Problem List

Run Submit

Description | Editorial | Solutions | Submissions

70. Climbing Stairs

Easy Topics Companies Hint

You are climbing a staircase. It takes `n` steps to reach the top.

Each time you can either climb `1` or `2` steps. In how many distinct ways can you climb to the top?

Example 1:

```
Input: n = 2
Output: 2
Explanation: There are two ways to climb to the top.
1. 1 step + 1 step
2. 2 steps
```

Example 2:

```
Input: n = 3
Output: 3
Explanation: There are three ways to climb to the top.
1. 1 step + 1 step + 1 step
2. 1 step + 2 steps
3. 2 steps + 1 step
```

Constraints:

- `1 <= n <= 45`

Code

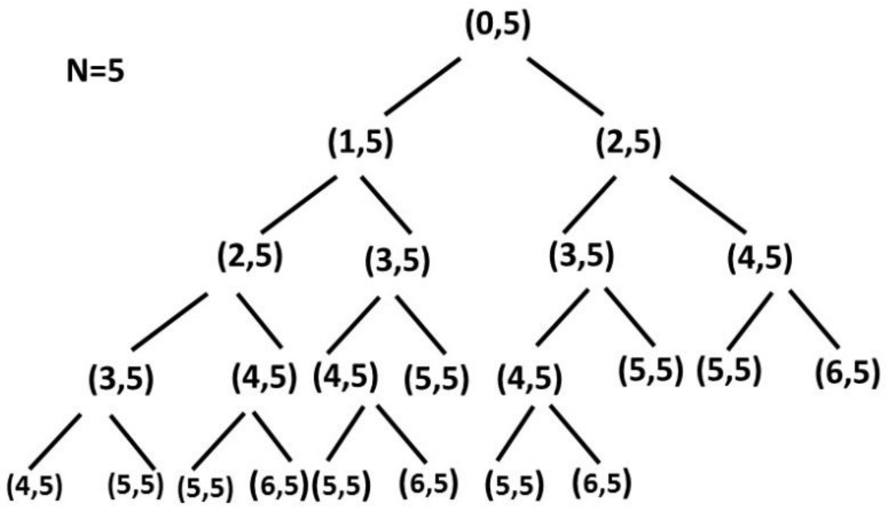
C++ • Auto

```cpp
class Solution {
public:
    int climbStairs(int n) {

    }
};
```

