有, 加微信: study322

一维坐标 = 行坐标 * 列数 + 列坐标

	0	1	2
0	0*3+0=0	0*3+1=1	0*3+2=2
1	1*3+0=3	1*3+1=4	1*3+2=5
2	2*3+0=6	2*3+1=7	2*3+2=8

扫一扫 加我免费送课



拓扑排序 Topological Sorting

九章课程都有，加微信：study322

很多公司都可能有拓扑排序的面试题

BFS or DFS?

建议使用BFS，实现更简单

扫一扫 加我免费送课



入度 (In-degree) :

有向图 (Directed Graph) 中指向当前节点的点的个数 (或指向当前节点的边的条数)

算法描述 :

1. 统计每个点的入度
2. 将每个入度为 0 的点放入队列 (Queue) 中作为起始节点
3. 不断从队列中拿出一个点，去掉这个点的所有连边 (指向其他点的边)，其他点的相应的入度 - 1
4. 一旦发现新的入度为 0 的点，丢回队列中

拓扑排序并不是传统的排序算法

一个图可能存在多个拓扑序 (Topological Order)，也可能不存在任何拓扑序

扫一扫 加我免费送课



求任意一个拓扑排序

问是否存在拓扑排序

求是否存在且仅存在一个拓扑排序

求字典序最小的拓扑排序

九章课程都有，加微信：study322

扫一扫 加我免费送课



616. Course Schedule II (问是否存在拓扑排序)

There are a total of n courses you have to take, labeled from 0 to n - 1.

Some courses may have prerequisites, for example to take course 0 you have to first take course 1, which is expressed as a pair: [0,1]

Given the total number of courses and a list of prerequisite pairs, return the ordering of courses you should take to finish all courses.

There may be multiple correct orders, you just need to return one of them. If it is impossible to finish all courses, return an empty array.

你需要去上n门九章的课才能获得offer，这些课被标号为 0 到 n-1。

有一些课程需要“前置课程”，比如如果你要上课程0，你需要先学课程1，我们用一个匹配来表示他们：[0,1]

给你课程的总数量和一些前置课程的需求，返回你为了学完所有课程所安排的学习顺序。

可能会有多个正确的顺序，你只要返回一种就可以了。如果不可能完成所有课程，返回一个空数组。

输入:

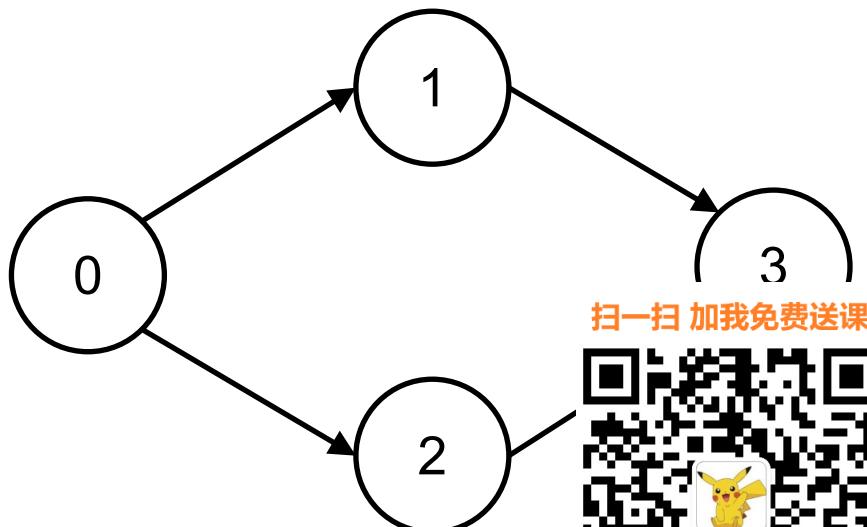
n = 4, prerequisites = [[1,0],[2,0],[3,1],[3,2]]

输出:

[0,1,2,3] or [0,2,1,3]

