Aho–Corasick algorithm

高频题为此颜色

普通题为此色

算法题目一般从毫无感觉 --> 看懂别人的思路 --> 培养自己的题感 --> 写出最优解法

爬楼梯

fibnacci

当前一步 等于 前一步可能性 + 前前一步可能性

```
class Solution {
    public int climbStairs(int n) {
        if (n <= 2) return n;
        int firstStep = 1;
        int secondStep = 2;
        int total = 0;
        for (int i = 3; i <= n; ++i) {
            total = firstStep + secondStep;
            firstStep = secondStep;
            secondStep = total;
        }
        return total;
    }
}
```

做题思路

- 做题思路
  - 写下思路步骤
  - 1，2，3
  - 写所有可能的解法，不要有约束跟顾虑
  - coding: first, check input valid
- 对题目懵逼了
  - 化繁为简，列出点最简单的情况
  - 能否聚为一类举例情况是？
  - 找最近子问题的重复性，机器只能 for 或 recursion
- 嵌套数组的遍历
  - 从左至右
  - 从右至左
- Linked List
  - 熟能生巧，套路固定

动态规划

Fibnacci

使用记忆化搜索缓存中间结果

Unique Paths

方法1

- 创建二维数组
- 先将 第一排跟第一列 的 items fill 1
- 使用递推公式:
  dp[i][j] = dp[i - 1][j] + dp[i][j - 1]
- return dp[m - 1][n - 1]

方法二（节省了一个维度的空间，复杂度与方法一一致）

只保留前一行的值，然后更新当前行每个 col 的值

- 初始化 new int[n], fill items to be 1
- 从 i(row) = 1, j(column) = 1 开始遍历二维数组
- curr[j] += curr[j - 1]; // += 这一步相当于把当前坐标的 col - 1 的值也加进去了，满足了 递推公式 dp[i][j] = dp[i - 1][j] + dp[i][j - 1]

方法三 牛逼

```
class Solution {
    public int uniquePaths(int m, int n) {
        if (m <= 0 || n <= 0) return 0;
        long res = 1;
        for (int i = 1; i < m; i++) {
            res = res * (i + n - 1) / i;
        }
        return (int)res;
    }
}
```

Longest Common Subsequence

		0	1	2	3	4	5	6
	str2 / str1	"	b	a	b	c	d	e
0	"	0	0	0	0	0	0	0
1	a	0	0	1	1	1	1	1
2	c	0	0	1	1	2	2	2
3	e	0	0	1	1	2	2	3

```
public int longestCommonSubsequence(String text1, String text2) {
    if (text1.length() == 0 || text2.length() == 0) return 0;
    int m = text1.length();
    int n = text2.length();
    int[][] dp = new int[m + 1][n + 1];
    for (int i = 1; i < m + 1; ++i)
        for (int j = 1; j < n + 1; ++j)
            dp[i][j] = text1.charAt(i - 1) == text2.charAt(j - 1) ?
                dp[i - 1][j - 1] + 1 :
                Math.max(dp[i - 1][j], dp[i][j - 1]);
    return dp[m][n];
}
```

```
class Solution {
    public int longestCommonSubsequence(String text1, String text2) {
        if (text1.length() == 0 || text2.length() == 0) return 0;
        char[] chars1 = text1.toCharArray();
        char[] chars2 = text2.toCharArray();
        int[][] dp = new int[chars1.length + 1][chars2.length + 1];
        for (int i = 0; i < chars1.length; ++i)
            for (int j = 0; j < chars2.length; ++j)
                //  相同则得到新增的子序列总数 + 1,
                //  不相同则比较 两个子序列分别减一个字符 谁总数最多取值
                dp[i + 1][j + 1] = chars1[i] == chars2[j] ? dp[i][j] + 1 : Math.max(dp[i + 1][j], dp[i][j + 1]);
        return dp[chars1.length][chars2.length];
    }
}
```

