1. DP

1.1 一维DP

`5 和 647 很像，都是extend类型的, if you can solve the 5 in the fastest way, you can skip 5 and 647.`

5. Longest Palindromic Substring

https://leetcode.com/problems/longest-palindromic-substring/description/

Given a string s, find the longest palindromic substring in s. You may assume that the maximum length of s is 1000.

Example 1:

```
Input: "babad"
Output: "bab"
Note: "aba" is also a valid answer.
```

Example 2:

```
Input: "cbbd"
Output: "bb"
```

```python
class Solution:
    def extend(self, s, i, j ):
        while i>=0 and j <len(s) and s[i] == s[j]:
            i-=1
            j+=1
        if j-i-1>self.maxLen:
            self.maxLen = j-i-1
            self.start = i+1

    def longestPalindrome(self, s):
        """ :type s: str :rtype: str """
        self.start = 0
        self.maxLen = 0

        for i in range(len(s)):
            self.extend(s, i, i)
            self.extend(s, i, i+1)
        return s[self.start: self.start+self.maxLen]
"""
one improvement according to the faster solution, is that your can skip those whose
"""
```

647. palindromic-substrings

https://leetcode.com/problems/palindromic-substrings/description/

Given a string, your task is to count how many palindromic substrings in this string.

The substrings with different start indexes or end indexes are counted as different substrings even they consist of same characters.

Example 1:

```
Input: "abc"
Output: 3
Explanation: Three palindromic strings: "a", "b", "c".
```

Example 2:

```
Input: "aaa"
Output: 6
Explanation: Six palindromic strings: "a", "a", "a", "aa", "aa", "aaa".
```

Note:

1. The input string length won't exceed 1000.

```
class Solution:
    def extend(self, s, i, j ):
        while i >= 0 and j < len(s) and s[i] == s[j]:
            i -= 1
            j += 1
            self.count += 1
    def countSubstrings(self, s):
        """ :type s: str :rtype: int """
        self.count = 0
        for i in range(len(s)):
            self.extend(s, i, i)
            self.extend(s, i, i+1)
        return self.count
```

`try to explain 53, don't need to solve 53, relative easy.`

53. maximum-subarray

Given an integer array `nums`, find the contiguous subarray (containing at least one number) which has the largest sum and return its sum.

Example:

```
Input: [-2,1,-3,4,-1,2,1,-5,4],
Output: 6
Explanation: [4,-1,2,1] has the largest sum = 6.
```

Follow up:

If you have figured out the O(*n*) solution, try coding another solution using the divide and conquer approach, which is more subtle.

```
class Solution:
    def maxSubArray(self, nums):
        """:type nums: List[int] :rtype: int"""
        res = nums[0]
        curSum = nums[0]
        for i in range(1, len(nums)):
            curSum = max(curSum + nums[i], nums[i])
            res = max(res, curSum)
        return res
```

---

152. maximum-product-subarray

https://leetcode.com/problems/maximum-product-subarray

Given an integer array `nums`, find the contiguous subarray within an array (containing at least one number) which has the largest product.

Example 1:

```
Input: [2,3,-2,4]
Output: 6
Explanation: [2,3] has the largest product 6.
```

Example 2:

```
Input: [-2,0,-1]
Output: 0
Explanation: The result cannot be 2, because [-2,-1] is not a subarray.
```

```
class Solution:
    def maxProduct(self, nums):
        """:type nums: List[int] :rtype: int"""
        max_ = min_ = res = nums[0]
        for i in range(1, len(nums)):
            max_, min_  = max(nums[i], max_*nums[i], min_*nums[i]) , min(nums[i], m
            res = max(max_, res, min_)
        return res
```

413 arithmetic slices

https://leetcode.com/problems/arithmetic-slices/description/

A sequence of number is called arithmetic if it consists of at least three elements and if the difference between any two consecutive elements is the same.

For example, these are arithmetic sequence:

```
1, 3, 5, 7, 9
7, 7, 7, 7
3, -1, -5, -9
```

The following sequence is not arithmetic.

```
1, 1, 2, 5, 7
```

A zero-indexed array A consisting of N numbers is given. A slice of that array is any pair of integers (P, Q) such that 0 less or equal to P less than Q less than N.

A slice (P, Q) of array A is called arithmetic if the sequence:
A[P], A[p + 1], ..., A[Q - 1], A[Q] is arithmetic. In particular, this means that P + 1 less than Q.

The function should return the number of arithmetic slices in the array A.

