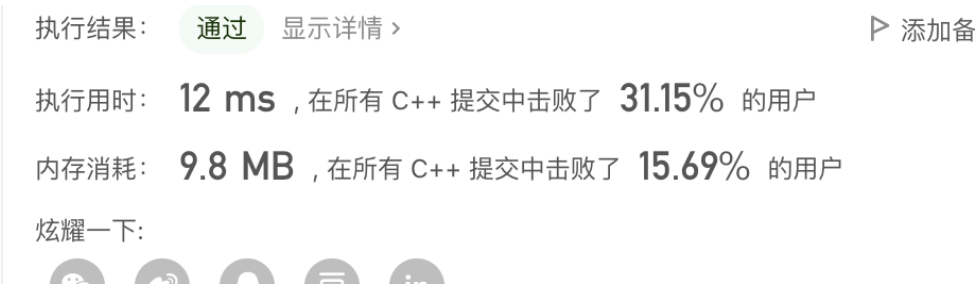
804 Unique Morse Code Words

```
    static String[] alpha = {".-","-...","-.-.","-..",".","..-.","-
-.","....","..",".---","-.-",".-..","--","-.","---",".--.","--.-",".-.","...","-
","..-","...-",".--","-..-","-.--","--.."};
    public int uniqueMorseRepresentations(String[] words) {
        Set<String> set = new HashSet<>();
        for(String str : words)
            set.add(getStr(str));
        return set.size();
    }

    private String getStr(String str){
        StringBuilder res = new StringBuilder();
        for(char ch : str.toCharArray()){
            res.append(alpha[ch - 'a']);
        }

        return res.toString();
    }
```

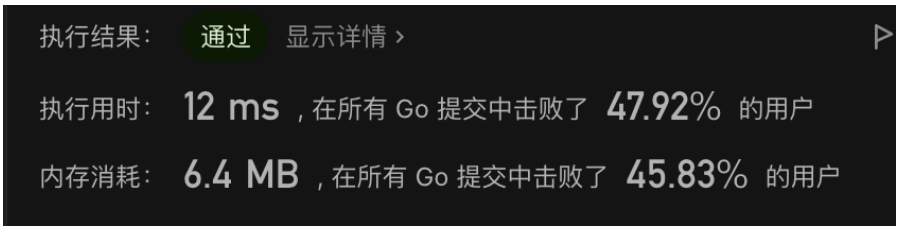
807 Max Increase to Keep City Skyline




```
class Solution {
public:
    int min(int i, int j){
        return i >= j ? j : i;
    }

    int maxIncreaseKeepingSkyline(vector<vector<int>>& grid) {
        int row = grid.size();
        int col = grid[0].size();

        vector<int> rowLine(row, 0);
```

```
        vector<int> colLine(col ,0);

        for(int i = 0; i < row; i++){
            for(int j = 0; j < col; j++){
                rowLine[i] = rowLine[i] > grid[i][j] ? rowLine[i] : grid[i][j];
                colLine[j] = colLine[j] > grid[i][j] ? colLine[j] : grid[i][j];
            }
        }

        int res = 0;
        for(int i = 0; i < row; i++){
            for(int j = 0; j < col; j++){
                int standard =  min(rowLine[i], colLine[j]);
                if(grid[i][j] < standard){
                    res += standard - grid[i][j];
                }
            }
        }

        return res;
    }
};
```

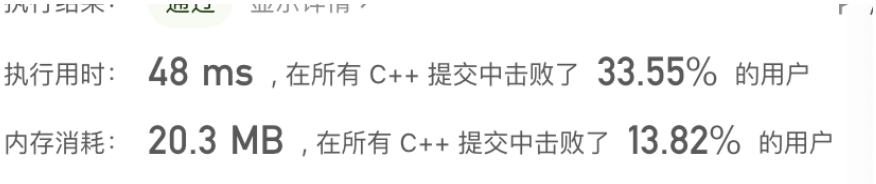
811 Subdomain Visit Count




```
func subdomainVisits(cpdomains []string) []string {
  m := make(map[string]int)

  for _, str := range cpdomains{
    splits := strings.Fields(str)
    strs :=  strings.Split(splits[1], ".")
    num, ok := strconv.Atoi(splits[0])
    if ok == nil{
      rawString := ""

      for i := len(strs) - 1; i >= 0; i--{
```

```
                newString := ""
                if i == len(strs) - 1{
                    newString = strs[i]
                }else{
            newString = strs[i] + "." + rawString
                }

        m[newString] += num
        rawString = newString
      }
    }
  }

  res := make([]string, 0)
  for key, val := range m{
    res = append(res, strconv.Itoa(val) + " " + key)
  }

  return res
}
```

