

## 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， $O(K \log K * N)$ ，(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

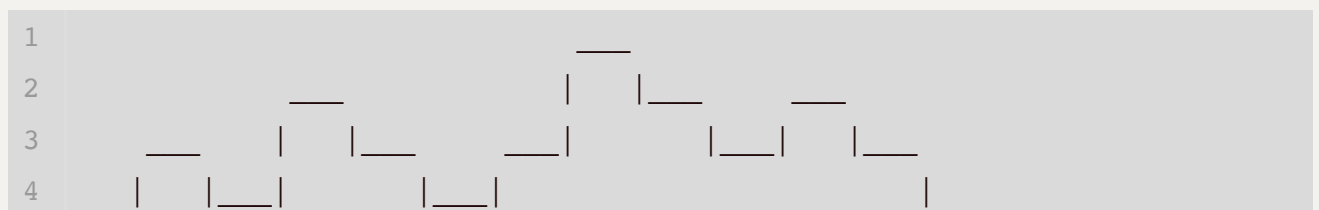
二分查找, [解析](#)

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](#)  $\sqrt{}$ ], 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?])

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

或者在C++11后

```
1 for(auto it : mapping01)
2     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

```
1 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](#)]



```

1 while (idx <= r) {
2     if (nums[idx] == 0) swap(nums, l++, idx++); // 变量 l 为下一个
        填入 0 的位置 (
3     else if (nums[idx] == 1) idx++; //
4     else swap(nums, idx, r--); // 变量 r 为下一个
        填入 2 的位置
5 }

```

## 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：

```

1 vector<vector<int>> subsets(vector<int>& nums) {
2     for (int mask = 0; mask < (1 << nums.size()); ++mask) {
3         t.clear();
4         for (int i = 0; i < nums.size(); ++i)
5             if (mask & (1 << i))
6                 t.push_back(nums[i]);
7
8         ans.push_back(t);
9     }
10    return ans;
11 }

```

## 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

```
1 class Solution {
2     public boolean exist(char[][] board, String word) {
3         for (int i = 0; i < board.length; i++)
4             for (int j = 0; j < board[0].length; j++)
5                 if (search(board, word, i, j, 0))
6                     return true;
7         return false;
8     }
9     boolean search(char[][] board, String word, int i, int j,
10 int k) {
11         if (k >= word.length()) return true;
12         if (i < 0 || i >= board.length || j < 0 || j >= board[0].length ||
13 board[i][j] != word.charAt(k))
14             return false;
15         board[i][j] += 256;
16         boolean result = search(board, word, i-1, j,
17 k+1) || search(board, word, i+1, j, k+1) ||
18 search(board, word, i, j-1,
19 k+1) || search(board, word, i, j+1, k+1);
20         board[i][j] -= 256;
21         return result;
22     }
23 }
```

## 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）

```
1  class Solution {
2  public:
3      vector<vector<int>> ret;
4      void level(TreeNode* root, int lev) {
5          if(!root) return;
6          if (lev >= ret.size())
7              ret.push_back(vector<int>());
8
9          ret[lev].push_back(root -> val);
10         level(root -> left, lev + 1);
11         level(root -> right, lev + 1);
12     }
13     vector<vector<int>> levelOrder(TreeNode* root) {
14         level(root, 0);
15         return ret;
16     }
17 };
```

