PDF

- C++
- go
- Java

1

1 num[i] ^ num[j]

i - 1 i

0 1 1 0

(1) dfs

dfs nums i xorSum

i == nums.length

(2)

n nums 1 0
nums n 2 ^ n
1 nums

1

I fast slow

(fast slow fast slow)

2

head slow

a + b a ( ) b f fast s slow

f=2s 2 2
f = s + nb (fast n

slow s = nb
head a + nb slow nb slow a

head slow a

for for
O(n^2) O(n)

+1

cur pre
cur.next cur.next
cur ==null

`ListNode dummy= new ListNode()`
`dummy.next = head;`

---

key

1

H(key) = a * key + b a b

2

m p

H(key) = key % p, p < m

3

4

1

2

3

12,22,32,...,n2

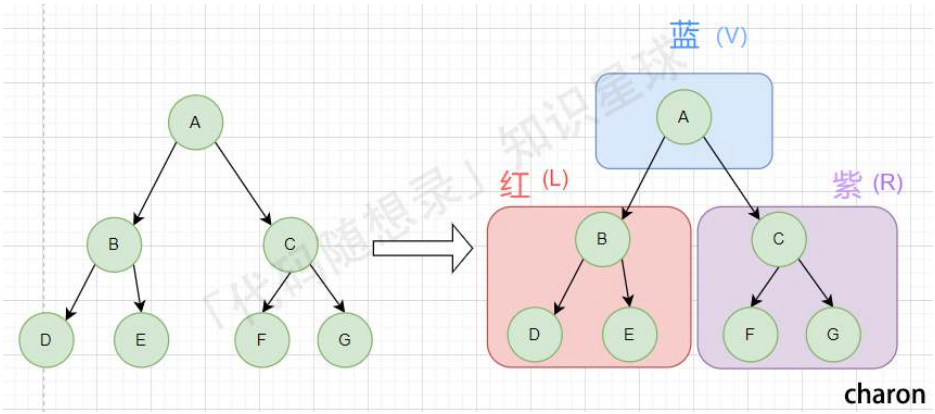
4

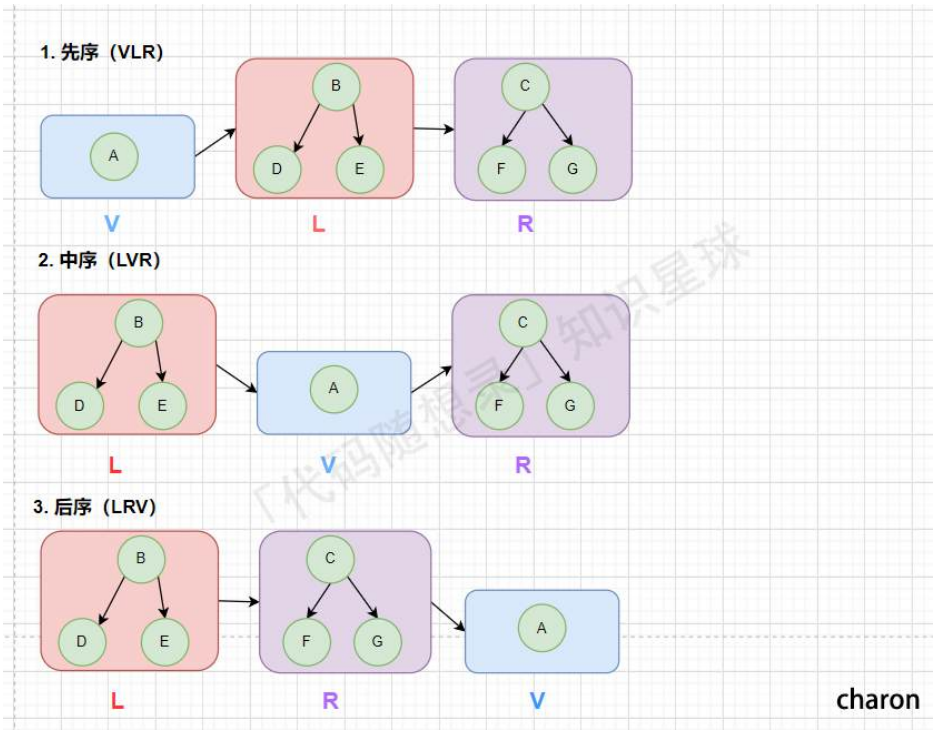
s1 s2 s1 s2 1 m-1 m s2

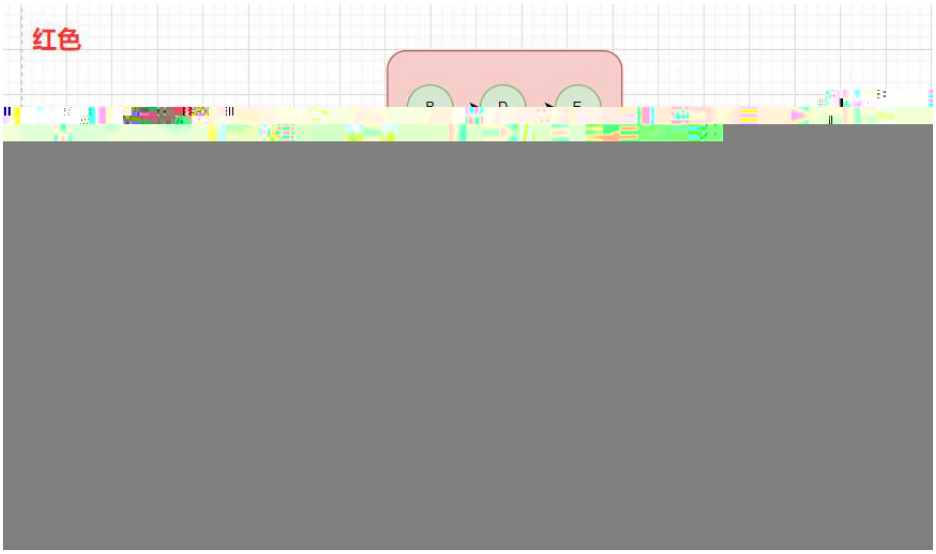
5

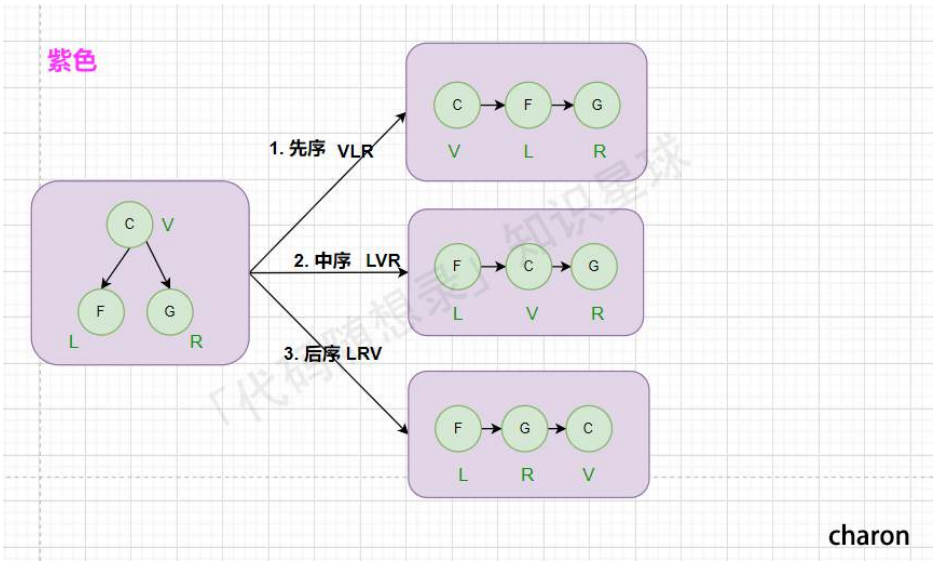
head 8 6

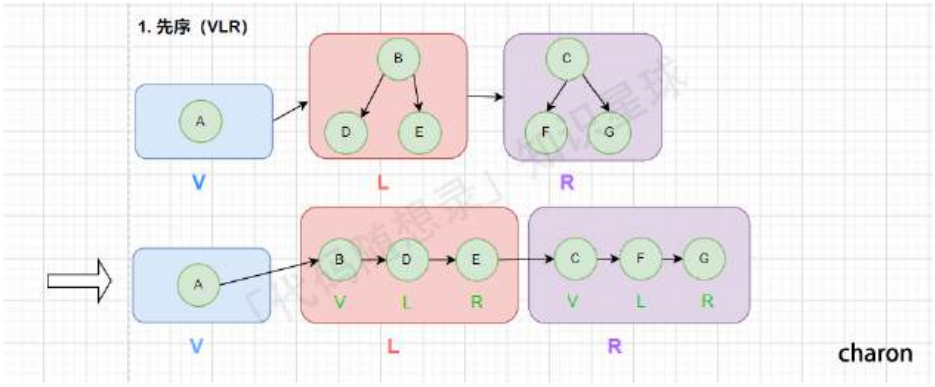
VLR LVR LRV

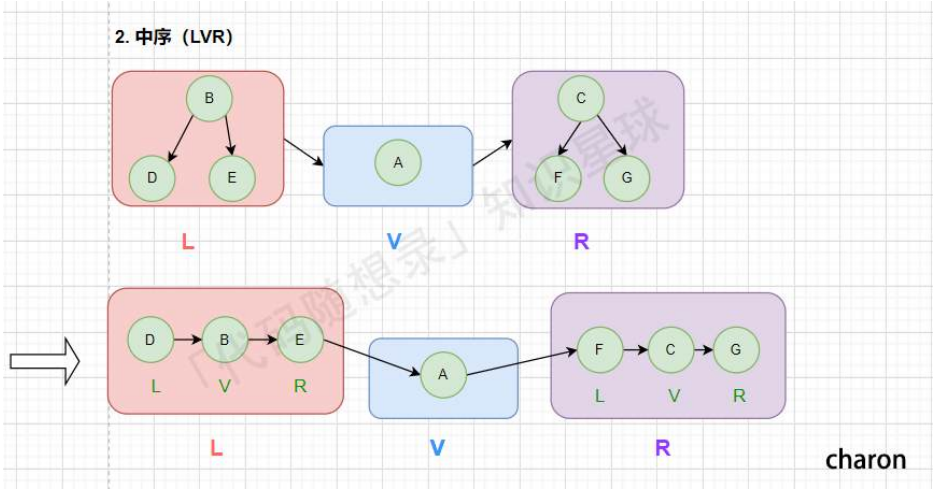

蓝 (V)
红 (L)
紫 (R)
A
B
C
D
E
F
G
charon

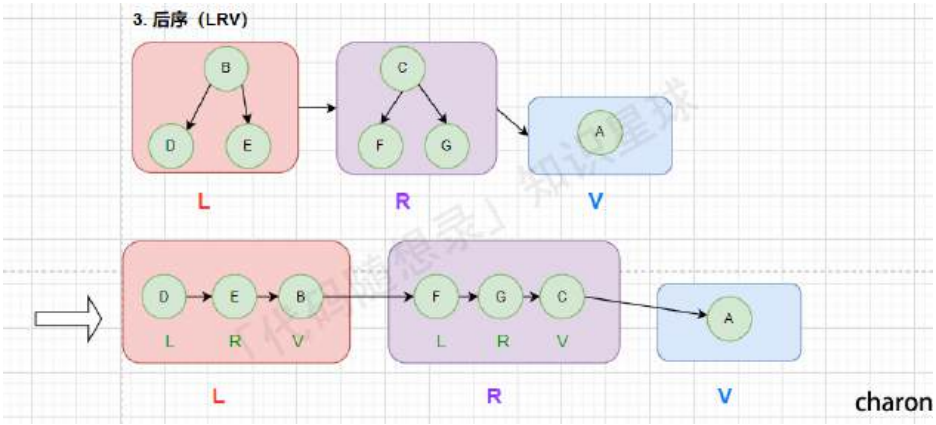

1. 先序（VLR）
V
L
R
2. 中序（LVR）
L
V
R
3. 后序（LRV）
L
R
V
charon


红色


紫色
1. 先序 VLR
2. 中序 LVR
3. 后序 LRV
C
V
F
G
L
R
charon


1. 先序（VLR）
A
B
C
D
E
F
G
V
L
R
charon





AVL

AVL

1.
2. ;
3. NULL NULL ;
4.
5. NULL ;

AVL

AVL

AVL

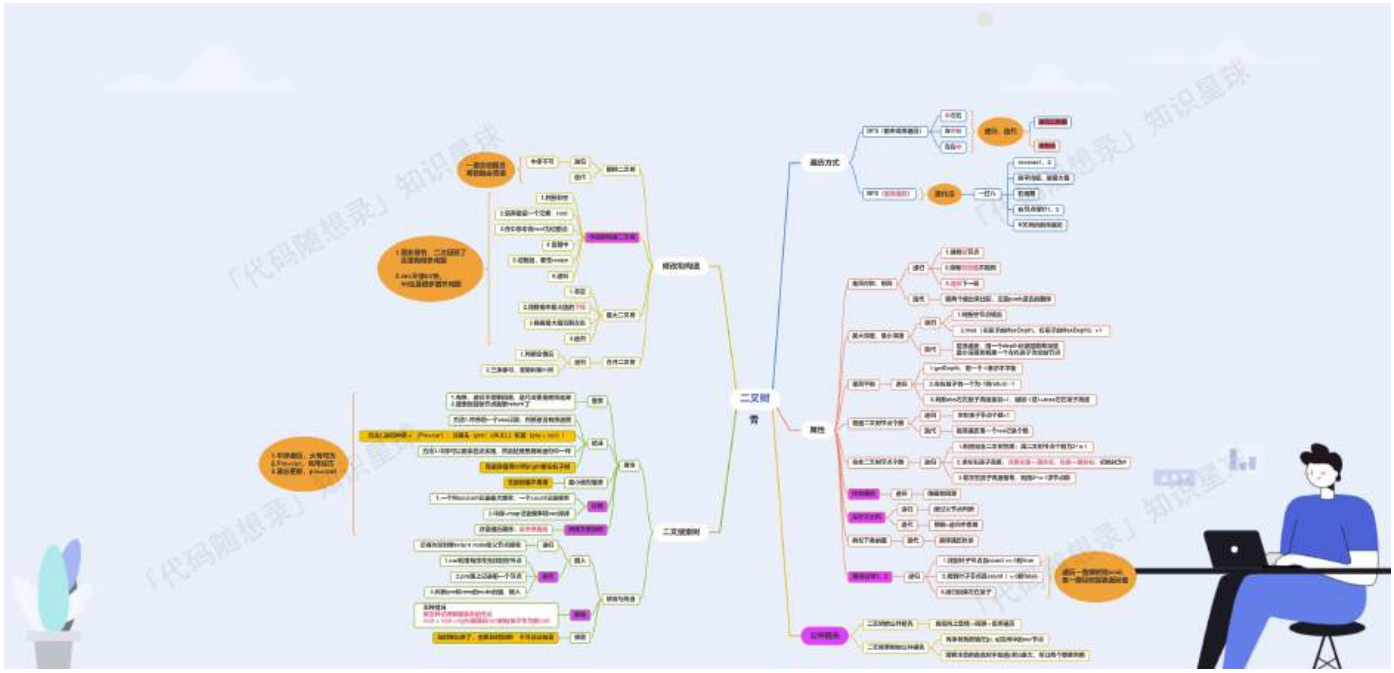
1

AVL

AVL rebalance

O(logn) STL set map

1.
2.
3.



t106.

nums

1. ,return null
2. index = left + right-left / 2
3. TreeNode root = new TreeNode(nums[index]);

> nums return root

4. [0, index) [index + 1, len)
5.