图 + 有依赖关系/有向 + 无环

→ 拓扑排序



扫一扫 加我免费送课



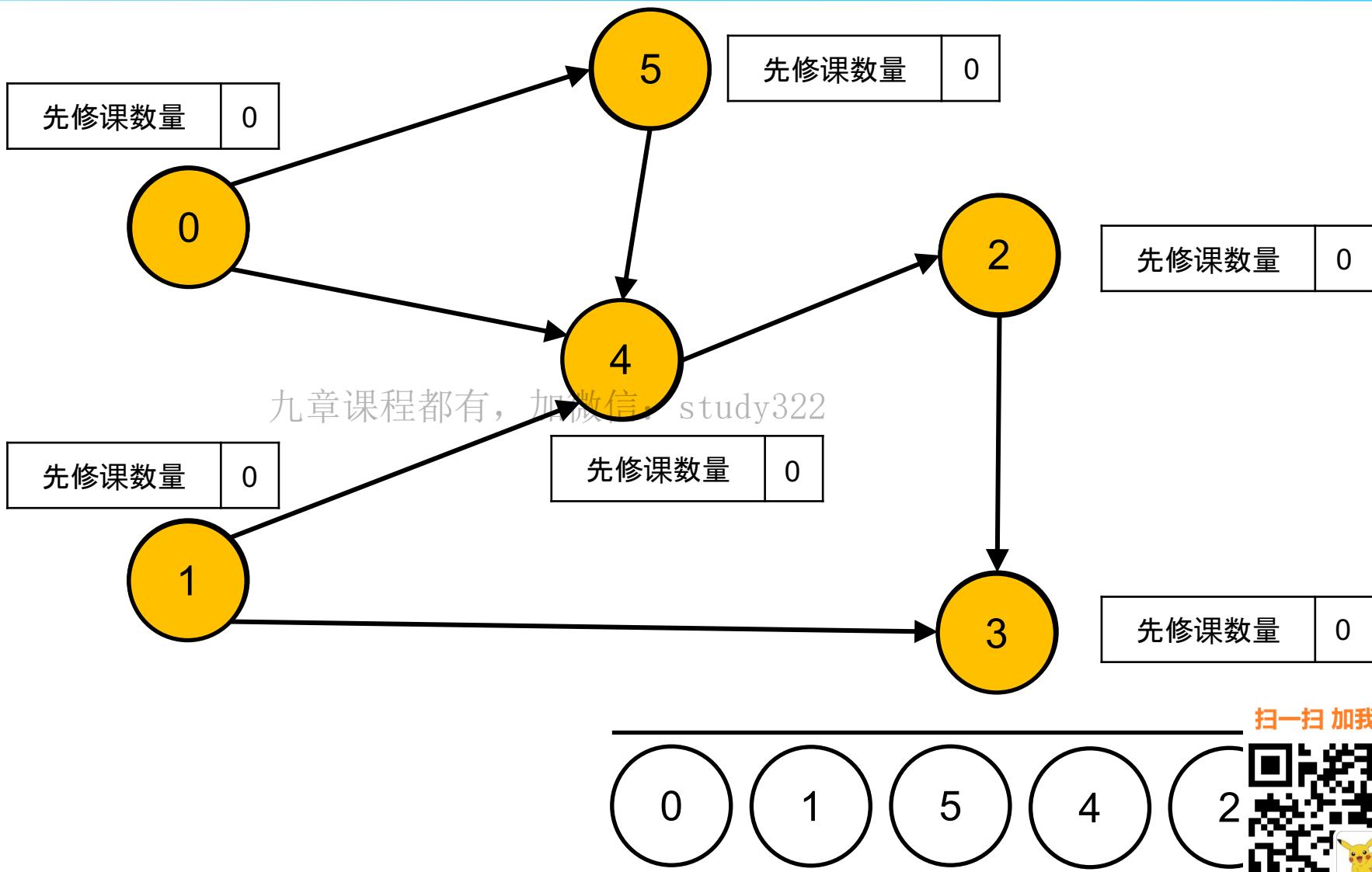
算法描述：

- ## 1. 统计每个点的入度

2. 将每个入度为 0 的点放入队列 (Queue) 中作为起始节点

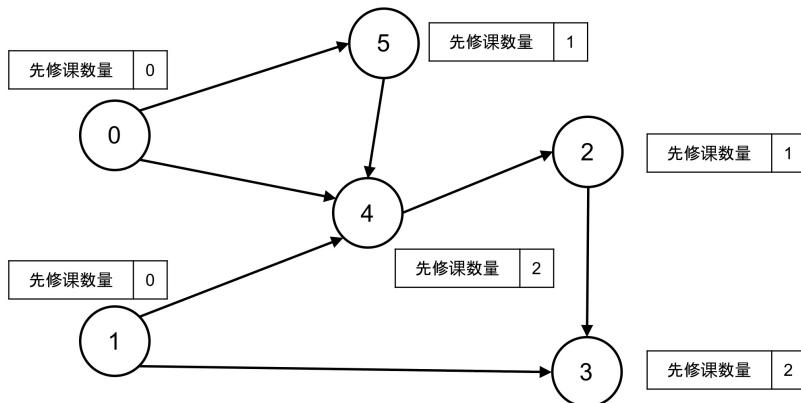
3. 不断从队列中拿出一个点，去掉这个点的所有连边（指向其他点的边），其他点的相应的入度 - 1

