RECURSION/DFS

Question: Flood fill problem.

YOU HAVE 15 MINUTES

https://leetcode.com/problems/flood-fill/

1	1	1
1	1	0
0	0	0

2	2	2
2	2	0
0	0	0

1	1	1
1	1	0
0	0	0

2	1	1
1	1	0
0	0	0

2	2	1
2	1	0
0	0	0

2	2	2
2	2	0
0	0	0

```python
# Flood fill
class Solution(object):
    def utilDFS(self,image,r,c,newColor,color):
        if(image[r][c] == color):
            image[r][c] = newColor
            if(r>=1):
                self.utilDFS(image,r-1,c,newColor,color)
            if(c>=1):
                self.utilDFS(image,r,c-1,newColor,color)
            if(r<len(image) -1):
                self.utilDFS(image,r+1,c,newColor,color)
            if(c<len(image[0]) -1):
                self.utilDFS(image,r,c+1,newColor,color)

    def floodFill(self, image, sr, sc, newColor):
        color = image[sr][sc]
        if (color != newColor):
            self.utilDFS(image,sr,sc,newColor,color);
        return image
```

TIME COMPLEXITY - O(N), SPACE COMPLEXITY - O(N) [Call stack for DFS]

Question: Given a non-empty 2D array grid of 0's and 1's, an island is a group of 1's (representing land) connected 4-directionally (horizontal or vertical.) You may assume all four edges of the grid are surrounded by water.

YOU HAVE 15 MINUTES

https://leetcode.com/problems/max-area-of-island/

In the context of this question, what is the area of the island ?

0	0	1	0	0	0	0
1	1	1	0	0	0	0
0	0	1	0	1	1	1
0	0	0	0	0	0	0
0	0	1	1	0	0	0
0	1	0	0	1	0	0
0	0	0	1	0	0	0

In the context of this question, what is the area of the island ?

0	0	1	0	0	0	0
1	1	1	0	0	0	0
0	0	1	0	1	1	1
0	0	0	0	0	0	0
0	0	1	1	0	0	0
0	1	0	0	0	0	0
0	0	0	1	1	0	0

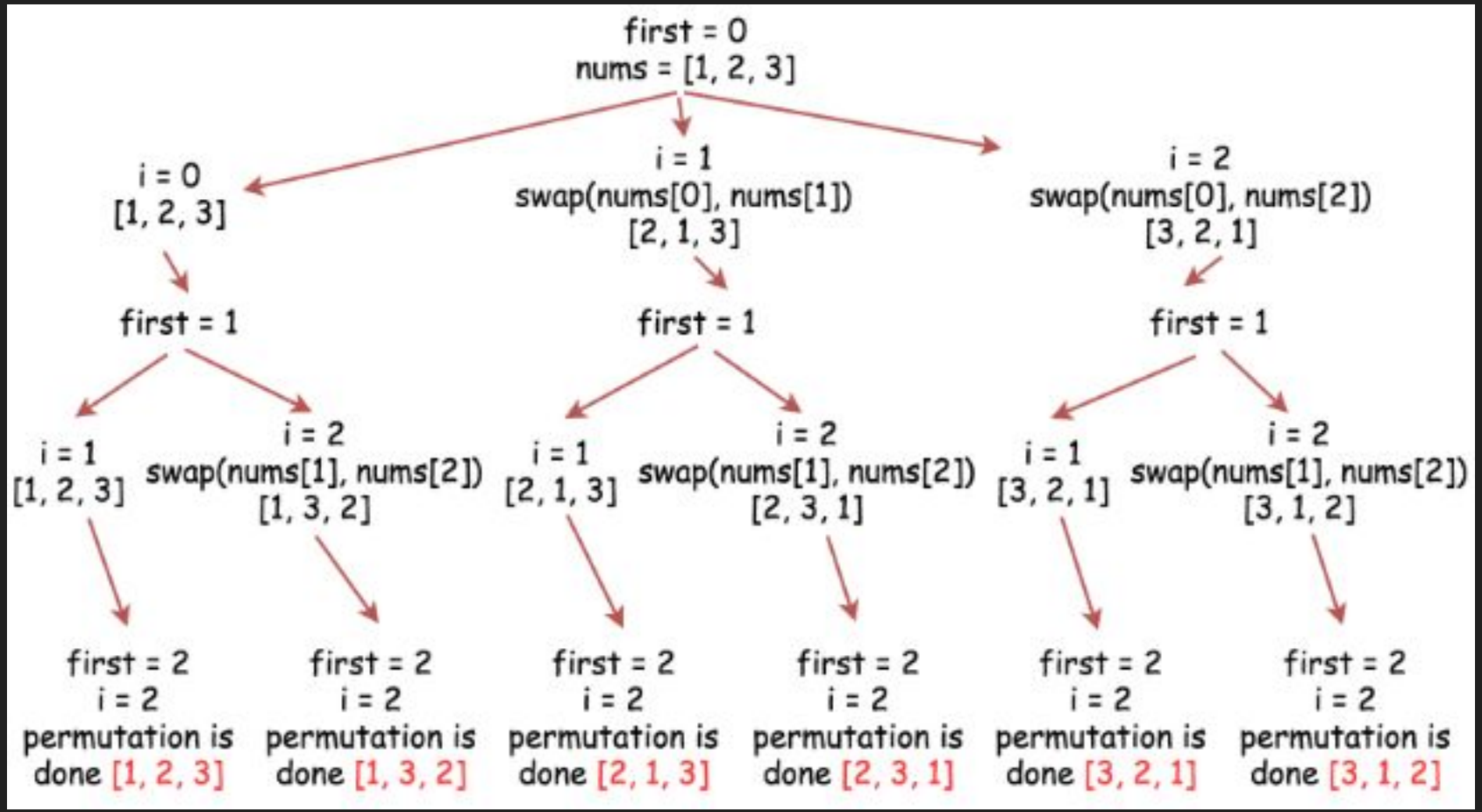
```python
# Max area of island
class Solution(object):
    def maxAreaOfIsland(self, grid):
        seen = set()
        def area(r, c):
            if not (0 <= r < len(grid) and 0 <= c < len(grid[0])
                    and (r, c) not in seen and grid[r][c]):
                return 0
            seen.add((r, c))
            return (1 + area(r+1, c) + area(r-1, c) +
                    area(r, c-1) + area(r, c+1))

        return max(area(r, c)
                   for r in range(len(grid))
                   for c in range(len(grid[0])))
```

