

121. 买卖股票的最佳时机

121. 买卖股票的最佳时机

难度 **简单** 912 收藏 分享 切换为英文 关注 反馈

给定一个数组，它的第 i 个元素是一支给定股票第 i 天的价格。

如果你最多只允许完成一笔交易（即买入和卖出一支股票一次），设计一个算法来计算你能获取的最大利润。

注意：你不能在买入股票前卖出股票。

示例 1:

输入: [7,1,5,3,6,4]

输出: 5

解释: 在第 2 天（股票价格 = 1）的时候买入，在第 5 天（股票价格 = 6）的时候卖出，最大利润 = 6-1 = 5。

注意利润不能是 7-1 = 6，因为卖出价格需要大于买入价格；同时，你不能在买入前卖出股票。

示例 2:

输入: [7,6,4,3,1]

输出: 0

解释: 在这种情况下，没有交易完成，所以最大利润为 0。

```
class Solution {
public:
    int maxProfit(vector<int>& prices) {
        if(prices.size() < 2) return 0;
        int profit = 0, low = prices[0];
        for(int i = 0; i < prices.size(); i++)
        {
            profit = max(profit, prices[i] - low);
            low = min(low, prices[i]);
        }
        return profit;
    }
};
```

122. 买卖股票的最佳时机 II

122. 买卖股票的最佳时机 II

难度 简单

669

收藏

分享

切换为英文

关注

反馈

给定一个数组，它的第 i 个元素是一支给定股票第 i 天的价格。

设计一个算法来计算你所能获取的最大利润。你可以尽可能地完成更多的交易（多次买卖一支股票）。

注意：你不能同时参与多笔交易（你必须在再次购买前出售掉之前的股票）。

示例 1:

输入: [7,1,5,3,6,4]

输出: 7

解释: 在第 2 天（股票价格 = 1）的时候买入，在第 3 天（股票价格 = 5）的时候卖出，这笔交易所能获得利润 = $5 - 1 = 4$ 。

随后，在第 4 天（股票价格 = 3）的时候买入，在第 5 天（股票价格 = 6）的时候卖出，这笔交易所能获得利润 = $6 - 3 = 3$ 。

示例 2:

输入: [1,2,3,4,5]

输出: 4

解释: 在第 1 天（股票价格 = 1）的时候买入，在第 5 天（股票价格 = 5）的时候卖出，这笔交易所能获得利润 = $5 - 1 = 4$ 。

注意你不能在第 1 天和第 2 天接连购买股票，之后再将它们卖出。

因为这样属于同时参与了多笔交易，你必须在再次购买前出售掉之前的股票。

示例 3:

输入: [7,6,4,3,1]

输出: 0

解释: 在这种情况下，没有交易完成，所以最大利润为 0。

```
class Solution {
public:
    int maxProfit(vector<int>& prices) {
        int sum = 0;
        for(int i = 1; i < prices.size(); i++)
            if(prices[i] > prices[i - 1])
                sum += prices[i] - prices[i - 1];
        return sum;
    }
};
/*
class Solution {
public:
    int maxProfit(vector<int>& prices) {
        int n = prices.size();
        int f, g;
        f = 0;
        g = -1000000000;
        for (int i = 0; i < n; i++) {
```

```

        int last_f = f, last_g = g;
        f = max(last_f, last_g + prices[i]);
        g = max(last_g, last_f - prices[i]);
    }
    return f;
}
};
*/

```

123. 买卖股票的最佳时机 III

123. 买卖股票的最佳时机 III

难度 **困难**  349  收藏  分享  切换为英文  关注  反馈

给定一个数组，它的第 i 个元素是一支给定的股票在第 i 天的价格。

设计一个算法来计算你能获取的最大利润。你最多可以完成 *两笔* 交易。

注意: 你不能同时参与多笔交易（你必须在再次购买前出售掉之前的股票）。

示例 1:

输入: [3,3,5,0,0,3,1,4]