4. 一旦发现新的入度为 0 的点，丢回队列中



版权归九章算法(杭州)科技有限公司所有，贩卖和传播盗版将被追究刑事责任

代码解析



```

2     # @param {int} numCourses a total of n courses
3     # @param {int[][]} prerequisites a list of prerequisite pairs
4     # @return {int[]} the course order
5     def findOrder(self, numCourses, prerequisites):
6         # 构建图，代表（先修课->多个后修课）的映射
7         # 图的初始化，每个先修课->空后修课List
8         graph = [[] for i in range(numCourses)]
9
10        # 1. 统计每个点的入度，并构建图
11        in_degree = [0] * numCourses
12        for node_in, node_out in prerequisites:
13            graph[node_out].append(node_in)
14            in_degree[node_in] += 1
15
16        # 2. 将每个入度为 0 的点放入队列（Queue）中作为起始节点
17        queue = collections.deque()
18        for i in range(numCourses):
19            if in_degree[i] == 0:
20                queue.append(i)
21
22        # 记录已修课程的数量
23        num_choose = 0
24
25        # 记录拓扑顺序
26        topo_order = []
27
28        # 3. 不断从队列中拿出一个点，去掉这个点的所有连边（指向其他点的边）
29        # 其他点的相应的入度-1
30        while queue:
31            now_pos = queue.popleft()
32            topo_order.append(now_pos)
33            num_choose += 1
34            # 当前点的所有邻居的入度减1。表示每个后修课的一门先修课已经完成
35            for next_pos in graph[now_pos]:
36                in_degree[next_pos] -= 1
37                # 4. 一旦发现新的入度为 0 的点，丢回队列中。
38                # 表示一门后修课的所有先修课已经完成，可以被修了
39                if in_degree[next_pos] == 0:
40                    queue.append(next_pos)
41
42            # 如果全部课程已经被修过，那么返回拓扑排序，否则返回空
43            return topo_order if num_choose == numCourses else []
44
45
46
47
48
49
50

```

```

2     public int[] findOrder(int numCourses, int[][] prerequisites) {
3         // 构建图，代表（先修课->多个后修课）的映射
4         List[] graph = new ArrayList[numCourses];
5         // 图的初始化，每个先修课->空后修课List
6         for (int i = 0; i < numCourses; i++) {
7             graph[i] = new ArrayList<Integer>();
8         }
9
10        // 1. 统计每个点的入度，并构建图
11        int[] inDegree = new int[numCourses];
12        for (int[] edge: prerequisites) {
13            graph[edge[1]].add(edge[0]);
14            inDegree[edge[0]]++;
15        }
16
17        Queue queue = new LinkedList();
18
19        // 2. 将每个入度为 0 的点放入队列（Queue）中作为起始节点
20        for(int i = 0; i < inDegree.length; i++){
21            if (inDegree[i] == 0) {
22                queue.add(i);
23            }
24        }
25
26        // 记录已修课程的数量
27        int numChoose = 0;
28        // 记录拓扑顺序
29        int[] topoOrder = new int[numCourses];
30
31        // 3. 不断从队列中拿出一个点，去掉这个点的所有连边（指向其他点的边）， 其他点的相应的入度 - 1
32        while (!queue.isEmpty()) {
33            int nowPos = (int)queue.poll();
34            topoOrder[numChoose] = nowPos;
35            numChoose++;
36            for (int i = 0; i < graph[nowPos].size(); i++) {
37                int nextPos = (int)graph[nowPos].get(i);
38                // 当前点的邻居的入度减1。表示每个后修课的一门先修课已经完成
39                inDegree[nextPos]--;
40                // 4. 一旦发现新的入度为 0 的点，丢回队列中。
41                // 表示一门后修课的所有先修课已经完成，可以被修了
42                if (inDegree[nextPos] == 0) {
43                    queue.add(nextPos);
44                }
45            }
46        }
47
48        // 如果全部课程已经被修过，那么返回拓扑排序，否则返回空
49        return (numChoose == numCourses) ? topoOrder : []
50

```



扫一扫 加我免费送课

127 Topological Sorting 拓扑排序（求任意一个拓扑排序）

Given an directed graph, a topological order of the graph nodes is defined as follow:

For each directed edge $A \rightarrow B$ in graph, A must before B in the order list.