TIME COMPLEXITY and SPACE COMPLEXITY - O(M*N), where M x N are dimensions

Question: Given a collection of distinct integers, return all possible permutations.

YOU HAVE 15 MINUTES

https://leetcode.com/problems/permutations/


first = 0
nums = [1, 2, 3]
i = 0
[1, 2, 3]
i = 1
swap(nums[0], nums[1])
[2, 1, 3]
i = 2
swap(nums[0], nums[2])
[3, 2, 1]
first = 1
first = 1
first = 1
i = 1
[1, 2, 3]
i = 2
swap(nums[1], nums[2])
[1, 3, 2]
i = 1
[2, 1, 3]
i = 2
swap(nums[1], nums[2])
[2, 3, 1]
i = 1
[3, 2, 1]
i = 2
swap(nums[1], nums[2])
[3, 1, 2]
first = 2
i = 2
permutation is done [1, 2, 3]
first = 2
i = 2
permutation is done [1, 3, 2]
first = 2
i = 2
permutation is done [2, 1, 3]
first = 2
i = 2
permutation is done [2, 3, 1]
first = 2
i = 2
permutation is done [3, 2, 1]
first = 2
i = 2
permutation is done [3, 1, 2]

```python
# Permutations
class Solution:
    def backtrack(self,nums,first,n,output):
        # if all integers are used up
        if first == n:
            output.append(nums[:])
        for i in range(first, n):
            # place i-th integer first
            # in the current permutation
            nums[first], nums[i] = nums[i], nums[first]
            # use next integers to complete the permutations
            self.backtrack(nums,first + 1,n,output)
            # backtrack
            nums[first], nums[i] = nums[i], nums[first]


    def permute(self, nums):
        """
        :type nums: List[int]
        :rtype: List[List[int]]
        """
        n = len(nums)
        output = []
        self.backtrack(nums,0,n,output)
        return output
```