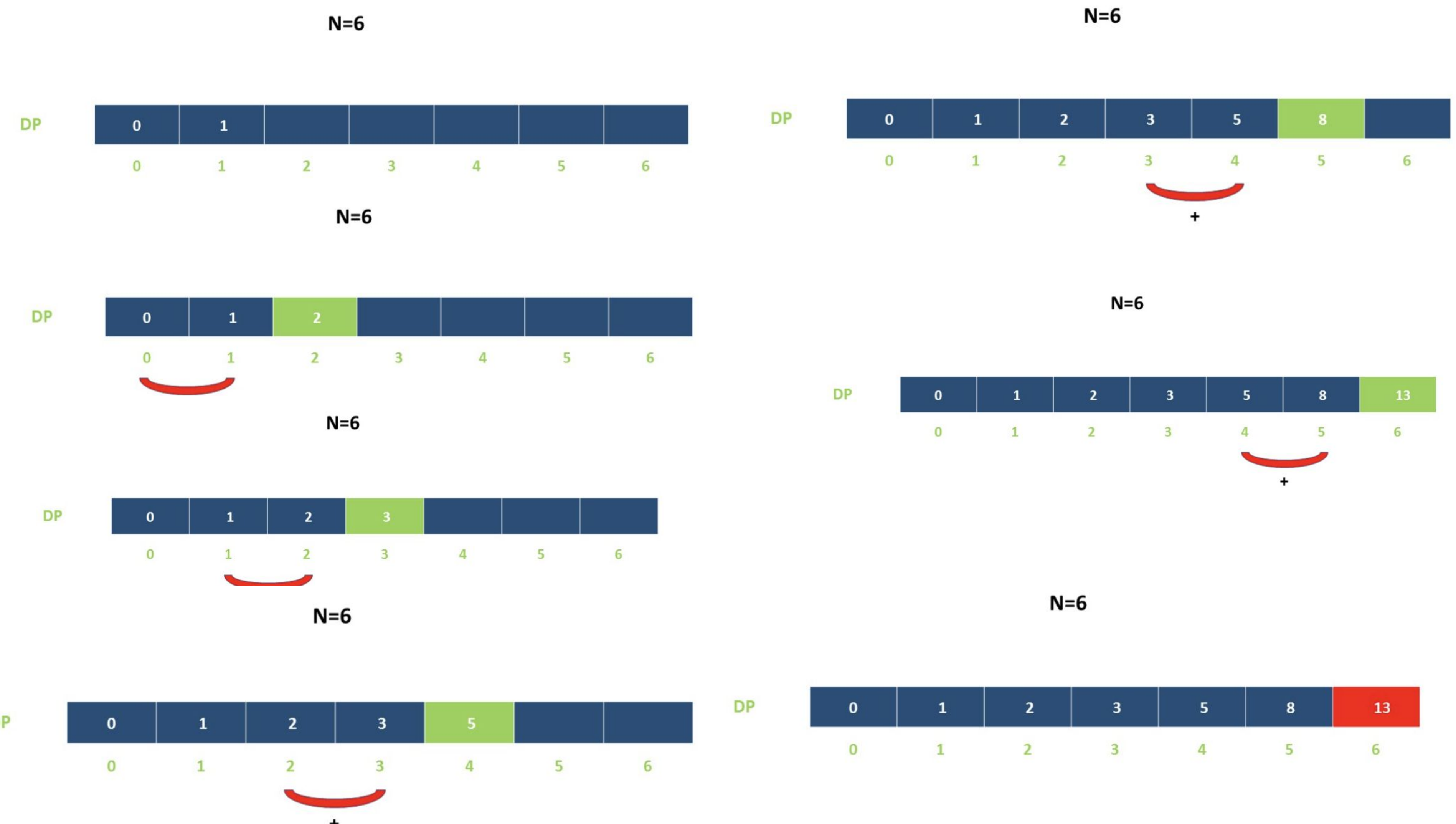
Approach 1: Brute Force

```
# Python3
class Solution:
    def climbStairs(self, n: int) -> int:
        return self.climb_Stairs(0, n)

    def climb_Stairs(self, i: int, n: int) -> int:
        if i > n:
            return 0
        if i == n:
            return 1
        return self.climb_Stairs(i + 1, n) + self.climb_Stairs(i + 2, n)
```

- Time complexity : $O(2^n)$.



Number of Nodes = $O(2^n)$


N=6
DP
0 1
0 1 2 3 4 5 6
N=6
DP
0 1 2
0 1 2 3 4 5 6
N=6
DP
0 1 2 3
0 1 2 3 4 5 6
N=6
P
0 1 2 3 5
0 1 2 3 4 5 6
+
N=6
DP
0 1 2 3 5 8
0 1 2 3 4 5 6
+
N=6
DP
0 1 2 3 5 8 13
0 1 2 3 4 5 6
+
N=6
DP
0 1 2 3 5 8 13
0 1 2 3 4 5 6

Problem List

Run Submit

0 Premium

Description | Editorial | Solutions | Submissions

70. Climbing Stairs

Solved

Easy Topics Companies Hint

You are climbing a staircase. It takes `n` steps to reach the top.

Each time you can either climb `1` or `2` steps. In how many distinct ways can you climb to the top?

Example 1:

```
Input: n = 2
Output: 2
Explanation: There are two ways to climb to the top.
1. 1 step + 1 step
2. 2 steps
```

Example 2:

```
Input: n = 3
Output: 3
Explanation: There are three ways to climb to the top.
1. 1 step + 1 step + 1 step
2. 1 step + 2 steps
3. 2 steps + 1 step
```

Code

Python3 Auto

```python
# Python3
class Solution:
    def climbStairs(self, n: int) -> int:
        if n == 1:
            return 1
        dp = [0 for _ in range(n + 1)]
        dp[1] = 1
        dp[2] = 2
        for i in range(3, n + 1):
            dp[i] = dp[i - 1] + dp[i - 2]
        return dp[n]
```

Description | Editorial | Solutions | Submissions

300. Longest Increasing Subsequence

Medium | Topics | Companies

Given an integer array `nums`, return *the length of the longest* ***strictly increasing subsequence***.

