1. 两数之和

遍历找到和为定值的两个数（可以是相同的数值）

- `vector.push_back()`
- `vector.size()`
- `vector` 不要轻易赋初始值

2. 两数相加

两个列表同时遍历，对应位置的数值求和（列表头尾的保存或逆序）

- `ListNode*` 需要初始化空值
- 列表元素多少不好找
- 列表插入的操作还需要更熟练！

3. 无重复字符最长子串

最长子串里不能有一样的字符

- 使用 `CHAR` 到 `INT` 的映射记录出现的字符
- 对`map`的更新，对子串的遍历查找

4. 找两个正序数组中位数

有时间可以学习低空间时间的更优解

5. 最长回文串 （与最长公共子串区分）

1. 暴力解法
2. 动态规划 可视化理解
3. Macacher 中心扩散，有时间学习

10. 正则表达式匹配

使用模式 `p` 字符串，具有`.` 和 `*` 规则，匹配目标模式串`s`
讲解及示意图 动态规划
使用案例包括

	S	P
1	abaaa	a.
2	abaaad	aba*cd
3	ab	.*
4	aab	c*a*b

11. 盛最多水的容器

底只会变小，向中心收拢，哪边板子低就不要哪边

15. 三数之和

暴力卡时间，利用去重的思想，使用左右指针节省遍历次数

17. 电话号码的字母组合

- [回溯] 介绍了由多层循环到回溯的转化 / [递归] / [DFS]
- 有时间学习利用队列的解法 BFS

19.删除链表的倒数第`n`个结点

遍历两遍， 注意删除唯一一个和删除第一个

> *如何遍历一遍解决问题?*
>
> *双指针，`A` 先走 `k`步，`A`、`B` 一起走 `n-k` 步*

20. 有效的括号

一看就是 `stack` ，注意压入弹出的时机

21. 合并两个有序链表

- 遍历节点和记录节点声明先后顺序
- 循环的及时终止

22. 括号生成

递归 DFS

23. 合并k个升序链表

- 暴力，所有的放在一起`sort`，最后放入一个链表里面
- 或者暴力两两合并，`O(K^2*N)`

  > *分治与二分的区别 以及对于分治降低复杂度的理解*

- 优先队列设计小顶堆，`O(KlogK*N)`,(结构体运算符重载<1>,<2题解>,) 各种堆

31. 下一个排列 *

确实没能思考出等价条件 => 题解
题目等同于更大的相邻整数

32. 最长有效括号

- 暴力结果超时
- 巧解1 扫描两遍
- 巧解2 dp（意识到了但是没有自己解出来）

33. 旋转数组

考察二分，在有序的里面二分

34. 在排序数组中查找元素的第一个和最后一个位置

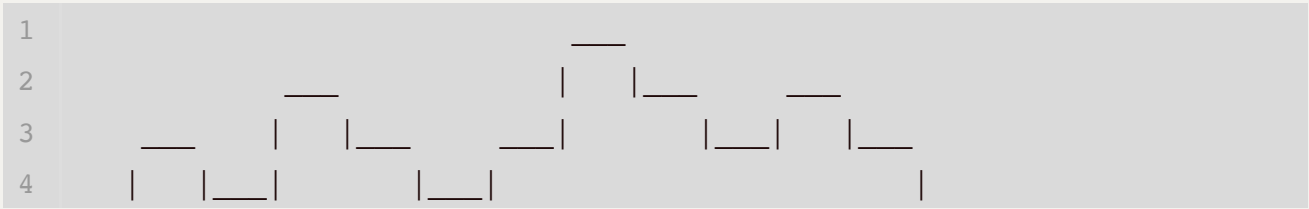
二分查找，解析

找左界和右界使用不同的中值划分，可以不用在循环外判断左右界更新

39. 组合总和

集合中的数可重复选，和为定值，排列组合使用. `DFS`

42. 接雨水 *



由建模按列求解，[到 dp 优化 √]，再到 双指针 (图 解) 优化

46. 全排列

`DFS` + `vector.push_back`

48. 旋转图像

- 自外向内一共有不超过 `n/2` 层， 4 份 `len-1` 的边. (`len = nums.size() - (layer * 2)`)
- (时间消耗大) 先转置再镜像 (注意循环边界)