The first node in the order can be any node in the graph with no nodes direct to it.

Find any topological order for the given graph.

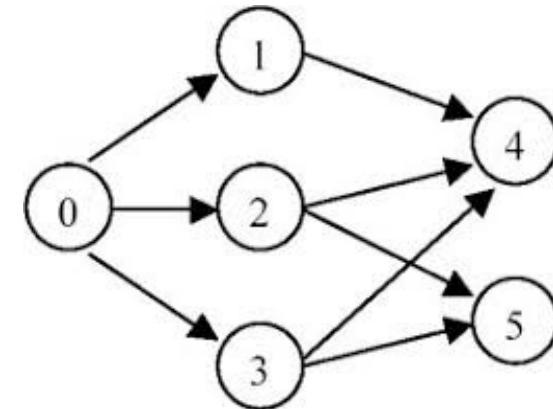
给定一个有向图，图节点的拓扑排序定义如下：

对于图中的每一条有向边 $A \rightarrow B$ ，在拓扑排序中A一定在B之前。

拓扑排序中的第一个节点可以是图中的任何一个没有其他节点指向它的节点。

针对给定的有向图找到任意一种拓扑排序的顺序。 九章课程都有，加微信：study322

你可以假设图中至少存在一种拓扑排序



拓扑排序可以为：

[0, 1, 2, 3, 4, 5]

[0, 2, 3, 1, 5, 4]

扫一扫 加我免费送课



代码解析

```

1  """
2  class DirectedGraphNode:
3      def __init__(self, x):
4          self.label = x
5          self.neighbors = []
6  """
7
8  class Solution:
9      def topSort(self, graph):
10         # 1. 统计每个点的入度
11         node_to_indegree = self.get_indegree(graph)
12
13         # 2. 将每个入度为 0 的点放入队列 (Queue) 中作为起始节点
14         start_nodes = [n for n in graph if node_to_indegree[n] == 0]
15         queue = collections.deque(start_nodes)
16
17         # 记录拓扑顺序
18         order = []
19
20         # 3. 不断从队列中拿出一个点，去掉这个点的所有连边（指向其他点的边），其他点的相应的入度 - 1
21         while queue:
22             node = queue.popleft()
23             order.append(node)
24             for neighbor in node.neighbors:
25                 node_to_indegree[neighbor] -= 1
26                 # 4. 一旦发现新的入度为 0 的点，丢回队列中
27                 if node_to_indegree[neighbor] == 0:
28                     queue.append(neighbor)
29
30         return order
31
32     # 统计每个点的入度。如果一个点的入度为0，那么这个点依然存在于dict中，对应入度为0
33     def get_indegree(self, graph):
34         # 初始化所有点的入度为0
35         node_to_indegree = {x: 0 for x in graph}
36
37         for node in graph:
38             # 所有邻居的入度加1
39             for neighbor in node.neighbors:
40                 node_to_indegree[neighbor] += 1
41
42         return node_to_indegree

```

```

1  /**
2  * Definition for Directed graph.
3  * class DirectedGraphNode {
4  *     int label;
5  *     List<DirectedGraphNode> neighbors;
6  *     DirectedGraphNode(int x) {
7  *         label = x;
8  *         neighbors = new ArrayList<DirectedGraphNode>();
9  *     }
10 */
11
12
13 public class Solution {
14     public ArrayList<DirectedGraphNode> topSort(ArrayList<DirectedGraphNode> graph) {
15         // 1. 统计每个点的入度
16         Map<DirectedGraphNode, Integer> indegree = getIndegree(graph);
17
18         // 2. 将每个入度为 0 的点放入队列 (Queue) 中作为起始节点
19         Queue<DirectedGraphNode> queue = new ArrayDeque<>();
20         for (DirectedGraphNode node : graph) {
21             if (!indegree.containsKey(node)) {
22                 queue.offer(node);
23             }
24         }
25
26         // 记录拓扑顺序
27         ArrayList<DirectedGraphNode> order = new ArrayList<>();
28
29         // 3. 不断从队列中拿出一个点，去掉这个点的所有连边（指向其他点的边），其他点的相应的入度 - 1
30         while (!queue.isEmpty()) {
31             DirectedGraphNode node = queue.poll();
32             order.add(node);
33             for (DirectedGraphNode neighbor : node.neighbors) {
34                 // 当前点的所有邻居的入度减1
35                 indegree.put(neighbor, indegree.get(neighbor) - 1);
36                 // 4. 一旦发现新的入度为 0 的点，丢回队列中
37                 if (indegree.get(neighbor) == 0) {
38                     queue.offer(neighbor);
39                 }
40             }
41         }
42
43         return order;
44     }