> [ ) [ ] [ ]

/ /

1. / BST
2. sum sum
3. ->
4. ->
5. ->

> / /

1.

1. startIndex
2. i

i

2. +

```
if(sum > target) return;
// 1.
sum + candidates[i] <= tartget;
// 2.
```

3.

`s.substr(startIndex,i-startIndex+1);`

4.

true

`vector<bool> used(candidates.size(),false);`

vector

-

  1. vector used(candidates.size(),false);
  2.
     - `used[i - 1] ==true` ;
     - `used[i - 1] == false` `candidates[i - 1]` ;
     - `used[i] =true` `used[i] = false`;
  3. ;
  4. if(i>0&&candidates[i-1] == candidates[i] && used[i-1] ==false) continue;

---

0 1

num1 num0 odd1 odd0

num1 num0 1 -1

1010... 0101...

1 1 0 (odd0)

0 0 1 (odd1)

010... 101....

1 num1 > num0 odd0

0 num0 > num1 odd1

---

dp

1. dp dp table
2.
3. dp
4.
5. dp

- dp case dp index ....
  - 509. if (n <= 1) return n;
  - n==1 dp[1] ...

dp debug

dp dp

> dp dp 2 3 4
> dp

dp dp

DP

509.

> n

dp

1. dp dp table
2. int[] dp = new int[n + 1];
3. dp[i] i fib
4. dp[i] = dp[i- 1] + dp[i - 2]
5. dp[0] = 0; dp[1] = 1;
6.
7. dp 9 dp 0 1 1 2 3 5 8 13 21

70.

> n
> 1 2

509.

746.

dp

1. dp[i] i
2. : dp[i] = Math.min(dp[i - 1] + cost[i - 1], dp[i - 2] + cost[i - 2]);
3. dp[0] = dp[1] = 0;
4.
5.
6. cost {1, 100, 1, 1, 1, 100, 1, 1, 100, 1}
7. dp {0, 0, 1, 2, 2, 3, 3, 4, 4, 5}

return dp[len - 1] dp[n-1] .

dp[n-1] dp[n-2]

return Math.min(dp[len - 1] + cost[len - 1], cost[len - 2] + dp[len - 2]);

62. _ . -

:

> m x n “Start”
>
> “Finish”

dp

1. dp dp[i][j]
2. int dp = new intm;
3. dp[i][j] (i,j)
4. dp[i][j]= dp[i][j]- 1 + dp[i][j];
5.
6.
7. 1
8. if (i == 0 || j == 0) dp[i][j]= 1;
9.
10. dp

63. _ . - II

> m x n “Start”
>
> “Finish”

62.

1. dp[i][j] = 0
2. dp0=0
3. carl
4. obstacleGridi == 1 dpi 1
5. for (int i = 0; i < m && obstacleGrid[i][i] == 0; i++) dp[i][j] = 1;

343. - -

> n

dp

1. dp dp[i]
    1. int[] dp = new int[n + 1];
    2. dp[i] i

2. dp[i] = Math.max(dp[i], Math.max(j *(i - j)*, *j* dp[i - j]));
    1. dp[i]
        1. j * (i - j)
        2. j * dp[i - j]
    2. dp[i]
    3. dp[2] = 1;
        1. dp[0] dp[1]
    4.
    5. dp

96. _ . _____

> n n 1 n

dp

1. dp[i] 1 i BST
2. dp[i] += dp[j - 1] * dp[i - j];
3.
    1. dp[0] = 1;
    2. dp[1] = 1;
    3. dp[2] = 2;
4.
5. dp

322. _ _ .

> coins amount
> -1

518. II 518. II

1.
   - dp[j] = Math.min(dp[j], dp[j - coins[i]] + 1);
   - 518. II dp[j] += dp[j - coins[i]]; //
2.
   - dp[0] = 0;
   - 518. II dp[0] = 1;
3.
   -
   - 518. II for for

> dp[j - coins[i]] if (dp[j - coins[i]] == INT_MAX) continue;

dp

1. dp[i] i
2. dp[j] = Math.min(dp[j], dp[j - coins[i]] + 1);
3.
   - dp[0]=0;
   -
4.
   -
   - 01 dp
5. dp

> dp[j - coins[i]] , if (dp[j - coins[i]] == INT_MAX) continue;

279. _ _ _ (_ _)

> n 1, 4, 9, 16, ... n
>
> n n
>
> 1
>
> 4 9 16 3 11

322.

1. dp[j] = Math.min(dp[j], dp[j - i * i] + 1);
2. dp if (j >= i * i && dp[j - i * i] != Integer.MAX_VALUE)
    -
    -

dp

1. dp[i] i
2. dp[j] = Math.min(dp[j], dp[j - i * i] + 1);
3. 322.
    - dp[0]=0;
    -
4.
    -
    - for (int i = 1; i < dp.length; i++) //
        - for (int j = i * i; j < dp.length; j++) //
5. dp

---

1 2 ... m

> m steps

1. 1 ..m
2.
3.
    1.
    2. (1,2) 2,1

dp

1. dp[i] i
2. dp[j] += dp[j - i];
3. dp[0] = 1;
    - 0
4.
5. dp

(01)

01

01

1. / /
   -
     - / dp[j] = max or min(dp[j], dp[j - weight[i]] + value[i]);
     - / dp[j] = Math.max or min(dp[j], dp[j - coins[i]] + 1);
   -
     - dp[0]=0;
     - 0 coins dp
2.
   - dp[j] += dp[j - nums[i]];
   -
     - for for
     - for for

1. 01
   - 01 dp for
     -
   - 01 dp for
     -
2.
   - dp for

/ for

/ for

		01背包问题
遍历顺序		先正序遍历物品，后**逆序**遍历容量
题型	装满背包，求最大价值 416.分割等子集 1049.最后一块石头的重量	1．dp的初始化：vector<int> dp(bag + 1,0); 2．数量：一般是包的容量 + 1；便于取到dp[bag] 3．值：全部设置为0 4．递推式：**dp[j] = max(dp[j], dp[j - weight[i] + value[i])** 5．拓展：最后多了一步判断：是否能达到最大价值 【416】；或者均分成两份之后，剩余多的部分【1049.】
	装满背包的组合数 494.目标和	1．dp的初始化：vector<int> dp(bag + 1,0); 2．数量：一般是包的容量 + 1；便于取到dp[bag] 3．值：**dp[0] = 1**；便于递推；其他的全部初始化为0 4．递推式：**dp[j] += dp[j - nums[i]]** 5．拓展：关键是找到包的容量！！！
	装满背包的**最多**物品数目 474.一和零	1．dp的初始化：vector<int> dp(bag + 1,0); 2．数量：一般是包的容量 + 1；便于取到dp[bag] 3．值：全部设置为0 4．递推式： **dp[j] = max(dp[j],dp[j - nums[i]] + 1)**【一维包的容量】 dp[i][j] = max(dp[i][j],dp[i - num1][j - num2] + 1)【二维包的容量】

题型	遍历顺序	完全背包问题
组合数 518 零钱兑换 II	先正序遍历**物品**，后正序遍历**背包**	1．dp的初始化：vector<int> dp(bag + 1,0); 2．数量：一般是包的容量 + 1；便于取到dp[bag] 3．值：dp[0] = 1;其余的为0 4．递推式：**dp[j] += dp[j - nums[i]]** 5．**细节：两个数相加，同样考虑是否越界：dp[j] < INT_MAX - dp[j - nums[i]]**
排列数 70 爬楼梯 377 组合总和 IV	先正序遍历**背包**，后正序遍历**物品**	
求最少物品数目 279 完全平方数 322 零钱兑换	组合数和排列数的两种遍历方式均可	1．dp的初始化：vector<int> dp(bag + 1,INT_MAX); 2．数量：一般是包的容量 + 1；便于取到dp[bag] 3．值：dp[0] = 0;其余的为INT_MAX，避免覆盖最小值 4．递推式：**dp[j] = min(dp[j],dp[j - nums[i]] +1)** 5．**细节：初始化赋值了最大值，所以在求解递推式时，注意考虑越界的情况：if(dp[j -nums[i]] != INT_MAX)才能使用递推式**
包是否装满 **193.单词拆分***** **难**	均可；但是选择先遍历背包，后遍历物品【不需要额外花费空间存储】	dp[i]：表示长度为i的字符串是否能由字典中的单词组成；是bool型 初始化：dp[0] = true【便于递推】；其他的全部初始化为false 递推式：dp[j] = true && [j,i]区间的字符串在字典中，可以推出dp[i]为true

674.

> 300.
>
> ...

1. for nums[i] > nums[i - 1]
2. 300. for nums[i] > nums[j] j [0, i]

dp

1. dp[i] i 0
2.
   1. if (nums[i] > nums[i - 1]) // &
   2. dp[i] = dp[i - 1] + 1; // 2
3.
   1. 1 dp[i] 1
4.
5. dp

dp[n -1] return dp max

392.

> s t s t
>
> "ace" "abcde" "aec"
>
> ok

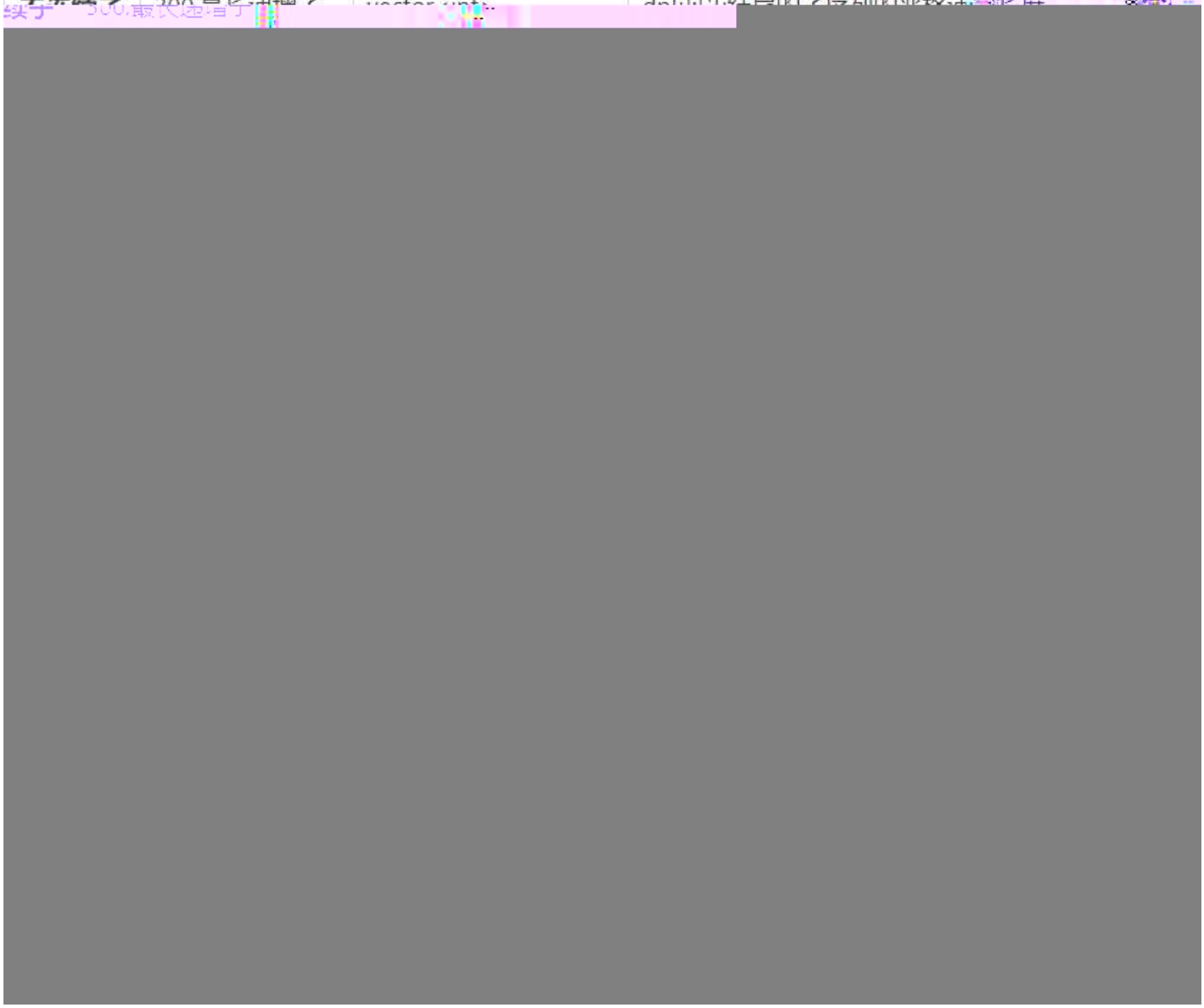
dp

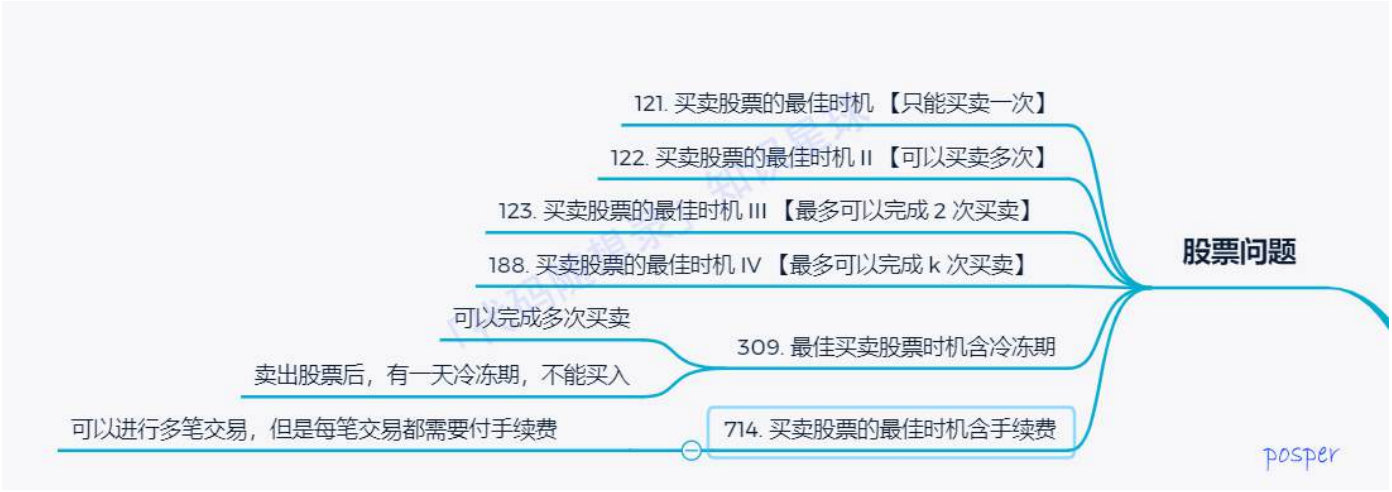
1. dpi s [0,i-1] t [0,j-1] dpi
2.
   - if (s.charAt(i - 1) == t.charAt(j - 1))
     - dpi = dpi - 1 + 1;
     - s[i-1] t[j-1] dpi = dpi - 1 + 1;
   - if (s.charAt(i - 1) != t.charAt(j - 1)) // t[j]
     - dpi = dpi;
     - s[i-1] t[i-2]
3.
   - dp0 = 0; s t [0,j-1] 0
   - dpi = 0; t s [0,i-1] 0
4.

