Dynamic Programming

Fibonacci / 1-Dimensional style

Min Cost Climbing Stairs (LC 746)

746. Min Cost Climbing Stairs
Easy

You are given an integer array cost where `cost[i]` is the cost of `i`th step on a staircase. Once you pay the cost, you can either climb one or two steps.

You can either start from the step with index 0, or the step with index 1.

Return the minimum cost to reach the top of the floor.

Example 1:

```
Input: cost = [10,15,20]
Output: 15
Explanation: You will start at index 1.
- Pay 15 and climb two steps to reach the top.
The total cost is 15.
```

Example 2:

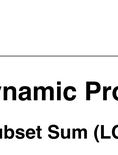
```
Input: cost = [1,100,1,1,1,100,1,1,100,1]
Output: 6
Explanation: You will start at index 0.
- Pay 1 and climb two steps to reach index 2.
- Pay 1 and climb two steps to reach index 4.
- Pay 1 and climb two steps to reach index 6.
- Pay 1 and climb one step to reach index 7.
- Pay 1 and climb two steps to reach index 9.
- Pay 1 and climb one step to reach the top.
The total cost is 6.
```


```
import collections
class Solution:
    def minCostClimbingStairs(self, cost: List[int]) -> int:
        for i in range(len(cost) - 3, -1, -1):
            cost[i] += min(cost[i + 1], cost[i + 2])

        return min(cost[0], cost[1])
```

example 2:
cost = [1,100,1,1,1,100,1,1,100,1]

```
    len(cost) -> 10
    for i in range(7, -1, -1)
i = 7 { cost[i] = min(cost[i+1], cost[i+2])
      {[1,100,1,1,1,100,1,2,100,1]

i = 6 { cost[i] = min(cost[i+1], cost[i+2])
      {[1,100,1,1,1,100,2,2,100,1]
                3      3
i = 5 { cost[i] = min(cost[i+1], cost[i+2])
      {[1,100,1,1,1,102,2,2,100,1]
              min      3
i = 4 { cost[i] = min(cost[i+1], cost[i+2])
      {[1,100,1,1,4,102,2,2,100,1]
        .
        .
        .
```

cost[0] because it records the smallest (minimum) between cost[i+1], cost[i+2] is the smallest possible value



House Robber (LC 198)

198. House Robber
Medium

You are a professional robber planning to rob houses along a street. Each house has a certain amount of money stashed, the only constraint stopping you from robbing each of them is that adjacent houses have security systems connected and it will automatically contact the police if two adjacent houses were broken into on the same night.

Given an integer array nums representing the amount of money of each house, return the maximum amount of money you can rob tonight without alerting the police.

Example 1:

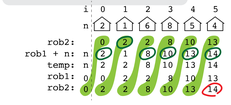
```
Input: nums = [1,2,3,1]
Output: 4
Explanation: Rob house 1 (money = 1) and then rob
house 3 (money = 3).
Total amount you can rob = 1 + 3 = 4.
```

Example 2:

```
Input: nums = [2,7,9,3,1]
Output: 12
Explanation: Rob house 1 (money = 2), rob house 3
(money = 9) and rob house 5 (money = 1).
Total amount you can rob = 2 + 9 + 1 = 12..
```

```
Q1: houses- [2, 1, 1, 2];       solution- 4
Q2: houses- [2, 7, 9, 3, 1];    solution- 12
Q3: houses- [2, 1, 6, 8, 5, 4]; solution- 14
Q4: houses- [2, 1, 8, 6, 5, 4]; solution- 15
```

```
class Solution:
    def rob(self, nums: List[int]) -> int:
        rob1, rob2 = 0, 0
        for n in nums:
            temp = max(n + rob1, rob2)
            rob1 = rob2
            rob2 = temp
        return rob2
```



1. declare two variables that keep track of previous values
2. alternate the values between the two variables to ensure they are not adjacent

House Robber II (LC 213)

213. House Robber II
Medium

You are a professional robber planning to rob houses along a street. Each house has a certain amount of money stashed. All houses at this place are arranged in a circle. That means the first house is the neighbor of the last one. Meanwhile, adjacent houses have a security system connected, and it will automatically contact the police if two adjacent houses were broken into on the same night.

