BITTIGER

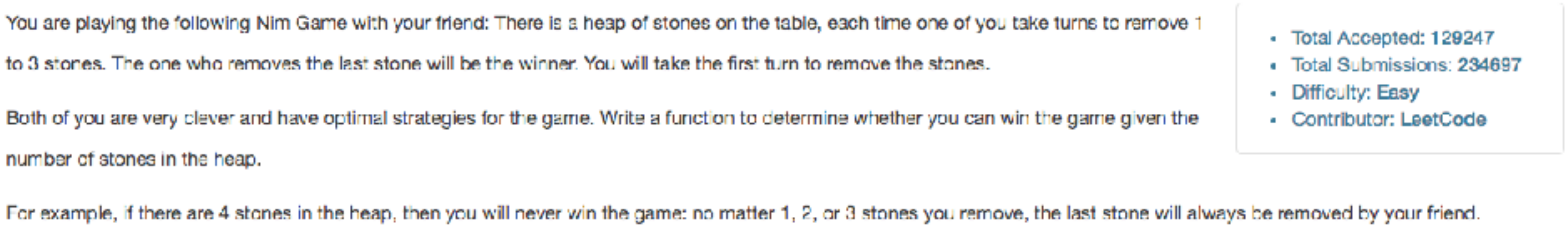
CLASS_6 COMPREHENSIVE TRAINING

Ben Gong

Content of Class_5

292 Nim Game
464 Can I Win
22 Generate Paentheses
56 Merge Intervals
473 Path Sum 3

292. Nim Game



You are playing the following Nim Game with your friend: There is a heap of stones on the table, each time one of you take turns to remove 1 to 3 stones. The one who removes the last stone will be the winner. You will take the first turn to remove the stones.

Both of you are very clever and have optimal strategies for the game. Write a function to determine whether you can win the game given the number of stones in the heap.

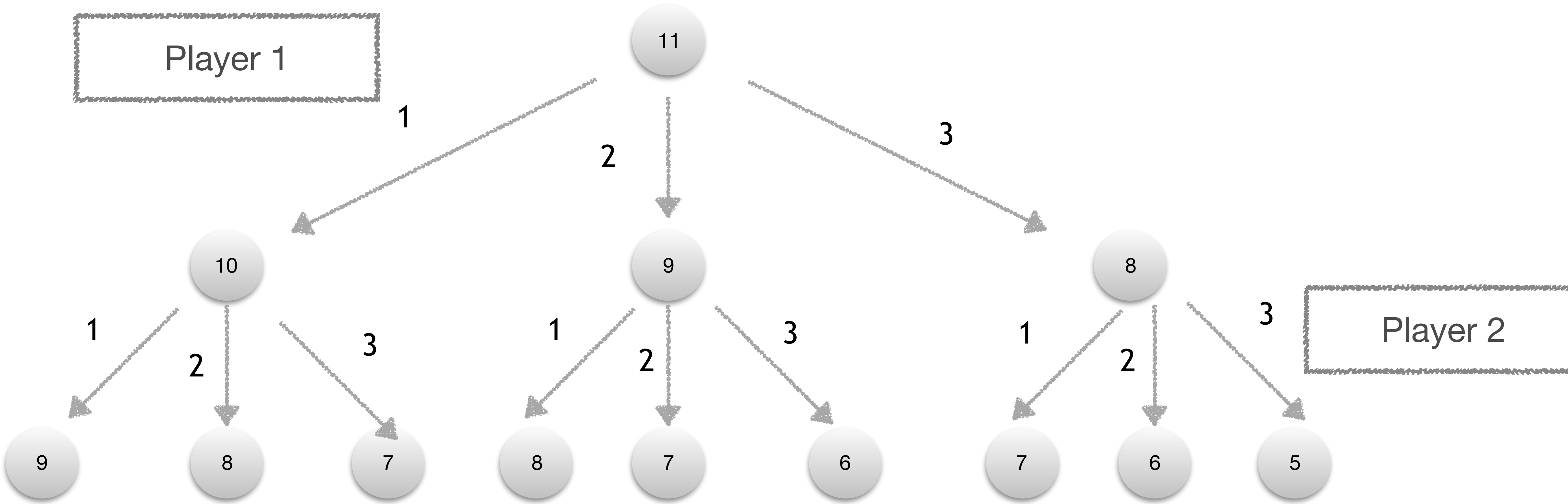
For example, if there are 4 stones in the heap, then you will never win the game: no matter 1, 2, or 3 stones you remove, the last stone will always be removed by your friend.

- Total Accepted: 129247
- Total Submissions: 234697
- Difficulty: Easy
- Contributor: LeetCode