输出: 6

解释: 在第 4 天（股票价格 = 0）的时候买入，在第 6 天（股票价格 = 3）的时候卖出，这笔交易所能获得利润 = $3 - 0 = 3$ 。

随后，在第 7 天（股票价格 = 1）的时候买入，在第 8 天（股票价格 = 4）的时候卖出，这笔交易所能获得利润 = $4 - 1 = 3$ 。

示例 2:

输入: [1,2,3,4,5]

输出: 4

解释: 在第 1 天（股票价格 = 1）的时候买入，在第 5 天（股票价格 = 5）的时候卖出，这笔交易所能获得利润 = $5 - 1 = 4$ 。

注意你不能在第 1 天和第 2 天接连购买股票，之后再将它们卖出。

因为这样属于同时参与了多笔交易，你必须在再次购买前出售掉之前的股票。

示例 3:

输入: [7,6,4,3,1]

输出: 0

解释: 在这个情况下，没有交易完成，所以最大利润为 0。

```

class Solution {
public:
    int maxProfit(vector<int>& prices) {
        int n = prices.size();
        if (!n) return 0;
        vector<int> f(n, 0);
        int minv = INT_MAX;
        for (int i = 0; i < n; i++) {
            if (i) f[i] = f[i - 1];

```

```

        if (prices[i] > minv)
            f[i] = max(f[i], prices[i] - minv);
        minv = min(minv, prices[i]);
    }
    int res = f[n - 1];
    int maxv = INT_MIN;
    for (int i = n - 1; i > 0; i -- )
    {
        if (prices[i] < maxv)
            res = max(res, maxv - prices[i] + f[i - 1]);
        maxv = max(maxv, prices[i]);
    }
    return res;
}
};

```

124. 二叉树中的最大路径和

124. 二叉树中的最大路径和

难度 **困难**  386  收藏  分享  切换为英文  关注  反馈

给定一个非空二叉树，返回其最大路径和。

本题中，路径被定义为一条从树中任意节点出发，达到任意节点的序列。该路径至少包含一个节点，且不一定经过根节点。

示例 1:

输入: [1,2,3]

```

    1
   / \
  2   3

```

输出: 6

示例 2:

输入: [-10,9,20,null,null,15,7]

```

    -10
   /  \
  9    20
 /  \
15   7

```

输出: 42

```

/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}

```

```

    * };
    */
class Solution {
public:
    int ans = INT_MIN;
    int maxPathSum(TreeNode* root) {
        dfs(root);
        return ans;
    }

    int dfs(TreeNode *root)
    {
        if(!root) return 0;

        auto left = dfs(root->left);
        auto right = dfs(root->right);
        ans = max(ans, left + root->val + right);
        return max(0, root->val + max(left, right));
    }
};

```

125. 验证回文串

125. 验证回文串

难度 **简单** 178 收藏 分享 切换为英文 关注 反馈

给定一个字符串，验证它是否是回文串，只考虑字母和数字字符，可以忽略字母的大小写。

说明：本题中，我们将空字符串定义为有效的回文串。

示例 1:

输入: "A man, a plan, a canal: Panama"

输出: true

示例 2:

输入: "race a car"

输出: false

```

class Solution {
public:
    bool check(char c)
    {
        return c >= '0' && c <= '9' || c >= 'a' && c <= 'z' || c >= 'A' && c <= 'Z';
    }

    bool isPalindrome(string s) {
        int i = 0, j = (int)s.size() - 1;
        while(i < j)
        {
            while(i < j && !check(s[i])) i++;
            while(i < j && !check(s[j])) j--;
        }
    }
};

```

```
        if(s[i] != s[j] && s[i] != (s[j] ^ 32)) return false;
        i ++, j --;
    }
    return true;
}
};
```

126. 单词接龙 II

126. 单词接龙 II

难度 困难 147 收藏 分享 切换为英文 关注 反馈

给定两个单词 (*beginWord* 和 *endWord*) 和一个字典 *wordList*，找出所有从 *beginWord* 到 *endWord* 的最短转换序列。转换需遵循如下规则：

1. 每次转换只能改变一个字母。
2. 转换过程中的中间单词必须是字典中的单词。

说明:

- 如果不存在这样的转换序列，返回一个空列表。
- 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。

示例 1:

输入：
beginWord = "hit",
endWord = "cog",
wordList = ["hot","dot","dog","lot","log","cog"]

输出：
[
 ["hit","hot","dot","dog","cog"],
 ["hit","hot","lot","log","cog"]
]

示例 2:

输入：
beginWord = "hit"
endWord = "cog"
wordList = ["hot","dot","dog","lot","log"]
...