49. 字母异位词分组 *

`ate, eta` 在一组， 同理讲字符串分组

- 我用的 DFS （可以处理一个单词内的相同字符了，但是）超时
- 题解 使用到了 `map` (字符串映射，字符数量映射，[质数映射?])

```
map<string, int>::iterator it;
for (it = m2.begin(); it != m2.end(); it++)
    string s = it->first;
```

或者在c++11后

```
for(auto it : mapping01)
  cout << it.first <<" "<< it.second <<endl;
```

51. N 皇后 （帮助同学）

DFS, `slope_array * 2`, `column`, `row`, `boundary (n)`, `vector`, `vector<vector>>`

53. 最大子序和

dp 遍历一遍 O (n)

有时间学习尝试分治法

55. 跳跃游戏

一维动规，看前一个的覆盖距离

56. 合并区间

- 对于sort中比较函数的定义

```
static bool toCompare (vector<int> a, vector<int> b) {
return (a[0] < b[0]); }
```

- 左右指针，排序后三种情况，循环一层仍然费空间费时间...

有时间学习优解

62. 不同路径 *

排列组合 `路径数量 = C（向下和向右操作数量，向右操作数量）`

- 组合数的优化计算，时间空间节省，但是可迁移性差，需要顾及不能整型越界（`DFS` 更不用提）
- 题解动态规划，尝试一下

64. 最小路径和

同右下的模拟操作，动规解决，注意初始状态为0

70. 爬楼梯

DFS 超时，动态规划 `O(n)` 可行 （空间可优化为0维）

72. 编辑距离 ***

动态规划

> *`D[i][j]` 表示 `A` 的前 `i` 个字母和 `B` 的前 `j` 个字母之间的编辑距离*

可加一个字符，可删一个字符，可替换一个字符，求由单词A到B的最短距离

> *本质不同的操作实际上只有三种：*
>
> - *在单词 `A` 中插入一个字符；*
> - *在单词 `B` 中插入一个字符；*
> - *修改单词 `A` 的一个字符。*

也可以理解成 转化到 `dp[i][j]`

> *`dp[i-1][j-1]` 表示替换操作，`dp[i-1][j]` 表示删除操作，`dp[i][j-1]` 表示插入操作*

75. 颜色分类

就是手写排序：两层循环

以及双指针 => [三路快排 O(n) 常数空间的题解]

```
while (idx <= r) {
 if (nums[idx] == 0) swap(nums, l++, idx++); // 变量 l 为下一个填入 0 的位置 (
 else if (nums[idx] == 1) idx++;             //
 else swap(nums, idx, r--);                  // 变量 r 为下一个填入 2 的位置
}
```

76. 最小覆盖子串 *

题解： 左右指针滑动窗口 （不用dp）
一上来我count比较字符串包含关系，超时

题解的 distance 思想: 比较A字符串包含B字符串中字符的个数，绝了
有一道题case267还是超时了

78. 子集

DFS 我可行

或者二进制枚举，题解：

```
vector<vector<int>> subsets(vector<int>& nums) {
 for (int mask = 0; mask < (1 << nums.size()); ++mask) {
     t.clear();
     for (int i = 0; i < nums.size(); ++i)
         if (mask & (1 << i))
             t.push_back(nums[i]);

     ans.push_back(t);
 }
 return ans;
}
```

79. 单词搜索 * DFS_grid寻单词

加标记就不会环起来 (只有一行或者只有一列或者在边界) （什么时候返回 true，什么时候返回 false ）

简介JAVA版答案

```java
class Solution {
 public boolean exist(char[][] board, String word) {
     for (int i = 0; i < board.length; i++)
         for (int j = 0; j < board[0].length; j++)
             if (search(board, word, i, j, 0))
                 return true;
     return  false;
 }
 boolean search(char[][] board, String word, int i, int j, int k) {
     if (k >= word.length()) return true;
     if (i<0||i>=board.length||j<0||j>=board[0].length || board[i][j]!=word.charAt(k))
         return false;
     board[i][j] += 256;
     boolean result = search(board, word, i-1, j, k+1)||search(board, word, i+1, j, k+1)||
                      search(board, word, i, j-1, k+1)||search(board, word, i, j+1, k+1);
     board[i][j] -= 256;
     return result;
 }
}
```