Example 1:

```
Input: nums = [10,9,2,5,3,7,101,18]
Output: 4
Explanation: The longest increasing subsequence is [2,3,7,101], therefore the length is 4.
```

Example 2:

```
Input: nums = [0,1,0,3,2,3]
Output: 4
```

Example 3:

```
Input: nums = [7,7,7,7,7,7,7]
Output: 1
```

Constraints:

- `1 <= nums.length <= 2500`
- $-10^4 \le nums[i] \le 10^4$

Follow up: Can you come up with an algorithm that runs in `O(n log(n))` time complexity?

Code

C++ • Auto

```cpp
class Solution {
public:
    int lengthOfLIS(vector<int>& nums) {
        
    }
};
```

Saved

Ln 1, Col 1

nums

10	9	2	5	3	7

dp

1	1	1	1	1	1

dp[i] represents the length of the longest increasing subsequence that ends at index i

A Framework to Solve Dynamic Programming Problems

1. First, we need some function or array that represents the answer to the problem from a given state
2. Second, we need a way to transition between states, such as dp[5] and dp[7]. This is called a recurrence relation
3. The third component is: we need a base case. For this problem, we can initialize every element of dp to 1, since every element on its own is technically an increasing subsequence.

Description | Editorial | Solutions | Submissions

300. Longest Increasing Subsequence

Solved

Medium | Topics | Companies

Given an integer array `nums`, return *the length of the longest* ***strictly increasing subsequence***.

Example 1:

```
Input: nums = [10,9,2,5,3,7,101,18]
Output: 4
Explanation: The longest increasing subsequence is [2,3,7,101], therefore the length is 4.
```

Example 2:

```
Input: nums = [0,1,0,3,2,3]
Output: 4
```

Example 3:

```
Input: nums = [7,7,7,7,7,7,7]
Output: 1
```

Constraints:

- `1 <= nums.length <= 2500`
- $-10^4 <= nums[i] <= 10^4$

Follow up: Can you come up with an algorithm that runs in `O(n log(n))` time complexity?

Code

Python3 | Auto

```python
class Solution:
    def lengthOfLIS(self, nums: List[int]) -> int:
        dp = [1] * len(nums)
        for i in range(1, len(nums)):
            for j in range(i):
                if nums[i] > nums[j]:
                    dp[i] = max(dp[i], dp[j] + 1)

        return max(dp)
```

Problem List

Run Submit

Description | Editorial | Solutions | Submissions

62. Unique Paths

Solved

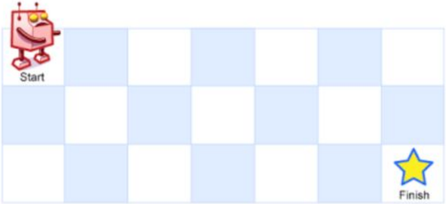
Medium | Topics | Companies

There is a robot on an `m x n` grid. The robot is initially located at the **top-left corner** (i.e., `grid[0][0]`). The robot tries to move to the **bottom-right corner** (i.e., `grid[m - 1][n - 1]`). The robot can only move either down or right at any point in time.

Given the two integers `m` and `n`, return *the number of possible unique paths that the robot can take to reach the bottom-right corner*.

The test cases are generated so that the answer will be less than or equal to $2 * 10^9$.

Example 1:




```
Input: m = 3, n = 7
Output: 28
```

Example 2:

```
Input: m = 3, n = 2
Output: 3
Explanation: From the top-left corner, there are a total of 3 ways to reach the bottom-right corner:
1. Right -> Down -> Down
2. Down -> Down -> Right
3. Down -> Right -> Down
```

Code

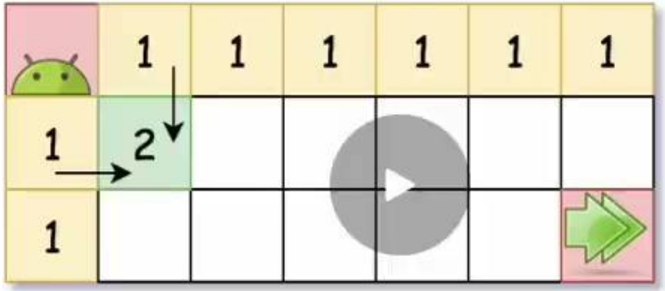
C++ • Auto

```cpp
class Solution {
public:
    int uniquePaths(int m, int n) {

    }
};
```

Saved

Ln 1, Col 1

$$d[1][1] = d[0][1] + d[1][0] = 2$$



Description | Editorial | Solutions | Submissions

62. Unique Paths

Solved

Medium | Topics | Companies

There is a robot on an `m x n` grid. The robot is initially located at the **top-left corner** (i.e., `grid[0][0]`). The robot tries to move to the **bottom-right corner** (i.e., `grid[m - 1][n - 1]`). The robot can only move either down or right at any point in time.