20 min

```
public boolean canWinNim(int n) {

}
```

Memorized Search V.S. Dynamic Programming

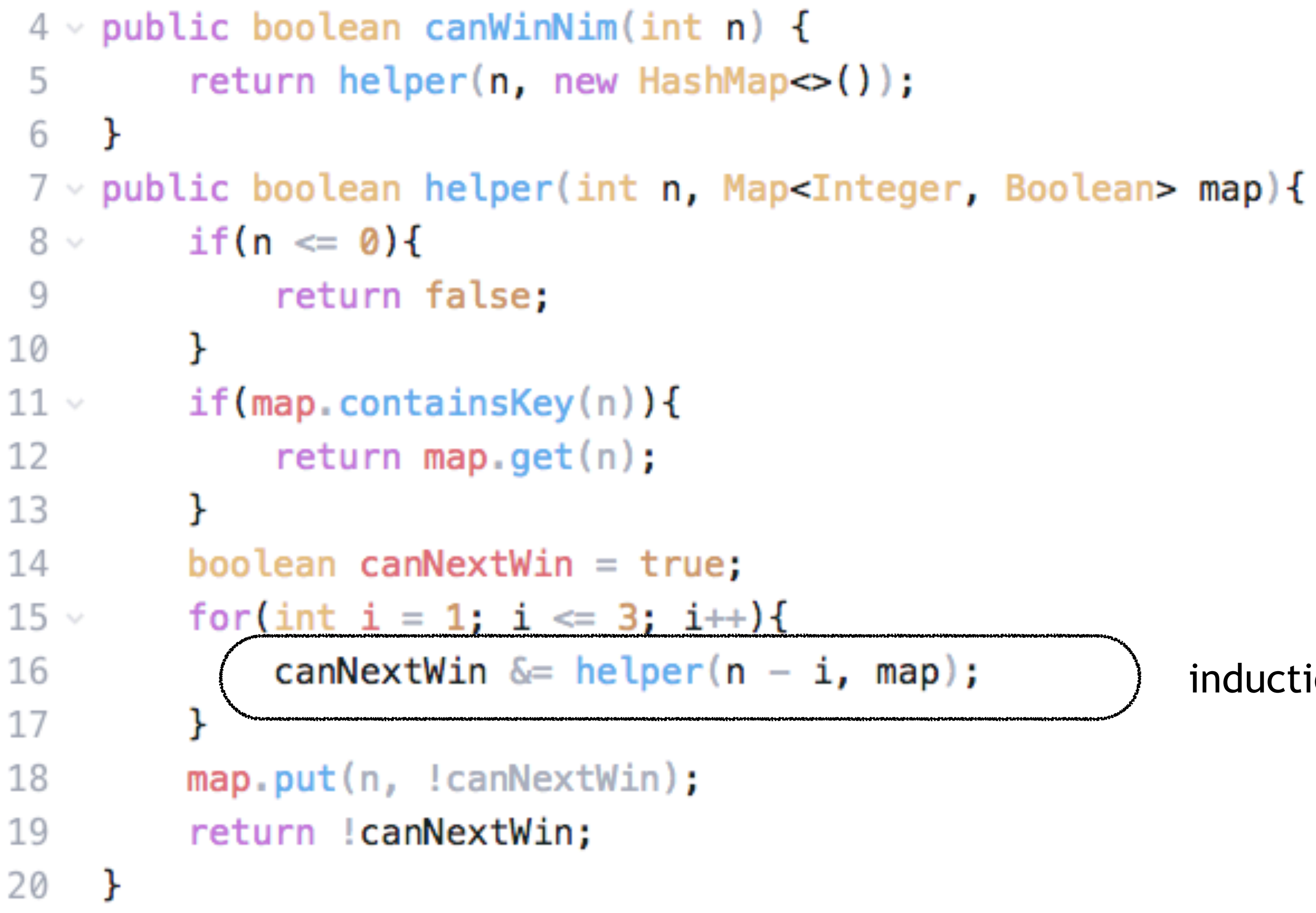


if NextLayer has a chance to loose, then current layer has a chance to win

翻译为编程语言

```java
public boolean canWinNim(int n) {
    return canWinNimDp(n);
}

public boolean canWinNimDp(int n){
    if(n <= 3){
        return true;
    }
    boolean[] dp = new boolean[4];
    dp[0] = false;
    dp[1] = true;
    dp[2] = true;
    dp[3] = true;
    for(int i = 4; i <= n; i++){
        dp[i % 4] = !(dp[(i - 1) % 4] && dp[(i - 2) % 4] && dp[(i - 3) % 4]);
    }
    return dp[n % 4];
}
```

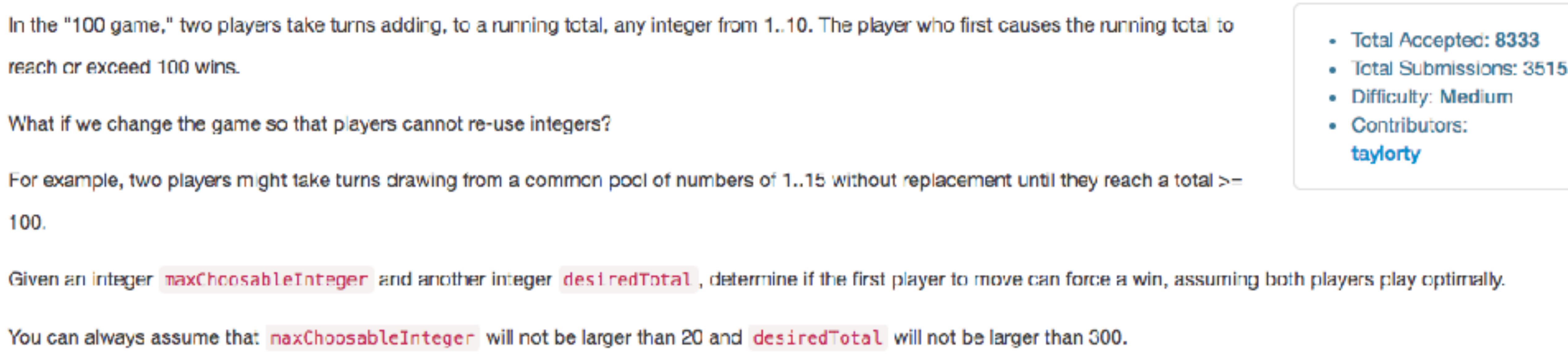
Rolling Array



```java
public boolean canWinNim(int n) {
    return helper(n, new HashMap<>());
}
public boolean helper(int n, Map<Integer, Boolean> map){
    if(n <= 0){
        return false;
    }
    if(map.containsKey(n)){
        return map.get(n);
    }
    boolean canNextWin = true;
    for(int i = 1; i <= 3; i++){
        canNextWin &= helper(n - i, map);
    }
    map.put(n, !canNextWin);
    return !canNextWin;
}
```

induction rule

464. Can I Win



In the "100 game," two players take turns adding, to a running total, any integer from 1..10. The player who first causes the running total to reach or exceed 100 wins.

What if we change the game so that players cannot re-use integers?

For example, two players might take turns drawing from a common pool of numbers of 1..15 without replacement until they reach a total >= 100.

Given an integer `maxChoosableInteger` and another integer `desiredTotal`, determine if the first player to move can force a win, assuming both players play optimally.

You can always assume that `maxChoosableInteger` will not be larger than 20 and `desiredTotal` will not be larger than 300.

- Total Accepted: **8333**
- Total Submissions: 3515
- Difficulty: **Medium**
- Contributors: **taylorty**