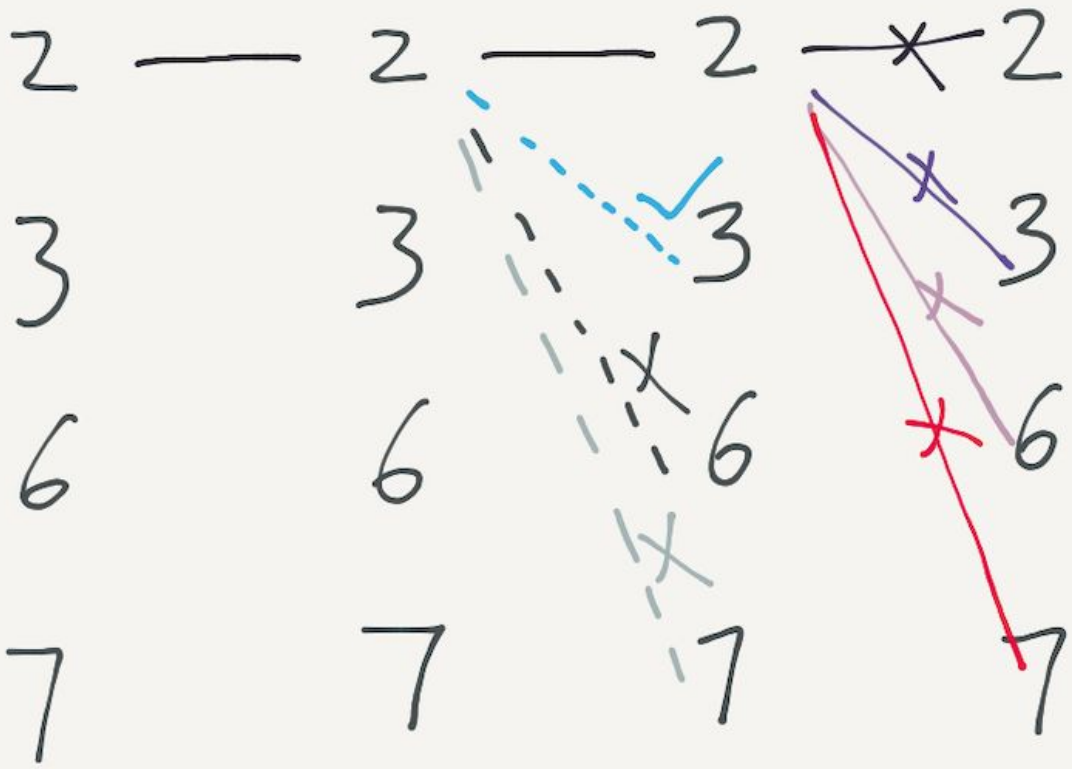
TIME COMPLEXITY - O(N*N!) and SPACE COMPLEXITY - O(N!)

Question: Given a set of candidate numbers (candidates) (without duplicates) and a target number (target), find all unique combinations in candidates where the candidate numbers sums to target.