127. 单词接龙

127. 单词接龙

难度 中等

👍 276

📖 收藏

🔗 分享

🌐 切换为英文

🔔 关注

📝 反馈

给定两个单词 (*beginWord* 和 *endWord*) 和一个字典，找到从 *beginWord* 到 *endWord* 的最短转换序列的长度。转换需遵循如下规则：

1. 每次转换只能改变一个字母。
2. 转换过程中的中间单词必须是字典中的单词。

说明:

- 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。

示例 1:

输入:

```
beginWord = "hit",
endWord = "cog",
wordList = ["hot","dot","dog","lot","log","cog"]
```

输出: 5

解释: 一个最短转换序列是 "hit" -> "hot" -> "dot" -> "dog" -> "cog", 返回它的长度 5。

示例 2:

输入:

```
beginWord = "hit"
endWord = "cog"
wordList = ["hot","dot","dog","lot","log"]
```

输出: 0

解释: *endWord* "cog" 不在字典中，所以无法进行转换。

```
class Solution {
public:
    bool check(string a, string b)
    {
        int res = 0;
        for (int i = 0; i < a.size(); i++) res += a[i] != b[i];
        return res == 1;
    }

    int ladderLength(string beginWord, string endWord, vector<string>& wordList)
    {
        unordered_map<string, int> dist;
        queue<string> que;
        que.push(beginWord), dist[beginWord] = 1;
        while (!que.empty())
        {
            string t = que.front();
```

```

        que.pop();

        if (t == endword) return dist[t];
        for (auto &word : wordList)
            if (check(t, word) && !dist[word])
            {
                dist[word] = dist[t] + 1;
                que.push(word);
            }
    }
    return 0;
}
};

```

128. 最长连续序列

128. 最长连续序列

难度 **困难** 293 收藏 分享 切换为英文 关注 反馈

给定一个未排序的整数数组，找出最长连续序列的长度。

要求算法的时间复杂度为 $O(n)$ 。

示例:

输入: [100, 4, 200, 1, 3, 2]

输出: 4

解释: 最长连续序列是 [1, 2, 3, 4]。它的长度为 4。






```

class Solution {
public:
    int longestConsecutive(vector<int>& nums) {
        int res = 0;
        unordered_map<int, int> tr_left, tr_right;
        for(auto & x : nums)
        {
            int left = tr_right[x - 1];
            int right = tr_left[x + 1];
            tr_left[x - left] = max(tr_left[x - left], left + 1 + right);
            tr_right[x + right] = max(tr_right[x + right], left + 1 + right);
            res = max(res, left + 1 + right);
        }
        return res;
    }
};

```

129. 求根到叶子节点数字之和

129. 求根到叶子节点数字之和

难度 中等  132  收藏  分享  切换为英文  关注  反馈

给定一个二叉树，它的每个结点都存放一个 0-9 的数字，每条从根到叶子节点的路径都代表一个数字。

例如，从根到叶子节点路径 1->2->3 代表数字 123。

计算从根到叶子节点生成的所有数字之和。

说明: 叶子节点是指没有子节点的节点。

示例 1:

输入: [1,2,3]

```
  1
 / \
```

```
2   3
```

输出: 25

解释:

从根到叶子节点路径 1->2 代表数字 12。

从根到叶子节点路径 1->3 代表数字 13。

因此，数字总和 = 12 + 13 = 25。

示例 2:

输入: [4,9,0,5,1]

```
  4
 / \
```

```
9   0
```

```
 / \
```

```
5   1
```

输出: 1026

解释:

从根到叶子节点路径 4->9->5 代表数字 495。

从根到叶子节点路径 4->9->1 代表数字 491。

从根到叶子节点路径 4->0 代表数字 40。

因此，数字总和 = 495 + 491 + 40 = 1026。

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    int ans = 0;
    int sumNumbers(TreeNode* root) {
        dfs(root,0);
        return ans;
    }

    void dfs(TreeNode* u,int s)
```

```

{
    if(!u) return;
    s = s * 10 + u->val;
    if(u->left) dfs(u->left,s);
    if(u->right) dfs(u->right,s);
    if(!u->left && !u->right) ans += s;
}
};

```

130. 被围绕的区域

130. 被围绕的区域

难度 中等

199

收藏

分享

切换为英文

关注

反馈

给定一个二维的矩阵，包含 'X' 和 'O'（字母 O）。

找到所有被 'X' 围绕的区域，并将这些区域里所有的 'O' 用 'X' 填充。

示例:

```

X X X X
X O O X
X X O X
X O X X

```

运行你的函数后，矩阵变为：

```

X X X X
X X X X
X X X X
X O X X

```

解释:

被围绕的区间不会存在于边界上，换句话说，任何边界上的 'O' 都不会被填充为 'X'。任何不在边界上，或不与边界上的 'O' 相连的 'O' 最终都会被填充为 'X'。如果两个元素在水平或垂直方向相邻，则称它们是“相连”的。

```

class Solution {
public:
    vector<vector<bool>> st;
    int n, m;

    void solve(vector<vector<char>>& board) {
        if (board.empty()) return;
        n = board.size(), m = board[0].size();
        for (int i = 0; i < n; i++)
        {
            vector<bool> temp;
            for (int j = 0; j < m; j++)
                temp.push_back(false);
            st.push_back(temp);
        }

        for (int i = 0; i < n; i++)
        {

```

```

        if (board[i][0] == 'o') dfs(i, 0, board);
        if (board[i][m - 1] == 'o') dfs(i, m - 1, board);
    }
    for (int i = 0; i < m; i++)
    {
        if (board[0][i] == 'o') dfs(0, i, board);
        if (board[n - 1][i] == 'o') dfs(n - 1, i, board);
    }

    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
            if (!st[i][j])
                board[i][j] = 'X';
}

void dfs(int x, int y, vector<vector<char>>&board)
{
    st[x][y] = true;
    int dx[4] = {-1, 0, 1, 0}, dy[4] = {0, 1, 0, -1};
    for (int i = 0; i < 4; i++)
    {
        int a = x + dx[i], b = y + dy[i];
        if (a >= 0 && a < n && b >= 0 && b < m && !st[a][b] && board[a][b]
== 'o')
            dfs(a, b, board);
    }
}
};

```

131. 分割回文串

131. 分割回文串

难度 中等  263  收藏  分享  切换为英文  关注  反馈

给定一个字符串 s ，将 s 分割成一些子串，使每个子串都是回文串。

返回 s 所有可能的分割方案。

示例:

```

输入: "aab"
输出:
[
  ["aa","b"],
  ["a","a","b"]
]

```

```

class Solution {
public:
    vector<vector<string>> ans;
    vector<string> path;

    vector<vector<string>> partition(string s) {
        dfs("", 0, s);
    }

```

```

        return ans;
    }

    bool check(string &now)
    {
        if (now.empty()) return false;
        for (int i = 0, j = now.size() - 1; i < j; i ++, j -- )
            if (now[i] != now[j])
                return false;
        return true;
    }

    void dfs(string now, int u, string &s)
    {
        if (u == s.size())
        {
            if (check(now))
            {
                path.push_back(now);
                ans.push_back(path);
                path.pop_back();
            }
            return;
        }

        if (check(now))
        {
            path.push_back(now);
            dfs("", u, s);
            path.pop_back();
        }

        dfs(now + s[u], u + 1, s);
    }
};

```

132. 分割回文串 II

132. 分割回文串 II

难度 **困难**  129  收藏  分享  切换为英文  关注  反馈

给定一个字符串 s ，将 s 分割成一些子串，使每个子串都是回文串。

返回符合要求的最少分割次数。

示例:

输入: "aab"

输出: 1

解释: 进行一次分割就可将 s 分割成 ["aa","b"] 这样两个回文子串。

