链滴

回溯法套路总结与应用

作者：vcjmhg

原文链接：https://ld246.com/article/1605623439879

来源网站：链滴

概述

回溯法常用于遍历一个列表元素的所有所有子集，比如全排列问题。可以说深度优先搜索就是回溯法一种特殊形式。该方法的时间复杂度比较大一般为O(N!)，它不像动态规划存在重叠子问题可以优化当然在某些情况下，我们可以通过剪枝来进行优化，当然这个技巧性可能较高。本篇文章主要从回溯的基本思想入手，总结出一套模板，灵活运用模板便可以讲解大部分回溯算法的问题。

模板与算法

回溯法其实核心思想就是以深度优先进行暴力穷举，最重要的是做到补充不漏，整个过程跟决策树的历是类似的。其通用的一个算法模板如下：

<pre spellcheck="false" class="md-fences md-end-block md-fences-with-lineno ty-contain-m modeLoaded" lang="python" cid="n3" mdtype="fences"> <san class="cm-variable">result = []
 <spanlass="cm-keyword">def backtrack(路径, 选择列表):
 <spanole="presentation"> if 足结束条件:
   result.add(路径)
   return</pan>

span role="presentation"> for 选择 in 选择列:
   做选择/span>
   backtrac(路径, 选择列表</spn>)
   撤销选择</spn></pre>