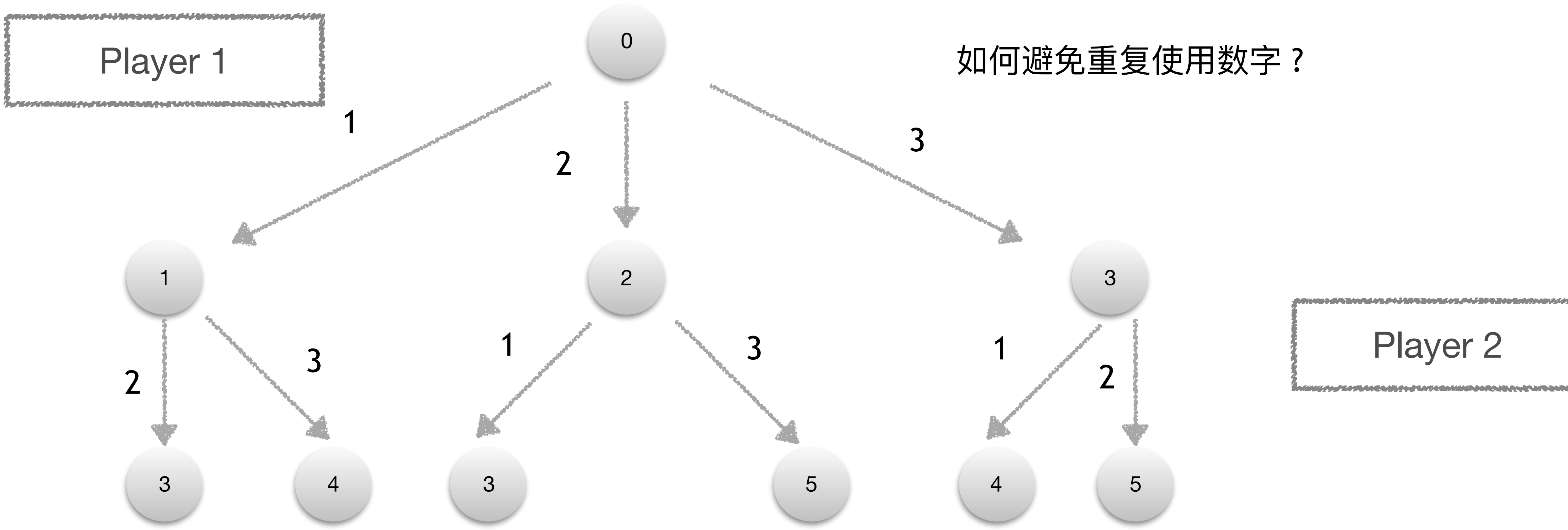
20 min

```
public boolean canIWin(int maxChoosableInteger, int desiredTotal) {

}
```

Memorized Search V.S. Dynamic Programming

如何记录重复结果（memorized search）？

如何避免重复使用数字？



if NextLayer has a chance to loose, then current layer has a chance to win

翻译为编程语言

```java
public boolean canIWin(int maxChoosableInteger, int desiredTotal) {
    if (desiredTotal <= 0) {
        return true;
    }
    if (maxChoosableInteger * (maxChoosableInteger + 1) / 2 < desiredTotal){
        return false;
    }

    return canIWin(desiredTotal, maxChoosableInteger, 0, new HashMap<>());
}
private boolean canIWin(int total, int n, int state, HashMap<Integer, Boolean> cache) {

    if(total <= 0){
        return false;
    }

    if (cache.containsKey(state)) {
        return cache.get(state);
    }

    for (int i = 0;i < n; i++) {
        if ((state & (1 << i)) != 0) {
            continue;
        }
        if (!canIWin(total-(i+1), n, state | (1<<i), cache)) {
            cache.put(state, true);
            return true;
        }
    }

    cache.put(state, false);
    return false;
}
```

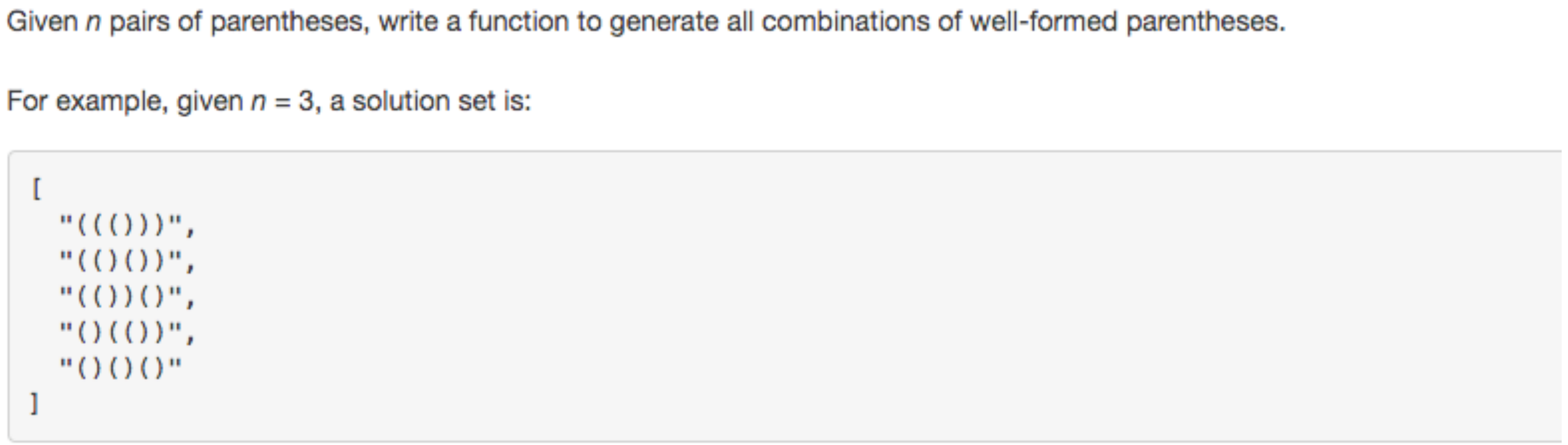
corner case

控制器

state 记录用过数字， cache 记录重复路径

why not DP ?