```

class Solution {
public:
    int minCut(string s) {

```

```

int n = s.size();
vector<int> f(n + 1);
vector<vector<bool>> st(n,vector<bool>(n,false));

for(int i = 0;i < n;i ++)
    for(int j = i;j >= 0;j --)
        if(i - j <= 1) st[j][i] = s[j] == s[i];
        else st[j][i] = s[j] == s[i] && st[j + 1][i - 1];

f[0] = 0;
for(int i = 1;i <= n;i ++)
{
    f[i] = INT_MAX;
    for(int j = 0;j < i;j ++)
        if(st[j][i - 1])
            f[i] = min(f[i],f[j] + 1);
}
return max(0,f[n] - 1);
}
};

```

133. 克隆图

113. 路径总和 II

难度 中等  204  收藏  分享  切换为英文  关注  反馈

给定一个二叉树和一个目标和，找到所有从根节点到叶子节点路径总和等于给定目标和的路径。

说明: 叶子节点是指没有子节点的节点。

示例:

给定如下二叉树，以及目标和 `sum = 22`，



返回:

```

[
  [5,4,11,2],
  [5,8,4,5]
]

```

```

/**
 * Definition for undirected graph.
 * struct UndirectedGraphNode {
 *     int label;
 *     vector<UndirectedGraphNode *> neighbors;
 *     UndirectedGraphNode(int x) : label(x) {};
 * };

```

```

*/
class Solution {
public:
    unordered_map<UndirectedGraphNode*, UndirectedGraphNode*> hash;

    UndirectedGraphNode *cloneGraph(UndirectedGraphNode *node) {
        if (!node) return 0;
        auto root = new UndirectedGraphNode(node->label);
        hash[node] = root;
        dfs(node);
        return root;
    }

    void dfs(UndirectedGraphNode* node)
    {
        for (auto &neighbor : node->neighbors)
        {
            if (!hash.count(neighbor))
            {
                hash[neighbor] = new UndirectedGraphNode(neighbor->label);
                dfs(neighbor);
            }
            hash[node]->neighbors.push_back(hash[neighbor]);
        }
    }
};

```

134. 加油站

134. 加油站

难度 中等  269  收藏  分享  切换为英文  关注  反馈

在一条环路上有 N 个加油站，其中第 i 个加油站有汽油 $gas[i]$ 升。

你有一辆油箱容量无限的汽车，从第 i 个加油站开往第 $i+1$ 个加油站需要消耗汽油 $cost[i]$ 升。你从其中的一个加油站出发，开始时油箱为空。

如果你可以绕环路行驶一周，则返回出发时加油站的编号，否则返回 -1。

说明：

- 如果题目有解，该答案即为唯一答案。
- 输入数组均为非空数组，且长度相同。
- 输入数组中的元素均为非负数。

示例 1:

输入：

$gas = [1,2,3,4,5]$

$cost = [3,4,5,1,2]$

输出：3

解释：

从 3 号加油站(索引为 3 处)出发，可获得 4 升汽油。此时油箱有 $= 0 + 4 = 4$ 升汽油
开往 4 号加油站，此时油箱有 $4 - 1 + 5 = 8$ 升汽油
开往 0 号加油站，此时油箱有 $8 - 2 + 1 = 7$ 升汽油
开往 1 号加油站，此时油箱有 $7 - 3 + 2 = 6$ 升汽油
开往 2 号加油站，此时油箱有 $6 - 4 + 3 = 5$ 升汽油
开往 3 号加油站，你需要消耗 5 升汽油，正好足够你返回到 3 号加油站。
因此，3 可为起始索引。

示例 2:

输入：

$gas = [2,3,4]$

$cost = [3,4,3]$

输出：-1


```
class Solution {
public:
    int canCompleteCircuit(vector<int>& gas, vector<int>& cost) {
        int n = gas.size();
        vector<int> sum = vector<int>(n * 2, 0);
        for (int i = 0; i < n * 2; i++)
            sum[i] = gas[i % n] - cost[i % n];

        int start = 0, end = 0, tot = 0;
        while (start < n && end <= 2 * n) {
            tot += sum[end];
            while (tot < 0) {
                tot -= sum[start];
                start++;
            }
            if (end - start + 1 == n)
                return start;
            end++;
        }
        return -1;
    }
};
```