```

// 统计每个点的入度。如果一个点的入度为0，那么这个点就不存在于map中。

```

45     private Map<DirectedGraphNode, Integer> getIndegree(ArrayList<DirectedGraphNode> graph) {
46         Map<DirectedGraphNode, Integer> indegree = new HashMap<DirectedGraphNode, Integer>();
47         for (DirectedGraphNode node : graph) {
48             for (DirectedGraphNode neighbor : node.neighbors) {
49                 // if (indegree.containsKey(neighbor))
50                 //     indegree.put(neighbor, indegree.get(neighbor) + 1);
51                 // } else {
52                 //     indegree.put(neighbor, 1);
53                 // }
54                 // 下面这一行代码可以取代上面5行代码。如
55                 // 所有邻居入度加1
56                 indegree.put(neighbor, indegree.get(neighbor) + 1);
57             }
58         }
59     }
60
61     return indegree;
62 }

```



605 · Sequence Reconstruction (问拓扑排序是否唯一)

Check whether the original sequence org can be uniquely reconstructed from the sequences in seqs. The org sequence is a permutation of the integers from 1 to n. Reconstruction means building a shortest common supersequence of the sequences in seqs (i.e., a shortest sequence so that all sequences in seqs are subsequences of it). Determine **whether there is only one** sequence that can be reconstructed from seqs and it is the org sequence.

判断是否序列 org 能唯一地由 seqs 重构得出。org 是一个由从 1 到 n 的正整数排列而成的序列。重构表示组合成 seqs 的一个最短的父序列 (意思是，一个最短的序列使得所有 seqs 里的序列都是它的子序列)。

判断是否有且 **仅有** 一个能从 seqs 重构出来的序列，并且这个序列是 org。
上一章课和本章课都有加微信: study322

例1:

```
输入: org = [1,2,3], seqs = [[1,2],[1,3]]  
输出: false  
解释:  
[1,2,3] 并不是唯一可以被重构出的序列，还可以重构出 [1,3,2]
```

例2:

```
输入: org = [1,2,3], seqs = [[1,2]]  
输出: false  
解释:  
能重构出的序列只有 [1,2].
```

例3:

```
输入: org = [1,2,3], seqs = [[1,2],[1,3],[2,3]]  
输出: true  
解释:  
序列 [1,2], [1,3]，和 [2,3] 可以唯一重构出 [1,2,3].
```

扫一扫 加我免费送课



```

def sequenceReconstruction(self, org, seqs):
    graph = self.build_graph(seqs)
    topo_order = self.topological_sort(graph)
    return topo_order == org

def build_graph(self, seqs):
    # initialize graph
    graph = {}
    for seq in seqs:
        for node in seq:
            if node not in graph:
                graph[node] = set()
            for seq in seqs:
                for i in range(1, len(seq)):
                    graph[seq[i - 1]].add(seq[i])

    return graph

def get_indegrees(self, graph):
    indegrees = {
        node: 0
        for node in graph
    }

    for node in graph:
        for neighbor in graph[node]:
            indegrees[neighbor] += 1

    return indegrees
  
```

```

def topological_sort(self, graph):
    indegrees = self.get_indegrees(graph)

    queue = []
    for node in graph:
        if indegrees[node] == 0:
            queue.append(node)

    topo_order = []
    while queue:
        if len(queue) > 1:
            # there must exist more than one topo orders
            return None

        node = queue.pop()
        topo_order.append(node)
        for neighbor in graph[node]:
            indegrees[neighbor] -= 1
            if indegrees[neighbor] == 0:
                queue.append(neighbor)

    if len(topo_order) == len(graph):
        return topo_order

    return None
  