22. Generate Parentheses



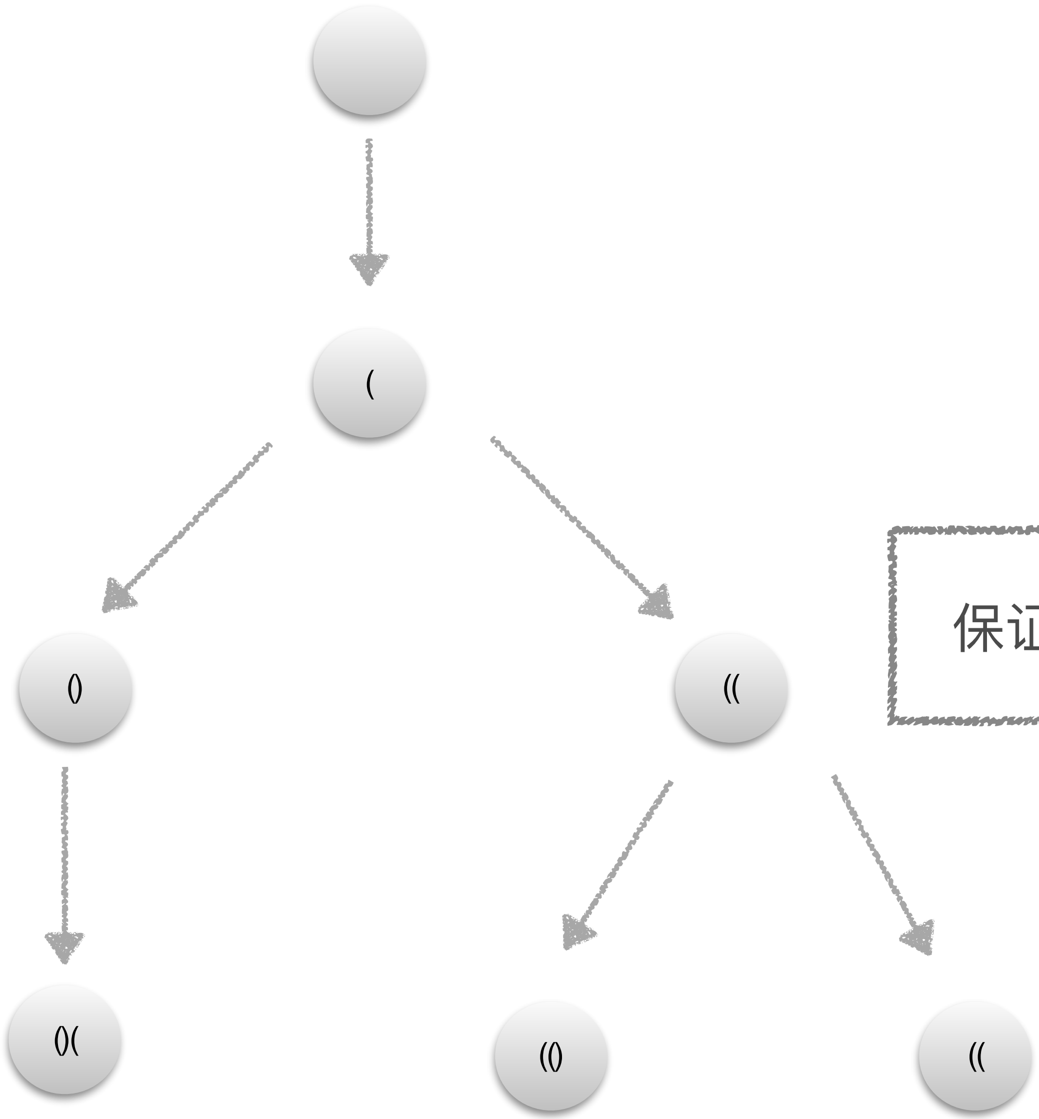
Given *n* pairs of parentheses, write a function to generate all combinations of well-formed parentheses.

For example, given *n* = 3, a solution set is:

```
[
  "((()))",
  "(()())",
  "(())()",
  "()(())",
  "()()()"
]
```

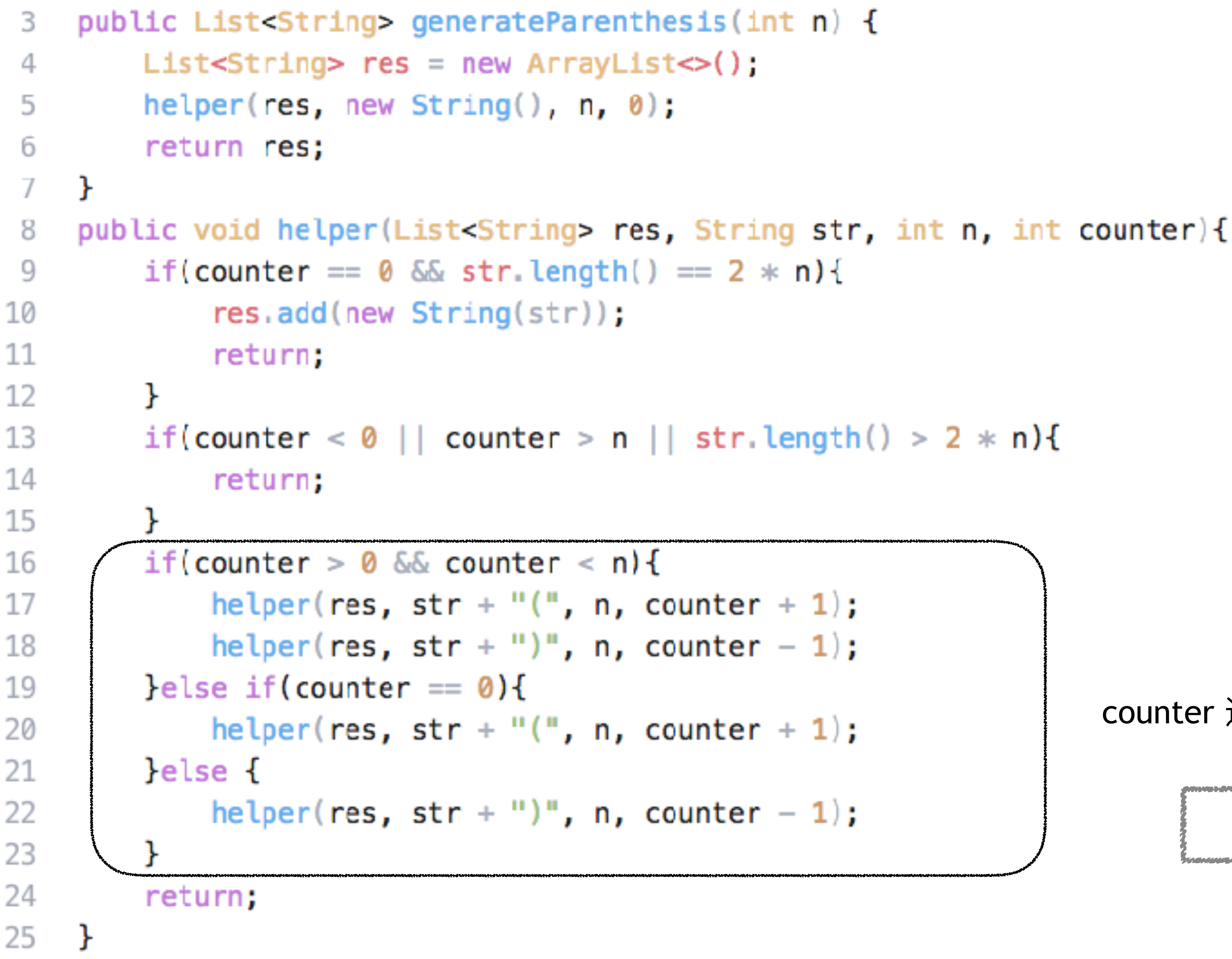
10 min

```
public List<String> generateParenthesis(int n) {

}
```