84. 柱状图中最大的矩形 $ *

单调栈，42题同样可以使用，还有739、496、316、901、402、581 题

单调栈做法解释
右边没有递增的了，挨个出栈找到延展到左边最大的矩形面积（还要保证左边不能比右边小，否则继续压入单调栈）

85. 最大矩形 $

动态规划 一下逐行转化为 84 柱状图中最大的矩形 $ 模型问题求解

94. 二叉树的中序遍历

左--中--右

96. 不同的二叉搜索树

公式 $F(i,n) = G(i-1) * G(n-i)$

```
1
2
3
4
5
6
```

$G(n) = \sum_{i=1}^{n} G(i-1) * G(n-i)$

98. 验证二叉搜索树

中序， 同时考虑上下界才行 *

101. 对称二叉树

左左等于右右，左右等于右左

102. 二叉树的层序遍历

队列 BFS （迭代）或者
递归（有点绕）

```
class Solution {
public:
    vector<vector<int>> ret;
    void level(TreeNode* root, int lev) {
        if(!root) return;
        if (lev >= ret.size())
            ret.push_back(vector<int>());

        ret[lev].push_back(root -> val);
        level(root -> left, lev + 1);
        level(root -> right, lev + 1);
    }
    vector<vector<int>> levelOrder(TreeNode* root) {
        level(root, 0);
        return ret;
    }
};
```

104. 二叉树的最大深度

一般的 dfs

105. 从前序与中序遍历序列构造二叉树 *** （待办）

已经遗忘了一

分别记录左子树、右子树各自的的前序和中序坐标，简单粗暴，

一定要学习一下简洁的解法：详细通俗的思路分析，多解法 - 从前序与中序遍历序列构造二叉树

114. 二叉树展开为链表

自己创建空间也要保证原地址...

学习原地

```
class Solution {
public:
    void flatten(TreeNode* root) {                          //   1
      1        1
        TreeNode *curr = root;                              // 2   3
     2          2
        while (curr != nullptr) {                           //4 5 6 7
     4 5        4 5
            if (curr->left != nullptr) {                    //
       3          3
                auto next = curr->left;                     //
      6 7        6 7
                auto predecessor = next;                    //
                while (predecessor->right != nullptr) {
                    predecessor = predecessor->right;
                }
                predecessor->right = curr->right;
                curr->left = nullptr;
                curr->right = next;
            }
            curr = curr->right;
```

```
        }
    }
};
```

121. 买卖股票的最佳时机

二维暴力超时。
单调栈可行，不过时间空间占用较差

> *单调栈的作用是：用 O(n) 的时间得知所有位置两边第一个比他大(或小)的数的位置。*

还可以采取动态规划

> *重叠子问题，即买卖股票的最佳时机是由之前买或不买的状态决定，而之前买不买又由更早状态决定*

> *状态压缩：详解 压缩状态动态规划* ***

124. 二叉树中的最大路径和

dfs 六种情况其中三种情况可以向上累加

128. 最长连续序列

我的暴力法（排序后看相邻）时间超o(n)，

还有放入hash set去重，遍历两遍，在第二遍找到一个前面无连续的位置往后查

```
unordered_set<int> num_set;
for (const int& num : num_set) {}
```

136. 只出现一次的数字

我的暴力法（排序后看相邻）？ 时间o(nlogn) 空间o(logn)

不考虑其他，可以通过 集合存储出入 / 哈希 / Double之后减去集合之和

线性时间和常数空间下位运算

```
int singleNumber(vector<int>& nums) {
    int ret = 0;
    for (auto e: nums) ret ^= e;
    return ret;
}
```

139. 单词拆分 *** 重要

DFS, 剪枝,去除多余操作, 记忆化方法,

> *记录以startIndex开始的子串是不可以被拆分的*

BFS, 避免访问重复的节点, 记录访问过的指针 学习

动态规划... 官解

`dp[i]` 表示前i个可拆分 `s[0, j-1]` (`dp[j]`) 和 `s[j, i - 1]`

141. 环形链表

我是改数值遍历， 还有快慢指针(速度2倍)法： 代码随想录 有230道 这一百道刷完了再刷

142. 环形链表 II

同上道题

146. LRU 缓存 背诵记忆 LFU <--> LRU 模拟题还是不熟练，再刷几遍

LRU (最近最少使用) 缓存

先声明后创建

```
ListNode* dummyHead;
dummyHead = new ListNode();
```

可以思考，学习如何节省更多的时间

`unordered_map<int, ListNode*> map;` or

`list<pair<int, int>> cache;`
`unordered_map<int, list<pair<int, int>>::iterator> map;`

148. 排序链表

递归排序三部曲：1，快慢指针找中点；2，递归调用mergeSort，3，合并两个链表

除了归并排序还需掌握 快排

待续(基准点左右插链，还没有写)

152.乘积最大子数组

相较53最大子数组组合，需要维护最小值保证负负相消 （我本来想`vector<struct dp {int raw; int abs;}>` 为绝对值排序，意思相近但是不好实现）

分治方法待学习

155.最小栈 *

原来说的不是单调栈，同时存raw数据和当前最小数值和152我开始的想法一样，
省空间的话，记录差值 *

还有人用链表实现...