```

保持队列中有且仅有一个元素



```

public boolean sequenceReconstruction(int[] org, int[][] seqs) {
    Map<Integer, Set<Integer>> graph = buildGraph(seqs);
    List<Integer> topoOrder = getTopoOrder(graph);

    if (topoOrder == null || topoOrder.size() != org.length) {
        return false;
    }
    for (int i = 0; i < org.length; i++) {
        if (org[i] != topoOrder.get(i)) {
            return false;
        }
    }
    return true;
}

private Map<Integer, Set<Integer>> buildGraph(int[][] seqs) {
    Map<Integer, Set<Integer>> graph = new HashMap();
    for (int[] seq : seqs) {
        for (int i = 0; i < seq.length; i++) {
            if (!graph.containsKey(seq[i])) {
                graph.put(seq[i], new HashSet<Integer>());
            }
        }
    }
    for (int[] seq : seqs) {
        for (int i = 1; i < seq.length; i++) {
            graph.get(seq[i - 1]).add(seq[i]);
        }
    }
    return graph;
}

private Map<Integer, Integer> getIndegrees(Map<Integer, Set<Integer>> graph) {
    Map<Integer, Integer> indegrees = new HashMap();
    for (Integer node : graph.keySet()) {
        indegrees.put(node, 0);
    }
    for (Integer node : graph.keySet()) {
        for (Integer neighbor : graph.get(node)) {
            indegrees.put(neighbor, indegrees.get(neighbor) + 1);
        }
    }
    return indegrees;
}

private List<Integer> getTopoOrder(Map<Integer, Set<Integer>> graph) {
    Map<Integer, Integer> indegrees = getIndegrees(graph);
    Queue<Integer> queue = new LinkedList();
    List<Integer> topoOrder = new ArrayList();

    for (Integer node : graph.keySet()) {
        if (indegrees.get(node) == 0) {
            queue.offer(node);
            topoOrder.add(node);
        }
    }

    while (!queue.isEmpty()) {
        if (queue.size() > 1) {
            return null;
        }
        Integer node = queue.poll();
        for (Integer neighbor : graph.get(node)) {
            indegrees.put(neighbor, indegrees.get(neighbor) - 1);
            if (indegrees.get(neighbor) == 0) {
                queue.offer(neighbor);
                topoOrder.add(neighbor);
            }
        }
    }

    if (graph.size() == topoOrder.size()) {
        return topoOrder;
    }
    return null;
}

```

九章课程都有，加微信: study322



There is a new alien language which uses the latin alphabet. However, the order among letters are unknown to you. You receive a list of **non-empty** words from the dictionary, where words are **sorted lexicographically by the rules of this new language**. Derive the order of letters in this language.

有一种新的使用拉丁字母的外来语言。但是，你不知道字母之间的顺序。你会从词典中收到一个**非空的**单词列表，其中的单词**在这种新语言的规则下按字典顺序排序**。请推导出这种语言的字母顺序。

你可以假设所有的字母都是小写。

如果a是b的前缀且b出现在a之前，那么这个顺序是无效的。如果顺序是无效的，则返回空字符串。
这里可能有多个有效的字母顺序，返回以正常字典顺序看来最小的。

输入 : ["z", "x"]

输出 : "zx"

解释 : 从 "z" 和 "x" , 我们可以得到 'z' < 'x' 所以返回"zx"

输入 : ["wrt", "wrf", "er", "ett", "rftt"]

输出 : "wertf"

解释 :

从 "wrt"和"wrf" , 我们可以得到 't'<'f'

从 "wrt"和"er" ,我们可以得到'w'<'e'

从 "er"和"ett" ,我们可以得到 'r'<'t'

从 "ett"和"rftt" ,我们可以得到 'e'<'r'

所以返回 "wertf"

扫一扫 加我免费送课



思路解析 & 代码解析

输入：

["wwt", "wwf", "wer", "wett", "rftt"]

输出：

"wertf"

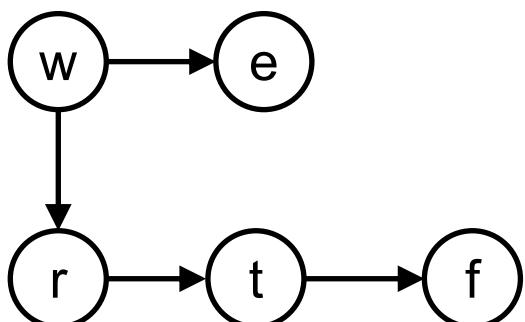
解释：

从 "wwt" 和 "wwf"，可以得到 't' < 'f'

从 "wwf" 和 "wer"，可以得到 'w' < 'e'