- S1 = "ABAZDC"
- S2 = "BACBAD"
- if S1[-1] != S2[-1]: LCS[s1, s2] = Max(LCS[s1-1, s2], LCS[s1, s2-1])
- LCS[s1, s2] = Max(LCS[s1-1, s2], LCS[s1, s2-1], LCS[s1-1, s2-1])
- if S1[-1] == S2[-1]: LCS[s1, s2] = LCS[s1-1, s2-1] + 1
- LCS[s1, s2] = Max(LCS[s1-1, s2], LCS[s1, s2-1], LCS[s1-1, s2-1], LCS[s1-1][s2-1] + 1)

Maximum Product Subarray

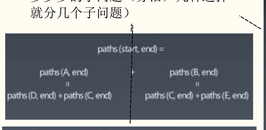
```
class Solution:
    def maxProduct(self, nums: List[int]) -> int:
        mi = ma = res = nums[0]
        for i in range(1, len(nums)):
            if nums[i] < 0: mi, ma = ma, mi
            ma = max(ma * nums[i], nums[i])
            mi = min(mi * nums[i], nums[i])
            res = max(res, ma)
        return res
```

Maximum Subarray

```
/**
 * 状态转移方程: sum[i] = max{sum[i - 1] + a[i], a[i]}
 * 如果 sum > 0, 则说明 sum 对结果有增益效果, 则 sum 保留并加上当前遍历数字
 * 如果 sum <= 0, 则说明 sum 对结果无增益效果, 需要舍弃, 则 sum 直接更新为当前遍历数字
 * 每次比较 sum 和 ans的大小, 将最大值置为ans, 遍历结束返回结果
 * 时间复杂度: O(n)
 */
public int maxSubArray(int[] nums) {
    int result = nums[0];  // base case
    int sum = nums[0];
    for (int i = 1; i < nums.length; ++i) {
        // sum[i] = max{sum[i - 1] + a[i], a[i]}
        sum = Math.max(sum + nums[i], nums[i]);
        //  update maximum
        result = Math.max(result, sum);
    }
    return result;
}
```

Count the paths

paths 拆分成 求A、B到终点路径有多少步的子问题（分治，几种选择就分几个子问题）




```
int countPaths (boolean [][]grid  int row  int col ){
  if (!validSquare(grid  row  col)) return 0;
  if (isAtEnd(grid  row  col)) return 1;
  return countPaths (grid  row + 1  col ) + countPaths (grid  row  col + 1);
}
```

状态转移方程（DP方程）

opt[i , j] = opt[i + 1 , j] + opt[i , j + 1]

完整逻辑：

if a[i, j] = '空地'：

opt[i , j] = opt[i + 1 , j] + opt[i , j + 1]

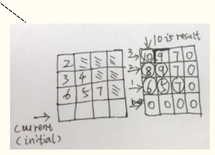
else:

opt[i , j] = 0

triangle

https://leetcode-cn.com/problems/triangle/solution/di-gui-ji-yi-hua-sou-suo-zai-dao-dp-by-crsm/

https://leetcode.com/problems/triangle/discuss/38735/Python-easy-to-understand-solutions-(top-down-bottom-up)




```
public int minimumTotal(List<List<Integer>> triangle) {
    //  bottom to top
    int m = triangle.size();
    int[] dp = new int[m + 1];
    for (int i = m - 1; i >= 0; --i)
        for (int j = 0; j <= i; ++j)
            dp[j] = Math.min(dp[j], dp[j + 1]) + triangle.get(i).get(j);
    return dp[0];
}
```

注意 j 的 loop 条件是 j <= i, 因为 dp 的长度是 m + 1;

Coin Change

https://leetcode-cn.com/problems/coin-change/solution/dong-tai-gui-hua-tao-lu-xiang-jie-by-wei-lai-bu-ke/

子问题，不断靠近0元，BFS(层数代表用了几个硬币，找层数用的最少的)