Given an integer array nums representing the amount of money of each house, return the maximum amount of money you can rob tonight without alerting the police.

Example 1:


```
Input: nums = [2,3,2]
Output: 3
Explanation: You cannot rob house 1 (money = 2) and
then rob house 3 (money = 2), because they are
adjacent houses.
```

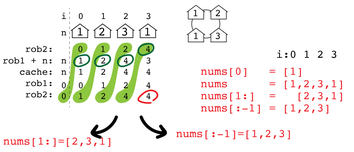
Example 2:


```
Input: nums = [1,2,3,1]
Output: 4
Explanation: Rob house 1 (money = 1) and then rob
house 3 (money = 3).
Total amount you can rob = 1 + 3 = 4.
```

Example 3:

```
Input: nums = [1,2,3]
Output: 3
```

```
class Solution:
    def rob(self, nums: List[int]) -> int:
        return max(nums[0], self.analysis(nums[1:]), self.analysis(nums[:-1]))

    def analysis(self, nums):
        rob1, rob2 = 0, 0
        for n in nums:
            cache = max(rob1 + n, rob2)
            rob1 = rob2
            rob2 = cache
        return rob2
```




```
i:0 1 2 3
nums[0]   = [1]
nums      = [1,2,3,1]
nums[1:]  =   [2,3,1]
nums[:-1] = [1,2,3]
```

perform the 'house robber' algorithm on two parts of the nums list, that either (1) excludes the first number, or (2) excludes the last number

---

Knapsack Dynamic Programming

Partition Equal Subset Sum (LC 416)

416. Partition Equal Subset Sum
Medium

Given an integer array nums, return true if you can partition the array into two subsets such that the sum of the elements in both subsets is equal or false otherwise.

Example 1:

```
Input: nums = [1,5,11,5]
Output: true
Explanation: The array can be partitioned as
[1, 5, 5] and [11].
```

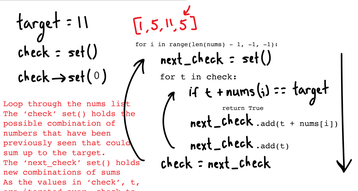
Example 2:

```
Input: nums = [1,2,3,5]
Output: false
Explanation: The array cannot be partitioned
into equal sum subsets.
```

Constraints:

```
1 <= nums.length <= 200
1 <= nums[i] <= 100
```

```
from typing import List

class Solution:
    def canPartition(self, nums: List[int]) -> bool:
        if sum(nums) % 2 != 0:
            return False

        target = sum(nums) // 2
        check = set()
        check.add(0)

        for i in range(len(nums) - 1, -1, -1):
            next_check = set()
            for t in check:
                if (t + nums[i]) == target:
                    return True
                next_check.add(t + nums[i])
                next_check.add(t)
            check = next_check
        return False
```



1. Loop through the nums list.
2. The 'check' set() holds the possible combination of numbers that have been previously seen that could sum up to the target.
3. The 'next_check' set() holds new combinations of sums
4. As the values in 'check', t, are iterated over, check to see if nums[i] + t is equal to target. If so, return True (otherwise, if the loop completes, return False)

Decode Ways (LC 91)

Dynamic Programming on Strings

Longest Palindromic Substring (LC 5)

5. Longest Palindromic Substring
Medium

Given a string s, return the longest palindromic substring in s.