DFS 同时保证合法性

保证'('的数量大于等于')'的数量

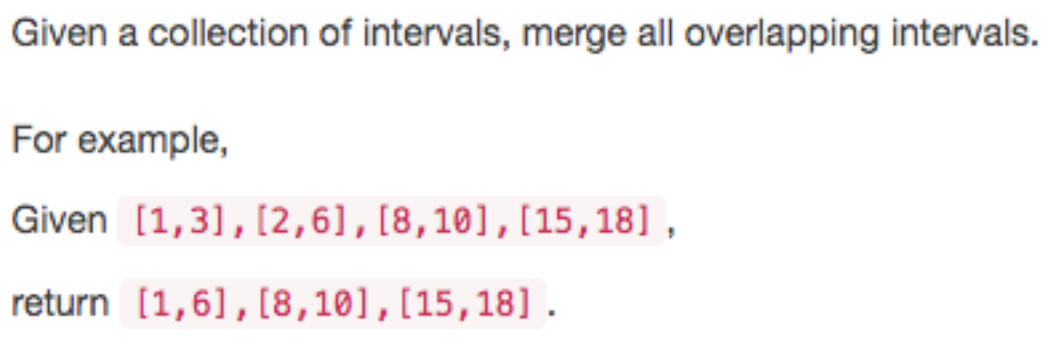


```java
public List<String> generateParenthesis(int n) {
    List<String> res = new ArrayList<>();
    helper(res, new String(), n, 0);
    return res;
}
public void helper(List<String> res, String str, int n, int counter){
    if(counter == 0 && str.length() == 2 * n){
        res.add(new String(str));
        return;
    }
    if(counter < 0 || counter > n || str.length() > 2 * n){
        return;
    }
    if(counter > 0 && counter < n){
        helper(res, str + "(", n, counter + 1);
        helper(res, str + ")", n, counter - 1);
    }else if(counter == 0){
        helper(res, str + "(", n, counter + 1);
    }else {
        helper(res, str + ")", n, counter - 1);
    }
    return;
}
```

counter 追踪 (和)

why not DP ?

56. Merge Intervals



Given a collection of intervals, merge all overlapping intervals.