```
        return start;
    end++;
}
return -1;
}
};
```

135. 分发糖果

135. 分发糖果

难度 **困难**  188  收藏  分享  切换为英文  关注  反馈

老师想给孩子们分发糖果，有 N 个孩子站成了一条直线，老师会根据每个孩子的表现，预先给他们评分。

你需要按照以下要求，帮助老师给这些孩子分发糖果：

- 每个孩子至少分配到 1 个糖果。
- 相邻的孩子中，评分高的孩子必须获得更多的糖果。

那么这样下来，老师至少需要准备多少颗糖果呢？

示例 1:

输入: [1,0,2]

输出: 5

解释: 你可以分别给这三个孩子分发 2、1、2 颗糖果。

示例 2:

输入: [1,2,2]

输出: 4

解释: 你可以分别给这三个孩子分发 1、2、1 颗糖果。

第三个孩子只得到 1 颗糖果，这已满足上述两个条件。

136. 只出现一次的数字 !!!

136. 只出现一次的数字

难度 简单

👍 1163

📖 收藏

🔗 分享

🌐 切换为英文

👤 关注

📝 反馈

给定一个非空整数数组，除了某个元素只出现一次以外，其余每个元素均出现两次。找出那个只出现了一次的元素。

说明：

你的算法应该具有线性时间复杂度。你可以不使用额外空间来实现吗？

示例 1:

输入: [2,2,1]
输出: 1

示例 2:

输入: [4,1,2,1,2]
输出: 4

```
class Solution {  
public:  
    int singleNumber(vector<int>& nums) {  
        for (int i = 1; i < nums.size(); i++)  
            nums[0] ^= nums[i];  
        return nums[0];  
    }  
};
```

137. 只出现一次的数字 II

137. 只出现一次的数字 II

难度 中等

👍 296

📖 收藏

🔗 分享

🌐 切换为英文

👤 关注

📝 反馈

给定一个非空整数数组，除了某个元素只出现一次以外，其余每个元素均出现了三次。找出那个只出现了一次的元素。

说明：

你的算法应该具有线性时间复杂度。你可以不使用额外空间来实现吗？

示例 1:

输入: [2,2,3,2]
输出: 3

示例 2:

输入: [0,1,0,1,0,1,99]
输出: 99

```

class Solution {
public:
    int singleNumber(vector<int>& nums) {
        int ans = 0;
        for (int bit = 0; bit < 32; bit++) {
            int counter = 0;
            for (int i = 0; i < nums.size(); i++)
                counter += (nums[i] >> bit) & 1;
            ans += (counter % 3) << bit;
        }
        return ans;
    }
};

```

138. 复制带随机指针的链表

138. 复制带随机指针的链表

难度 中等  246  收藏  分享  切换为英文  关注  反馈

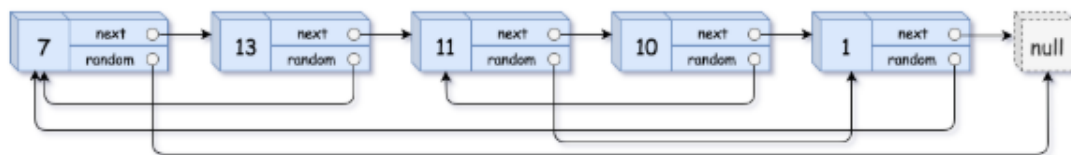
给定一个链表，每个节点包含一个额外增加的随机指针，该指针可以指向链表中的任何节点或空节点。

要求返回这个链表的 **深拷贝**。

我们用一个由 n 个节点组成的链表来表示输入/输出中的链表。每个节点用一个 `[val, random_index]` 表示：

- `val`：一个表示 `Node.val` 的整数。
- `random_index`：随机指针指向的节点索引（范围从 `0` 到 `n-1`）；如果不指向任何节点，则为 `null`。

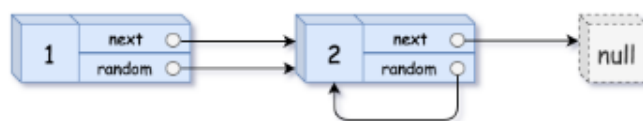
示例 1：



输入：head = [[7,null],[13,0],[11,4],[10,2],[1,0]]

输出：[[7,null],[13,0],[11,4],[10,2],[1,0]]

示例 2：



输入：head = [[1,1],[2,1]]

输出：[[1,1],[2,1]]

/*