Example:

```
A = [1, 2, 3, 4]

return: 3, for 3 arithmetic slices in A: [1, 2, 3], [2, 3, 4] and [1, 2, 3, 4] itse
```

`注意是连续的，跳着的不算，比如 1, 2, 3, 4, 5,  [1, 3, 5]就不算。`

```

```

338. counting-bits

Given a non negative integer number num. For every numbers i in the range 0 ≤ i ≤ num calculate the number of 1's in their binary representation and return them as an array.

Example 1:

```
Input: 2
Output: [0,1,1]
```

Example 2:

```
Input: 5
Output: [0,1,1,2,1,2]
```

Follow up:

- It is very easy to come up with a solution with run time O(n*sizeof(integer)). But can you do it in linear time O(n) /possibly in a single pass?
- Space complexity should be O(n).
- Can you do it like a boss? Do it without using any builtin function like __builtin_popcount in c++ or in any other language.

---

279. perfect-squares

Given a positive integer *n*, find the least number of perfect square numbers (for example, `1, 4, 9, 16, ...`) which sum to *n*.

Example 1:

```
Input: n = 12
Output: 3
Explanation: 12 = 4 + 4 + 4.
```

Example 2:

```
Input: n = 13
Output: 2
Explanation: 13 = 4 + 9.
```

`DP方法 和 数学方法，hint只有1，2，3，4四种可能，公式是 4k*(8m+7) -> 4`

1.2 二维DP

10. regular-expression-matching

Given an input string (`s`) and a pattern (`p`), implement regular expression matching with support for `'.'` and `'*'`.

```
'.' Matches any single character.
'*' Matches zero or more of the preceding element.
```

The matching should cover the entire input string (not partial).

Note:

- `s` could be empty and contains only lowercase letters `a-z`.
- `p` could be empty and contains only lowercase letters `a-z`, and characters like `.` or `*`.

Example 1:

```
Input:
s = "aa"
p = "a"
Output: false
Explanation: "a" does not match the entire string "aa".
```

Example 2:

```
Input:
s = "aa"
p = "a*"
Output: true
Explanation: '*' means zero or more of the precedeng element, 'a'. Therefore, by re
```

Example 3:

```
Input:
s = "ab"
p = ".*"
Output: true
Explanation: ".*" means "zero or more (*) of any character (.)".
```

Example 4:

```
Input:
s = "aab"
p = "c*a*b"
Output: true
Explanation: c can be repeated 0 times, a can be repeated 1 time. Therefore it matc
```

Example 5:

```
Input:
s = "mississippi"
p = "mis*is*p*."
Output: false
```

---

85. maximal-rectangle

Given a 2D binary matrix filled with 0's and 1's, find the largest rectangle containing only 1's and return its area.

Example:

```
Input:
[
  ["1","0","1","0","0"],
  ["1","0","1","1","1"],
  ["1","1","1","1","1"],
  ["1","0","0","1","0"]
]
Output: 6
```

---

312. burst-balloons

Given `n` balloons, indexed from `0` to `n-1`. Each balloon is painted with a number on it represented by array `nums`. You are asked to burst all the balloons. If the you burst balloon `i` you will get `nums[left] * nums[i] * nums[right]` coins. Here `left` and `right` are adjacent indices of `i`. After the burst, the `left` and `right` then becomes adjacent.

Find the maximum coins you can collect by bursting the balloons wisely.

Note:

- You may imagine `nums[-1] = nums[n] = 1`. They are not real therefore you can not burst them.
- 0 ≤ `n` ≤ 500, 0 ≤ `nums[i]` ≤ 100

Example:

```
Input: [3,1,5,8]
Output: 167
Explanation: nums = [3,1,5,8] --> [3,5,8] -->   [3,8]   -->  [8]  --> []
             coins =  3*1*5      +  3*5*8    +  1*3*8      + 1*8*1   = 167
```

120. triangle

Given a triangle, find the minimum path sum from top to bottom. Each step you may move to adjacent numbers on the row below.

For example, given the following triangle

```
[
     [2],
    [3,4],
   [6,5,7],
  [4,1,8,3]
]
```

The minimum path sum from top to bottom is `11` (i.e., 2 + 3 + 5 + 1 = 11).

Note:

Bonus point if you are able to do this using only *O*(*n*) extra space, where *n* is the total number of rows in the triangle.

188 & 123 best-time-to-buy-and-sell-stock-iv

Say you have an array for which the *i*th element is the price of a given stock on day *i*.

Design an algorithm to find the maximum profit. You may complete at most k transactions.

Note:
You may not engage in multiple transactions at the same time (ie, you must sell the stock before you buy again).