- [i, i]
- dp[i] = 1; //
- 0

4. dp[i]
5. dp

动态规划之子序列问题		
题型	初始化	dp状态以及递推式
不连续子	300.最长递增子	vector dp...





买股票的最佳时机

注:使用二维数组 dp[][]

	限制条件	dp 数组的状态(关注 j)	初始化过程
I	买卖一次	1. 持有 dp[i][0] 2. 不持有 dp[i][1]	dp[0][0]= -prices[0] dp[0][1]= 0
II	买卖无数次		
III	买卖 2 次	1. 从未买入 dp[i][0] 2. 第一次买入 dp[i][1] 3. 第一次卖出 dp[i][2] 4. 第二次买入 dp[i][3] 5. 第二次卖出 dp[i][4]	dp[0][0]=0 dp[0][1]=-prices[0] dp[0][3]=-prices[0] dp[0][3]初始化为-price[0]的原因:当天买入第二次,可看做当天第一次买入花费 price[0]后,当天卖出又赚了 price[0],所以第二次买入花费为 price[0]
IV	买卖 k 次	1. 从未买入 dp[i][0] 2. 第 j 次买入 dp[i][2t-1] 3. 第 j 次卖出 dp[i][2t] (t 为整数)	for(int j=1;j<=2*k;j+=2) dp[0][j]=-prices[0];

含冷冻期	买卖无数次&&冷冻期	1. 持有 dp[0][0]
		2. 不持有&&当天卖出 dp[0][1]
		3. 不持有&&当天不卖出 dp[0][2]
			dp[0][0]=-prices[0];
			dp[0][1]=0;
			dp[0][2]=0;

含手续费	买卖无数次&&手续费	1. 持有 dp[i][0]
		2. 不持有 dp[i][1]
			dp[0][0]=-fee-prices[0];
			//fee 为手续费
			dp[0][1]=0;

本人

714.

122. II

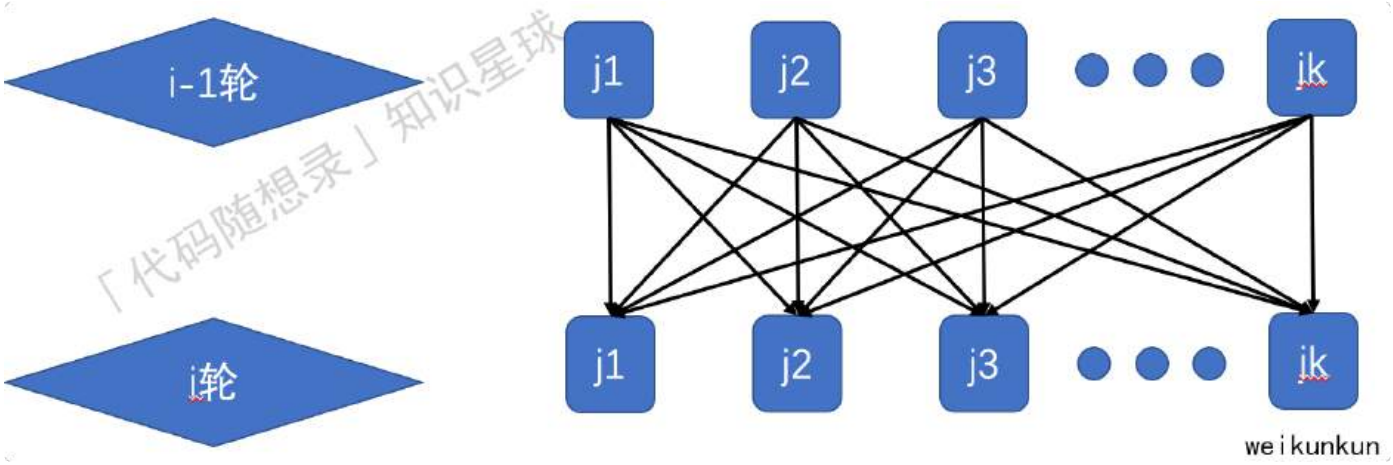
1.
2.

dp

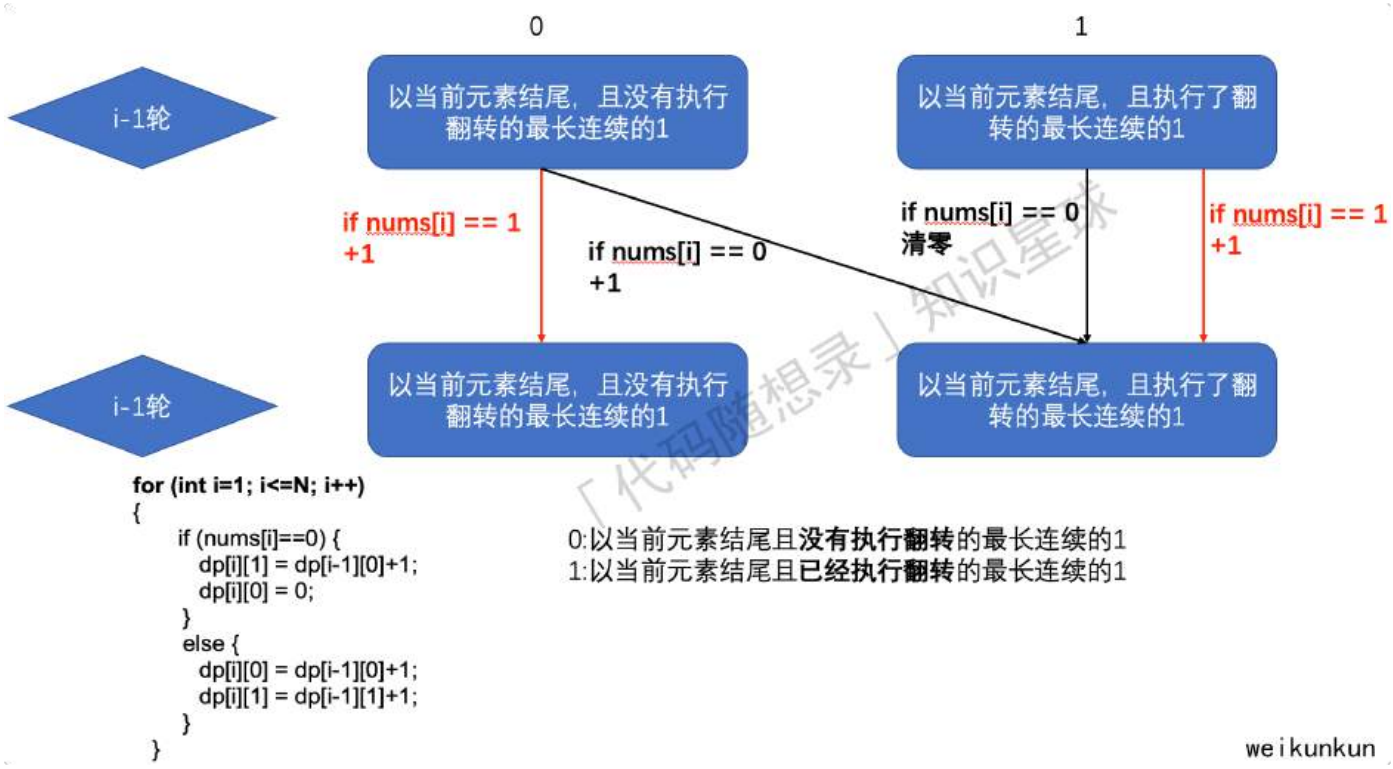
1. dp
    1. 2
        1. dp[i]: i
        2. dp[i]: i
    2. `int[] dp = new int[n];`
2.
    1. `dpi = Math.max(dpi - 1, dpi - 1 - prices[i]);`
        1. dp[i] - 1
        2. dp[i] - 1 - prices[i]
    2. max
    3. dp[i] = Math.max(dp[i] - 1, dp[i] - 1 + prices[i] - fee);
        1.
        2. dp[i] - 1
        3.
    4. max
3.
    1. `dp[0] = -prices[0];`
    2. 0
4.
5. dp

((/) “ ” “ ” “ ”
)

1. dp dp[i][j] ith j ;
2. dp[i][j] dp[i-1][j] ;
3. dp

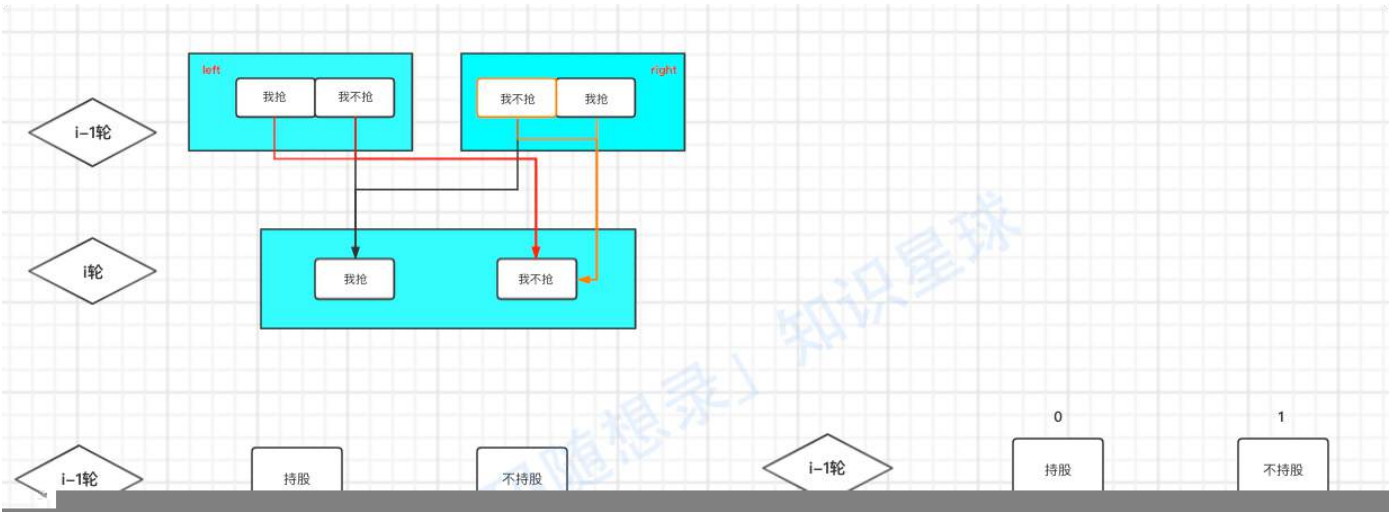


LC_487. 1 II



III DP

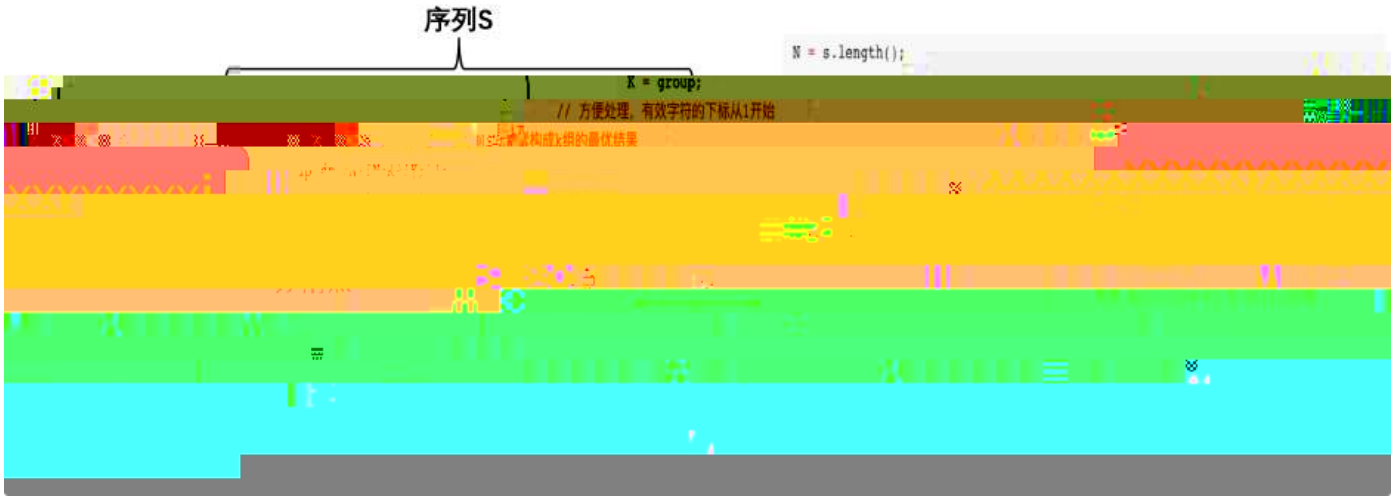
dp


left
right
我抢
我不抢
我不抢
我抢
i-1轮
i轮
我抢
我不抢

i-1轮
持股
不持股
i-1轮
0
1
持股
不持股

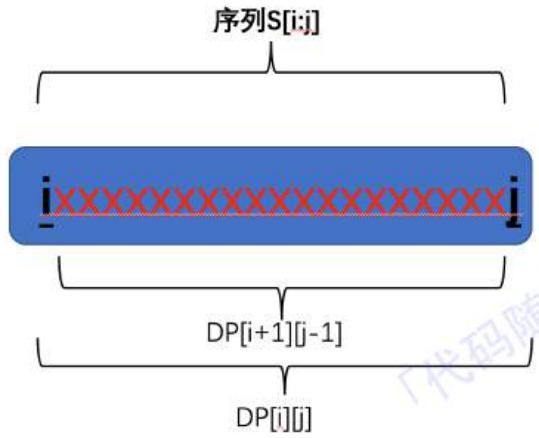
DP

K



LC813
LC1278 III
LC410
LC1335.

DP




```
初始化;
for (int len = 2; len <= N; len++) { // 先遍历区间大小
    for (int i = 0; i + len - 1 < N; i++) { // 再遍历起始位置（初始位置）
        int j = i + len - 1; // 右端点位置
        // 进行状态转移
        // 具体的状态转移还是要根据题意来
        // 下面的属于一种情况
        // 1. 在[i:j]区间找到最优的结果
        for (int k = i; k <= j; k++) {
            dp[i][j] = best();
        }
    }
}

return dp[0:N];
```

weikunkun

dp

664. Strange-Printer
1547.
1682.
1690.
1745.
1000.

664.

1. dp dp[i][j] [i, j]
2.
    1. s[i] == s[j] s[i] s[j]
       i j j dp[i][j] = dp[i][j - 1]
    2. s[i] != s[j] [i, k] [k + 1, j] k [i, j) d
       dp[i][j] = min(dp[i][k] + dp[k + 1][j])
3. 1 i == j dp[i][i] = 1
4. i j s[i] != s[j] dp[i][j] dp[i][k], dp[k + 1][j]
    1. i 0 k i dp[k + 1][j] i + 1
    2. i n - 1 k i dp[k +1][j] i + 1 i
       dp[k + 1][j]
5. j i + 1 k i dp[i][k]

BFS

1.
2.
3.