Given the two integers `m` and `n`, return *the number of possible unique paths that the robot can take to reach the bottom-right corner*.

The test cases are generated so that the answer will be less than or equal to $2 * 10^9$.

Example 1:


```
Input: m = 3, n = 7
Output: 28
```

Example 2:

```
Input: m = 3, n = 2
Output: 3
Explanation: From the top-left corner, there are a total of 3 ways to reach the
bottom-right corner:
1. Right -> Down -> Down
2. Down -> Down -> Right
3. Down -> Right -> Down
```

Code

Python3 | Auto

```python
class Solution:
    def uniquePaths(self, m: int, n: int) -> int:
        d = [[1] * n for _ in range(m)]

        for col in range(1, m):
            for row in range(1, n):
                d[col][row] = d[col - 1][row] + d[col][row - 1]

        return d[m - 1][n - 1]
```

Saved | Ln 9, Col 31

Testcase | >_ Test Result

Backtracking

Run Submit

0

Description | Editorial | Solutions | Submissions

17. Letter Combinations of a Phone Number

Solved

Medium | Topics | Companies

Given a string containing digits from `2-9` inclusive, return all possible letter combinations that the number could represent. Return the answer in **any order**.

A mapping of digits to letters (just like on the telephone buttons) is given below. Note that 1 does not map to any letters.

Example 1:

```
Input: digits = "23"
Output: ["ad","ae","af","bd","be","bf","cd","ce","cf"]
```

Example 2:

```
Input: digits = ""
Output: []
```

Example 3:

```
Input: digits = "2"
Output: ["a","b","c"]
```

Code

C++ • Auto

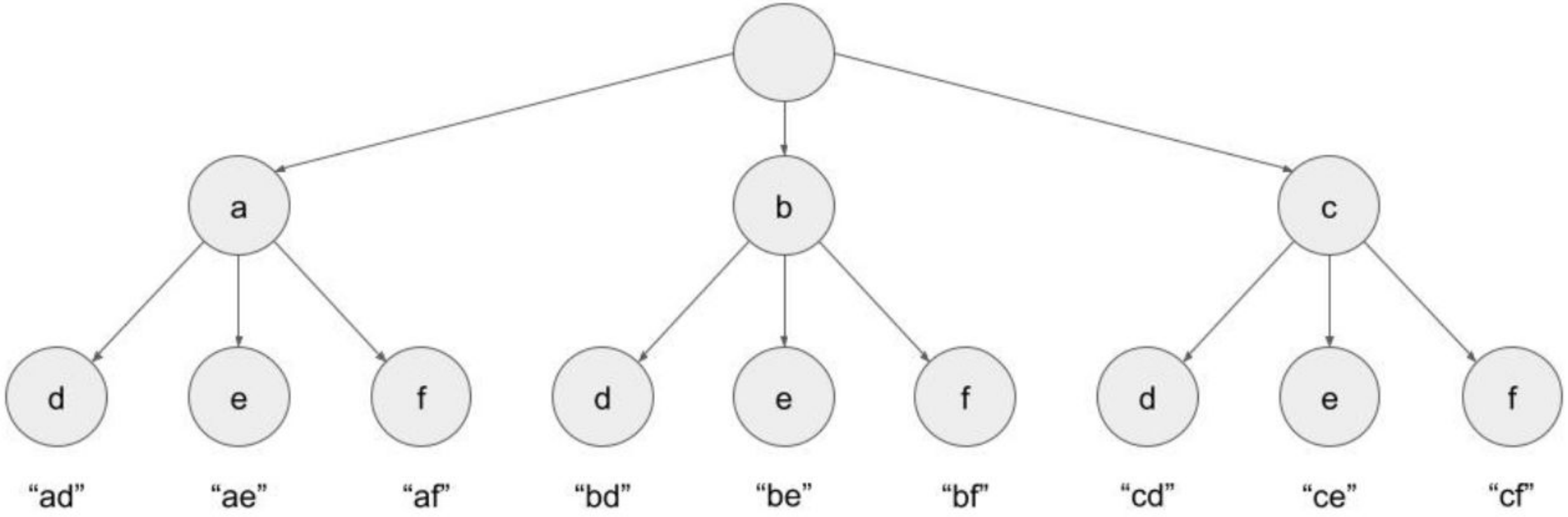
```
class Solution {
public:
    vector<string> letterCombinations(string digits) {
        
    }
};
```