从 "wer" 和 "wett"，可以得到 get 'r' < 't'

从 "wett" 和 "rftt"，可以得到 'w' < 'r'



```
1  from heapq import heappush, heappop, heapify
2
3  class Solution:
4      """
5          @param words: a list of words
6          @return: a string which is correct order
7      """
8
9      def alienOrder(self, words):
10         graph = self.build_graph(words)
11         # 如果数据不合理, graph为空, 返回空字符串
12         if not graph:
13             return ""
14         return self.topological_sort(graph)
15
16     def build_graph(self, words):
17         # 存放(字母->右面的多个字母)的映射关系
18         graph = {}
19         # 生成所有的点, 每个点的后续点暂时为空
20         for word in words:
21             for c in word:
22                 if c not in graph:
23                     graph[c] = set()
24
25         # 生成所有的边, 找到一个点之后的点, 并建立连接
26         n = len(words)
27         for i in range(n - 1):
28             for j in range(min(len(words[i]), len(words[i + 1]))):
29                 if words[i][j] != words[i + 1][j]:
30                     graph[words[i][j]].add(words[i + 1][j])
31                     break
32                 # 如果输入["abc", "ab"], "abc"出现在"ab"前面, 不合法, 返回null
33                 if j == min(len(words[i]), len(words[i + 1])) - 1:
34                     if len(words[i]) > len(words[i + 1]):
35                         return None
36
37         return graph
```

```
39
40     def topological_sort(self, graph):
41         """
42             # 1. 统计每个点的入度
43             indegree = self.get_indegree(graph)
44
45             # 2. 将每个入度为 0 的点放入队列中作为起始节点
46             queue = [node for node in graph if indegree[node] == 0]
47             # 要求: 这里可能有多个有效的字母顺序, 返回以正常字典顺序看来最小的。
48             # 所以这里要heapify, 从所有可以出队的元素中, 先出队字典序比较小的元素
49             heapify(queue)
50
51             # 记录拓扑顺序(外星人字典排序)
52             topo_order = ""
53
54             # 3. 不断从队列中拿出一个点, 去掉这个点的所有连边(指向其他点的边),
55             # 其他点的相应的入度 - 1
56             while queue:
57                 node = heappop(queue)
58                 topo_order += node
59                 for neighbor in graph[node]:
60                     # 当前点的邻居的入度减1
61                     indegree[neighbor] -= 1
62                     # 4. 一旦发现新的入度为 0 的点, 丢回队列中
63                     if indegree[neighbor] == 0:
64                         heappush(queue, neighbor)
65
66             # 如果有些字母没有出现在字典排序中, 那么没有答案
67             return topo_order if len(topo_order) == len(graph) else ""
68
69         # 统计每个点的入度。如果一个点的入度为0, 那么这个点依然存在于dict中, 对应入度为0
70         def get_indegree(self, graph):
71             """
72                 # 初始化所有点的入度为0
73                 indegree = {node: 0 for node in graph}
74
75                 for node in graph:
76                     # 所有邻居的入度加1
77                     for neighbor in graph[node]:
78                         indegree[neighbor] = indegree[neighbor] + 1
79
80             return indegree
```

扫一扫 加我免费送课



代码解析

```

2  public String alienOrder(String[] words) {
3      Map<Character, Set<Character>> graph = constructGraph(words);
4      // 如果数据不合理, graph为空, 返回空字符串
5      if (graph == null) {
6          return "";
7      }
8      return topologicalSorting(graph);
9  }
10
11
12 private Map<Character, Set<Character>> constructGraph(String[] words) {
13     // 存放(字母->右面的多个字母)的映射关系
14     Map<Character, Set<Character>> graph = new HashMap<>();
15
16     // 生成所有的点, 每个点的后续点暂时为空
17     for (int i = 0; i < words.length; i++) {
18         for (int j = 0; j < words[i].length(); j++) {
19             char c = words[i].charAt(j);
20             if (!graph.containsKey(c)) {
21                 graph.put(c, new HashSet<Character>());
22             }
23         }
24     }
25
26     // 生成所有的边, 找到一个点之后的点, 并建立连接
27     for (int i = 0; i < words.length - 1; i++) {
28         int index = 0;
29         while (index < words[i].length() && index < words[i + 1].length()) {
30             if (words[i].charAt(index) != words[i + 1].charAt(index)) {
31                 graph.get(words[i].charAt(index)).add(words[i + 1].charAt(index));
32                 break;
33             }
34             index++;
35         }
36         // 如果输入为 ["abc", "ab"], "abc"出现在"ab"前面不合法, graph为null
37         if (index == Math.min(words[i].length(), words[i + 1].length())) {
38             if (words[i].length() > words[i + 1].length()) {
39                 return null;
40             }
41         }
42     }
43
44     return graph;
45 }