1.
2.
3. dp dp[i][j] i-1 j-1

	子序列之编辑距离		
题型		初始化	dp含义以及递推式
删除长串	392.判断子序列	根据dp的含义均初始化为0	dp[i][j]：以i-1结尾字符串和以j-1结尾的字符串，相同子序列的**长度** s[i-1] == t[j-1]： **dp[i][j] = dp[i-1][j-1] + 1** 不相等：**dp[i][j] = dp[i][j-1]**//只删除长串t 结果：最终判断dp[s.size()][t.size()] 与字符串s的长度是否相等
	115.不同子序列	t不为空时，s子串为空时：dp[i][0] = 1 t为空时，s子串不为空时：dp[0][j] = 0	dp[i][j]：以i-1为结尾的s子序列中出现以j-1为结尾的t的**个数** **s[i-1] == t[j-1]:dp[i][j] = dp[i-1][j-1] + dp[i-1][j]** **[不使用s[i-1]进行匹配！！！]** **dp[i][j] = dp[i-1][j]**//只删除长串s 结果：dp[s.size()][t.size()] 为最终不同子序列的个数

两个字符串均可删除	583.两个字符串的删除	word1不为空时，word2为空时：
		for(int i = 0;i <= word1.size();i++)
		dp[i][0] = i;
		word1为空时，word2不为空时：
		for(int j = 0;j <= word2.size();j++)
		dp[0][j] = j;
			dp[i][j]：以i-1为结尾的字符串word1，和以j-1位结尾的字符串word2，想要达到相等，所需要删除元素的**最少次数**
			相等：**dp[i][j] = dp[i-1][j-1]**
			不相等：**dp[i][j] = min(dp[i-1][j],dp[i][j-1]) + 1**

	72.编辑距离【比583.多了一步替换操作】		dp[i][j] 表示以下标i-1为结尾的字符串word1，和以下标j-1为结尾的字符串word2，最近编辑距离 相等：**dp[i][j] = dp[i-1][j-1]** 不相等：**dp[i][j] = min(dp[i-1][j-1],min(dp[i-1][j],dp[i][j-1])) + 1**
总结： dp的定义式：vector<vector<int>> dp(word1.size() + 1,vector<int>(word2.size() + 1))；增加考虑空串，便于初始化以及递推式 递推式分两步：比较word1[i-1]与word2[j-1]是否相等；①相等时的递推式② 不相等的递推式			

1.
2. dp

3.
4. dp i <= j

(1) bool
(2) int i=j
1 0

5. dp

(1) i s.size()-1 0 j i s.size()-1
(2) i s.size()-1 0 j i+1 s.size()-1 i+1 i=j

子序列之回文串问题			
题型		初始化	递推式
回文子串	647.回文子串	全部初始化为false；默认均不是	当s[i] == s[j]时； if(j-i <=1 || dp[i+1][j-1]) dp[i][j] = true//区间中要么元素个数小于等于1；要么内部区间为真 当为true的情况，作相应操作： • 统计回文子串个数 • 更新最大回文子串长度 以及对应回文子串起始位置
	5.最长回文子串可拓展求长度		
回文子序列	517.最长回文子序列	区间中只有一个元素的位置赋值为1(dp[i][i])；其余赋值为0	s[i] == s[j] dp[i][j] = dp[i+1][j-1] + 2; 不相等： dp[i][j] = max(dp[i+1][j],dp[i][j-1]);
回文子串总结： 1.布尔类型的dp[i][j]：表示区间范围[i,j] （注意是左闭右闭）的子串是否是回文子串，如果是dp[i][j]为true，否则为false; **vector<vector<bool>> dp(s.size(),vector<bool>(s.size(),false));** 2.同样分两种情况讨论：s[i] 与s[j]是否相等；但是只需要处理相等的；不相等肯定不是；			

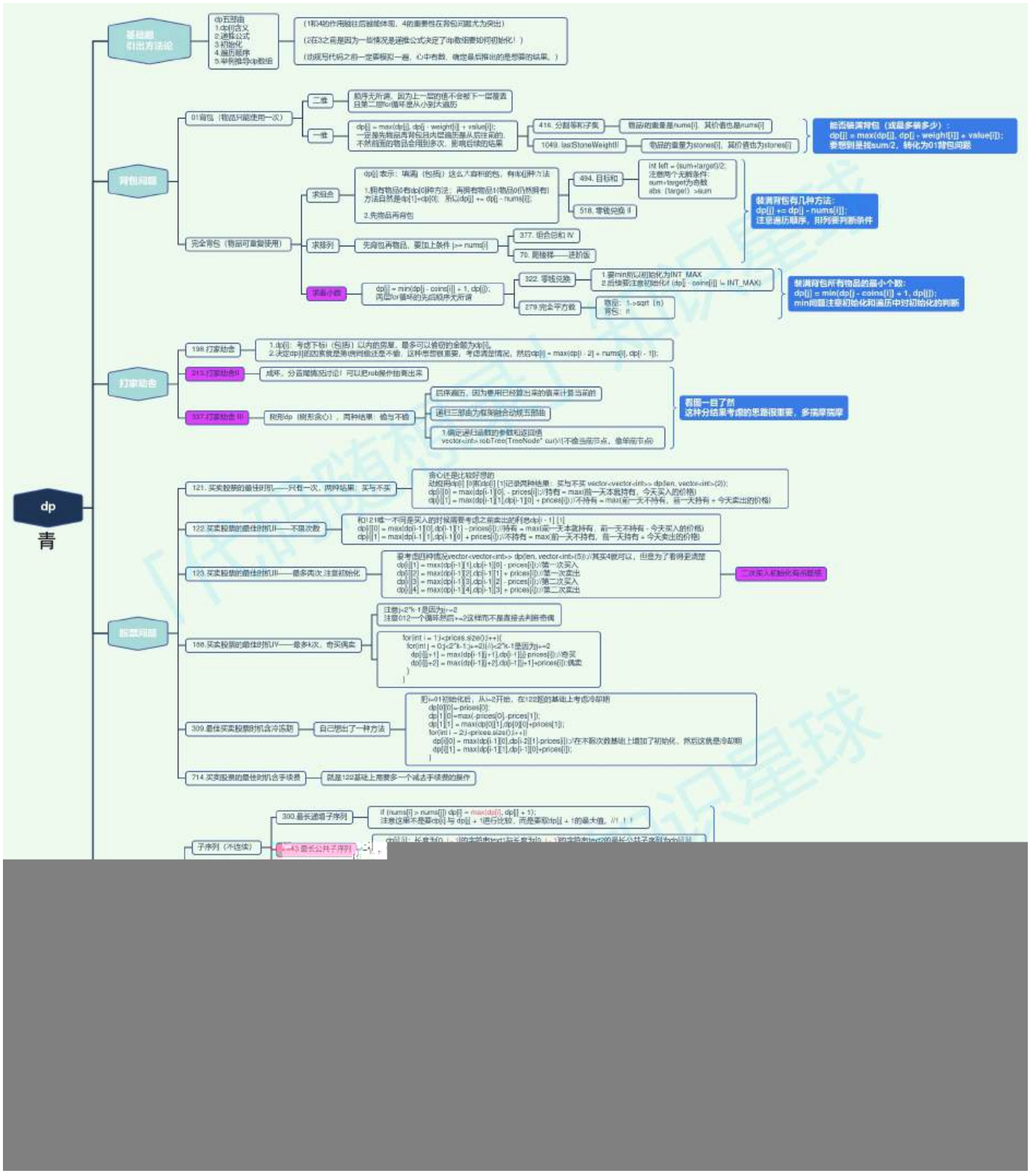
DP

动态规划各类问题的 dp 数组特点

本人

	dp 数组维度	dp 数组定义特点	解决的问题	最终结果与 dp 数组的关系
背包问题	一维 二维		求最值	最终的结果为 dp 数组下标的最大值: dp[n]或 dp[n][m]
打家劫舍	一维 二维	dp[i]:第 i 天偷窃金额的最大值，考虑 nums[i]，但不一定包含		
股票问题	二维	dp[i][j]:第 i 天 j 状态下的利润最大值，考虑 nums[i]，但不一定包含		
子序列问题	一维 二维	当遍历到 dp[i][j]或 dp[i]时，无论是字符串还是数组，必须包含 str[i]或 nums[i]。	求最值 求子串 求子序列	对于数组:最终结果为 dp 数组所有元素的最大或最值:res=max(res, dp[i])
				对于字符串:最终的结果为 dp 数组下标的最值: dp[n]或 dp[n][m]

DP


dp
青
基础篇
引出方法论
dp五部曲
1.dp[i]含义
2.递推公式
3.初始化
4.遍历顺序
5.举例推导dp数组
(1和4的作用就往后越能体现，4的重要性在背包问题尤为突出)
(2在3之前是因为一些情况是递推公式决定了dp数组要如何初始化！)
(动规写代码之前一定要模拟一遍，心中有数，确定最后推出的是想要的结果。)
背包问题
01背包（物品只能使用一次）
二维
顺序无所谓，因为上一层的值不会被下一层覆盖
且第二层for循环是从小到大遍历
一维
dp[i] = max(dp[i], dp[j - weight[i]] + value[i]);
一定是先物品再背包且内层循环是从后往前的，
不然前面的物品会用到多次，影响后续的结果
416. 分割等和子集
物品的重量是nums[i]，其价值也是nums[i]
1049. lastStoneWeightII
物品的重量为stones[i]，其价值也为stones[i]
能否装满背包（或最多装多少）：
dp[j] = max(dp[j], dp[j - weight[i]] + value[i]);
变形到装我sum/2，转化为01背包问题
完全背包（物品可重复使用）
求组合
dp[i]表示：填满（包括i）这么大容积的包，有dp[i]种方法
1.拥有物品i有dp[j]种方法，再拥有物品i+1物品j仍然拥有i方法自然是dp[i+1]+=dp[i]，所以dp[j] += dp[j - nums[i]];
2.先物品再背包
494. 目标和
int left = (sum+target)/2;
注意两个无解条件:
sum+target为奇数
abs（target）>sum
518. 零钱兑换 II
装满背包有几种方法:
dp[j] += dp[j - nums[i]];
注意遍历顺序，排列要判断条件
求排列
先背包再物品，要加上条件 j>= nums[i]
377. 组合总和 IV
70. 爬楼梯——进阶版
求最小值
dp[j] = min(dp[j - coins[i]] + 1, dp[j]);
两层for循环的先后顺序无所谓
322. 零钱兑换
1.要min所以初始化为INT_MAX
2.后续要注意初始化if (dp[j - coins[i]] != INT_MAX)
279.完全平方数
物品：1->sqrt（n）
背包：n
装满背包所有物品的最小个数:
dp[j] = min(dp[j - coins[i]] + 1, dp[j]);
min问题注意初始化和遍历中对初始化的判断
打家劫舍
198.打家劫舍
1.dp[i]：考虑下标i（包括i）以内的房屋，最多可以偷窃的金额为dp[i]。
2.决定dp[i]的因素就是第i间房偷还是不偷，这种思想很重要，考虑清楚情况，然后dp[i] = max(dp[i - 2] + nums[i], dp[i - 1]);
213.打家劫舍II
成环，分首尾情况讨论！可以把rob操作抽离出来
337.打家劫舍 III
树形dp（树形贪心），两种结果：偷与不偷
后序遍历，因为要用已经算出来的值来计算当前的
递归三部曲为框架融合动规五部曲
1.确定递归函数的参数和返回值
vector<int> robTree(TreeNode* cur)//cur当前节点，偷or不偷当前节点
看图一目了然
这种分结果考虑的思路很重要，多练早脱坑
股票问题
121. 买卖股票的最佳时机——只有一次，两种结果：买与不买
贪心还是比较好想的
动规用dp[i][0]和dp[i][1]记录两种结果：买与不买 vector<vector<int>> dp(len, vector<int>(2));
dp[i][0] = max(dp[i-1][0], - prices[i])//持有 = max(前一天本就持有，今天买入的价格)
dp[i][1] = max(dp[i-1][1],dp[i-1][0] + prices[i])//不持有 = max(前一天不持有，前一天持有 + 今天卖出的价格)
122.买卖股票的最佳时机II——不限次数
和121唯一不同是买入的时候需要考虑之前卖出的利润dp[i - 1][1]
dp[i][0] = max(dp[i-1][0],dp[i-1][1] - prices[i])//持有 = max(前一天本就持有，前一天不持有 - 今天买入的价格)
dp[i][1] = max(dp[i-1][1],dp[i-1][0] + prices[i])//不持有 = max(前一天不持有，前一天持有 + 今天卖出的价格)
123.买卖股票的最佳时机III——最多两次,注意初始化
要考虑四种情况vector<vector<int>> dp(len, vector<int>(5));//其实4就可以，但是为了看得更清楚
dp[i][1] = max(dp[i-1][1],dp[i-1][0] - prices[i]);//第一次买入
dp[i][2] = max(dp[i-1][2],dp[i-1][1] + prices[i]);//第一次卖出
dp[i][3] = max(dp[i-1][3],dp[i-1][2] - prices[i]);//第二次买入
dp[i][4] = max(dp[i-1][4],dp[i-1][3] + prices[i]);//第二次卖出
二次买入初始化有点意思
188.买卖股票的最佳时机IV——最多k次，奇买偶卖
注意k=2*k-1是因为i=2
注意012一个循环然后+=2这样而不是直接去判断奇偶
for(int i = 1;i<prices.size();i++){
for(int j = 0;j<2*k-1;j+=2){//j<2*k-1是因为i+=2
dp[i][j+1] = max(dp[i-1][j+1],dp[i-1][j] - prices[i]);//奇买
dp[i][j+2] = max(dp[i-1][j+2],dp[i-1][j+1]+prices[i]);偶卖
}
}
309.最佳买卖股票时机含冷冻期
自己想出了一种方法
把i=01初始化后，从i=2开始，在122题的基础上考虑冷却期
dp[0][0]=-prices[0];
dp[1][0]=max(-prices[0],-prices[1]);
dp[1][1] = max(dp[0][1],dp[0][0]+prices[1]);
for(int i = 2;i<prices.size();i++){
dp[i][0] = max(dp[i-1][0],dp[i-2][1]-prices[i]);//在不断次数基础上增加了初始化，然后这就是冷却期
dp[i][1] = max(dp[i-1][1],dp[i-1][0]+prices[i]);
}
714.买卖股票的最佳时机含手续费
就是122基础上需要多一个减去手续费的操作
子序列（不连续）
300.最长递增子序列
if (nums[i] > nums[j]) dp[i] = max(dp[i], dp[j] + 1);
注意这里不是要dp[i] 与 dp[j] + 1进行比较，而是要取dp[j] + 1的最大值。//！！！
1143.最长公共子序列

1

2

1.
2. K

3

1.
2.

4

topK

Prim Dijkstra

1.

1

1 2 3 4 5 5 4 3 2 1
1 2 3 4 5 ;

2

push push ;

3

2.

1

add pollall polldog pollcat ;

2

3.

1

head1 head2 ;

2

4.

1

5.

1

-- return f(n - 1) + f(n - 2), 2 n ;

-- n n ;

--F n =F n-1 +F n-2
2×2 N N
logn N
1 2 4 8...

1.

hash

1 A B 2T A B

1. N
2. A hash
   A1-AN B B1-BN
3. <A1,B1>
   hashmap

2 A,B 50 url 64Byte 4G url

1. 1000 300M
2. A URL hash
A1-AN B B1-BN A/B
3. <A1,B1> URL
hashmap

3 1000w

1.
2. hash_set
3. hash_set

2. TOP N

hash_map
N

hash hash_map

1 N

1

1. hash
2. hash_map
3. N

2

1. hash_map
2. N

2 IP 1

1. IP % 1000 1000
2. IP
3.
4. 1000 IP map
5. 1000 IP

3 1000w 10 2

4 1G 1 1 16 1M 100 2

1. % 5000 5000
2. 200K
3. 1M 200K
4. map
5. 5000

3. TOP N

1 100

1.
    1.
    2.
    3. -> map ->
    4. 100 TOP10
2.