## 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

```
1  class Solution {
2  public:
3      void flatten(TreeNode* root) {                                // 1
4          // 1 1
5          // 2 2
6          // 3 3
7          // 4 5 4 5
8          // 6 7 6 7
9          // 10 11
10         // 12 13
11         // 14 15
12         // 16
13         // 17
14         // 18
15         // 19
16         // 20
17         // 21
18         // 22
19         // 23
20         // 24
21         // 25
22         // 26
23         // 27
24         // 28
25         // 29
26         // 30
27         // 31
28         // 32
29         // 33
30         // 34
31         // 35
32         // 36
33         // 37
34         // 38
35         // 39
36         // 40
37         // 41
38         // 42
39         // 43
40         // 44
41         // 45
42         // 46
43         // 47
44         // 48
45         // 49
46         // 50
47         // 51
48         // 52
49         // 53
50         // 54
51         // 55
52         // 56
53         // 57
54         // 58
55         // 59
56         // 60
57         // 61
58         // 62
59         // 63
60         // 64
61         // 65
62         // 66
63         // 67
64         // 68
65         // 69
66         // 70
67         // 71
68         // 72
69         // 73
70         // 74
71         // 75
72         // 76
73         // 77
74         // 78
75         // 79
76         // 80
77         // 81
78         // 82
79         // 83
80         // 84
81         // 85
82         // 86
83         // 87
84         // 88
85         // 89
86         // 90
87         // 91
88         // 92
89         // 93
90         // 94
91         // 95
92         // 96
93         // 97
94         // 98
95         // 99
96         // 100
97         // 101
98         // 102
99         // 103
100        // 104
101        // 105
102        // 106
103        // 107
104        // 108
105        // 109
106        // 110
107        // 111
108        // 112
109        // 113
110        // 114
111        // 115
112        // 116
113        // 117
114        // 118
115        // 119
116        // 120
117        // 121
118        // 122
119        // 123
120        // 124
121        // 125
122        // 126
123        // 127
124        // 128
125        // 129
126        // 130
127        // 131
128        // 132
129        // 133
130        // 134
131        // 135
132        // 136
133        // 137
134        // 138
135        // 139
136        // 140
137        // 141
138        // 142
139        // 143
140        // 144
141        // 145
142        // 146
143        // 147
144        // 148
145        // 149
146        // 150
147        // 151
148        // 152
149        // 153
150        // 154
151        // 155
152        // 156
153        // 157
154        // 158
155        // 159
156        // 160
157        // 161
158        // 162
159        // 163
160        // 164
161        // 165
162        // 166
163        // 167
164        // 168
165        // 169
166        // 170
167        // 171
168        // 172
169        // 173
170        // 174
171        // 175
172        // 176
173        // 177
174        // 178
175        // 179
176        // 180
177        // 181
178        // 182
179        // 183
180        // 184
181        // 185
182        // 186
183        // 187
184        // 188
185        // 189
186        // 190
187        // 191
188        // 192
189        // 193
190        // 194
191        // 195
192        // 196
193        // 197
194        // 198
195        // 199
196        // 200
197        // 201
198        // 202
199        // 203
200        // 204
201        // 205
202        // 206
203        // 207
204        // 208
205        // 209
206        // 210
207        // 211
208        // 212
209        // 213
210        // 214
211        // 215
212        // 216
213        // 217
214        // 218
215        // 219
216        // 220
217        // 221
218        // 222
219        // 223
220        // 224
221        // 225
222        // 226
223        // 227
224        // 228
225        // 229
226        // 230
227        // 231
228        // 232
229        // 233
230        // 234
231        // 235
232        // 236
233        // 237
234        // 238
235        // 239
236        // 240
237        // 241
238        // 242
239        // 243
240        // 244
241        // 245
242        // 246
243        // 247
244        // 248
245        // 249
246        // 250
247        // 251
248        // 252
249        // 253
250        // 254
251        // 255
252        // 256
253        // 257
254        // 258
255        // 259
256        // 260
257        // 261
258        // 262
259        // 263
260        // 264
261        // 265
262        // 266
263        // 267
264        // 268
265        // 269
266        // 270
267        // 271
268        // 272
269        // 273
270        // 274
271        // 275
272        // 276
273        // 277
274        // 278
275        // 279
276        // 280
277        // 281
278        // 282
279        // 283
280        // 284
281        // 285
282        // 286
283        // 287
284        // 288
285        // 289
286        // 290
287        // 291
288        // 292
289        // 293
290        // 294
291        // 295
292        // 296
293        // 297
294        // 298
295        // 299
296        // 300
297        // 301
298        // 302
299        // 303
300        // 304