Example 1:

```
Input: s = "babad"
Output: "bab"
Explanation: "aba" is also a valid answer.
```

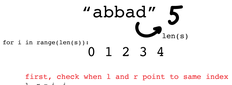
Example 2:

```
Input: s = "cbbd"
Output: "bb"
```

```
from typing import List

class Solution:
    def longestPalindrome(self, s: str) -> str:
        result = ""

        for i in range(len(s)):
            l, r = i, i
            while l >= 0 and r < len(s) and s[l] == s[r]:
                if (r - l + 1) > len(result):
                    result = s[l:r + 1]
                l -= 1
                r += 1

            l, r = i, i + 1
            while l >= 0 and r < len(s) and s[l] == s[r]:
                if (r - l + 1) > len(result):
                    result = s[l:r + 1]
                l -= 1
                r += 1
        return result
```

simplified

```
from typing import List

class Solution:
    def longestPalindrome(self, s: str) -> str:
        result = ""

        for i in range(len(s)):
            for l, r in ((i,i), (i,i+1)):
                while l >= 0 and r < len(s) and s[l] == s[r]:
                    if (r - l + 1) > len(result):
                        result = s[l:r + 1]
                    l -= 1
                    r += 1

        return result
```



first, check when l and r point to same index

```
l, r = i, i
while l >= 0 and r < len(s) and s[l] == s[r]:
```

l and r pointers are in bound, and s[l] and s[r] are palindromes

(r-l+1) is the size of the substring, if (r-l+1) is greater than length of result, then save the new result

```
    if (r - l + 1) > len(result):
        result = s[l:r + 1]
    l -= 1
    r += 1
```

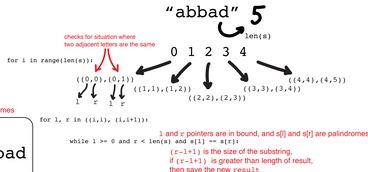
next, check when l and r point to adjacent index

```
l, r = i, i + 1
while l >= 0 and r < len(s) and s[l] == s[r]:
```

l and r pointers are in bound, and s[l] and s[r] are palindromes

(r-l+1) is the size of the substring, if (r-l+1) is greater than length of result, then save the new result

```
    if (r - l + 1) > len(result):
        result = s[l:r + 1]
    l -= 1
    r += 1
```

l and r pointers are in bound, and s[l] and s[r] are palindromes

(r-l+1) is the size of the substring, if (r-l+1) is greater than length of result, then save the new result

```
while l >= 0 and r < len(s) and s[l] == s[r]:
    if (r - l + 1) > len(result):
        result = s[l:r + 1]
    l -= 1
    r += 1
```

Palindromic Substring (LC 647)

647. Palindromic Substrings
Medium

Given a string s, return the number of palindromic substrings in it.

A string is a palindrome when it reads the same backward as forward.

A substring is a contiguous sequence of characters within the string.

Example 1:

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

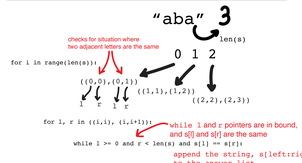
Example 2:

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

Constraints:

```
1 <= s.length <= 1000
s consists of lowercase English letters.
```

```
class Solution:
    def countSubstrings(self, s: str) -> int:
        answer = []

        for i in range(len(s)):
            for left, right in ((i,i), (i,i+1)):
                while left >= 0 and right < len(s) and s[left] == s[right]:
                    answer.append(s[left:right+1])
                    left -= 1
                    right += 1

        return len(answer)
```



while l and r pointers are in bound, and s[l] and s[r] are the same

append the string, s[left:right+1] to the answer list

```
    answer.append(s[left:right+1])
    l -= 1
    r += 1
```

1. Loop through the string, s.
2. There are two parts:
   a. set left (l) and right (r) pointers the same as i, expand left (l-=1) and right (r+=1), and check to see if the s[l] and s[r] are the same.
   b. set left (l) as i, and right (r) as i + 1, so that it accounts for situations where there are two adjacent characters that are the same, and expand left(l-=1) and right (r+=1), and check to see if the s[l] and s[r] are the same.

General 1D Dynamic Programming

Decode Ways (LC 91)