- hash
  TOP10

Astar

1. Astar

1.
2.
3. Astar

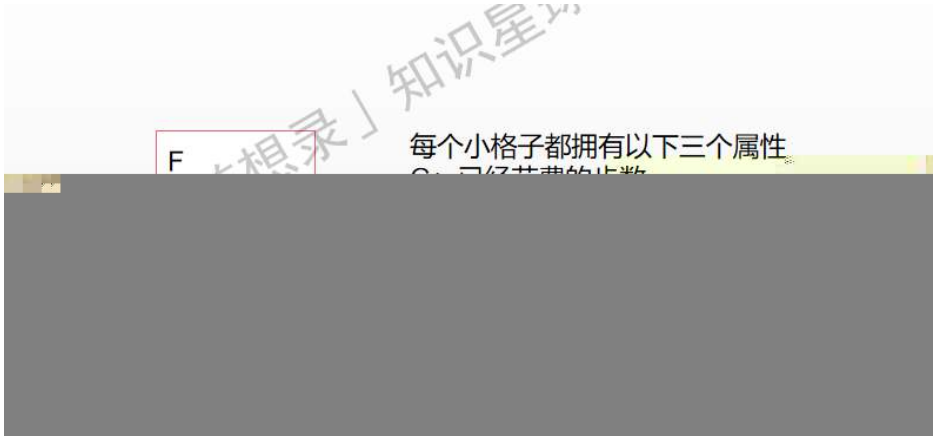
2. Astar +

1

```
1. Open
2. Close
```

2

1 F = G + H



2

1. G
2. H
3. F G H

3.

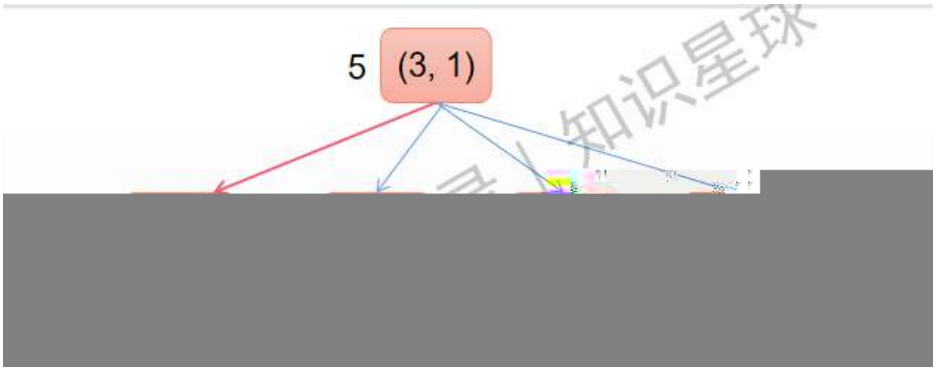
1

1. open

2. Open F Open Close
3.

Open Close F Open

4.



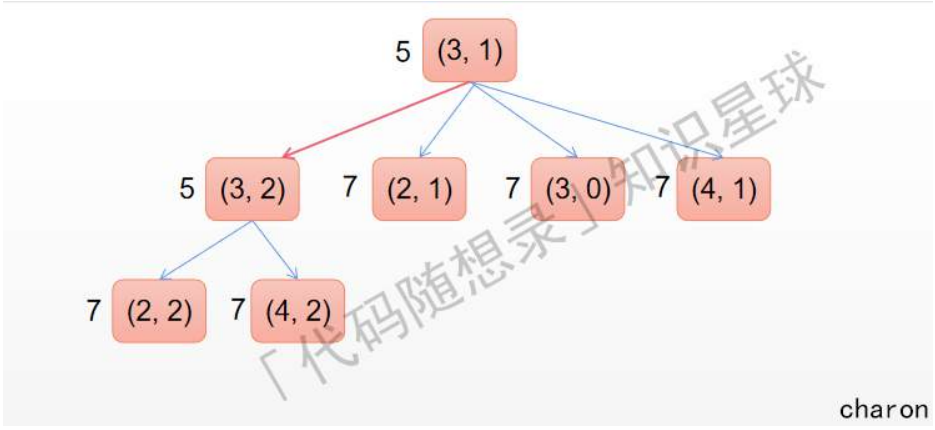
2

Open end

1. Open F Open Close

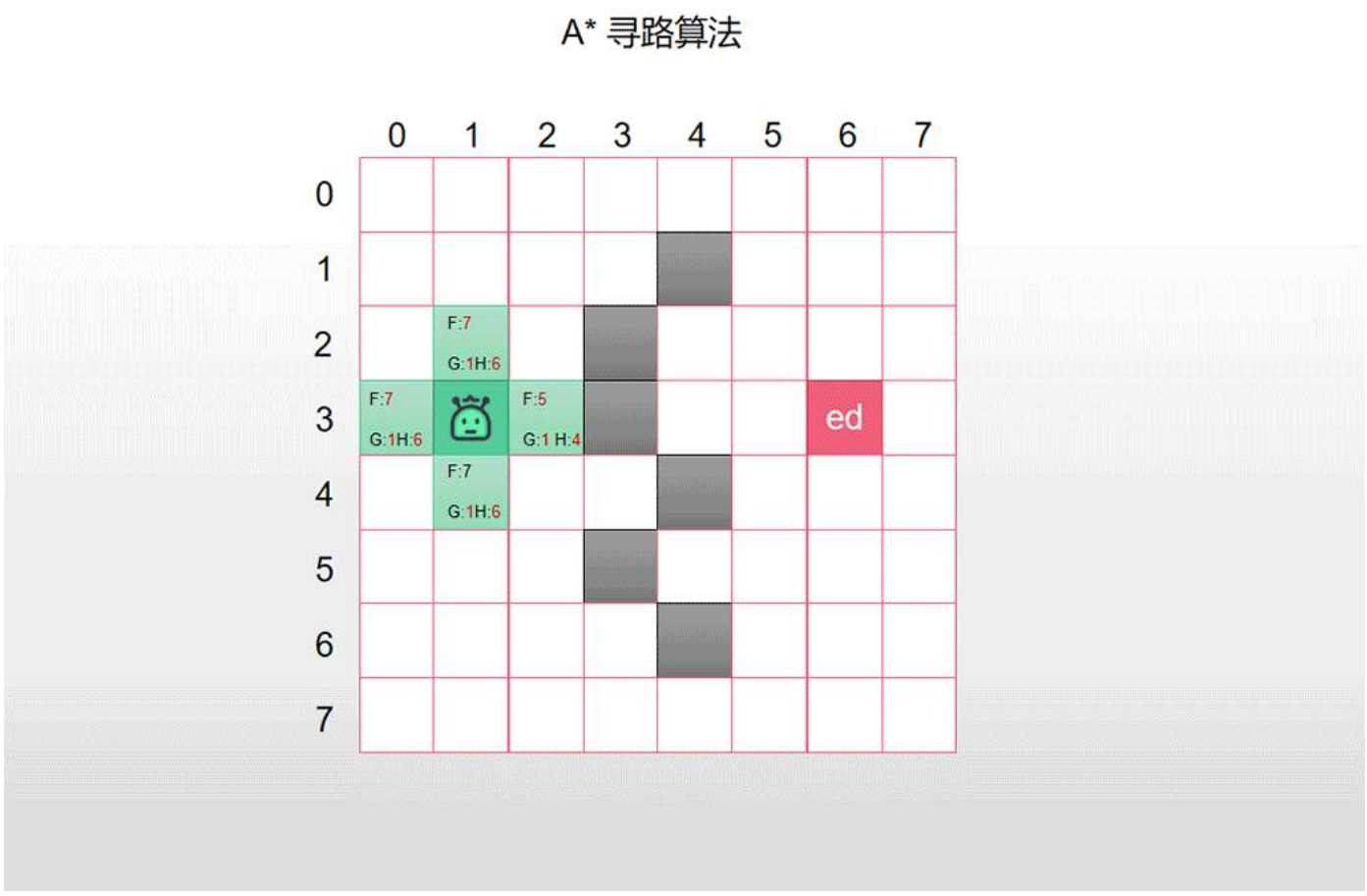
2. Open
   Close F Open

3. 3 3 3 1 Close Open

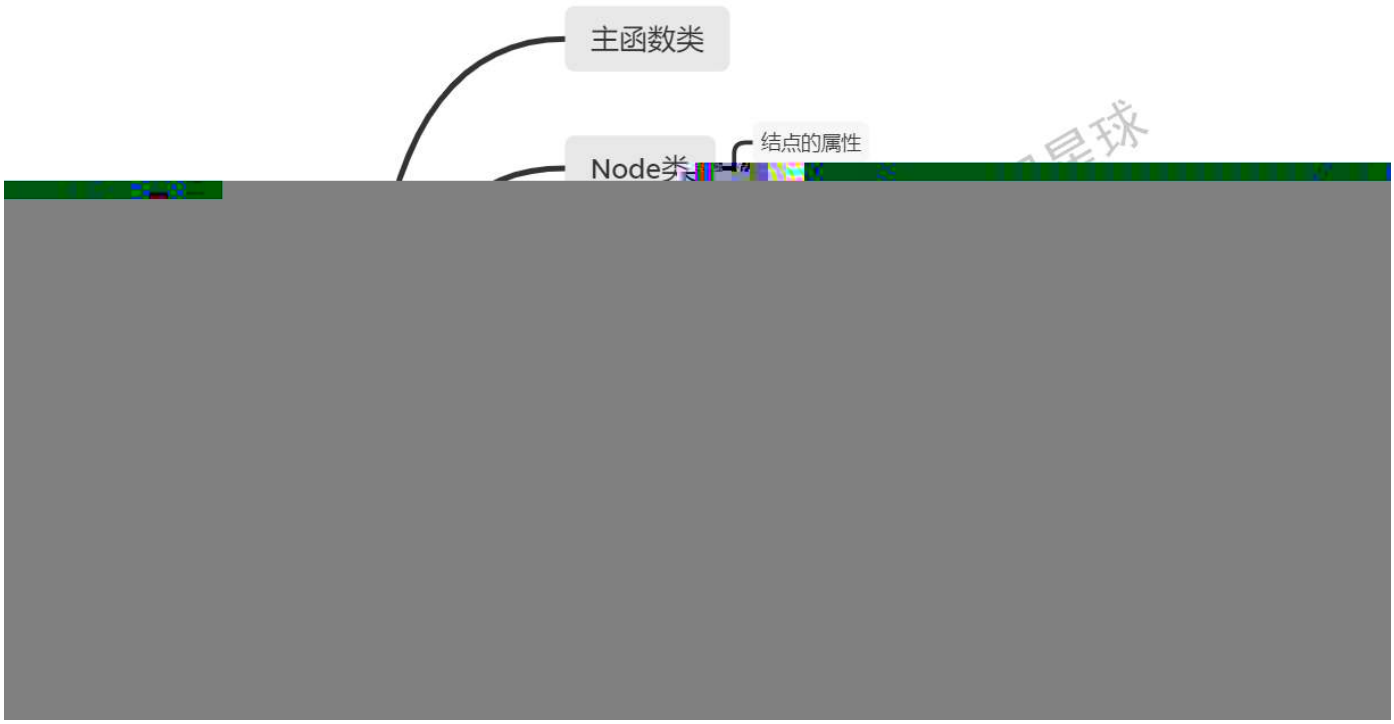


3

1. Open F Close
2. Open
   F F


A* 寻路算法
0 1 2 3 4 5 6 7
0
1
2
3
4
5
6
7
F:7
G:1H:6
F:7
G:1H:6
F:5
G:1 H:4
F:7
G:1H:6
ed

4.


主函数类
Node类
结点的属性

1. Open Open Java

2. Java Exist
   Open Close
3. Close

Node Node

```
//
//                                        Comparable
class Node implements Comparable<Node> {
public int x; //x
public int y; //y
public int F; //F
public int G; //G
public int H; //H
public Node Father;  //
//
public Node(int x, int y) {
    this.x = x;
    this.y = y;
}
//                                  F   G   H
//
public void init_node(Node father, Node end) {
    this.Father = father;
    if (this.Father != null) {
        //
        this.G = father.G + 1;
    } else { //
        this.G = 0;
    }
    //                                      H
    this.H = Math.abs(this.x - end.x) + Math.abs(this.y - end.y);
    this.F = this.G + this.H;
}
    //              Node
    @Override
    public int compareTo(Node o) {
        return Integer.compare(this.F, o.F);
    }
}
```

ps: public

Solution

```
// -1 ->       1 ->     2 ->
public int[][] map = {
   {-1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
   {-1, 0, 0, 0, 0, 0, 0, 0, 0, -1},
   {-1, 0, 0, 0, 0, -1, 0, 0, 0, -1},
   {-1, 0, 0, 0, -1, 0, 0, 0, 0, -1},
   {-1, 0, 1, 0, -1, 0, 0, 2, 0, -1},
   {-1, 0, 0, 0, 0, -1, 0, 0, 0, -1},
   {-1, 0, 0, 0, -1, 0, 0, 0, 0, -1},
   {-1, 0, 0, 0, 0, -1, 0, 0, 0, -1},
   {-1, 0, 0, 0, 0, 0, 0, 0, 0, -1},
   {-1, -1, -1, -1, -1, -1, -1, -1, -1, -1}
   };
```

map

-1 x y

Open Close Exist

Open Close

Open Exist Open

Close

```
//Open
//Open        F
public PriorityQueue<Node> Open = new PriorityQueue<Node>();
//Close
public ArrayList<Node> Close = new ArrayList<Node>();
//Exist
public ArrayList<Node> Exist = new ArrayList<Node>();
```

is_exist

```
public boolean is_exist(Node node)
{
    for (Node exist_node : Exist) {
    //           Exist           true
        if (node.x == existnode.x && node.y == existnode.y) {
        return true;
        }
    }
    //          false
    return false;
}
```

is_valid

```
public boolean is_valid(int x, int y) {
    //                    -1
    if (map[x][y] == -1) return false;
    for (Node node : Exist) {
        //
        if (is_exist(new Node(x, y))) {
            return false;
        }
    }
    //
    return true;
}
```

(extendcurrentnode)

```
public ArrayList<Node> extendcurrentnode(Node current_node) {
    //           x, y
    int x = current_node.x;
    int y = current_node.y;
    //                            neighbour_node
    ArrayList<Node> neighbour_node = new ArrayList<Node>();
    if (is_valid(x + 1, y)) {
        Node node = new Node(x + 1, y);
        neighbour_node.add(node);
    }
    if (is_valid(x - 1, y)) {
        Node node = new Node(x -1, y);
        neighbour_node.add(node);
    }
    if (is_valid(x, y + 1)) {
        Node node = new Node(x, y + 1);
        neighbour_node.add(node);
    }
    if (is_valid(x, y - 1)) {
        Node node = new Node(x, y - 1);
        neighbour_node.add(node);
    }
    //
    return neighbour_node;
}
```

Astar (astarSearch)

```
public Node astarSearch(Node start, Node end) {
//                  Open
    this.Open.add(start);
    this.Exist.add(start);
```

```
    //
    while (Open.size() > 0) {
    //                              Open
    Node current_node = Open.poll();
    //             Close
    Close.add(current_node);
    //
    ArrayList<Node> neighbournode = extendcurrentnode(currentnode);
        //
        for (Node node : neighbour_node) {
            if (node.x == end.x && node.y == end.y) {//
                //init_node
                //        G   F   H
                node.initnode(currentnode,end);
                return node;
            }
            if (!is_exist(node)) {
                //                 Open
                //        G, F, H
                node.initnode(currentnode, end);
                Open.add(node);
                Exist.add(node);
            }
        }
    }
    //                                          null
    return null;
 }
```