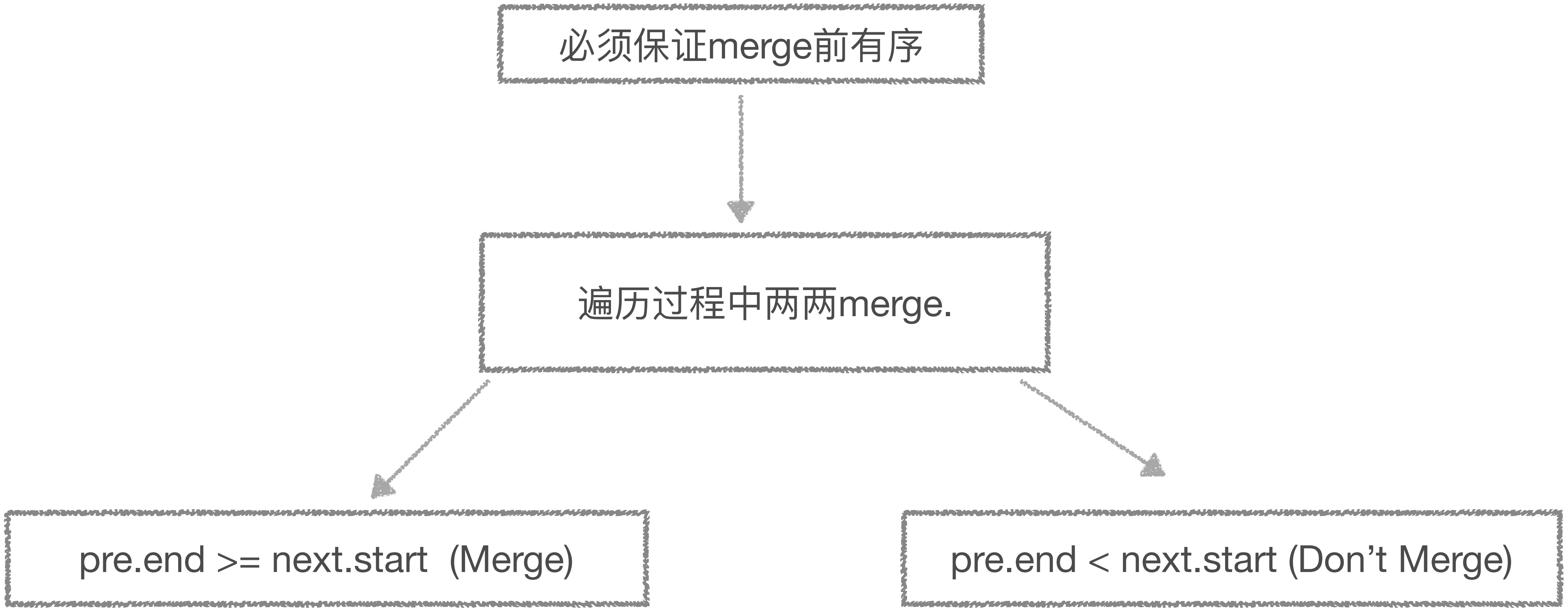
For example,

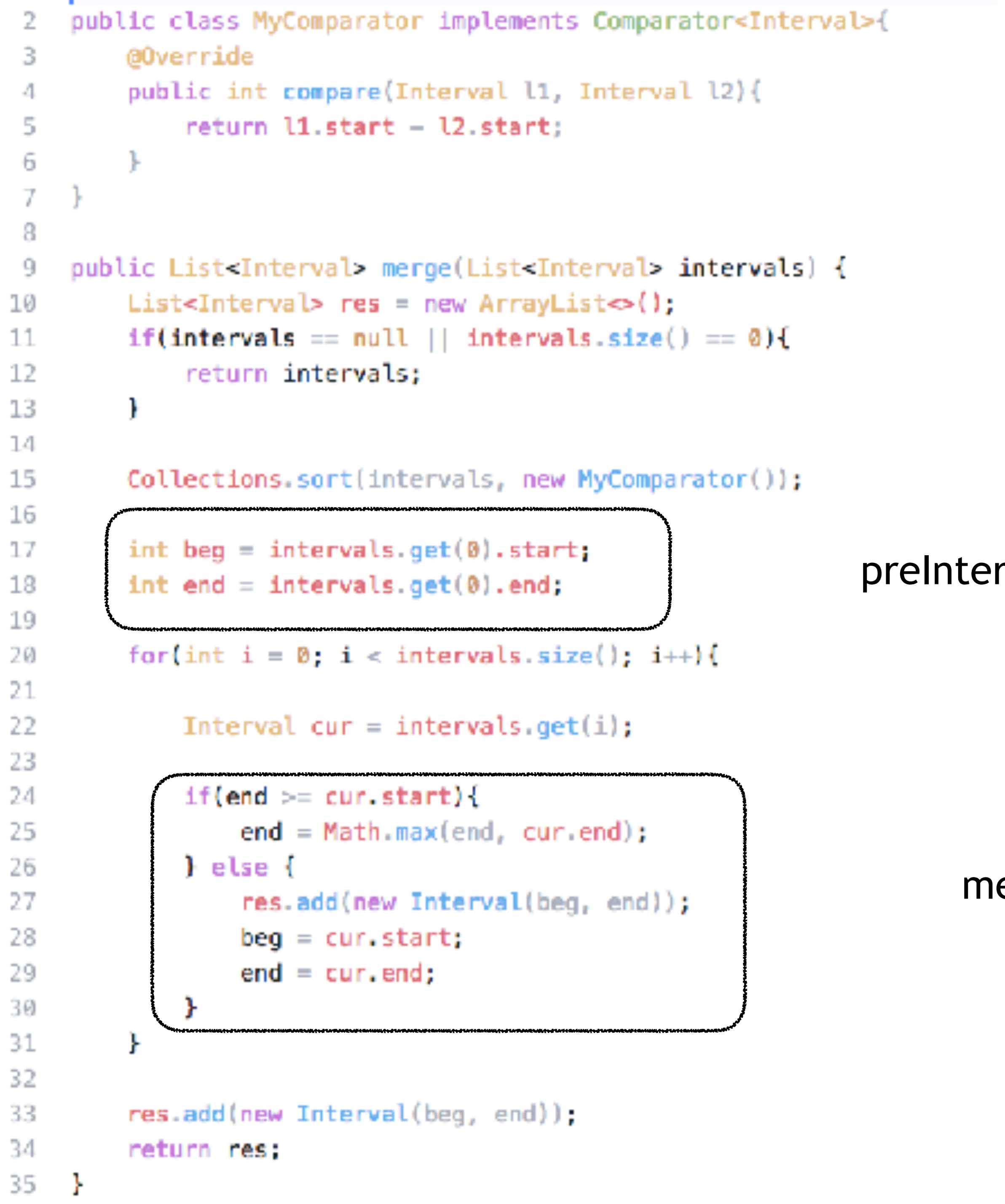
Given `[1,3],[2,6],[8,10],[15,18]`,

return `[1,6],[8,10],[15,18]`.

10 min

```
public List<Interval> merge(List<Interval> intervals) {

}
```


必须保证merge前有序
遍历过程中两两merge.
pre.end >= next.start (Merge)
pre.end < next.start (Don’t Merge)

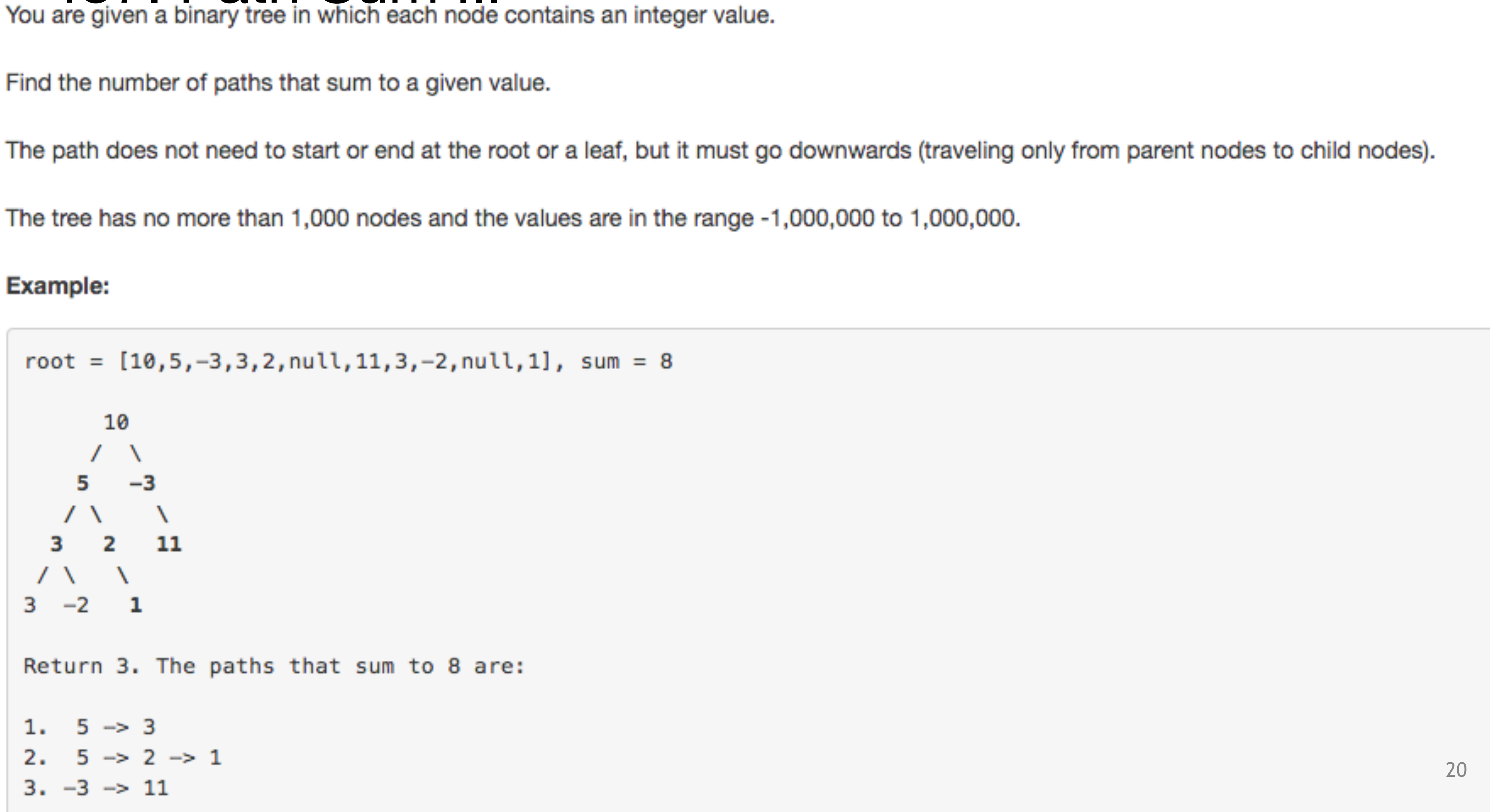


```java
public class MyComparator implements Comparator<Interval>{
    @Override
    public int compare(Interval l1, Interval l2){
        return l1.start - l2.start;
    }
}

public List<Interval> merge(List<Interval> intervals) {
    List<Interval> res = new ArrayList<>();
    if(intervals == null || intervals.size() == 0){
        return intervals;
    }

    Collections.sort(intervals, new MyComparator());

    int beg = intervals.get(0).start;
    int end = intervals.get(0).end;

    for(int i = 0; i < intervals.size(); i++){

        Interval cur = intervals.get(i);

        if(end >= cur.start){
            end = Math.max(end, cur.end);
        } else {
            res.add(new Interval(beg, end));
            beg = cur.start;
            end = cur.end;
        }
    }

    res.add(new Interval(beg, end));
    return res;
}
```