169.多数元素

找到大于整体一般数量的元素，摩尔投票法 :)

198.打家劫舍

状态方程没有看出来 `dp[i] = nums[i] + max(dp[i-2], dp[i-3])`

200.岛屿数量

dfs 铲平岛屿

206. 反转链表

迭代（保存一个之后的一个之前的）走一遍链
递归（想象k之后的都已经反转好，后面的操作是 a->next->next = a, a->next = nullptr,一直返回头节点也就是正序的最后一个）

207.课程表

拓扑, visit所有的边分三种状态，dfs开始0，结束2，doing1，据源点 visit 所有的终点

208.前缀树 Trie

```
    struct TreeNode {
        bool isEnd;
        TreeNode* next[26];
        TreeNode() : isEnd(false), next() {}
    };
    TreeNode* root;
```

215 数组中的第K个最大元素 *** 要学习堆排序

sort没什么可说的，但是要求了o(n)复杂度好吧，题解提供了个快排（带加速随机化），一个堆排序

> *快排递归，但是不关系左序列和右序列是否有序，`l->r, swap(nums[l--], nums[r])`*
> *堆排 - 大根堆的实现方法，`***` 堆-LeetBook*

221. 最大正方形

dp题目，可以想到左上到右下斜线成方形，那么dp的思路，由这条线作对角线构成的方形作状态推演

226. 翻转二叉树

DFS. easy

234. 回文链表

栈

236. 二叉树的最近公共祖先

思路不清晰

1. 递归，看左孩子和右孩子的包含关系

> *这个判断条件有点意思，左和右子树同时涵盖两个节点，另一种情况是在一条链上*

2. 存储父节点

> *这个方法也有点意思，先遍历一遍，有所标记（Hash 存储），第二次遍历找到 LCA 节点*
> `unordered_map<int, TreeNode*> fa; unordered_map<int, bool> vis;`

238. 除自身以外数组的乘积

前缀后缀乘积

学习 O(1) 空间复杂度解法: 额，其实就是左右同时在遍历，一次遍历就乘两次

239. 滑动窗口最大值

单调队列，当存储idx之后，只需要保证top在窗口内即可

240. 搜索二维矩阵 II

首先，能根据大小关系思考到游走，只不过游走的方向是从右上向左或下游走！

> *其他方法包括二分查找和其他 有时间要学习*

279. 完全平方数

看出了动规，状态方程没推出来，有点方：
$f(x) = 1 + \min_{j=1}^{\sqrt{i}} f(i - j^2)$

时间复杂度优先队列 `nlogn` 超了...

283. 移动零元素

可以双指针，也可以记录非0的数量直接赋值，

```java
public void moveZeroes(int[] nums) {
    for(int i = 0, count = 0;i < nums.length; i++){
        if(nums[i] != 0){
            if (count != i) {
                nums[count] = nums[i];
                nums[i] = 0;
            }
            count++;
        }
    }
}
```

287. 寻找重复数

sort容易如何找到O(n)时间复杂度和O(1) 空间复杂度呢?

1. 快慢指针
2. 2分查找
   查找数量上的二分，

   > *如果没有重复，小于等于它的个数 m 肯定比这个数小，但是如果大了，说明就是重复的那个数。"计数排序" 就是这么个原理*

> *快慢指针 形象讲解*

297. Design 二叉树的序列化与反序列化

DFS掌握熟练，序列化只需一种遍历方式，存储转换字符串

```
    store += to_string(tree->val) + ",";
    tree = new TreeNode(stoi(tmp));
```