DP:

a. subproblems

b. DP array:
   f(n) = min(f(n - k), for k in [1, 2, 5]) + 1

c. DP 方程:

```
class Solution {
    public int coinChange(int[] coins, int amount) {
        int max = amount + 1;
        int[] dp = new int[amount + 1]; // 0..amount
        Arrays.fill(dp, max);
        dp[0] = 0;
        // from 0..amount
        for (int i = 1; i <= amount; ++i) {
            //  each coin
            for (int j = 0; j < coins.length; ++j) {
                //  if coin can make a change
                if (coins[j] <= i) {
                    //  if this coin have no answer, will still using max as dp[i]'s value
                    //  otherwise:
                    //  f(n) = f(n - k) + 1, for k in coins
                    //  explain: (1 count + (amount - coin = previous amount's count))
                    dp[i] = Math.min(dp[i], dp[i - coins[j]] + 1);
                }
            }
        }
        return dp[amount] > amount ? -1 : dp[amount];
    }
}
```

House Robber

对当前房子可偷可不偷，

技巧: 开二维数组来存储偷不偷状态，对于复杂问题要开多个维度记录夹带的条件

a[i][0, 1]: 0: i偷, 1:不偷

DP公式

dp[i][0] = Max(a[i - 1][0], a[i-1][1])

dp[i][1] = Max(a[i - 1][0], nums[i])

```
public int rob(int[] nums) {
    if (nums == null || nums.length == 0) return 0;
    int n = nums.length;
    //  create [][] for record 偷不偷(像括号生成题的选择一样)
    int[][] a = new int[n][2];
    a[0][0] = 0; //  不偷
    a[0][1] = nums[0]; //  偷

    //  for each room, 偷 或 不偷
    for (int i = 1; i < nums.length; ++i) {
        a[i][0] = Math.max(a[i - 1][0], a[i - 1][1]);
        a[i][1] = a[i - 1][0] + nums[i];
    }

    //  最后一间房偷 或 不偷
    return Math.max(a[n - 1][0], a[n - 1][1]);
}
```

解法2

max = max(prevPax + x, currMax)

```
public int rob(int[] num) {
    int prevMax = 0;
    int currMax = 0;
    for (int x : num) {
        int temp = currMax;
        currMax = Math.max(prevMax + x, currMax);
        prevMax = temp;
    }
    return currMax;
}
```

贪心 & 二分查找

Coin Change

Assign Cookies

对小孩, cookie 进行排序

从胃口最小的小孩开始分配

最小的 cookie

```
class Solution {
    public int findContentChildren(int[] children, int[] cookies) {
        Arrays.sort(children);
        Arrays.sort(cookies);
        int j = 0;
        for (int i = 0; i < cookies.length && j < children.length; ++i)
            if (children[j] <= cookies[i]) ++ j;
        return j;
    }
}
```

Best Time to Buy and Sell Stock II

遍历时间线, 如果前一天比当前一天钱多，就买入 复杂度 O(n)

Jump Game

从后往前贪心

看当前步数 + 当前 index 是否能达到前一步

Sqrtx

二分查找或牛顿迭代法(用直线代替曲线 无限逼近平方根的值)

https://www.beyond3d.com/content/articles/8/

```
class Solution(object):
    def mySqrt(self, x):
        r = x
        while r*r > x:
            r = (r + x/r) / 2
        return r
```

```
class Solution {
    public int mySqrt(int x) {
        if(x <= 1) return x;
        int left = 1, right = x;
        while(left <= right){
            int mid = left + (right -left) / 2; //  prevent overflow
            //  prevent overflow via using divided, not multiple(mid * mid == x)
            int sqrt = x / mid;
            if(sqrt == mid)
                return mid;
            else if(sqrt < mid)
                right = mid - 1;
            else
                left = mid + 1;
        }
        return right;
    }
}
```

Search a 2D Matrix

遍历 Matrix 找到 target 可能存在的 row

对 target row 使用二分查找

Valid Perfect Square

牛顿迭代法

```
class Solution {
  public boolean isPerfectSquare(int num) {
    if (num < 2) return true;
    long left = 2, right = num / 2, x, guessSquared;
    while (left <= right) {
      x = left + (right - left) / 2;
      guessSquared = x * x;
      if (guessSquared == num)
        return true;
      if (guessSquared > num)
        right = x - 1;
      else
        left = x + 1;
    }
    return false;
  }
}
```