828 Count Unique Characters of All Substrings of a Given String




```
class Solution {
public:
    int uniqueLetterString(string s) {
        int size = s.size();

        auto leftIndex  = vector<int>(size, -1);
        auto rightIndex = vector<int>(size, -1);
        auto alpha      = vector<int>(26, -1);

        for(int i = 0; i < size; i++){
            char ch = s[i];
            int pos = ch - 'A';

            if(alpha[pos] != -1)
                leftIndex[i] = alpha[pos];
```

```
16
17              alpha[pos] = i;
18          }
19
20          alpha = vector<int>(26, -1);
21          for(int i = size - 1; i >= 0; i--){
22              char ch = s[i];
23              int pos = ch  - 'A';
24
25              if(alpha[pos] != -1)
26                  rightIndex[i] = alpha[pos];
27
28              alpha[pos] = i;
29
30  //              cout << rightIndex[i] << " ";
31          }
32
33          int res = 0;
34          for(int i = 0; i < size; i++){
35              int left;
36              int right;
37              if(leftIndex[i] == -1)
38                  left = i + 1;
39              else
40                  left = i - leftIndex[i];
41
42              if(rightIndex[i] == -1)
43                  right = size - i;
44              else
45                  right = rightIndex[i] - i;
46
47              res += left * right;
48          }
49
50          return res;
51      }
52  };
```

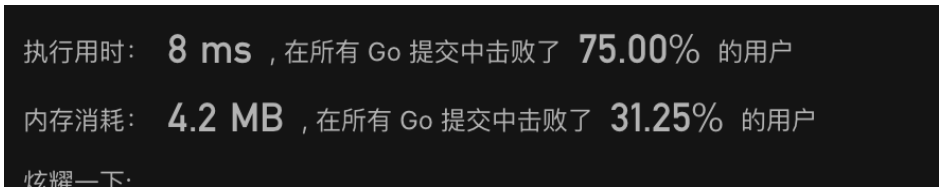
```
1  //思路实际上就是 通过扩散
2  /*
3    找到 每一个字符的不重复长度，然后进行求解
4
```

```
  比如 L E E T C O D E
            3
    找到 T 的不重复空间 [0, size - 1]
    那么 可以组成的字符串个数为

     4 * 5 = 20

*/
class Solution {
public:
    const int mod = 1e9 + 7;
    int uniqueLetterString(string s) {
        int size = s.size();
        int left = 0, right = size - 1;

        vector<int> alpha(26, -1);
        vector<int> leftIndex(size, -1);
        vector<int> rightIndex(size, -1);

        for(int i = 0; i < size; i++){
            leftIndex[i] = alpha[s[i] - 'A'];
            alpha[s[i] - 'A'] = i;
        }

        std::fill(alpha.begin(), alpha.end(), size);

        for(int i = size - 1; i >= 0; i--){
            rightIndex[i] = alpha[s[i] - 'A'];
            alpha[s[i] - 'A'] = i;
        }

        int res = 0;
        for(int i = 0 ; i < size; i++){
            long left = i - leftIndex[i];
            long right = rightIndex[i] - i;

            res = (res + (left * right) % mod) % mod;
        }

        return res;
    }
};
```

836 Overlapping Rec 很棒的思路

```cpp
class Solution {
public:
    bool isRectangleOverlap(vector<int>& rec1, vector<int>& rec2) {
        if(rec1[0] == rec1[2] || rec1[1] == rec1[3] ||
            rec2[0] == rec2[2] || rec2[1] == rec2[3])
            return false;

        bool horizontal = !(rec1[2] <= rec2[0] || rec2[2] <= rec1[0]);
        bool vertical   = !(rec1[3] <= rec2[1] || rec2[3] <= rec1[1]);

        return horizontal && vertical;
    }
};
```