Design 待做题库

---

300. 最长递增子序列

想了单调栈 X,
想了动规，不会推, 一看题解原来就是O(n2)的复杂度

还可以二分 之后记得学习 贪心

301. 删除无效的括号 * 没有完全独立，再刷一次

面对最少需要次数，BFS 可以提前结束
这个必须要去重 set，其他超时 （已经蕴含了剪枝和提前终止操作）
还是要学习题解
还有回溯剪枝 和 枚举状态子集

309. 最佳买卖股票时机含冷冻期 * 更复杂 dp 没有思路

最佳买卖股票时机含冷冻期-代码随想录 只是结果复杂度较大

考虑了4状态, 达到买入持有， 达到卖出不持有（过了冷冻期但是没有持有），达到冷冻期，卖出达到卖出不持有

312.戳气球，动态规划

window 在扩张，在 window 中分区找到子问题

1. 戳气球 - 戳气球 - 力扣（LeetCode）
2. [这个菜谱, 自己在家也能做] 关键思路解释 - 戳气球 - 力扣（LeetCode）
3. 图解：动态规划解决戳气球问题，思路清晰简明，注释详细 - 戳气球 - 力扣（LeetCode）
4. 超详细回溯到分治到DP - 戳气球 - 力扣（LeetCode）

记住思路

322. 零钱兑换

动规划，初始化的值是最大值，dp存的是张数，没有其他特别的
可以动规划的题，可以记忆化搜索自顶向下，以及DFS，之后练习

337. 打家劫舍-iii

- dfs 可以在判断左支右支非空的情况下，分别计算子孙节点与自己的和 和 孩子节点的和，两种情况的比较
- 动态规划怎么做呢
  $f(o)$ 表示选择 o 节点的情况下，o 节点的子树上被选择的节点的最大权值和
  $g(o)$ 表示不选择 o 节点的情况下，o 节点的子树上被选择的节点的最大权值和
  选不选左谁最大不一定，选不选右谁最大也不一定

338. 比特位计数

靠奇偶，while 做除法
还有位运算 比如Brian Kernighan 算法 题解 或动规

347. 前 K 个高频元素

map, -> vector sort,官方： 小顶堆 / 快排

- `priority_queue<pair<int, int>, vector<pair<int, int>>, decltype(&cmp)> q(cmp);`
  堆的思路：如果q里面数量还没到k，就不停的加，直到满了，（堆已排序）出现次数比最少的更多的进来
- 快排有点复杂，回头再说

394. 字符串解码

设想用栈 + dfs，一塌糊涂 ...
单独 DFS 可以做出，设计，先记录数字，在`'['`的时候 dfs，保留当前 idx 并引用传递右括号所在 idx，在 `']'` 的时候传回字符串
还要学习 使用 双栈的做法

---

399. 除法求值

- 带权并查集
- Floyd 算法 √恐怖，先记住中规中矩的答法
- 广度优先搜索

这真的只是一道 Medium 吗...
这道题我爆了
先重新学会 Floyd 算法

---

406. 根据身高重建队列 * 又没思路

先对输入数组排序，h升序，k降序 从头循环遍历 当前这个人就是剩下未安排的人中最矮的人，他的k值就代表他在剩余空位的索引值 如果有多个人高度相同，要按照k值从大到小领取索引值

```
[ 0, 1, 2, 3, 4, 5 ] [ 4, 4 ] 4
[ 0, 1, 2, 3, 5 ]    [ 5, 2 ] 2
[ 0, 1, 3, 5 ]       [ 5, 0 ] 0
[ 1, 3, 5 ]          [ 6, 1 ] 3
[ 1, 5 ]             [ 7, 1 ] 5
[ 1 ]                [ 7, 0 ] 1
[ [ 5, 0 ], [ 7, 0 ], [ 5, 2 ], [ 6, 1 ], [ 4, 4 ], [ 7, 1 ] ]
```

> *What can you say about the position of the shortest person? If the position of the shortest person is i, how many people would be in front of the shortest person? 先考虑数组中最小身高的（可以有多个），这些 person 在新数组的位置就是 person[1]*
> *Once you fix the position of the shortest person, what can you say about the position of the second shortest person? 再考虑次小的*

从高到底考虑代码量更小 插空

416. 分割等和子集

在一个数组中找到某些元素和为定值

- DFS
- 动态规划 判断数组中是否存在若干元素和为目标值问题
  两层内增循环的背包可以设置 `dp[i][j]:`容量为`j`的前`i`个物品所能组成的最大价值
  还可以对空间进行优化至以一维 dp