5.

```
import java.util.ArrayList;
import java.util.PriorityQueue;

public class Astar {
    public static void main(String[] args) {
        int[][] map = {
        {-1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
        {-1, 0, 0, 0, 0, 0, 0, 0, 0, -1},
        {-1, 0, 0, 0, 0, -1, 0, 0, 0, -1},
        {-1, 0, 0, 0, -1, 0, 0, 0, 0, -1},
        {-1, 0, 1, 0, -1, 0, 0, 2, 0, -1},
        {-1, 0, 0, 0, 0, -1, 0, 0, 0, -1},
        {-1, 0, 0, 0, -1, 0, 0, 0, 0, -1},
        {-1, 0, 0, 0, 0, -1, 0, 0, 0, -1},
        {-1, 0, 0, 0, 0, 0, 0, 0, 0, -1},
        {-1, -1, -1, -1, -1, -1, -1, -1, -1, -1}
        };
```

```
        Node start = new Node(4, 2);
        start.Father = null;
        Node end = new Node(4, 7);
        Solution solution = new Solution();
        Node res_node = solution.astarSearch(start, end);
        //
        while (res_node != null) {
            map[res_node.x][res_node.y] = 11;
            res_node = res_node.Father;//
        }

        for (int i = 0; i < 10; i++) {
            for (int j = 0; j < 10; j++) {
                System.out.printf("%3d", map[i][j]);
            }
            System.out.println();
        }
    }
}
//
class Node implements Comparable<Node> {
    public int x; //x
    public int y; //y
    public int F; //F
    public int G; //G
    public int H; //H
    public Node Father;  //
    //
    public Node(int x, int y) {
        this.x = x;
        this.y = y;
    }
//                  F   G   H
//
    public void init_node(Node father, Node end) {
        this.Father = father;
        if (this.Father != null) {
            this.G = father.G + 1;
        } else { //
            this.G = 0;
        }
        //                                  H
        this.H = Math.abs(this.x - end.x) + Math.abs(this.y - end.y);
        this.F = this.G + this.H;
    }
//              Node
    @Override
    public int compareTo(Node o) {
        return Integer.compare(this.F, o.F);
```

```
    }
}

class Solution {
    //    -1          1        2
    public int[][] map = {
        {-1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
        {-1, 0, 0, 0, 0, 0, 0, 0, 0, -1},
        {-1, 0, 0, 0, 0, -1, 0, 0, 0, -1},
        {-1, 0, 0, 0, -1, 0, 0, 0, 0, -1},
        {-1, 0, 1, 0, -1, 0, 0, 2, 0, -1},
        {-1, 0, 0, 0, 0, -1, 0, 0, 0, -1},
        {-1, 0, 0, 0, -1, 0, 0, 0, 0, -1},
        {-1, 0, 0, 0, 0, -1, 0, 0, 0, -1},
        {-1, 0, 0, 0, 0, 0, 0, 0, 0, -1},
        {-1, -1, -1, -1, -1, -1, -1, -1, -1, -1}
        };
    // Open
    public PriorityQueue<Node> Open = new PriorityQueue<Node>();
    //Close
    public ArrayList<Node> Close = new ArrayList<Node>();
    //Exist
    public ArrayList<Node> Exist = new ArrayList<Node>();

    public Node astarSearch(Node start, Node end) {
        //                    Open
        this.Open.add(start);
        //                Exist
        this.Exist.add(start);
        //
        while (Open.size() > 0) {
            //                             Open
            Node current_node = Open.poll();
            //             Close
            Close.add(current_node);
            //
            ArrayList<Node> neighbour_node = extend_current_node(current_node);
            //
            //
            for (Node node : neighbour_node) {
                if (node.x == end.x && node.y == end.y) {//
                    node.init_node(current_node,end);
                    return node;
                }
                if (!is_exist(node)) { //                 Open
                    node.init_node(current_node, end);
                    Open.add(node);
                    Exist.add(node);
                }
```

```
            }
        }
        //                                          null
        return null;
    }

    public ArrayList<Node> extend_current_node(Node current_node) {
        int x = current_node.x;
        int y = current_node.y;
        ArrayList<Node> neighbour_node = new ArrayList<Node>();
        if (is_valid(x + 1, y)) {
            Node node = new Node(x + 1, y);
            neighbour_node.add(node);
        }
        if (is_valid(x - 1, y)) {
            Node node = new Node(x -1, y);
            neighbour_node.add(node);
        }
        if (is_valid(x, y + 1)) {
            Node node = new Node(x, y + 1);
            neighbour_node.add(node);
        }
        if (is_valid(x, y - 1)) {
            Node node = new Node(x, y - 1);
            neighbour_node.add(node);
        }
        return neighbour_node;
    }

    public boolean is_valid(int x, int y) {
        //              -1
        if (map[x][y] == -1) return false;
        for (Node node : Exist) {
            //
            //     if (node.x == x && node.y == y) {
            //       return false;
            //     }
            if (is_exist(new Node(x, y))) {
                return false;
            }
        }
        //
        return true;
    }

    public boolean is_exist(Node node)
    {
        for (Node exist_node : Exist) {
            if (node.x == exist_node.x && node.y == exist_node.y) {
```

```
                return true;
            }
        }
        return false;
    }
}
```

```
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1  0  0  0 11 11 11 11  0 -1
-1  0  0 11 11 -1  0 11  0 -1
-1  0  0 11 -1  0  0 11  0 -1
-1  0 11 11 -1  0  0 11  0 -1
-1  0  0  0  0 -1  0  0  0 -1
-1  0  0  0 -1  0  0  0  0 -1
-1  0  0  0  0 -1  0  0  0 -1
-1  0  0  0  0  0  0  0  0 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1
```

1.
2.

```
void DownAdjust(vector<int>& A, int parentIndex, int len) {
   int temp = A[parentIndex];
   int childIndex = 2 * parentIndex + 1;
   while(childIndex < len) {
       if(childIndex + 1 < len && A[childIndex + 1] < A[childIndex])
           childIndex++;
       if(temp >= A[childIndex]) break;
           A[parentIndex] = A[childIndex];
           parentIndex = childIndex;
           childIndex = 2 * childIndex + 1;
   }
   A[parentIndex] = temp;
}
void HeapSort( vector<int>& A) {
   int len = A.size();
   if(len <= 1) return;

   for(int i = len / 2; i >= 0; i--) {
       DownAdjust(A, i, len);
   }

   for(int i = len  - 1; i > 0; i--) {
       int temp = A[i];
       A[i] = A[0];
       A[0] = temp;
       DownAdjust(A, 0 , i);
   }
}
```

Floyd

```
int dist[400][400];
void Floyd(int n)
{
    for (int k = 1; k <= n; ++k)
        for (int i = 1; i <= n; ++i)
            for (int j = 1; j <= n; ++j)
                dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j]);
}
```

Floyd k i j

k i→k +k→j i→j

dist int_max i→i 0

Bellman-Ford :

dist[] S D S->P1->P2->...->D

-2

```
void Bellman_Ford(int n, int m)
{
    for (int j = 0; j < n - 1; ++j)
        for (int i = 1; i <= m; ++i)
            dist[edges[i].to] = min(dist[edges[i].to], dist[edges[i].from] +
edges[i].w);
}
```

SPFA :

Bellman-Ford S

S

SPFA

```
void SPFA(int s)  //s            inqueue
{
    queue<int> Q;
    Q.push(s);
    while (!Q.empty())
    {
        int p = Q.front();
        Q.pop();
        inqueue[p] = 0;
        for (int e = head[p]; e != 0; e = edges[e].next)
        {
            int to = edges[e].to;
            if (dist[to] > dist[p] + edges[e].w)
            {
                dist[to] = dist[p] + edges[e].w;
                if (!inqueue[to])
                {
                    inqueue[to] = 1;
                    Q.push(to);
                }
            }
        }
    }
}
```

O(mn)

Dijkstra

Dij 0

dist dist

dist

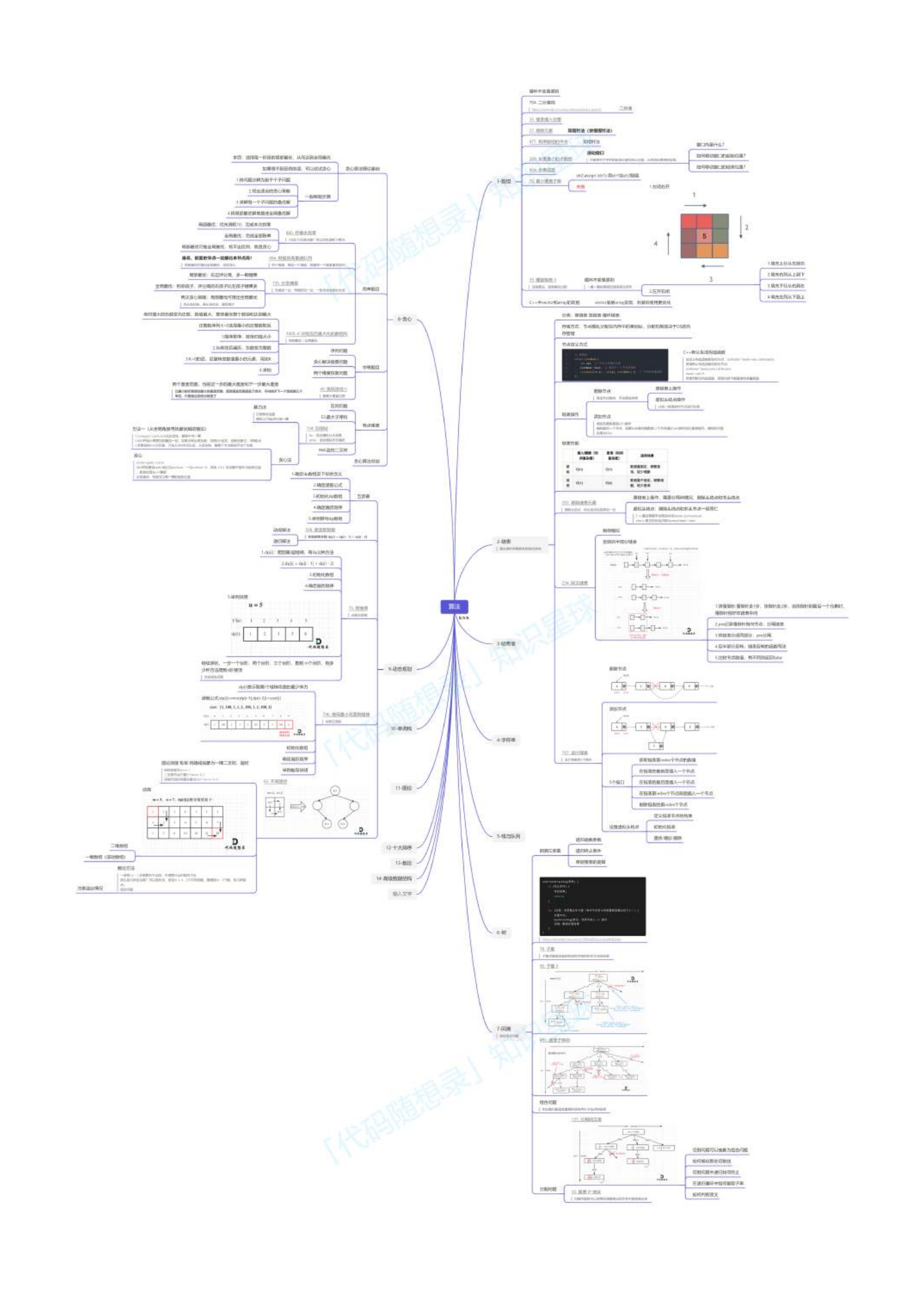
,

```
struct Polar{
    int dist, id;
    Polar(int dist, int id) : dist(dist), id(id){}
};
```

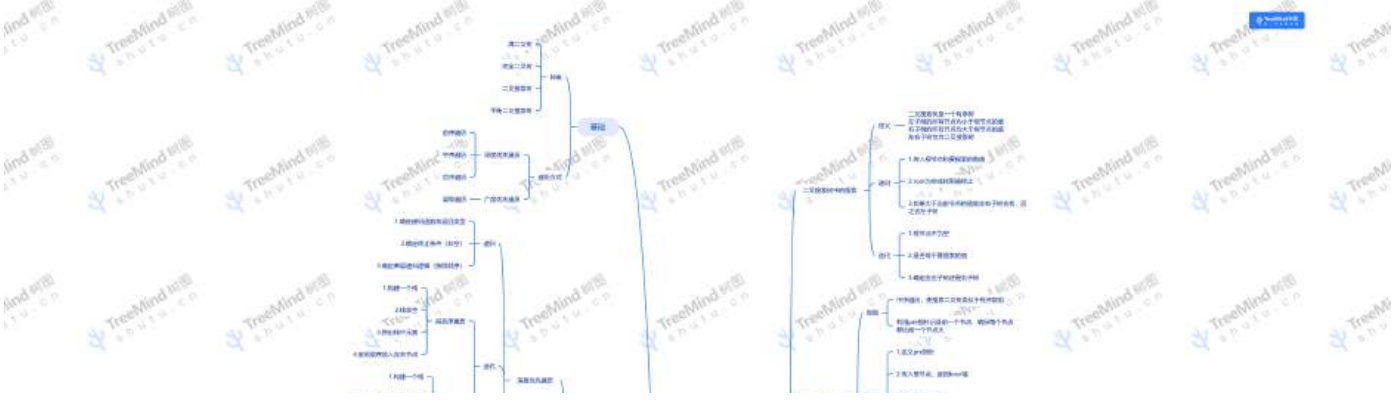
```
struct cmp{
    bool operator ()(Polar a, Polar b){ //    ()
        return a.dist > b.dist;    //
    }
};
priority_queue<Polar, vector<Polar>, cmp> Q;  //
int vis[n] //
```

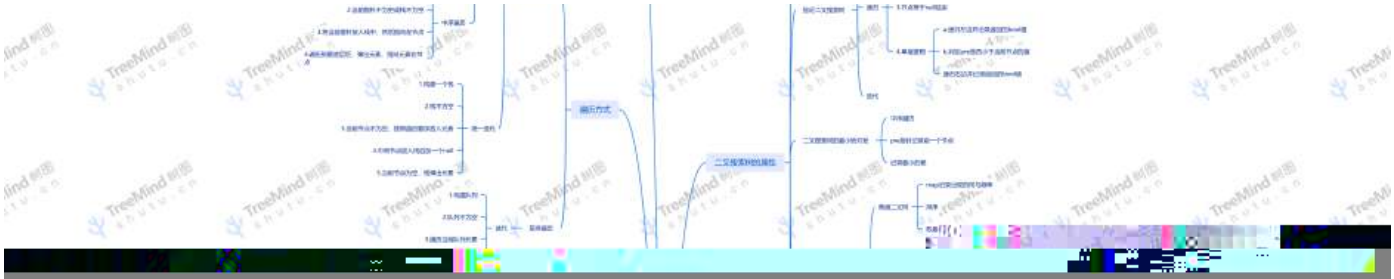
```
void Dij(int s)
{
    dist[s] = 0;
    Q.push(Polar(0, s));
    while (!Q.empty())
    {
        int p = Q.top().id;
        Q.pop();
        if (vis[p])
            continue;
        vis[p] = 1;
        for (int e = head[p]; e != 0; e = edges[e].next)
        {
            int to = edges[e].to;
            dist[to] = min(dist[to], dist[p] + edges[e].w);
            if (!vis[to])
                Q.push(Polar(dist[to], to));
        }
    }
}
```

pre dist pre


算法
1-数组
循环不变量原则
704. 二分查找
二分法
35. 搜索插入位置
27. 移除元素
双指针法（快慢指针法）
977. 有序数组的平方
双指针法
209. 长度最小的子数组
滑动窗口
904. 水果成篮
59. 螺旋矩阵 II
1.如何右开
2.左开右闭
3.左闭右开
1.填充上行从左到右
2.填充右列从上到下
3.填充下行从右到左
4.填充左列从下到上
C++中vector和array区别
2-链表
203. 移除链表元素
707. 设计链表
3-哈希表
4-字符串
5-栈与队列
6-树
7-回溯
8-贪心
9-动态规划
10-单调栈
11-图论
12-十大排序
13-数论
14-高级数据结构
输入文字