原文链接：回溯法套路总结与应用

从这个模板中，我们可以看出，要用好回溯算法其实重点就是解决三个问题：

1. 路径：也就是已经做出的选择
2. 选择列表
3. 结束条件

下边我们通过一个全排列问题来展开说明：

全排列问题

具体题目如下：

46. 全排列

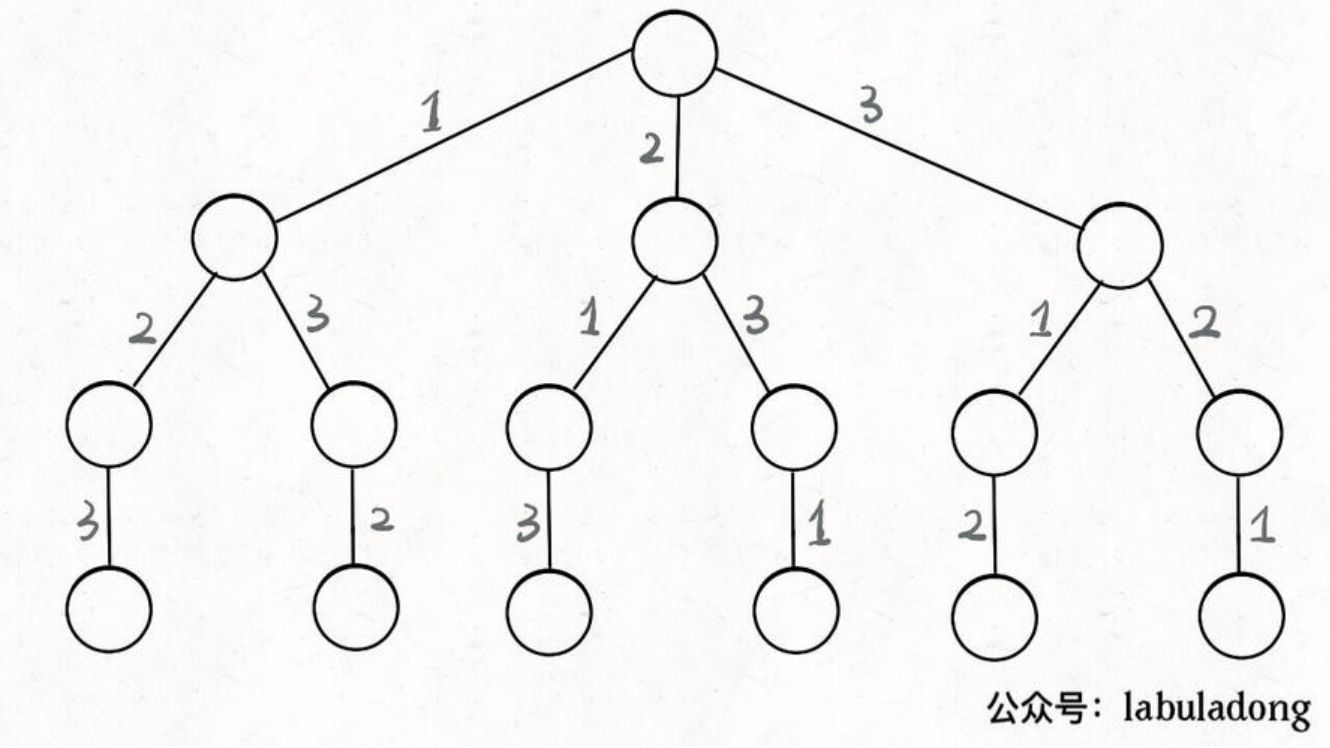
难度 中等 994 收藏 分享 切换为英文 接收动态 反馈

给定一个 **没有重复** 数字的序列，返回其所有可能的全排列。

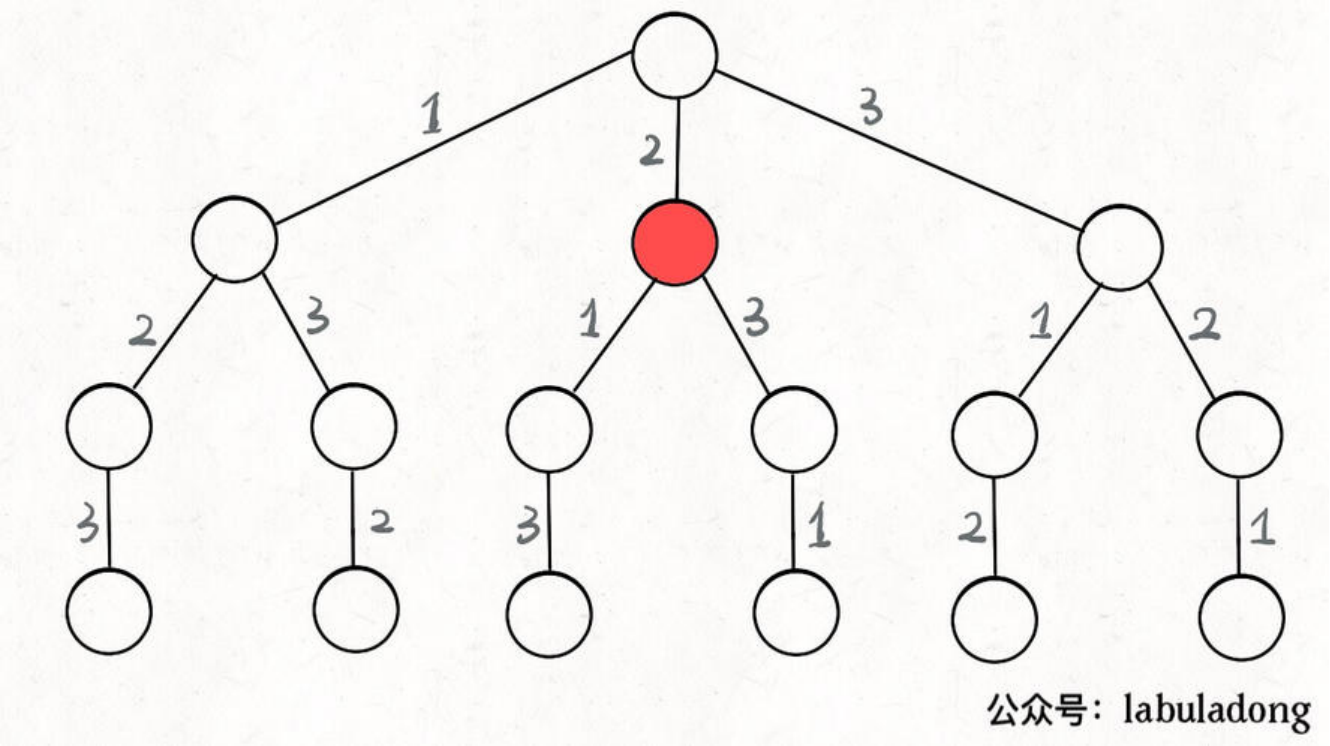
示例:

```
输入: [1,2,3]
输出:
[
  [1,2,3],
  [1,3,2],
  [2,1,3],
  [2,3,1],
  [3,1,2],
  [3,2,1]
]
```

问题本身很简单，我们高中的时候可能就会通过画决策树来解决整个问题：



为啥说这是决策树呢，因为你在每个节点上其实都在做决策。比如说你站在下图的红色节点上：



你现在就在做决策，可以选择 1 那条树枝，也可以选择 3 那条树枝。为啥只能在 1 和 3 之中选择呢，因为 2 这个树枝在你身后，这个选择你之前做过了，而全排列是不允许重复使用数字的。

通过前边的说明，针对该问题需要解决的三个问题分别为：

1. 路径：[2]就是路径，记录已经做过的选择
2. 选择列表：[1,3]就是选择列表，表示接下来可进行选择元素的列表
3. 结束条件：便利到树的底层，也就是说选择列表为空的时候结束

