[Leetcode] Backtracking回溯法(又称DFS,递归)全解

转

[Leetcode] Backtracking回溯法(又称DFS,递归)全解

2018年07月22日 23:10:01 阅读数：57更多

回溯全集

回溯是啥

用爬山来比喻回溯，好比从山脚下找一条爬上山顶的路,起初有好几条道可走,当选择一条道走到某处时,又有几条岔道可供选择,只能选择其中一条道往前走,若能这样子顺利爬上山顶则罢了,否则走到一条绝路上时,只好返回到最近的一个路口,重新选择另一条没走过的道往前走。如果该路口的所有路都走不通,只得从该路口继续回返。照此规则走下去,要么找到一条到达山顶的路,要么最终试过所有可能的道,无法到达山顶。
回溯是一种穷举，但与brute force有一些区别，回溯带了两点脑子的，并不多，brute force一点也没带。
第一点脑子是回溯知道回头；相反如果是brute force,发现走不通立刻跳下山摔死，换第二条命从头换一条路走。
第二点脑子是回溯知道剪枝；如果有一条岔路上放了一坨屎，那这条路我们不走，就可以少走很多不必要走的路。

还有一些爱混淆的概念：递归，回溯，DFS。
回溯是一种找路方法，搜索的时候走不通就回头换路接着走，直到走通了或者发现此山根本不通。
DFS是一种开路策略，就是一条道先走到头，再往回走一步换一条路走到头，这也

是回溯用到的策略。在树和图上回溯时人们叫它DFS。
递归是一种行为，回溯和递归如出一辙，都是一言不合就回到来时的路，所以一般回溯用递归实现；当然也可以不用，用栈。
以下以回溯统称，因为这个词听上去很文雅。

识别回溯

判断回溯很简单，拿到一个问题，你感觉如果不穷举一下就没法知道答案，那就可以开始回溯了。
一般回溯的问题有三种：

Find a path to success 有没有解

Find all paths to success 求所有解

求所有解的个数

求所有解的具体信息

Find the best path to success 求最优解

理解回溯：给一堆选择, 必须从里面选一个. 选完之后我又有了新的一组选择. This procedure is repeated over and over until you reach a final state. If you made a good sequence of choices, your final state is a goal state; if you didn't, it isn't.

回溯可以抽象为一棵树，我们的目标可以是找这个树有没有good leaf，也可以是问有多少个good leaf，也可以是找这些good leaf都在哪，也可以问哪个good leaf最好，分别对应上面所说回溯的问题分类。
good leaf都在leaf上。good leaf是我们的goal state，leaf node是final state，是解空间的边界。

对于第一类问题(问有没有解)，基本都是长着个样子的，理解了它，其他类别迎刃而解：

```
boolean solve(Node n) {
```

```
    if the leaf is a goal node, return true
    else return false
  } else {
    for each child c of n {
      if solve(c) succeeds, return true
    }
    return false
  }
}
```

请读以下这段话以加深理解：

Notice that the algorithm is expressed as a boolean function. This is essential to understanding the algorithm. If solve(n) is true, that means node n is part of a solution--that is, node n is one of the nodes on a path from the root to some goal node. We say that n is solvable. If solve(n) is false, then there is no path that includes n to any goal node.

还不懂的话请通读全文吧：[Backtracking - David Matuszek]

关于回溯的三种问题，模板略有不同，
第一种，返回值是true/false。
第二种，求个数，设全局counter，返回值是void；求所有解信息，设result，返回值void。
第三种，设个全局变量best，返回值是void。

第一种：

```
boolean solve(Node n) {
  if n is a leaf node {
    if the leaf is a goal node, return true
    else return false
  } else {
    for each child c of n {
      if solve(c) succeeds, return true
    }
    return false
  }
}
```

第二种：

```
void solve(Node n) {
    if n is a leaf node {
        if the leaf is a goal node, count++, return;
        else return
    } else {
        for each child c of n {
            solve(c)
        }
    }
}
```

第三种：

```
void solve(Node n) {
    if n is a leaf node {
        if the leaf is a goal node, update best result, return;
        else return
    } else {
        for each child c of n {
            solve(c)
        }
    }
}
```

题目

八皇后 N-Queens

问题

1.给个n，问有没有解；
2.给个n，有几种解；(Leetcode N-Queens II)
3.给个n，给出所有解；(Leetcode N-Queens I)

解答

1.有没有解

怎么做：一行一行的放queen，每行尝试n个可能，有一个可达，返回true；都不可达，返回false.