```

```

47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99

```

```

private String topologicalSorting(Map<Character, Set<Character>> graph) {
    // 1. 统计每个点的入度
    Map<Character, Integer> indegree = getIndegree(graph);

    // 要求: 这里可能有多个有效的字母顺序, 返回以正常字典顺序看来最小的。
    // 所以这里要选择PriorityQueue, 从所有可以出队的元素中, 先出队字典序比较小的元素
    Queue<Character> queue = new PriorityQueue<>();

    // 2. 将每个入度为 0 的点放入队列 (Queue) 中作为起始节点
    for (Character c : indegree.keySet()) {
        if (indegree.get(c) == 0) {
            queue.offer(c);
        }
    }
    // 记录拓扑顺序(外星人字典排序)
    StringBuilder sb = new StringBuilder();

    // 3. 不断从队列中拿出一个点, 去掉这个点的所有连边 (指向其他点的边), 其他点的相应的入度 - 1
    while (!queue.isEmpty()) {
        Character head = queue.poll();
        sb.append(head);
        for (Character neighbor : graph.get(head)) {
            // 当前点的邻居的入度减1
            indegree.put(neighbor, indegree.get(neighbor) - 1);
            // 4. 一旦发现新的入度为 0 的点, 丢回队列中
            if (indegree.get(neighbor) == 0) {
                queue.offer(neighbor);
            }
        }
    }

    // 如果有些字母没有出现在字典排序中, 那么没有答案
    return (sb.length() == indegree.size()) ? sb.toString() : "";
}

// 统计每个点的入度。如果一个点的入度为0, 那么这个点依然存在于dict中, 对应入度为0
private Map<Character, Integer> getIndegree(Map<Character, Set<Character>> graph) {
    Map<Character, Integer> indegree = new HashMap<>();

    // 初始化所有点的入度为0
    for (Character u : graph.keySet()) {
        indegree.put(u, 0);
    }

    for (Character u : graph.keySet()) {
        // 所有邻居的入度加1
        for (Character v : graph.get(u)) {
            indegree.put(v, indegree.get(v) + 1);
        }
    }

    return indegree;
}

```

九章课程都有，加微信

study322

扫一扫 加我免费送课



- 图上的 BFS
 - 判断一个图是否是一棵树
 - <http://www.lintcode.com/problem/graph-valid-tree/>
 - 搜索图中最近值为target的点
 - <http://www.lintcode.com/problem/search-graph-nodes/>
 - 无向图连通块 九章课程都有，加微信：study322
 - <http://www.lintcode.com/problem/connected-component-in-undirected-graph/>
- 矩阵上的 BFS
 - 僵尸多少天吃掉所有人
 - <http://www.lintcode.com/problem/zombie-in-matrix/>
 - 建邮局问题 Build Post Office II
 - <http://www.lintcode.com/problem/build-post-office-ii/>

扫一扫 加我免费送课



- 能用 BFS 的一定不要用 DFS (除非面试官特别要求)
- BFS 的三个使用场景
 - 连通块问题
 - 层级遍历问题
 - 拓扑排序问题
- 是否需要层级遍历
- 矩阵坐标变换数组
 - `deltaX`, `deltaY`
 - 是否在界内 : `isInBound` / `isValid`

九章课程都有，加微信：study322

扫一扫 加我免费送课



- 第三十二章 使用宽度优先搜索找出所有方案
 - 使用 BFS 求所有方案类问题
 - 使用 BFS 序列化二叉树
 - 什么是序列化与反序列化
- 第三十三章 【互动】双向宽度优先搜索算法上一章课业都有，加微信：study322
 - 双向宽度优先搜索算法
 - 双向宽度优先搜索到底优化了多少
 - 如何优雅的实现双向宽度优先搜索

扫一扫 加我免费送课



Thanks♪(・ω・)/



扫一扫 加我免费送课



同学们，下次再见挥手！记得课后复习，课前预习！😊

做作业

做ladder

群里提问题

看回放



课前预习

课后复习

刷题

扫一扫 加我免费送课

互