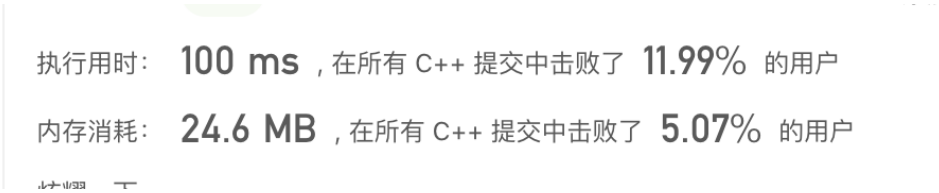
841 Key and Rooms




```go

func canVisitAllRooms(rooms [][]int) bool {
  queue := make([]int, 0)
  m := make(map[int]bool)

  queue = append(queue, 0)

  for ; len(queue) != 0; {
    size := len(queue)

    for i := 0; i < size; i++{
      cur := queue[0]
```

```
      queue = queue[1 : ]
      m[cur] = true
      canOpenRooms := rooms[cur]

      for j := 0;  j < len(canOpenRooms); j++{
        if !m[canOpenRooms[j]]{
          queue = append(queue, canOpenRooms[j])
        }
      }
    }
  }

  return len(m) == len(rooms)
}
```

847 Shortest Path Visiting All Nodes




```
/*
    根据 不同的二进制位,  表示是否访问过
*/
struct pair_hash
{
    template <class T1, class T2>
    std::size_t operator () (std::pair<T1, T2> const &pair) const
    {
        std::size_t h1 = std::hash<T1>()(pair.first);
        std::size_t h2 = std::hash<T2>()(pair.second);

        return h1 ^ h2;
    }
};

class Solution {
public:
    int shortestPathLength(vector<vector<int>>& graph) {
        int maxNum = graph.size() - 1;

        unordered_set<pair<int, int>, pair_hash> visited;
        queue<vector<int>> myQueue;
        const int finalState = (1 << (maxNum + 1)) - 1;
        for(int i = 0; i < maxNum + 1; i++){
```

```
            int state = 1 << i;
            myQueue.push({i, state});
        }

        int round = 0;
        while(!myQueue.empty()){
            int size = myQueue.size();

            for(int i = 0; i < size; i++){
                vector<int> curState = myQueue.front(); myQueue.pop();
                int curNode = curState[0];
                int key = curState[1];
                visited.insert({curNode, key});

                if(key == finalState)
                    return round;

                for(int nextPos : graph[curNode]){
                    int nextKey = key | (1 << nextPos);

                    pair<int, int> nextNode = {nextPos, nextKey};
                    if(visited.count(nextNode) == 0){
                        visited.insert(nextNode);
                        myQueue.push({nextPos, nextKey});
                    }
                }
            }

            round++;
        }

        return -1;
    }
};
```

857 Minimum Cost to Hire K Workers

857. Minimum Cost to Hire K Workers

难度 困难 120

There are n workers. You are given two integer arrays `quality` and `wage` where `quality[i]` is the quality of the i^{th} worker and `wage[i]` is the minimum wage expectation for the i^{th} worker.

We want to hire exactly `k` workers to form a paid group. To hire a group of `k` workers, we must pay them according to the following rules:

1. Every worker in the paid group should be paid in the ratio of their quality compared to other workers in the paid group.
2. Every worker in the paid group must be paid at least their minimum-wage expectation.

Given the integer `k`, return *the least amount of money needed to form a paid group satisfying the above conditions*. Answers within 10^{-5} of the actual answer will be accepted.

Example 1:

Input: quality = [10,20,5], wage = [70,50,30], k = 2
Output: 105.00000
Explanation: We pay 70 to 0^{th} worker and 35 to 2^{nd} worker.

```cpp
//ref: https://www.youtube.com/watch?v=ZHFRB58hlQw&t=548s
/*
  当 出现两个评价指标的时候， 可以考虑 sort + pq

  在本题中， 可以把 wage[i] / quality[i] 作为评价指标,
  因为这个代表 《单位工作量工人的工资》， 而 对于要找 k个人，肯定是希望平均的支配工资最低

  那么如果 必须是这个人，我希望总的工作量最小， 因此会维护一个 k - 1 大的 pq 大顶堆

  很不错的思路
*/
class Solution {
public:
    static bool cmp(pair<int, int>& p1, pair<int, int>& p2){
        return (p1.second * 1.0 / p1.first) < (p2.second * 1.0 / p2.first);
    }