边界条件leaf:放完第n行 或者 该放第n+1行(出界，返回)

目标条件goal:n行放满且isValid，即目标一定在leaf上

helper函数：
boolean solve(int i, int[][] matrix)
在进来的一瞬间，满足property：第i行还没有被放置，前i-1行放置完毕且valid
solve要在给定的matrix上试图给第i行每个位置放queen。

```
public static boolean solve1(int i, List<Integer> matrix, int n) {
    if (i == n) {
        if (isValid(matrix))
            return true;
        return false;
    } else {
        for (int j = 0; j < n; j++) {
            matrix.add(j);
            if (isValid(matrix)) {    //剪枝
                if (solve1(i + 1, matrix, n))
                    return true;
            }
            matrix.remove(matrix.size() - 1);
        }
        return false;
    }
}
```

2.求解的个数

怎么做：一行一行的放queen，每行尝试n个可能。这回因为要找所有，返回值就没有了意义，用void即可。在搜索时，如果有一个可达，仍要继续尝试；每个子选项都试完了，返回.

边界条件leaf:放完第n行 或者 该放第n+1行(出界，返回)

目标条件goal:n行放满且isValid，即目标一定在leaf上

helper函数：
void solve(int i, int[][] matrix)
在进来的一瞬间，满足property：第i行还没有被放置，前i-1行放置完毕且valid
solve要在给定的matrix上试图给第i行每个位置放queen。
这里为了记录解的个数，设置一个全局变量(static)int是比较efficient的做法。

```
public static void solve2(int i, List<Integer> matrix, int n) {
    if (i == n) {
        if (isValid(matrix))
            count++;
        return;
    } else {
        for (int j = 0; j < n; j++) {
            matrix.add(j);
            if (isValid(matrix)) {    //剪枝
                solve2(i + 1, matrix, n);
            }
            matrix.remove(matrix.size() - 1);
        }
    }
}
```

3.求所有解的具体信息

怎么做：一行一行的放queen，每行尝试n个可能。返回值同样用void即可。在搜索时，如果有一个可达，仍要继续尝试；每个子选项都试完了，返回.

边界条件leaf:放完第n行 或者 该放第n+1行(出界，返回)

目标条件goal:n行放满且isValid，即目标一定在leaf上

helper函数：
void solve(int i, int[][] matrix)
在进来的一瞬间，满足property：第i行还没有被放置，前i-1行放置完毕且valid
solve要在给定的matrix上试图给第i行每个位置放queen。
这里为了记录解的具体情况，设置一个全局变量(static)集合是比较efficient的做

法。
当然也可以把结果集合作为参数传来传去。

```
public static void solve3(int i, List<Integer> matrix, int n) {
    if (i == n) {
        if (isValid(matrix))
            result.add(new ArrayList<Integer>(matrix));
        return;
    } else {
        for (int j = 0; j < n; j++) {
            matrix.add(j);
            if (isValid(matrix)) {    //剪枝
                solve3(i + 1, matrix, n);
            }
            matrix.remove(matrix.size() - 1);
        }
    }
}
```

优化

上面的例子用了省空间的方法。
由于每行只能放一个，一共n行的话，用一个大小为n的数组，数组的第i个元素表示第i行放在了第几列上。

Utility(给一个list判断他的最后一行是否和前面冲突):

```
public static boolean isValid(List<Integer> list){
    int row = list.size() - 1;
    int col = list.get(row);
    for (int i = 0; i <= row - 1; i++) {
        int row1 = i;
        int col1 = list.get(i);
        if (col == col1)
            return false;
        if (row1 - row == col1 - col)
            return false;
```

```
            return false;
        }
        return true;

    }
```

新浪微博
微信
Twitter
Facebook

赞 | 9收藏 | 21

你可能感兴趣的文章

leetcode62. Unique Pathsraledongdynamic-programmingmatharraylistjavaleetcode
（一）Android Studio 安装部署 华丽躲坑guo289702431android
有向图欧拉回路的快速算法（POJ 2230题解）Michael_Lin图论poj算法
Combination Sum和深度优先搜索Depth-First-Search发条橙子算法javascript前端
技术人攻略访谈十一： CSDN社区技术大总管的十年技术人攻略技术人攻略访谈程序员
Uninformed search Python实现【译】fireflow算法
《深入浅出机器学习》之强化学习方老司机器学习
[LeetCode] #206: Reverse Linked List (递代&递归解法)海滩边儿的Codylinkedlistleetcodejava

https://segmentfault.com/a/1190000006121957

阅读更多

*leetcode*回溯算法（*backtracking*）总结

回溯算法的定义：回溯算法也叫试探法，它是一种系统地搜索问题的解的方法。回溯算法的基本思想是：从一条路往前走，能进则进，不能进则退回来，换一条路再试。回溯算法实际上一个类似枚举的搜索尝试过程，主要是在搜...

想对作者说点什么？ 我来说一句