Example 1:

```
Input: [2,4,1], k = 2
Output: 2
Explanation: Buy on day 1 (price = 2) and sell on day 2 (price = 4), profit = 4-2 =
```

Example 2:

```
Input: [3,2,6,5,0,3], k = 2
Output: 7
Explanation: Buy on day 2 (price = 2) and sell on day 3 (price = 6), profit = 6-2 =
             Then buy on day 5 (price = 0) and sell on day 6 (price = 3), profit =
```

115. distinct-subsequences

Given a string S and a string T, count the number of distinct subsequences of S which equals T.

A subsequence of a string is a new string which is formed from the original string by deleting some (can be none) of the characters without disturbing the relative positions of the remaining characters. (ie, `"ACE"` is a subsequence of `"ABCDE"` while `"AEC"` is not).

Example 1:

```
Input: S = "rabbbit", T = "rabbit"
Output: 3
Explanation:

As shown below, there are 3 ways you can generate "rabbit" from S.
(The caret symbol ^ means the chosen letters)

rabbbit
```

```
^^^^ ^^
rabbbit
^^ ^^^^
rabbbit
^^^ ^^^
```

Example 2:

```
Input: S = "babgbag", T = "bag"
Output: 5
Explanation:

As shown below, there are 5 ways you can generate "bag" from S.
(The caret symbol ^ means the chosen letters)

babgbag
^^ ^
babgbag
^^    ^
babgbag
^    ^^
babgbag
  ^  ^^
babgbag
    ^^^
```

---

2. Stack

32. longest-valid-parentheses

Given a string containing just the characters `'('` and `')'`, find the length of the longest valid (well-formed) parentheses substring.

Example 1:

```
Input: "(()"
Output: 2
Explanation: The longest valid parentheses substring is "()"
```

Example 2:

```
Input: ")()())"
Output: 4
Explanation: The longest valid parentheses substring is "()()"
```

---

20. valid-parentheses

Given a string containing just the characters `'('`, `')'`, `'{'`, `'}'`, `'['` and `']'`, determine if the input string is valid.

An input string is valid if:

1. Open brackets must be closed by the same type of brackets.
2. Open brackets must be closed in the correct order.

Note that an empty string is also considered valid.

Example 1:

```
Input: "()"
Output: true
```

Example 2:

```
Input: "()[]{}"
Output: true
```

Example 3:

```
Input: "(]"
Output: false
```

Example 4:

```
Input: "([)]"
Output: false
```

Example 5:

```
Input: "{[]}"
Output: true
```

---

155. min-stack

Design a stack that supports push, pop, top, and retrieving the minimum element in constant time.

- push(x) -- Push element x onto stack.
- pop() -- Removes the element on top of the stack.
- top() -- Get the top element.
- getMin() -- Retrieve the minimum element in the stack.

Example:

```
MinStack minStack = new MinStack();
minStack.push(-2);
minStack.push(0);
minStack.push(-3);
minStack.getMin();   --> Returns -3.
minStack.pop();
minStack.top();      --> Returns 0.
minStack.getMin();   --> Returns -2.
```

---

316. remove-duplicate-letters

Given a string which contains only lowercase letters, remove duplicate letters so that every letter appear once and only once. You must make sure your result is the smallest in lexicographical order among all possible results.

Example 1:

```
Input: "bcabc"
Output: "abc"
```

Example 2:

```
Input: "cbacdcbc"
Output: "acdb"
```

---

341. flatten-nested-list-iterator

341. flatten-nested-list-iterator

Given a nested list of integers, implement an iterator to flatten it.

Each element is either an integer, or a list -- whose elements may also be integers or other lists.

Example 1:

```
Input: [[1,1],2,[1,1]]
Output: [1,1,2,1,1]
Explanation: By calling next repeatedly until hasNext returns false,
             the order of elements returned by next should be: [1,1,2,1,1].
```

Example 2:

```
Input: [1,[4,[6]]]
Output: [1,4,6]
Explanation: By calling next repeatedly until hasNext returns false,
             the order of elements returned by next should be: [1,4,6].
```

---

224. basic-calculator

Implement a basic calculator to evaluate a simple expression string.

The expression string may contain open `(` and closing parentheses `)`, the plus `+` or minus sign `-`, non-negative integers and empty spaces ` `.

Example 1:

```
Input: "1 + 1"
Output: 2
```

Example 2:

```
Input: " 2-1 + 2 "
Output: 3
```

Example 3:

```
Input: "(1+(4+5+2)-3)+(6+8)"
Output: 23
```

Note:

- You may assume that the given expression is always valid.
- Do not use the `eval` built-in library function.