```

```
    double mincostToHireWorkers(vector<int>& quality, vector<int>& wage, int k) {
        vector<pair<int, int>> persons;
        for(int i = 0; i < quality.size(); i++)
            persons.push_back({quality[i], wage[i]});

        sort(persons.begin(), persons.end(), cmp);

        priority_queue<int> pq;
        int qualitySum = 0;
        double res = DBL_MAX;

        for(int i = 0; i < quality.size(); i++){
            while(pq.size() > k - 1){
                qualitySum -= pq.top();
                pq.pop();
            }

            double ratio = persons[i].second * 1.0 / persons[i].first;
            qualitySum += persons[i].first;

            if(pq.size() == k - 1)
                res = min(res, qualitySum * ratio);
            pq.push(persons[i].first);
        }

        return res;
    }
};
```

863 All Nodes Distance K in Binary Tree

863. All Nodes Distance K in Binary Tree

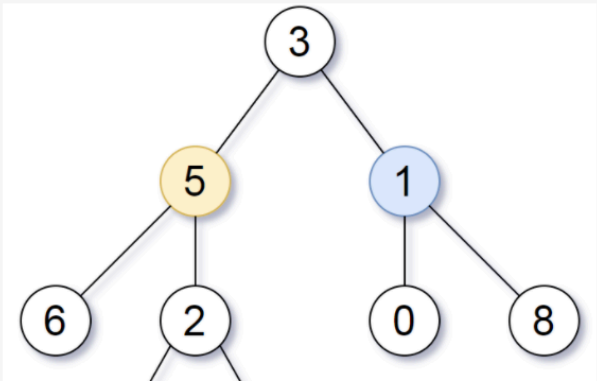
难度 中等 281

We are given a binary tree (with root node `root`), a `target` node, and an integer value `k`.

Return a list of the values of all nodes that have a distance `k` from the `target` node. The answer can be returned in any order.

Example 1:

```
Input: root = [3,5,1,6,2,0,8,null,null,7,4],
target = 5, k = 2

Output: [7,4,1]

Explanation:
The nodes that are a distance 2 from the target
node (with value 5)
have values 7, 4, and 1.
```




```java
public List<Integer> distanceK(TreeNode root, TreeNode target, int k) {
    Map<TreeNode, List<TreeNode>> map = new HashMap<>();
    List<Integer> res = new ArrayList<>();

    if(k == 0){
        if(root != null){
            res.add(target.val);
        }

        return res;
    }
    preorder(map, root);
    Deque<TreeNode> queue = new ArrayDeque<>();
    Set<TreeNode> visited = new HashSet<>();
    queue.add(target);
```

```java

        while(!queue.isEmpty()){
            int size = queue.size();
            k--;
            for(int i = 0; i < size; i++){
                TreeNode node = queue.removeFirst();
                visited.add(node);
                List<TreeNode> treeNodes = map.get(node);


                for(TreeNode node2 : treeNodes){
                    if(!visited.contains(node2))
                        queue.addLast(node2);
                }
            }

            if(k == 0){
                for(TreeNode node : queue){
                    res.add(node.val);
                }
            }
        }

        return res;
    }

    private void preorder(Map<TreeNode, List<TreeNode>> map, TreeNode root){
        if(root == null){
            return;
        }

        map.putIfAbsent(root, new ArrayList<>());
        if(root.left != null){
            map.get(root).add(root.left);
            map.putIfAbsent(root.left, new ArrayList<>());
            map.get(root.left).add(root);
        }

        if(root.right != null){
            map.get(root).add(root.right);
            map.putIfAbsent(root.right, new ArrayList<>());
            map.get(root.right).add(root);
        }

        preorder(map, root.left);
        preorder(map, root.right);
    }
```

864 Shortest Path to Get All Keys

864. Shortest Path to Get All Keys

难度 困难 86

You are given an `m x n` grid `grid` where:

- `'.'` is an empty cell.
- `'#'` is a wall.
- `'@'` is the starting point.
- Lowercase letters represent keys.
- Uppercase letters represent locks.

You start at the starting point and one move consists of walking one space in one of the four cardinal directions. You cannot walk outside the grid, or walk into a wall.

If you walk over a key, you can pick it up and you cannot walk over a lock unless you have its corresponding key.