preInterval

merge OR continue

437. Path Sum III



You are given a binary tree in which each node contains an integer value.

Find the number of paths that sum to a given value.

The path does not need to start or end at the root or a leaf, but it must go downwards (traveling only from parent nodes to child nodes).

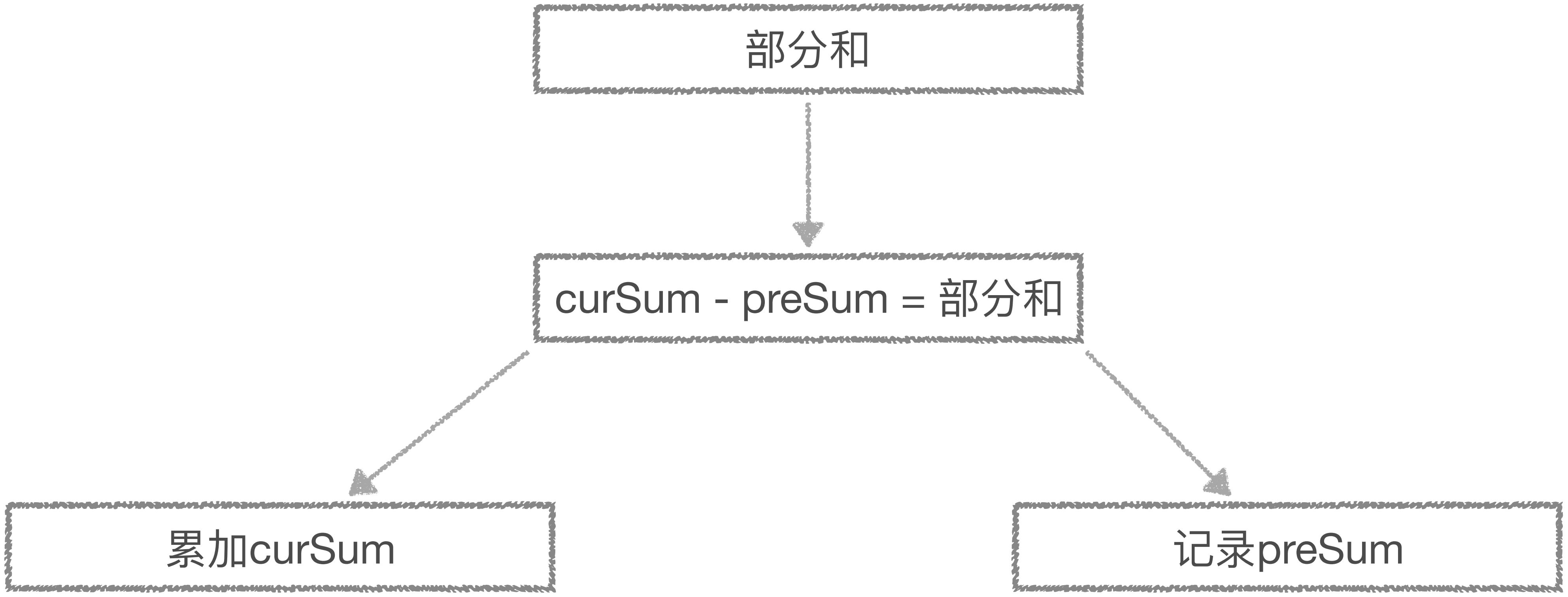
The tree has no more than 1,000 nodes and the values are in the range -1,000,000 to 1,000,000.

Example:

```
root = [10,5,-3,3,2,null,11,3,-2,null,1], sum = 8

      10
     /  \
    5   -3
   / \    \
  3   2   11
 / \   \
3  -2   1

Return 3. The paths that sum to 8 are:

1.  5 -> 3
2.  5 -> 2 -> 1
3. -3 -> 11
```