301        // 305
302        // 306
303        // 307
304        // 308
305        // 309
306        // 310
307        // 311
308        // 312
309        // 313
310        // 314
311        // 315
312        // 316
313        // 317
314        // 318
315        // 319
316        // 320
317        // 321
318        // 322
319        // 323
320        // 324
321        // 325
322        // 326
323        // 327
324        // 328
325        // 329
326        // 330
327        // 331
328        // 332
329        // 333
330        // 334
331        // 335
332        // 336
333        // 337
334        // 338
335        // 339
336        // 340
337        // 341
338        // 342
339        // 343
340        // 344
341        // 345
342        // 346
343        // 347
344        // 348
345        // 349
346        // 350
347        // 351
348        // 352
349        // 353
350        // 354
351        // 355
352        // 356
353        // 357
354        // 358
355        // 359
356        // 360
357        // 361
358        // 362
359        // 363
360        // 364
361        // 365
362        // 366
363        // 367
364        // 368
365        // 369
366        // 370
367        // 371
368        // 372
369        // 373
370        // 374
371        // 375
372        // 376
373        // 377
374        // 378
375        // 379
376        // 380
377        // 381
378        // 382
379        // 383
380        // 384
381        // 385
382        // 386
383        // 387
384        // 388
385        // 389
386        // 390
387        // 391
388        // 392
389        // 393
390        // 394
391        // 395
392        // 396
393        // 397
394        // 398
395        // 399
396        // 400
397        // 401
398        // 402
399        // 403
400        // 404
401        // 405
402        // 406
403        // 407
404        // 408
405        // 409
406        // 410
407        // 411
408        // 412
409        // 413
410        // 414
411        // 415
412        // 416
413        // 417
414        // 418
415        // 419
416        // 420
417        // 421
418        // 422
419        // 423
420        // 424
421        // 425
422        // 426
423        // 427
424        // 428
425        // 429
426        // 430
427        // 431
428        // 432
429        // 433
430        // 434
431        // 435
432        // 436
433        // 437
434        // 438
435        // 439
436        // 440
437        // 441
438        // 442
439        // 443
440        // 444
441        // 445
442        // 446
443        // 447
444        // 448
445        // 449
446        // 450
447        // 451
448        // 452
449        // 453
450        // 454
451        // 455
452        // 456
453        // 457
454        // 458
455        // 459
456        // 460
457        // 461
458        // 462
459        // 463
460        // 464
461        // 465
462        // 466
463        // 467
464        // 468
465        // 469
466        // 470
467        // 471
468        // 472
469        // 473
470        // 474
471        // 475
472        // 476
473        // 477
474        // 478
475        // 479
476        // 480
477        // 481
478        // 482
479        // 483
480        // 484
481        // 485
482        // 486
483        // 487
484        // 488
485        // 489
486        // 490
487        // 491
488        // 492
489        // 493
490        // 494
491        // 495
492        // 496
493        // 497
494        // 498
495        // 499
496        // 500
497        // 501
498        // 502
499        // 503
500        // 504
501        // 505
502        // 506
503        // 507
504        // 508
505        // 509
506        // 510
507        // 511
508        // 512
509        // 513
510        // 514
511        // 515
512        // 516
513        // 517
514        // 518
515        // 519
516        // 520
517        // 521
518        // 522
519        // 523
520        // 524
521        // 525
522        // 526
523        // 527
524        // 528
525        // 529
526        // 530
527        // 531
528        // 532
529        // 533
530        // 534
531        // 535
532        // 536
533        // 537
534        // 538
535        // 539
536        // 540
537        // 541
538        // 542
539        // 543
540        // 544
541        // 545
542        // 546
543        // 547
544        // 548
545        // 549
546        // 550
547        // 551
548        // 552
549        // 553
550        // 554
551        // 555
552        // 556
553        // 557
554        // 558
555        // 559
556        // 560
557        // 561
558        // 562
559        // 563
560        // 564
561        // 565
562        // 566
563        // 567
564        // 568
565        // 569
566        // 570
567        // 571
568        // 572
569        // 573
570        // 574
571        // 575
572        // 576
573        // 577
574        // 578
575        // 579
576        // 580
577        // 581
578        // 582
579        // 583
580        // 584
581        // 585
582        // 586
583        // 587
584        // 588
585        // 589
586        // 590
587        // 591
588        // 592
589        // 593
590        // 594
591        // 595
592        // 596
593        // 597
594        // 598
595        // 599
596        // 600
597        // 601
598        // 602
599        // 603
600        // 604
601        // 605
602        // 606