For some `1 <= k <= 6`, there is exactly one lowercase and one uppercase letter of the first `k` letters of the English alphabet in the grid. This means that there is exactly one key for each lock, and one lock for each key; and also that the letters used to represent the keys and locks were chosen in the same order as the English alphabet.

Return *the lowest number of moves to acquire all keys*. If it is impossible, return `-1`.

Example 1:

```
//source : https://happygirlzt.com/code/864.html
/*
  这个题目可以理解为一个加强版本的 BFS

  因为本身 如果只是搜集 钥匙， 没有锁， 那么 BFS 完全可以完成这个任务，

  但是因为有锁的限制， 所以 必须要对 锁的状态加以记录
  因此时间复杂度为 O(N * M * 2 ^ k)

  使用 vector<vector<vector<int>>> visited 进行状态的记录
*/
class Solution {
public:
```

```cpp
    int shortestPathAllKeys(vector<string>& grid) {
        int row = grid.size();
        int col = grid[0].size();
        vector<vector<int>> dir{{-1, 0}, {0, -1}, {1, 0}, {0, 1}};

        int keys = 0;
        queue<vector<int>> myQueue;
        for(int i = 0; i < row; i++){
            for(int j = 0; j < col; j++){
                char ch = grid[i][j];
                if(ch == '@'){
                            //    x, y, curMove, have keys
                    myQueue.push({i, j, 0, 0});
                }else if(ch >= 'a' && ch <= 'z'){
                    keys++;
                }
            }
        }

        keys = (1 << keys) - 1;
        auto visited =
            vector<vector<vector<int>>>(row, vector<vector<int>>(col, vector<int>(keys + 1, 0)));
        auto isInRange = [&](int i, int j){return i >= 0 && j >= 0 && i < row && j < col;};

        while(!myQueue.empty()){
            int size = myQueue.size();
            for(int i = 0; i < size; i++){
                vector<int> curState = myQueue.front(); myQueue.pop();

                int x     = curState[0];
                int y     = curState[1];
                int moves = curState[2];
                int key   = curState[3];

                if(key == keys)
                    return moves;

                for(int k = 0; k < 4; k++){
                    int newX = x + dir[k][0];
                    int newY = y + dir[k][1];

                    if(isInRange(newX, newY) && grid[newX][newY] != '#'){
                        char ch = grid[newX][newY];

                        if(ch >= 'A' && ch <= 'Z'){
                            if( ((key >> (ch - 'A')) & 1) && visited[newX][newY][key] == 0){
```

```
                                myQueue.push({newX, newY, moves + 1, key});
                                visited[newX][newY][key] = 1;
                            }
                        }else if(ch >= 'a' && ch <= 'z'){
                            int newKey = (key | (1 << (ch - 'a')));
                            if(visited[newX][newY][newKey] == 0){
                                myQueue.push({newX, newY, moves + 1, newKey});
                                visited[newX][newY][newKey] = 1;
                            }
                        }else{
                            if(visited[newX][newY][key] == 0){
                               myQueue.push({newX, newY, moves + 1, key});
                                visited[newX][newY][key] = 1;
                            }
                        }
                    }
                }
            }
        }

        return -1;
    }
};
```

887 Super Egg Drop

887. Super Egg Drop

难度 困难 655

You are given `k` identical eggs and you have access to a building with `n` floors labeled from `1` to `n`.

You know that there exists a floor `f` where `0 <= f <= n` such that any egg dropped at a floor **higher** than `f` will **break**, and any egg dropped **at or below** floor `f` will **not break**.

Each move, you may take an unbroken egg and drop it from any floor `x` (where `1 <= x <= n`). If the egg breaks, you can no longer use it. However, if the egg does not break, you may **reuse** it in future moves.

Return *the **minimum number of moves** that you need to determine **with certainty** what the value of* `f` is.