```

// Definition for a Node.
class Node {
public:
    int val;
    Node* next;
    Node* random;

    Node() {}

    Node(int _val, Node* _next, Node* _random) {
        val = _val;
        next = _next;
        random = _random;
    }
};
*/
class Solution {
public:
    Node *copyRandomList(Node *head) {
        if (!head) return 0;
        unordered_map<Node*, Node*> hash;
        Node *root = new Node(head->val, NULL, NULL);
        hash[head] = root;
        while (head->next)
        {
            if (hash.count(head->next) == 0)
                hash[head->next] = new Node(head->next->val, NULL, NULL);
            hash[head]->next = hash[head->next];

            if (head->random && hash.count(head->random) == 0)
                hash[head->random] = new Node(head->random->val, NULL, NULL);
            hash[head]->random = hash[head->random];

            head = head->next;
        }

        if (head->random && hash.count(head->random) == 0)
            hash[head->random] = new Node(head->random->val, NULL, NULL);
        hash[head]->random = hash[head->random];

        return root;
    }
};

```

139. 单词拆分

139. 单词拆分

难度 中等

👍 384

📖 收藏

🔗 分享

🌐 切换为英文

🔔 关注

📝 反馈

给定一个非空字符串 s 和一个包含非空单词列表的字典 $wordDict$ ，判定 s 是否可以被空格拆分为一个或多个在字典中出现的单词。

说明：

- 拆分时可以重复使用字典中的单词。
- 你可以假设字典中没有重复的单词。

示例 1：

输入：s = "leetcode", wordDict = ["leet", "code"]
输出：true
解释：返回 true 因为 "leetcode" 可以被拆分成 "leet code"。

示例 2：

输入：s = "applepenapple", wordDict = ["apple", "pen"]
输出：true
解释：返回 true 因为 "applepenapple" 可以被拆分成 "apple pen apple"。
注意你可以重复使用字典中的单词。

示例 3：

输入：s = "catsanddog", wordDict = ["cats", "dog", "sand", "and", "cat"]
输出：false

```
class Solution {
public:
    bool wordBreak(string s, vector<string>& wordDict) {
        unordered_set<string> dict(wordDict.begin(), wordDict.end());

        vector<bool> dp(s.size() + 1, false); // dp表示字符之间的隔板，n个字符有n+1个隔板
        dp[0] = true; // dp[0]是s[0]前面的隔板

        for(int i = 1; i <= s.size(); i++)
        {
            for(int j = i; j >= 0; j--)
                if(dict.find(s.substr(j, i - j)) != dict.end() && dp[j])
                {
                    dp[i] = true;
                    break;
                }
        }
        return dp[s.size()];
    }
};
```

140. 单词拆分 II

40. 组合总和 II

难度 中等

👁 238

♡ 收藏

🔗 分享

🌐 切换为英文

🔔 关注

💬 反馈

给定一个数组 `candidates` 和一个目标数 `target`，找出 `candidates` 中所有可以使数字和为 `target` 的组合。

`candidates` 中的每个数字在每个组合中只能使用一次。

说明：

- 所有数字（包括目标数）都是正整数。
- 解集不能包含重复的组合。

示例 1:

```
输入: candidates = [10,1,2,7,6,1,5], target = 8,
所求解集为:
[
  [1, 7],
  [1, 2, 5],
  [2, 6],
  [1, 1, 6]
]
```

示例 2:

```
输入: candidates = [2,5,2,1,2], target = 5,
所求解集为:
[
  [1,2,2],
  [5]
]
```