603        // 607
604        // 608
605        // 609
606        // 610
607        // 611
608        // 612
609        // 613
610        // 614
611        // 615
612        // 616
613        // 617
614        // 618
615        // 619
616        // 620
617        // 621
618        // 622
619        // 623
620        // 624
621        // 625
622        // 626
623        // 627
624        // 628
625        // 629
626        // 630
627        // 631
628        // 632
629        // 633
630        // 634
631        // 635
632        // 636
633        // 637
634        // 638
635        // 639
636        // 640
637        // 641
638        // 642
639        // 643
640        // 644
641        // 645
642        // 646
643        // 647
644        // 648
645        // 649
646        // 650
647        // 651
648        // 652
649        // 653
650        // 654
651        // 655
652        // 656
653        // 657
654        // 658
655        // 659
656        // 660
657        // 661
658        // 662
659        // 663
660        // 664
661        // 665
662        // 666
663        // 667
664        // 668
665        // 669
666        // 670
667        // 671
668        // 672
669        // 673
670        // 674
671        // 675
672        // 676
673        // 677
674        // 678
675        // 679
676        // 680
677        // 681
678        // 682
679        // 683
680        // 684
681        // 685
682        // 686
683        // 687
684        // 688
685        // 689
686        // 690
687        // 691
688        // 692
689        // 693
690        // 694
691        // 695
692        // 696
693        // 697
694        // 698
695        // 699
696        // 700
697        // 701
698        // 702
699        // 703
700        // 704
701        // 705
702        // 706
703        // 707
704        // 708
705        // 709
706        // 710
707        // 711
708        // 712
709        // 713
710        // 714
711        // 715
712        // 716
713        // 717
714        // 718
715        // 719
716        // 720
717        // 721
718        // 722
719        // 723
720        // 724
721        // 725
722        // 726
723        // 727
724        // 728
725        // 729
726        // 730
727        // 731
728        // 732
729        // 733
730        // 734
731        // 735
732        // 736
733        // 737
734        // 738
735        // 739
736        // 740
737        // 741
738        // 742
739        // 743
740        // 744
741        // 745
742        // 746
743        // 747
744        // 748
745        // 749
746        // 750
747        // 751
748        // 752
749        // 753
750        // 754
751        // 755
752        // 756
753        // 757
754        // 758
755        // 759
756        // 760
757        // 761
758        // 762
759        // 763
760        // 764
761        // 765
762        // 766
763        // 767
764        // 768
765        // 769
766        // 770
767        // 771
768        // 772
769        // 773
770        // 774
771        // 775
772        // 776
773        // 777
774        // 778
775        // 779
776        // 780
777        // 781
778        // 782
779        // 783
780        // 784
781        // 785
782        // 786
783        // 787
784        // 788
785        // 789
786        // 790
787        // 791
788        // 792
789        // 793
790        // 794
791        // 795
792        // 796
793        // 797
794        // 798
795        // 799
796        // 800
797        // 801
798        // 802
799        // 803
800        // 804
801        // 805
802        // 806
803        // 807
804        // 808
805        // 809
806        // 810
807        // 811
808        // 812
809        // 813
810        // 814
811        // 815
812        // 816
813        // 817
814        // 818
815        // 819
816        // 820
817        // 821
818        // 822
819        // 823
820        // 824
821        // 825
822        // 826
823        // 827
824        // 828
825        // 829
826        // 830
827        // 831
828        // 832
829        // 833
830        // 834
831        // 835
832        // 836
833        // 837
834        // 838
835        // 839
836        // 840
837        // 841
838        // 842
839        // 843
840        // 844
841        // 845
842        // 846
843        // 847
844        // 848
845        // 849
846        // 850
847        // 851
848        // 852
849        // 853
850        // 854
851        // 855
852        // 856
853        // 857
854        // 858
855        // 859
856        // 860
857        // 861
858        // 862
859        // 863
860        // 864
861        // 865
862        // 866
863        // 867
864        // 868
865        // 869
866        // 870
867        // 871
868        // 872
869        // 873
870        // 874
871        // 875
872        // 876
873        // 877
874        // 878
875        // 879
876        // 880
877        // 881
878        // 882
879        // 883
880        // 884
881        // 885
882        // 886
883        // 887
884        // 888
885        // 889
886        // 890
887        // 891
888        // 892
889        // 893
890        // 894
891        // 895
892        // 896
893        // 897
894        // 898
895        // 899
896        // 900
897        // 901
898        // 902
899        // 903
900        // 904
901        // 905
902        // 906
903        // 907
904        // 908
905        // 909