Example 1:

```
Input: k = 1, n = 2
Output: 2
Explanation:
Drop the egg from floor 1. If it breaks, we know
that f = 0.
Otherwise, drop the egg from floor 2. If it
breaks, we know that f = 1.
If it does not break, then we know f = 2.
Hence, we need at minimum 2 moves to determine
with certainty what the value of f is.
```

Example 2:

```cpp

//Author: guoguo
class Solution {
public:
    //                  eggs   floor
    int superEggDrop(int k, int n) {
        vector<vector<int>> dp(k + 1, vector<int>(n + 1, 0));
        for(int i = 1; i <= k; i++){
            dp[i][1] = 1;
        }

        for(int j = 1; j <= n; j++){
            dp[1][j] = j;
        }
```

```
15
16          for(int i = 2; i <= k; i++){
17              for(int j = 2; j <= n; j++){
18                  if(i > j){
19                      dp[i][j] = dp[i - 1][j];
20                  }else{
21                      //dp[i][j] = 1 + std::max(dp[i - 1][m - 1], dp[i][j - m]);
22
23                      int left = 1, right = j;
24                      int ans = left;
25                      while(left <= right){
26                          int mid = (left + right) / 2;
27
28                          int f = dp[i - 1][mid - 1];
29                          int g = dp[i][j - mid];
30
31                          if(f >= g){
32                              ans = mid;
33                              right = mid - 1;
34
35                              if(f == g)
36                                  break;
37                          }else{
38                              left = mid + 1;
39                          }
40                      }
41
42                      dp[i][j] = 1 + std::max(dp[i - 1][ans - 1], dp[i][j - ans]);
43                  }
44              }
45          }
46
47
48          return dp[k][n];
49      }
50  };
```

```
1   //超出时间限制，时间复杂度是 O(n2 * k)
2   class Solution {
3   public:
4       //          eggs   floor
5       int superEggDrop(int k, int n) {
6           vector<vector<int>> dp(k + 1, vector<int>(n + 1, 0));
7           for(int i = 1; i <= k; i++){
8               dp[i][1] = 1;
```

```
        }

        for(int j = 1; j <= n; j++){
            dp[1][j] = j;
        }

        for(int i = 2; i <= k; i++){
            for(int j = 2; j <= n; j++){
                if(i > j){
                    dp[i][j] = dp[i - 1][j];
                }else{
                    for(int m = 1; m <= j; m++){
                        if(m == 1)
                            dp[i][j] = 1 + std::max(dp[i - 1][m - 1], dp[i][j - m]);
                        else
                            dp[i][j] = std::min(dp[i][j], 1 + std::max(dp[i - 1][m - 1], dp[i][j - m]));
                    }
                }
            }
        }

        return dp[k][n];
    }
};

```

```
//po 一个 花花的超时代码
public:
  int superEggDrop(int K, int N){
    vector<vector<int>> m(K + 1, vector<int>(N + 1, INT_MAX));
    function<int(int, int)> dp = [&](int k, int n){
      if(k == 0)  return 0;
      if(k == 1)  return n;
      if(n <= 1)  return n;

      int& ans = m[k][n];
      if(ans != INT_MAX)  return ans;
      for(int i = 1; i <= n; i++)
          ans = min(ans, 1 + max(dp(k - 1, i -1), dp(k, n - i)));

      return ans;
    }
    return dp(K, N);
  }
```

889 Construct Binary Tree From Preorder and Postorder

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
 * };
 */
class Solution {
public:

//1 2 4 5 3 6 7
//4 5 2 6 7 3 1
//       i

    TreeNode* constructFromPrePost(vector<int>& preorder, vector<int>& postorder) {
        int size = preorder.size();
        if(size == 0)
            return nullptr;
        else if(size == 1)
            return new TreeNode(preorder[0]);

        TreeNode* root = new TreeNode(preorder[0]);

        int i = 0;
        for(; i < size; i++){
            if(preorder[1] == postorder[i]){
                i++;
                break;
            }
        }

//        cout << i << "----" << endl;
```

```
        vector<int> preorderLeft   = vector<int>(preorder.begin() + 1, preorder.begin() + i + 1);
//        cout << preorderLeft.size() << endl;
        vector<int> preorderRight  = vector<int>(preorder.begin() + i + 1, preorder.end());
//        cout << preorderRight.size() << endl;
        vector<int> postorderLeft  = vector<int>(postorder.begin(), postorder.begin() + i);
//        cout << postorderLeft.size() << endl;
        vector<int> postorderRight = vector<int>(postorder.begin() + i, postorder.end() - 1);
//        cout << postorderRight.size() << endl;

        root->left = constructFromPrePost(preorderLeft, postorderLeft);
        root->right = constructFromPrePost(preorderRight, postorderRight);

        return root;
    }
};
```