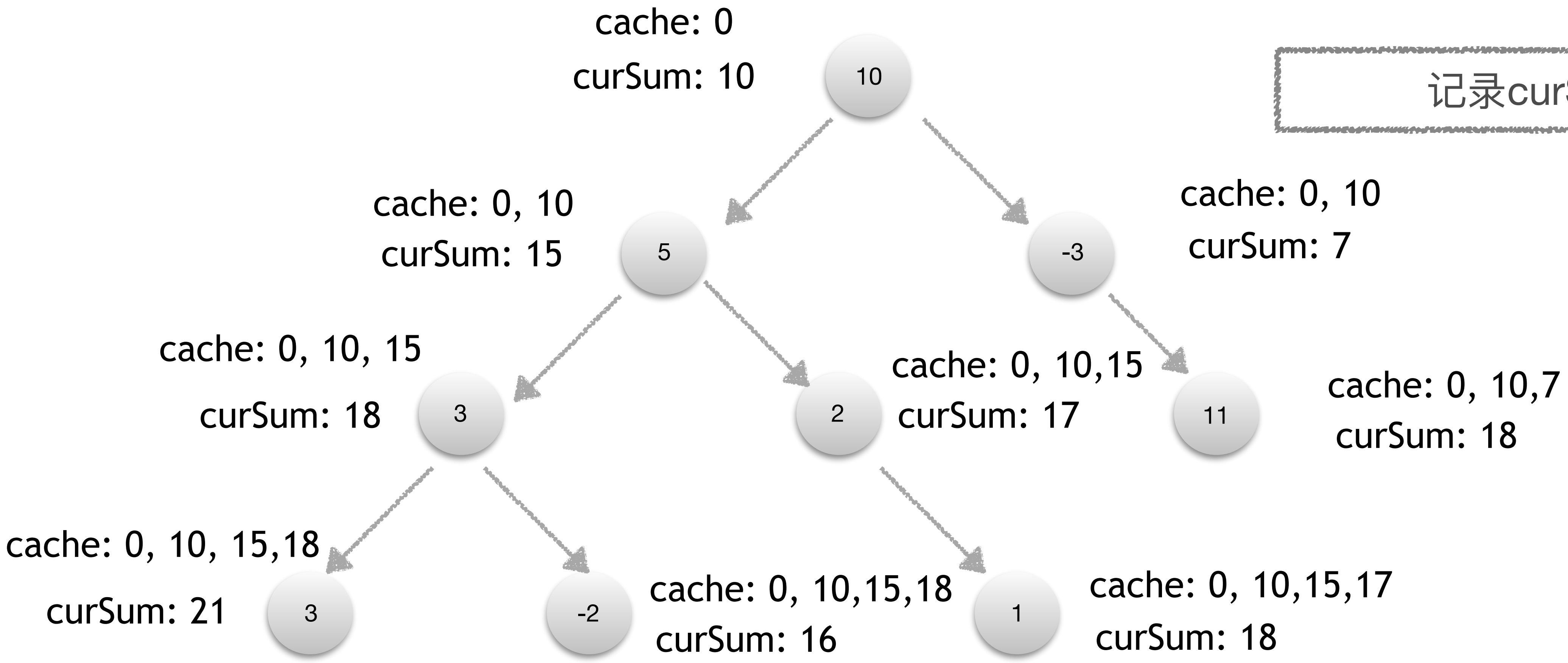
10 min

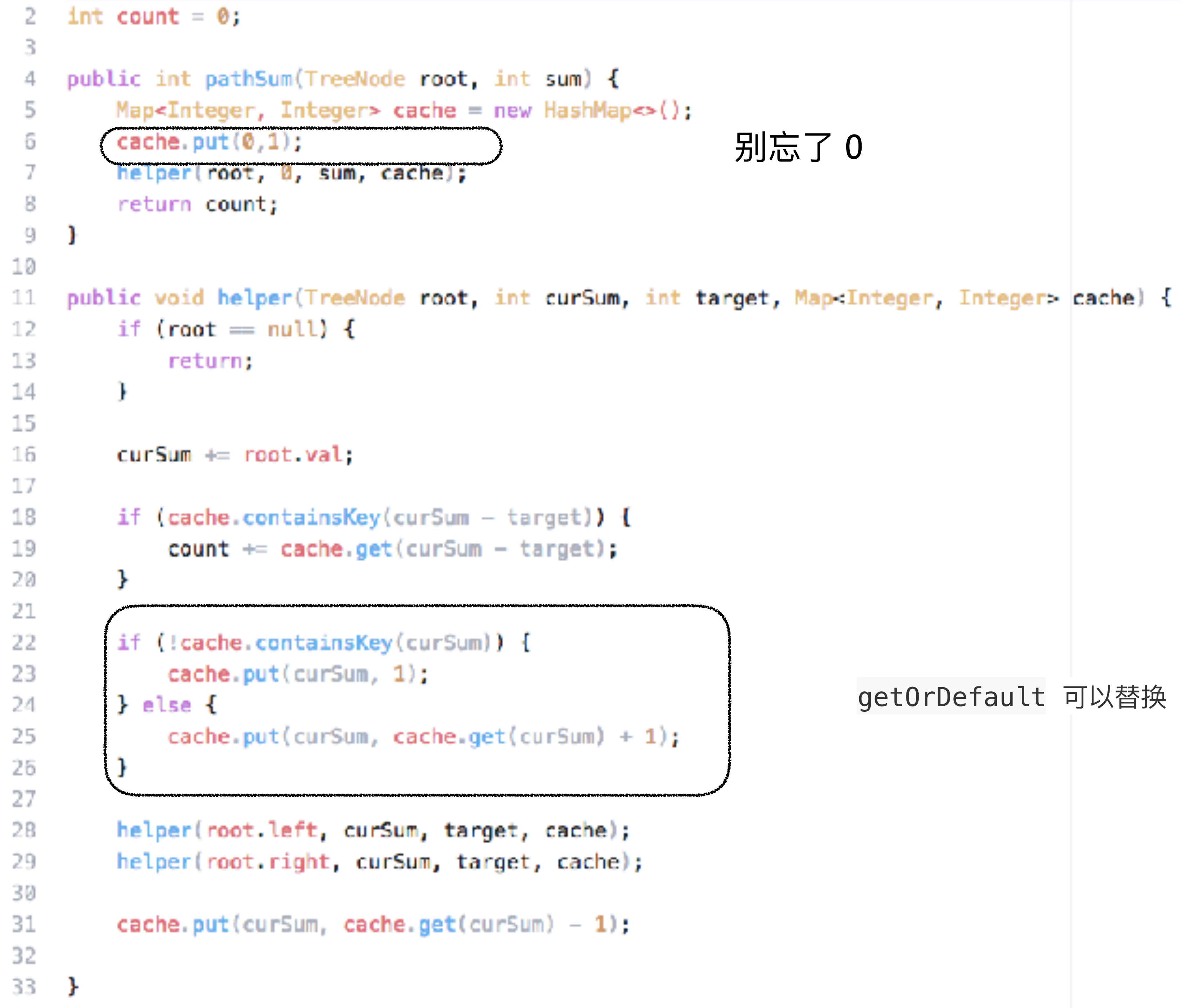
```
public int pathSum(TreeNode root, int sum) {

}
```


部分和
curSum - preSum = 部分和
累加curSum
记录preSum

SUM = 8

记录curSum





```java
int count = 0;

public int pathSum(TreeNode root, int sum) {
    Map<Integer, Integer> cache = new HashMap<>();
    cache.put(0,1);
    helper(root, 0, sum, cache);
    return count;
}

public void helper(TreeNode root, int curSum, int target, Map<Integer, Integer> cache) {
    if (root == null) {
        return;
    }

    curSum += root.val;

    if (cache.containsKey(curSum - target)) {
        count += cache.get(curSum - target);
    }

    if (!cache.containsKey(curSum)) {
        cache.put(curSum, 1);
    } else {
        cache.put(curSum, cache.get(curSum) + 1);
    }

    helper(root.left, curSum, target, cache);
    helper(root.right, curSum, target, cache);

    cache.put(curSum, cache.get(curSum) - 1);

}
```

别忘了 0

`getOrDefault` 可以替换

Homework

289 Game of Life
150 Evaluate Reverse Polish Notation
451 Sort Characters By Frequency
57 Insert Interval
378 Kth Smallest Element in a Sorted Matrix

Q & A

Thank you