906        // 910
907        // 911
908        // 912
909        // 913
910        // 914
911        // 915
912        // 916
913        // 917
914        // 918
915        // 919
916        // 920
917        // 921
918        // 922
919        // 923
920        // 924
921        // 925
922        // 926
923        // 927
924        // 928
925        // 929
926        // 930
927        // 931
928        // 932
929        // 933
930        // 934
931        // 935
932        // 936
933        // 937
934        // 938
935        // 939
936        // 940
937        // 941
938        // 942
939        // 943
940        // 944
941        // 945
942        // 946
943        // 947
944        // 948
945        // 949
946        // 950
947        // 951
948        // 952
949        // 953
950        // 954
951        // 955
952        // 956
953        // 957
954        // 958
955        // 959
956        // 960
957        // 961
958        // 962
959        // 963
960        // 964
961        // 965
962        // 966
963        // 967
964        // 968
965        // 969
966        // 970
967        // 971
968        // 972
969        // 973
970        // 974
971        // 975
972        // 976
973        // 977
974        // 978
975        // 979
976        // 980
977        // 981
978        // 982
979        // 983
980        // 984
981        // 985
982        // 986
983        // 987
984        // 988
985        // 989
986        // 990
987        // 991
988        // 992
989        // 993
990        // 994
991        // 995
992        // 996
993        // 997
994        // 998
995        // 999
996        // 1000
997        // 1001
998        // 1002
999        // 1003
1000        // 1004
1001        // 1005
1002        // 1006
1003        // 1007
1004        // 1008
1005        // 1009
1006        // 1010
1007        // 1011
1008        // 1012
1009        // 1013
1010        // 1014
1011        // 1015
1012        // 1016
1013        // 1017
1014        // 1018
1015        // 1019
1016        // 1020
1017        // 1021
1018        // 1022
1019        // 1023
1020        // 1024
1021        // 1025
1022        // 1026
1023        // 1027
1024        // 1028
1025        // 1029
1026        // 1030
1027        // 1031
1028        // 1032
1029        // 1033
1030        // 1034
1031        // 1035
1032        // 1036
1033        // 1037
1034        // 1038
1035        // 1039
1036        // 1040
1037        // 1041
1038        // 1042
1039        // 1043
1040        // 1044
1041        // 1045
1042        // 1046
1043        // 1047
1044        // 1048
1045        // 1049
1046        // 1050
1047        // 1051
1048        // 1052
1049        // 1053
1050        // 1054
1051        // 1055
1052        // 1056
1053        // 1057
1054        // 1058
1055        // 1059
1056        // 1060
1057        // 1061
1058        // 1062
1059        // 1063
1060        // 1064
1061        // 1065
1062        // 1066
1063        // 1067
1064        // 1068
1065        // 1069
1066        // 1070
1067        // 1071
1068        // 1072
1069        // 1073
1070        // 1074
1071        // 1075
1072        // 1076
1073        // 1077
1074        // 1078
1075        // 1079
1076        // 1080
1077        // 1081
1078        // 1082
1079        // 1083
1080        // 1084
1081        // 1085
1082        // 1086
1083        // 1087
1084        // 1088
1085        // 1089
1086        // 1090
1087        // 1091
1088        // 1092
1089        // 1093
1090        // 1094
1091        // 1095
1092        // 1096
1093        // 1097
1094        // 1098
1095        // 1099
1096        // 1100
1097        // 1101
1098        // 1102
1099        // 1103
1100        // 1104
1101        // 1105
1102        // 1106
1103        // 1107
1104        // 1108
1105        // 1109
1106        // 1110
1107        // 1111
1108        // 1112
1109        // 1113
1110        // 1114
1111        // 1115
1112        // 1116
1113        // 1117
1114        // 1118
1115        // 1119
1116        // 1120
1117        // 1121
1118        // 1122
1119        // 1123
1120        // 1124
1121        // 1125
1122        // 1126
1123        // 1127
1124        // 1128
1125        // 1129
1126        // 1130
1127        // 1131
1128        // 1132
1129        // 1133
1130        // 1134
1131        // 1135
1132        // 1136
1133        // 1137
1134        // 1138
1135        // 1139
1136        // 1140
1137        // 1141
1138        // 1142
1139        // 1143
1140        // 1144
1141        // 1145
1142        // 1146
1143        // 1147
1144        // 1148
1145        // 1149
1146        // 1150
1147        // 1151
1148        // 1152
1149        // 1153
1150        // 1154
1151        // 1155
1152        // 1156
1153        // 1157
1154        // 1158
1155        // 1159
1156        // 1160
1157        // 1161
1158        // 1162
1159        // 1163
1160        // 1164
1161        // 1165
1162        // 1166
1163        // 1167
1164        // 1168
1165        // 1169
1166        // 1170
1167        // 1171
1168        // 1172
1169        // 1173
1170        // 1174
1171        // 1175
1172        // 1176
1173        // 1177
1174        // 1178
1175        // 1179
1176        // 1180
1177        // 1181
1178        // 1182
1179        // 1183
1180        // 1184
1181        // 1185
1182        // 1186
1183        // 1187
1184        // 1
```

```
17         }  
18     }  
19 };
```