链表
操作
删除
解题要点：找到目标位置的前节点使其直接指向待删节点的next指针判断
移除链表元素（节点）
1、定义虚拟头节点dummyHead并指向head
2、定义一个cur指针指向dummyHead
3、在链表存在的情况下，遍历链表
4、判断链表中每个节点的值是否等于val
5、找到操作元素后，定义temp变量存储待删节点，再将cur->next = cur->next->next实现删除操作
6、释放删除节点temp的内存
删除链表的倒数第N个节点
1、定义虚拟头节点dummyHead并指向head
2、定义一个快指针fastNode和慢slow并指向head
3、在快指针不为空的前提下，快指针移动n+1次
4、让两个指针同时移动，直到快指针为空为止
5、找到目标节点后，定义temp变量存储待删节点，利用slow->next=slow->next->next实现删除操作
6、释放删除节点temp的内存
反转和交换
解题要点：找到链表的当前节点和前驱节点，并保存下一个节点的next，指针节点的位置，在更改指针前要记得保存下一节点next，如何断开后续链表
反转链表
1、定义双指针cur=head,pre=NULL
2、当链表不为空时，定义指针节点temp存储cur->next
3、让cur指向它的上一个节点pre
4、移动指针，使pre=cur
5、移动指针，使cur=temp
6、一直循环到链表为空（无节点），将链表指针节点temp的内存
7、返回头节点就变为了pre
K个一组反转链表
两两交换相邻链表节点
1、定义虚拟头节点dummyHead并指向head
2、定义节点指针cur指向dummyHead
3、当cur的下一个节点和下下一个节点不为NULL时，定义两个临时指针temp1=cur->next和temp2=cur->next->next
4、让cur指向它的下下一个节点，再让cur的下下一个节点指向temp1,最后让temp1指向temp2
5、移动cur指针：cur=cur->next->next
查找
解题要点：快慢指针，相遇点到入口/头节点的入口的距离相等
同向速度
1、定义双指针fast和slow，让其指向head
2、因为fast指针每次走两个节点，slow指针每次走一个节点，最后循环的判断条件总结最fast==NULL及fast->next==NULL
3、超fast=slow，则说明存在环
4、定义双指针index1=fast和index2=head
5、当index1=index2时，index1和index2两个指针退出循环
6、只要index1==index2，则index1和index2为环入口的第一个节点，省去if，则返回NULL
重点：一定要计算相遇时fast == slow
判断
判断是否成环
判断是否相交
解题要点：理解链表相交不是val相等，而是指针指向的节点地址相等
链表相交
1、对两个链表分别创建一个指针curA和curB，都指向各自的头节点
2、定义两个指针用长度差减去较大的对应的长度
3、让这两个指针再次指向自己的头指针
4、如果链表A的长度大于指向较长度，则较长指针先行交换，给将两个链表A长度大于等于较短的长度
5、求出两者距离gap，并向curA进行前移gap个
6、从curA遍历链表A，若两个链表指针相等，则返回其节点指针（表示找到其相交节点），并让两者指针逐步向前直至相等，若无则为，则返回NULL
重点：理解不合适应用循环,直接curA == curB 可作为退出循环
判断是否是回文链表
解题要点：回文可以通过双指针技术找出前后两段是否val相同来判断，注意判断相比较，也可以利用数组来对前后val值是否相同来进行判断
回文链表
1、定义一个vector数组
2、定义一个cur指针指向head
3、遍历链表，并且将每个值自然语都存储进vector数组中
4、利用双指针（前后下标）一前一后遍历vector数组，若有发现不相等，则返回false（不是回文链表）；否则就返回true
设计
设计链表
解题要点：链表链表设计实现，存储表达此了对链表相关操作(get、addAtHead、addAtTail、addAtIndex、deleteAtIndex)设置头两节和二分优化偏后节点个数
设计链表
链表节点的定义及初始化
定义链表大小初始化为0
定义一个虚拟头节点dummyHead并指向head
查找链表中下标为index的节点的值
1、若index不在范围内，返回-1
2、定义一个cur指针指向头节点
3、cur指针遍历到下标为index的位置
4、找到下标为index的节点后，返回节点的值val
向一个值为val的节点前插入新的节点
1、创建一个新节点并赋值val
2、让新节点的next指向下一个节点（头节点）
3、让dummyHead指向新节点（新头节点）
4、链表长度size+1
向一个值为val的节点后插入新的节点
1、创建一个新节点并赋值val
2、定义一个指针cur指向head
3、当cur的下一个节点不为NULL，cur就一直向后移
4、让cur指向新节点
5、链表长度size+1
向一个值为val的节点插入新的节点
1、创建一个新节点并赋值val
2、定义一个指针cur指向head
3、如果index大于链表长度，则直接返回，如果小于0，则指向index设置为0
4、指针cur遍历到index位
5、让新节点指向cur的下一个节点，让cur指向新节点
6、链表长度size+1
删除下标有效的第index个节点
1、定义一个指针cur指向head
2、若index不在范围内，直接返回
3、cur指针遍历到index位
4、定义一个临时指针temp存储cur的下一个节点
5、让cur指向它的下下一个节点
6、释放节点temp指向节点的内存
7、链表长度size-1
设计队列
设计队列缓存
设计LRU缓存

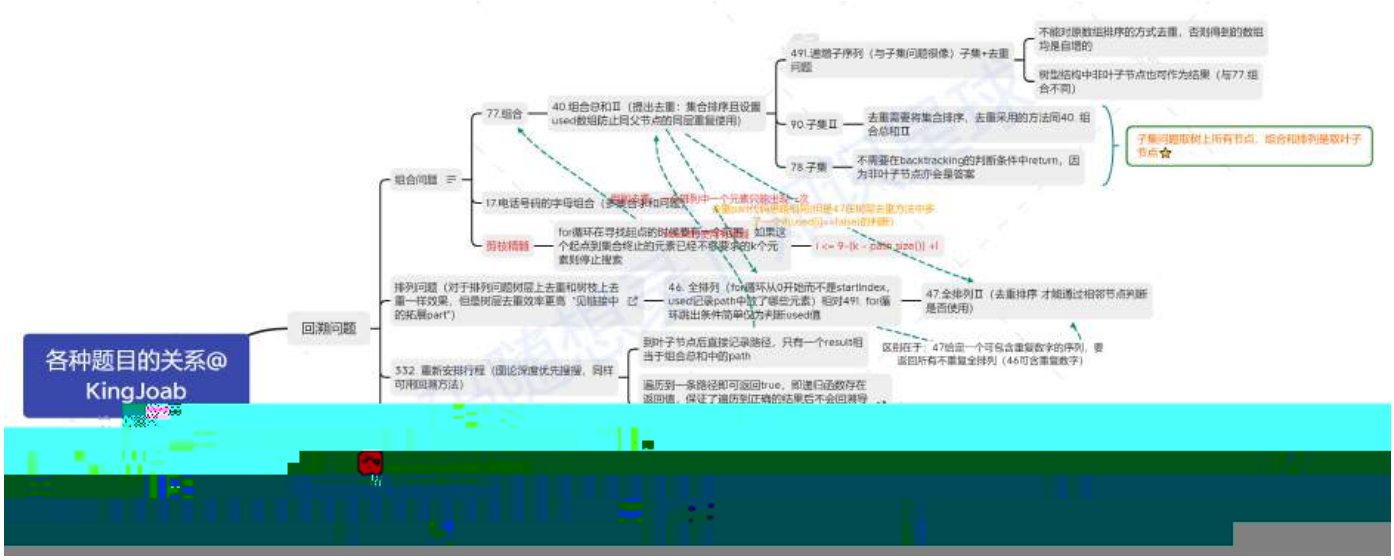


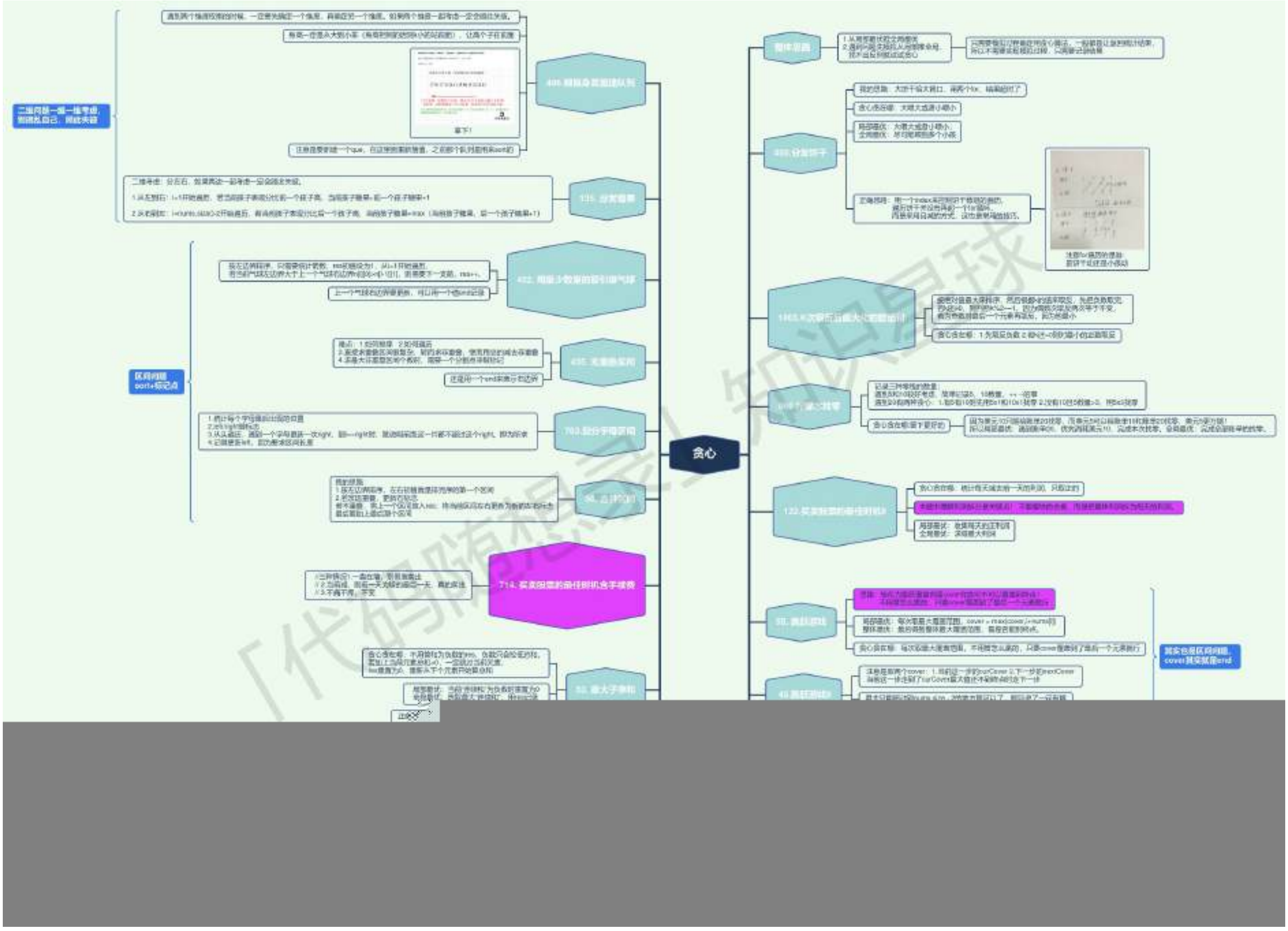


去重小结

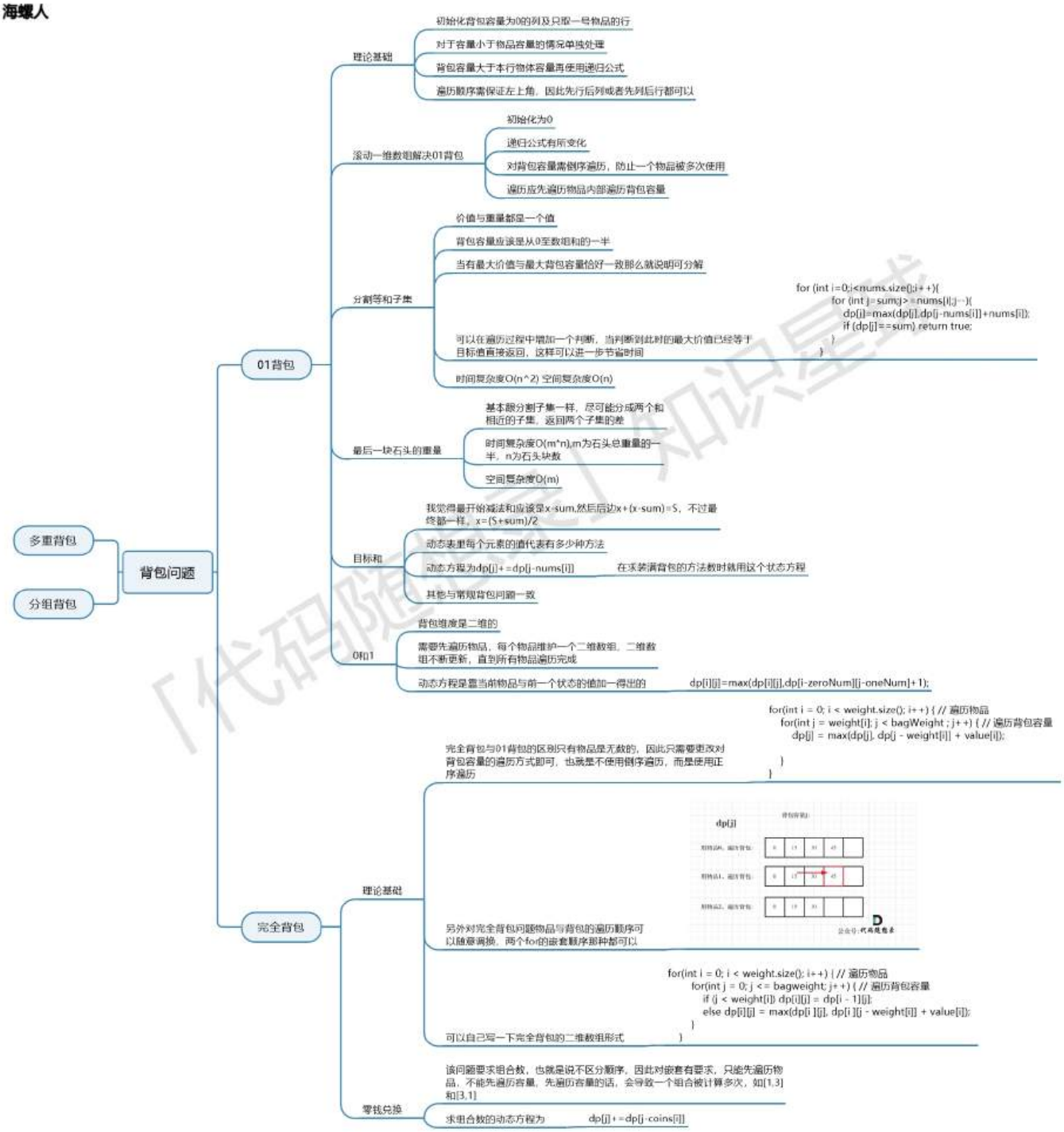
	StartIndex	Used[]数组	Flag[]数组
出现的原因	同一元素不能重复选取(下一层传入 i+1)	子集/组合问题中: 出现了数值相同的多个元素，且同一层数值相同的多个元素不能被重复选取	用于解决递增子序列问题，由于递增子序列问题不能对数组进行排序，否则不满足原始题目要求，但又要求同一树层的元素不能重复选取，所以在每一层中都创建flag[]数组为该层去重
	同一元素可以被重复选取(下一层传入 i)	全排列问题中: 1. 同一层数值相同的多个元素不能被重复选取 2. 树枝的所有节点不能出现重复的元素(即使没出现数值相同的多个元素也需要 used[]数组)	
需要满足的特殊条件	无	当出现重复元素时数组要进行排序	无
适用场景	组合问题，子集问题，递增子序列问题	组合问题，子集问题，排列问题	递增子序列问题