Saved

Ln 1, Col 1

digits = “23”

2 → “abc”, 3 → “def”



“ad”	“ae”	“af”	“bd”	“be”	“bf"	“cd”	“ce”	“cf”

Problem List

Run Submit

0 Premium

Description | Editorial | Solutions | Submissions

17. Letter Combinations of a Phone Number

Solved

Medium Topics Companies

Given a string containing digits from `2-9` inclusive, return all possible letter combinations that the number could represent. Return the answer in **any order**.

A mapping of digits to letters (just like on the telephone buttons) is given below. Note that 1 does not map to any letters.

Example 1:

```
Input: digits = "23"
Output: ["ad","ae","af","bd","be","bf","cd","ce","cf"]
```

Example 2:

```
Input: digits = ""
Output: []
```

Example 3:

```
Input: digits = "2"
Output: ["a","b","c"]
```

Code

Python3 Auto

```python
class Solution:
    def letterCombinations(self, digits: str) -> List[str]:
        # If the input is empty, immediately return an empty answer array
        if len(digits) == 0:
            return []
        # Map all the digits to their corresponding letters
        letters = {
            "2": "abc",
            "3": "def",
            "4": "ghi",
            "5": "jkl",
            "6": "mno",
            "7": "pqrs",
            "8": "tuv",
            "9": "wxyz",
        }
        def backtrack(index, path):
            # If the path is the same length as digits, we have a complete combination
            if len(path) == len(digits):
                combinations.append("".join(path))
                return  # Backtrack
            # Get the letters that the current digit maps to, and loop through them
            possible_letters = letters[digits[index]]
            for letter in possible_letters:
                # Add the letter to our current path
                path.append(letter)
                # Move on to the next digit
                backtrack(index + 1, path)
                # Backtrack by removing the letter before moving onto the next
                path.pop()
        # Initiate backtracking with an empty path and starting index of 0
        combinations = []
        backtrack(0, [])
        return combinations
```

Saved

Ln 30, Col 27

Backtracking

Run Submit

Description | Editorial | Solutions | Submissions

Code

C++ • Auto

90. Subsets II

Solved

Medium Topics Companies

Given an integer array `nums` that may contain duplicates, return *all possible* subsets *(the power set)*.

The solution set **must not** contain duplicate subsets. Return the solution in **any order**.

Example 1:

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

Example 2:

```
Input: nums = [0]
Output: [[],[0]]
```

Constraints:

- `1 <= nums.length <= 10`
- `-10 <= nums[i] <= 10`

Seen this question in a real interview before? 1/5

Yes No

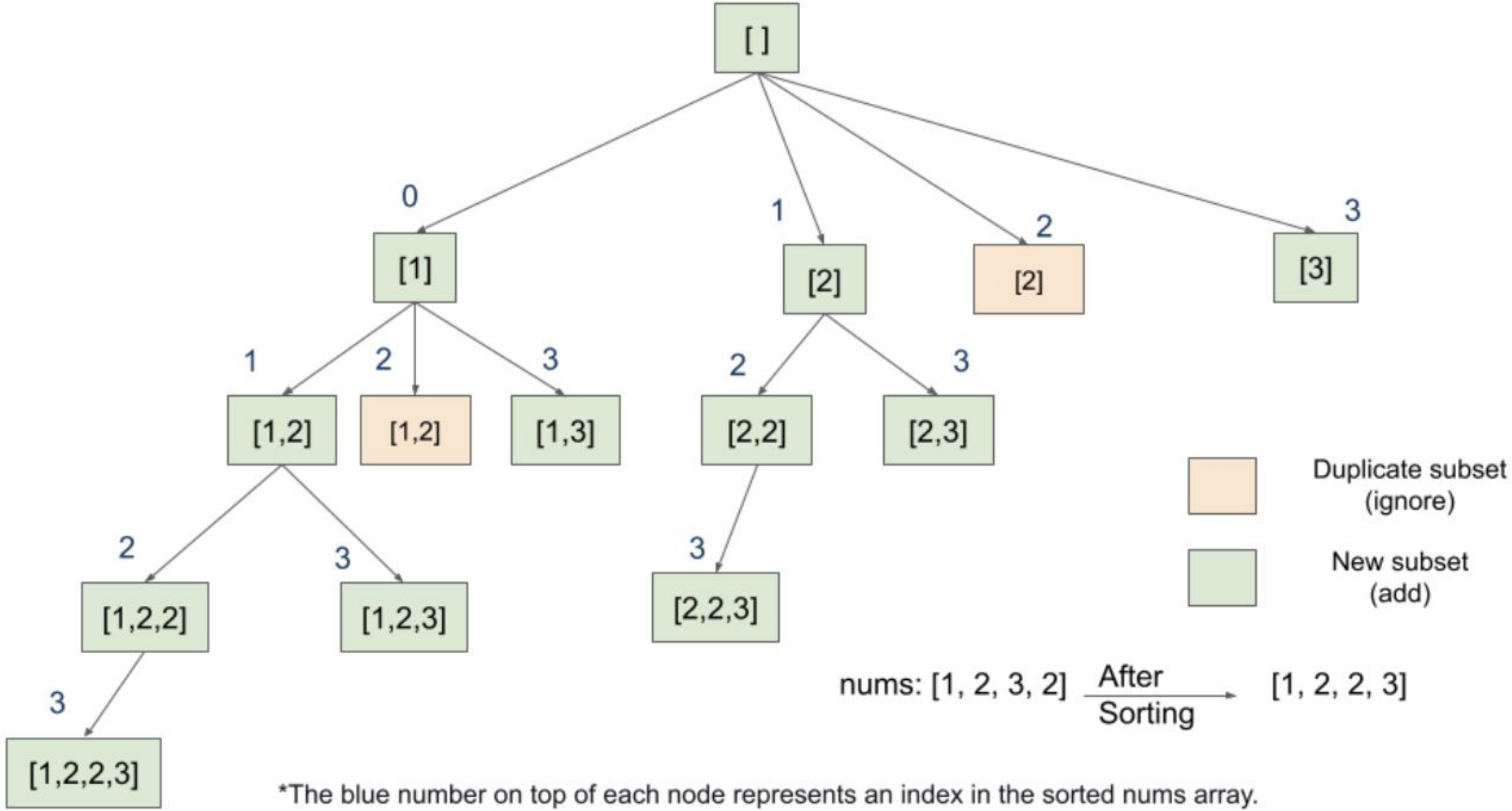
Accepted 1.1M | Submissions 1.8M | Acceptance Rate 58.4%

Topics

10K | 94 | 61 Online

Saved | Ln 1, Col 1

Testcase | Test Result


[ ]
0
1
2
3
[1]
[2]
[2]
[3]
1
2
3
2
3
[1,2]
[1,2]
[1,3]
[2,2]
[2,3]
Duplicate subset (ignore)
New subset (add)
2
3
3
[1,2,2]
[1,2,3]
[2,2,3]
3
[1,2,2,3]
nums: [1, 2, 3, 2]
After Sorting
[1, 2, 2, 3]
*The blue number on top of each node represents an index in the sorted nums array.

Problem List Run Submit 0 Premium

Description | Accepted | Editorial | Solutions | Submissions

90. Subsets II

Solved

Medium Topics Companies

Given an integer array `nums` that may contain duplicates, return *all possible* subsets *(the power set)*.

The solution set **must not** contain duplicate subsets. Return the solution in **any order**.