## 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 $O(n\log n)$  空间 $O(\log n)$

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

```
1  int singleNumber(vector<int>& nums) {  
2      int ret = 0;  
3      for (auto e: nums) ret ^= e;  
4      return ret;  
5  }
```

## 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

```
1  ListNode* dummyHead;  
2  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我开始的思路一样，省空间的话，记录差值 \*

还有人用链表实现...

## 169.多数元素

找到大于整体一半数量的元素，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  $a \rightarrow next \rightarrow next = a$ ,  $a \rightarrow next = nullptr$ , 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

```
1 struct TreeNode {
2     bool isEnd;
3     TreeNode* next[26];
4     TreeNode() : isEnd(false), next() {}
5 };
6 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  $n \log n$  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，

```
1 public void moveZeroes(int[] nums) {
2     for(int i = 0, count = 0; i < nums.length; i++){
3         if(nums[i] != 0){
4             if (count != i) {
5                 nums[count] = nums[i];
6                 nums[i] = 0;
7             }
8             count++;
9         }
10    }
11 }
```

## 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

```
1 store += to_string(tree->val) + ",";
2 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 $O(n^2)$ 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  $\sqrt{\quad}$  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

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

```
1  /* 滑动窗口算法框架 */
2  void slidingWindow(string s, string t) {
3      unordered_map<char, int> need, window;
4      for (char c : t) need[c]++;
5
6      int left = 0, right = 0;
7      int valid = 0;
8      while (right < s.size()) {
```

```

9      // c 是将移入窗口的字符
10     char c = s[right];
11     // 右移窗口
12     right++;
13     // 进行窗口内数据的一系列更新
14     ...
15
16     /** debug 输出的位置 */
17     printf("window: [%d, %d)\n", left, right);
18     /** */
19
20     // 判断左侧窗口是否要收缩
21     while (window needs shrink) {
22         // d 是将移出窗口的字符
23         char d = s[left];
24         // 左移窗口
25         left++;
26         // 进行窗口内数据的一系列更新
27         ...
28     }
29 }
30 }

```

## 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  $O(n)$

```

1  for (auto& num : nums) {
2      int x = (num - 1) % n;
3      nums[x] += n;
4  }

```

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](#)

```
1  int subarraySum(vector<int>& nums, int k) {
2      unordered_map<int, int> hash;
3      hash[0] = 1;
4      int preSum = 0, count = 0;
5      for (int i = 0; i < nums.size(); i++) {
6          preSum += nums[i];
7          count += hash[preSum - k];
8          hash[preSum]++;
9      }
10     return count;
11 }
```

## 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

```
1  class Solution {
2      public TreeNode mergeTrees(TreeNode t1, TreeNode t2) {
3          if (t1 == null) {
4              return t2;
5          }
6          if (t2 == null) {
```

```

7         return t1;
8     }
9     TreeNode merged = new TreeNode(t1.val + t2.val);
10    merged.left = mergeTrees(t1.left, t2.left);
11    merged.right = mergeTrees(t1.right, t2.right);
12    return merged;
13 }
14 }

```

## 621. 任务调度器

额，桶思想

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

当任务总数不超过  $(n+1) \times (\max-1) + \text{tot}$  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

```

1 for l in range(1, n):
2     for i in range(n - l):
3         j = l + i
4         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

```
1  vector<int> dailyTemperatures(vector<int>& temperatures) {
2      vector<int> res(temperatures.size());
3      stack<int> asc;
4      for (int i = 0; i < temperatures.size(); i++) {
5          while (!asc.empty() && temperatures[i] >
temperatures[asc.top()])
6              {
7                  int x = asc.top();
8                  asc.pop();
9                  res[x] = i - x;
10             }
11             asc.push(i);
12         }
13         return res;
14     }
```