本人



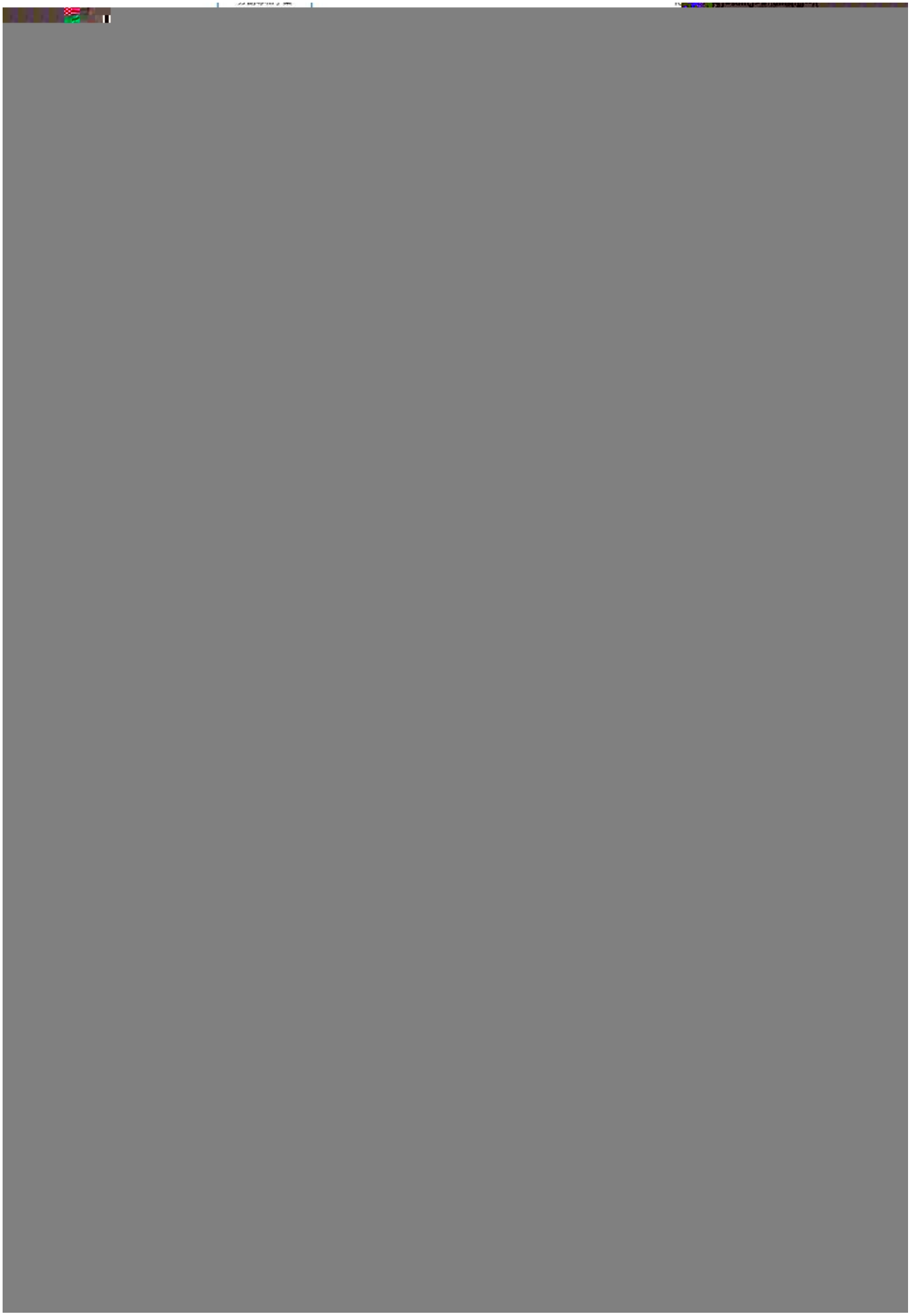

贪心

海螺人



海螺人

理论基础
初始化背包容量为0的列及只取一号物品的行
对于容量小于物品容量的情况单独处理
背包容量大于本行物体容量再使用递归公式
遍历顺序需保证左上角，因此先行后列或者先列后行都可以
滚动一维数组解决01背包
初始化为0
递归公式有所变化
对背包容量需倒序遍历，防止一个物品被多次使用
遍历应先遍历物品内部遍历背包容量
分割等和子集
价值与重量都是一个值
背包容量应该是从0至数组和的一半
当有最大价值与最大背包容量恰好一致那么就说明可分解



背包遍历顺序

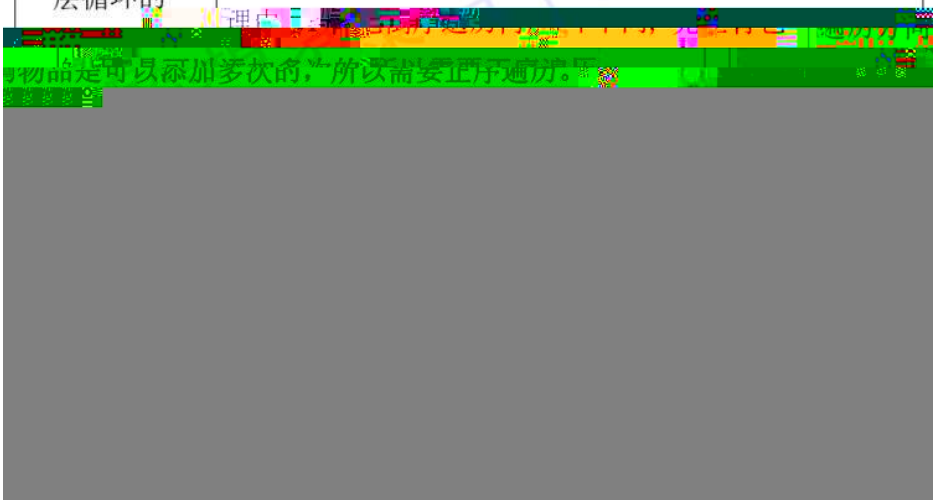
本人

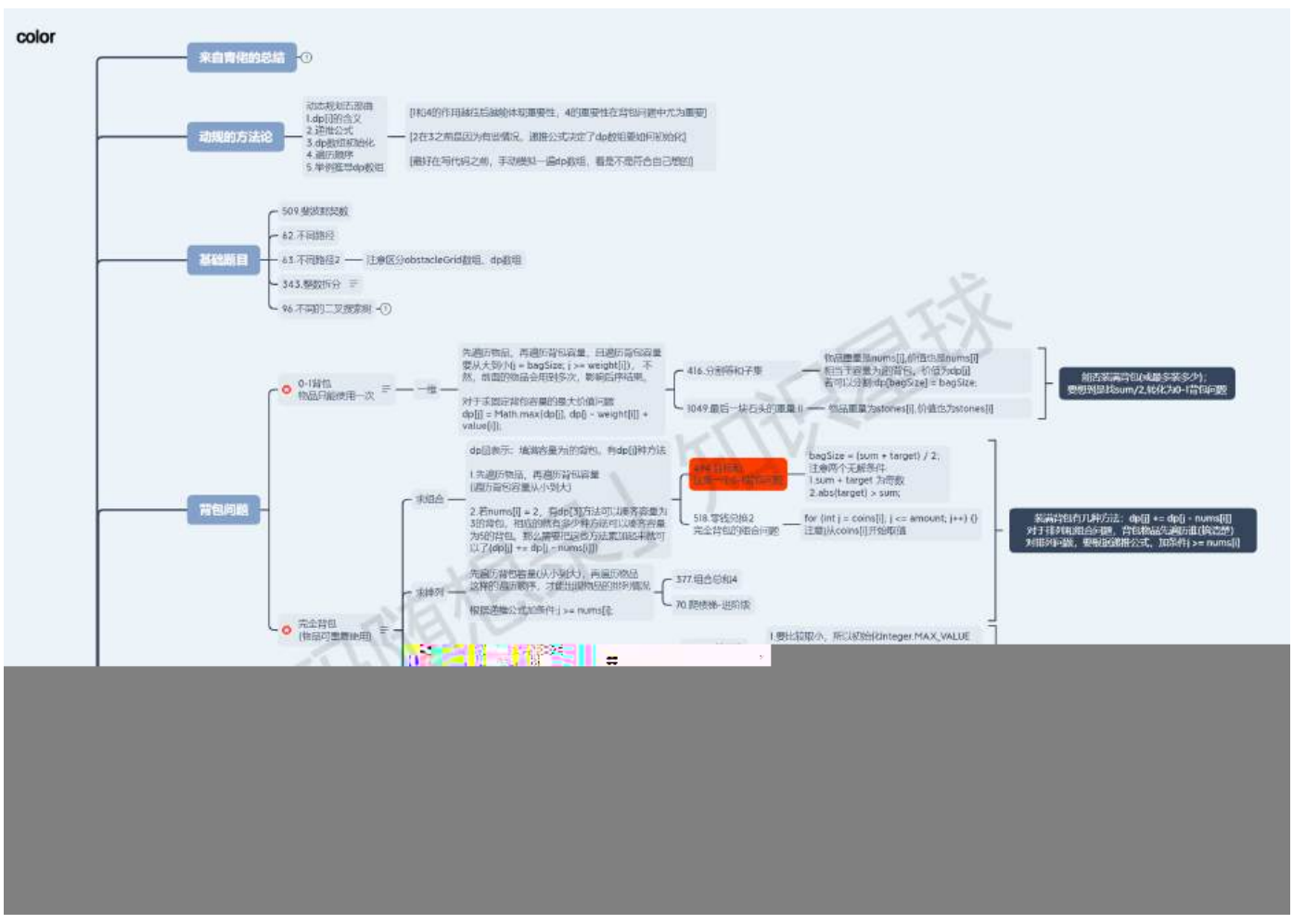
友情提示： 动手画出各种情况下的 dp 数组有助于加深理解		两层 for 循环的 遍历顺序	一层 for 循环的 遍历方向（正序/倒序）	
		先选择物品 or 先选择背包	初始化时的 遍历方向	循环递推 dp 数组时的 遍历方向
01 背包	二维 dp[i][j]	都可以 理由：dp[i][j]总是由 dp 数组左上角的值推导得到	倒序遍历 for j:bagWeight to weight[0] dp[0][j] = dp[0][j - weight[0]] + value[0]; 理由：为了保证在每个重量的背包中物品 i 只被放入一次	两层循环均为正序遍历 理由：dp[i][j]总是由 dp 数组左上角的值推导得到
	一维 dp[j]	先选择物品后选择背包 理由：如果先背包后物品任意重量的背包只会选择其中一个价值最大的物品	无 for(j:1 to bagWeight) dp[j]=0 一般初始化为 0 即可	遍历物品时：正序遍历 遍历背包时：倒序遍历 倒序遍历理由：为了保证物品 i 只被放入一次

完全背包遍历顺序

注:leetcode 中排列问题也有可能出现组合的字眼(如:377. 组合总和 Ⅳ),但是如果看到这句话: “顺序不同的序列被视作不同的组合”,则就是排列问题。

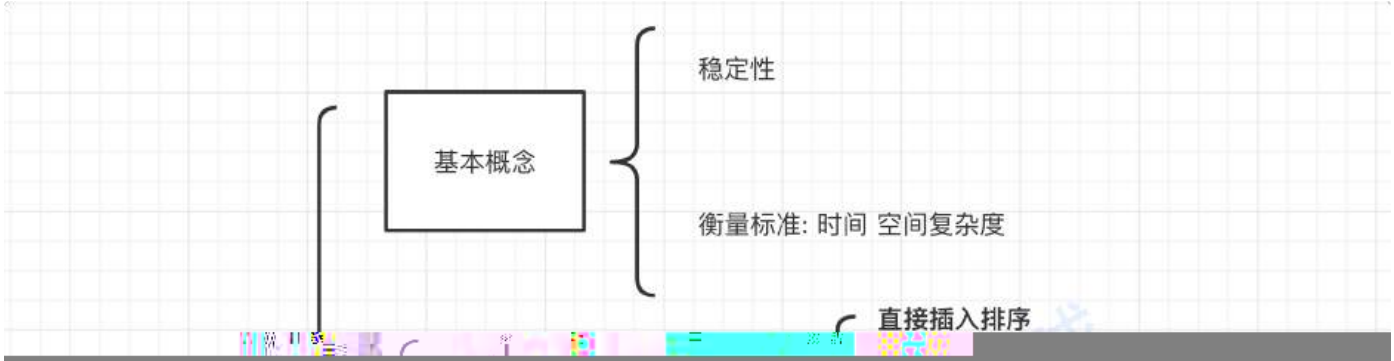
一维背包 dp[j]	纯完全背包	完全背包-组合	完全背包-排列
两层循环中每一层循环的	均为正序遍历,即 i++, j++		





https://t.zsxq.com/14OVuRtMY


冒泡排序
思想：两两比较相邻元素，如果前一个比后一个大，则交换之
优化：
(1) 增设一个标记，若本轮冒泡未交换元素，说明已经有序，无需下一轮冒泡；
(2) 每轮冒泡记录最后一次交换的下标位置（其后就是有序区，无需继续冒泡遍历）
思想：冒泡排序的升级版。鸡尾酒排序的元素比较和交换过程是双向的。第一轮从左到右，第二轮从右向左，第三轮再从左到右，



	时间复杂度	空间复杂度	是否为稳定排序	是否为原地排序
直接插入排序	$O(n^2)$	$O(1)$	✓	✓
折半插入排序	$O(n^2)$ (只优化了比较的过程)	$O(1)$	✓	✓
希尔排序	和增量有关，最坏$O(n^2)$	$O(1)$	✗	✓
冒泡排序	$O(n^2)$	$O(1)$	✓	✓
快速排序	$O(nlog_2n)$	$O(log_2n)$	✗	✓
简单选择排序	$O(n^2)$	$O(1)$	✗	✓
堆排序	$O(nlog_2n)$	$O(1)$	✗	✓
归并排序	$O(nlog_2n)$	$O(n)$	✓	✗
基数排序	$O(dn)$ d是位数	$O(d)$	✓	✗
桶排序	$O(n+k)$ k为桶个数，n为元素数量	$O(n+k)$	✓	✗

排序算法时间空间复杂度分析

		最好时间复杂度	最坏时间复杂度	平均时间复杂	复杂度