进而套用上边的模板，我们可以得出如下代码：

<pre spellcheck="false" class="md-fences md-end-block md-fences-with-lineno ty-contain-m modeLoaded" lang="java" cid="n65" mdtype="fences">List<List<Ineger>> result = new ArrayList<>();

/**
* 使用回溯法，解决全排列问题
* @param nums
* @return
*/
public List<List<Integer>> permute(int[] nums) {
<span r

le="presentation"> LinkedList<Integer> track = new LinkedList<>();
  backtrace(nums, track);
  return result;
  }

  private void backtrace(int[] nums, LinkedList<Integer> track) {
  // add track
  if (track.size() == nums.length) {
  result.add(new LinkedList<>(track));
  }
  for (int num : nums) {
  if (track.contains(num)) {
  continue
  }
  // make choice
  track.add(num);
  // recursive
  backtrace(nums, track);
  // backtrace choince
  track.removeLast();
  }
  }</pre>

应用

同样的再做一道题目练习一下：

47. 全排列 II

难度 中等 525 收藏 分享 切换为英文 接收动态 反馈

给定一个可包含重复数字的序列 `nums` ，**按任意顺序** 返回所有不重复的全排列。

示例 1：

```
输入：nums = [1,1,2]
输出：
[[1,1,2],
 [1,2,1],
 [2,1,1]]
```

示例 2：

```
输入：nums = [1,2,3]
输出：[[1,2,3],[1,3,2],[2,1,3],[2,3,1],[3,1,2],[3,2,1]]
```

该问题，较之上一个问题，最大的不同就是会有重复的元素，因此会增加重复元素的判断，但整体难也不大，套用模板可以得出如下解决方案：

<pre spellcheck="false" class="md-fences md-end-block md-fences-with-lineno ty-contain-m modeLoaded" lang="java" cid="n83" mdtype="fences"> <spa class="cm-variable">List<List<Inte er> result</spa > = new ArrayList<>();
  byte[] visited = <span class="cm-keyword" new byte[22</sp n>];

 < pan role="presentation"> public List<List</sp n><Integer<spa class="cm-operator">> permuteUnique(int[] nums) {
 span role="presentation"> LinkedList<Integer> track =</spa > new LinkedList< pan class="cm-operator"><>();
  Arrays.sort(nums);
  backtrace(nums, track);
   eturn result;
  }
 <b /> private void backtrace(int[] nums, L nkedList<Intege > track) { /span>
 <span role="presentation"> <span class="cm-comment">// 倡导到达nums.le

ght则记录路径
   if (nums.lengt == tracksize()) {
   result.add(new LinkedList<>(track));
   }
   for (int i = 0; i < nums.length; i++) {
   //已经添加过的元素直接跳过
   if (visited[i] == 1) {
   continue;
   }
   //与上一个元素相同，且没添加过直接跳过
   if (i != 0 && nums[i] == nums[i- 1] && visited[i - 1] == 0) {
   continue;
   }
   visited[i] = 1;
   track.add(nums[i]);
   backtrace(nums, track);
   track.removeLast();
   visited[i] = 0;
   }
   }</pre>

总结

回溯算法就是个多叉树的遍历问题，关键就是在前序遍历和后序遍历的位置做一些操作，算法框架如：

<pre spellcheck="false" class="md-fences md-end-block md-fences-with-lineno ty-contain-m modeLoaded" lang="java" cid="n88" mdtype="fences"> result = []
 def backtrack(路径, 选择表):
   if</sp

n> 满足结束条件:
 <span role="presentatio
"> result.add(<
pan class="cm-variable">路径)
  <spa
class="cm-keyword">return
 <span cm-t
xt="">
  fo
 选择 in <s
an class="cm-variable">选择列表:
  <s
an class="cm-variable">做选择
  <spa
class="cm-def">backtrack(路径, <span class="
m-variable">选择列表)
  <span class="
m-variable">撤销选择</pre>

写 backtrack 函数时，需要维护走过的「路径」和当前可以做的「选择列表」，当触发「结束条件」，将「路径」记入结果集。

只要确定了选择的条件，以及记录路径的调整，整个代码很容易便可以实现出来了。

参考

1. https://labuladong.gitbook.io/algo/di-ling-zhang-bi-du-xi-lie/hui-su-suan-fa-xiang-jie-xiu-ing-ban#san-zui-hou-zong-jie

2. https://greyireland.gitbook.io/algorithm-pattern/suan-fa-si-wei/backtrack#permutations