Example 1:

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

Example 2:

```
Input: nums = [0]
Output: [[],[0]]
```

Constraints:

- `1 <= nums.length <= 10`
- `-10 <= nums[i] <= 10`

Seen this question in a real interview before? 1/5

Yes No

Accepted 1.1M | Submissions 1.8M | Acceptance Rate 58.4%

Topics

Code

Python3 Auto

```python
class Solution:
    def subsetsWithDup(self, nums):
        nums.sort()
        subsets = []
        currentSubset = []
        self.subsetsWithDupHelper(subsets, currentSubset, nums, 0)
        return subsets

    def subsetsWithDupHelper(self, subsets, currentSubset, nums, index):
        # Add the subset formed so far to the subsets list.
        subsets.append(list(currentSubset))
        for i in range(index, len(nums)):
            # If the current element is a duplicate, ignore.
            if i != index and nums[i] == nums[i - 1]:
                continue
            currentSubset.append(nums[i])
            self.subsetsWithDupHelper(subsets, currentSubset, nums, i + 1)
            currentSubset.pop()
```

Saved Ln 18, Col 32

Description | Editorial | Solutions | Submissions

139. Word Break

Medium | Topics | Companies

Given a string `s` and a dictionary of strings `wordDict`, return `true` if `s` can be segmented into a space-separated sequence of one or more dictionary words.

Note that the same word in the dictionary may be reused multiple times in the segmentation.

Example 1:

```
Input: s = "leetcode", wordDict = ["leet","code"]
Output: true
Explanation: Return true because "leetcode" can be segmented as "leet code".
```

Example 2:

```
Input: s = "applepenapple", wordDict = ["apple","pen"]
Output: true
Explanation: Return true because "applepenapple" can be segmented as "apple pen apple".
Note that you are allowed to reuse a dictionary word.
```

Example 3:

```
Input: s = "catsandog", wordDict = ["cats","dog","sand","and","cat"]
Output: false
```

Constraints:

- `1 <= s.length <= 300`
- `1 <= wordDict.length <= 1000`
- `1 <= wordDict[i].length <= 20`
- `s` and `wordDict[i]` consist of only lowercase English letters.

Code

C++ • Auto

```cpp
class Solution {
public:
    bool wordBreak(string s, vector<string>& wordDict) {
        
    }
};
```

1.

dp

F	F	F	F	F	F	F	F
l	e	e	t	c	o	d	e

s = “leetcode”
wordDict = [“leet”, “code”]

The table holds boolean values T (true) and F (false). Under each element is the corresponding letter from the input string.

dp(i) indicates if it is possible to use words from wordDict to build the input string up to index i.

2.

dp

F	F	F	T	F	F	F	F
l	**e**	**e**	**t**	c	o	d	e

s = “leetcode”
wordDict = [“leet”, “code”]

The first criteria is that a word from wordDict can end at s[i]. The first occurrence of this is at index 3. The word “leet” can end here.

The second criteria is that the input string is formable up to the point before where the current string starts. Since this is the first word being used, the criteria is also met.

Both criteria are satisfied, dp(3) = true.

3.

dp

F	F	F	T	F	F	F	T
l	e	e	t	**c**	**o**	**d**	**e**

s = “leetcode”
wordDict = [“leet”, “code”]

The next time that the first criteria is satisfied is at index 7. The word “code” can end here.

The second criteria is also satisfied - the word “code” starts at index 4. The index before that, 3, is also “true”. Therefore, dp(7) = true.

Since this is also the last index, the answer to the problem is “true”.

Problem List Run Submit Premium

Description | Editorial | Solutions | Submissions

139. Word Break

Solved

Medium | Topics | Companies

Given a string `s` and a dictionary of strings `wordDict`, return `true` if `s` can be segmented into a space-separated sequence of one or more dictionary words.

Note that the same word in the dictionary may be reused multiple times in the segmentation.

Example 1:

```
Input: s = "leetcode", wordDict = ["leet","code"]
Output: true
Explanation: Return true because "leetcode" can be segmented as "leet code".
```

Example 2:

```
Input: s = "applepenapple", wordDict = ["apple","pen"]
Output: true
Explanation: Return true because "applepenapple" can be segmented as "apple pen
apple".
Note that you are allowed to reuse a dictionary word.
```

Example 3:

```
Input: s = "catsandog", wordDict = ["cats","dog","sand","and","cat"]
Output: false
```

Constraints:

- `1 <= s.length <= 300`
- `1 <= wordDict.length <= 1000`
- `1 <= wordDict[i].length <= 20`

Code

Python3 | Auto

```python
class Solution:
    def wordBreak(self, s: str, wordDict: List[str]) -> bool:
        @cache
        def dp(i):
            if i < 0:
                return True

            for word in wordDict:
                if s[i - len(word) + 1 : i + 1] == word and dp(i - len(word)):
                    return True

            return False

        return dp(len(s) - 1)
```

Saved | Ln 14, Col 30

Testcase | Test Result