437. 路径总和 III

DFS 但是先对根节点 DFS，之后向下遍历，在子节点再分别 DFS
（想要通过一次 DFS 会出现难以处理的冗余问题... ）

438. 找到字符串中所有字母异位词 *

看出来是滑动窗口了系列

```
/* 滑动窗口算法框架 */
void slidingWindow(string s, string t) {
    unordered_map<char, int> need, window;
    for (char c : t) need[c]++;

    int left = 0, right = 0;
    int valid = 0;
    while (right < s.size()) {
```

```
        // c 是将移入窗口的字符
        char c = s[right];
        // 右移窗口
        right++;
        // 进行窗口内数据的一系列更新
        ...

        /*** debug 输出的位置 ***/
        printf("window: [%d, %d)\n", left, right);
        /**********************/

        // 判断左侧窗口是否要收缩
        while (window needs shrink) {
            // d 是将移出窗口的字符
            char d = s[left];
            // 左移窗口
            left++;
            // 进行窗口内数据的一系列更新
            ...
        }
    }
}
```

448. 找到所有数组中消失的数字

我先用了排序后双指针的笨方法，还别忘了三种条件都有可能需要 push_back 进去缺失的元素
为了满足 O(n)

```
for (auto& num : nums) {
    int x = (num - 1) % n;
    nums[x] += n;
}
```

之后，nums 中小于等于 n 的则是没有出现过的

461. 汉明距离

异或，取 1 的个数

---

494. 目标和

$set(a) - set(b) = target$
子集求和转换
动态规划，只是 dp 存贮的是策略的数量
$dp[i][j] = dp[i-1][j] + dp[i-1][j-nums[i-1]]$
$dp[i][j] = dp[i-1][j]$

---

538. 把二叉搜索树转换为累加树

dfs 注意左枝返回值和 !root 返回值

543. 二叉树的直径

简单的 DFS，就是返回的时候叶子节点单独考虑，返回值 -> 就已经是路径长度了

560. 和为 K 的子数组

- 滑动窗口因为有负数，不可以被使用（全正数可以考虑）
- 枚举的话，外层从0往后，内层从外层起始点往前
- 前缀和 + Hash

> *[j..i] 这个子数组和为 k 表示为*
> $pre[i] - pre[j-1] == k$
> *官方动画*

```cpp
int subarraySum(vector<int>& nums, int k) {
    unordered_map<int, int> hash;
    hash[0] = 1;
    int preSum = 0, count = 0;
    for (int i = 0; i < nums.size(); i++) {
        preSum += nums[i];
        count += hash[preSum - k];
        hash[preSum]++;
    }
    return count;
}
```

581. 最短无序连续子数组

- 法一 排序后看不一样的位置，复杂度较高
- 法二，一遍遍历，
  从左往右，比最大值还小是右边，从右往左，比最小值还大是左边

617. 合并二叉树

同时遍历，我是左空接右枝
先判断孩子是否有空，在进入 DFS

官解很简练

```cpp
class Solution {
    public TreeNode mergeTrees(TreeNode t1, TreeNode t2) {
        if (t1 == null) {
            return t2;
        }
        if (t2 == null) {
```

```
            return t1;
        }
        TreeNode merged = new TreeNode(t1.val + t2.val);
        merged.left = mergeTrees(t1.left, t2.left);
        merged.right = mergeTrees(t1.right, t2.right);
        return merged;
    }
}
```

621. 任务调度器

额，桶思想

- (1)
- (2)
- (3)

> *当任务总数不超过 (n+1)×(max−1)+tot 时，我们总能将其他任务插到空闲时间中去，不会引入额外的冻结时间；而当任务数超过该值时，我们可以在将其横向添加每个 n+1 块的后面，同时不会引入额外的冻结时间*

`x.size()` 还需要 `int()` 一下

647. 回文子串

动态规划
斜着遍历

```
for l in range(1, n):
  for i in range(n - l):
    j = l + i
    dp[i][j] .....
```

- O(N)_O(N) Manacher

739. 每日温度

暴力超时，这道题竟然是单调栈?
只操作下标 简单来说，栈不空，单调递减

```
vector<int> dailyTemperatures(vector<int>& temperatures) {
    vector<int> res(temperatures.size());
    stack<int> asc;
    for (int i = 0; i < temperatures.size(); i++) {
        while (!asc.empty() && temperatures[i] >
temperatures[asc.top()])
        {
            int x = asc.top();
            asc.pop();
            res[x] = i - x;
        }
        asc.push(i);
    }
    return res;
}
```