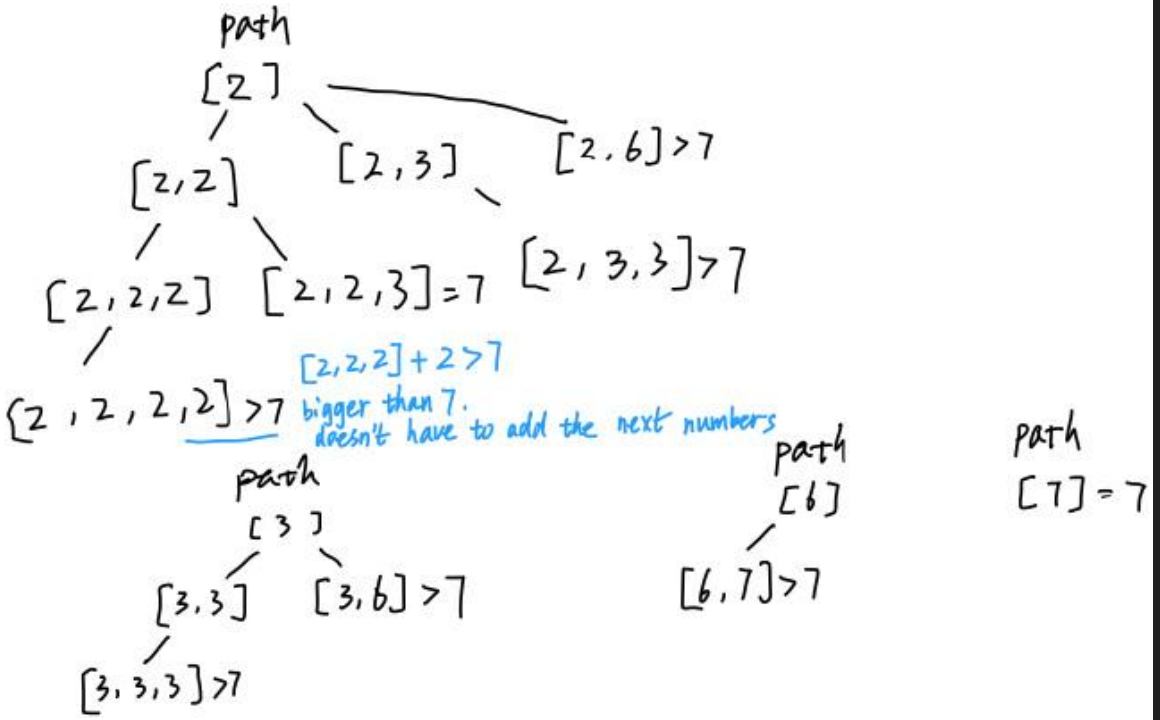
YOU HAVE 15 MINUTES

https://leetcode.com/problems/combination-sum/

Target = 7



candidates = [2, 3, 6, 7], Target = 7



⇒ res = [[2,2,3],[7]]

```python
# Combination Sum
class Solution:
    def recurse(self,curr_comb, curr_sum, j,T,C,sol):
            if curr_sum == T:
                sol.append(curr_comb.copy())
            elif curr_sum < T:
                for i in range(j, len(C)):
                    curr_comb.append(C[i])
                    self.recurse(curr_comb, curr_sum + C[i], i,T,C,sol)
                    curr_comb.pop()
    def combinationSum(self, C: List[int], T: int) -> List[List[int]]:
        sol = []
        self.recurse([], 0, 0,T,C,sol)
        return sol
```

[]
[2]
[2, 2]
[2, 2, 2]
[2, 2, 2, 2]
[2, 2, 2, 3]
[2, 2, 2, 6]
[2, 2, 2, 7]
[2, 2, 3]
[2, 2, 6]
[2, 2, 7]
[2, 3]
[2, 3, 3]
[2, 3, 6]
[2, 3, 7]
[2, 6]
[2, 7]
[3]
[3, 3]
[3, 3, 3]
[3, 3, 6]
[3, 3, 7]
[3, 6]
[3, 7]
[6]
[6, 6]
[6, 7]
[7]

Question: Given a 2D board and a word, find if the word exists in the grid.

YOU HAVE 15 MINUTES

https://leetcode.com/problems/word-search/

T	H	S	M	A	L	L	T	R	P	T	L	A
E	A	P	C	R	S	R	P	S	P	B	L	S
E	L	I	C	F	T	O	S	P	A	R	Q	H
N	I	H	D	E	T	S	E	R	I	U	V	C
N	B	C	D	W	U	S	J	J	I	Y	B	D
Y	M	A	E	S	Y	C	E	N	O	T	N	Y
P	I	E	T	G	N	L	N	G	T	D	S	J
P	S	C	U	D	U	E	G	C	A	A	G	G
O	T	G	G	C	B	W	U	W	J	E	J	S
I	Q	L	E	A	V	Q	K	Q	N	T	T	D
N	D	L	S	D	C	A	H	T	M	R	E	R
T	O	C	T	G	H	J	H	D	S	E	T	Y
M	G	M	I	J	R	T	Y	Y	U	I	O	P

```
[
  ['A','B','C','E'],
  ['S','F','C','S'],
  ['A','D','E','E']
]
```

word = `"ABCCED"`, -> returns `true`,

word = `"SEE"`, -> returns `true`,

word = `"ABCB"`, -> returns `false`.

```
[
  ['A','B','C','E'],
  ['S','F','C','S'],
  ['A','D','E','E']
]
```

word = `"ABCCED"`, -> returns `true`,

word = `"SEE"`, -> returns `true`,

word = `"ABCB"`, -> returns `false`.

```python
# Word Search
class Solution(object):
    def exist(self, board, word):
        self.ROWS = len(board)
        self.COLS = len(board[0])
        self.board = board
        for row in range(self.ROWS):
            for col in range(self.COLS):
                if self.backtrack(row, col, word):
                    return True
        return False

    def backtrack(self, row, col, suffix):
        if len(suffix) == 0:
            return True
        if row < 0 or row == self.ROWS or col < 0 or col == self.COLS \
                or self.board[row][col] != suffix[0]:
            return False

        ret = False
        self.board[row][col] = '#'
        for rowOffset, colOffset in [(0, 1), (1, 0), (0, -1), (-1, 0)]:
            ret = self.backtrack(row + rowOffset, col + colOffset, suffix[1:])
            if ret: break

        self.board[row][col] = suffix[0]
        return ret
```

Time of DFS :

$$\begin{array}{cccc} A & \rightarrow B & \rightarrow C & \rightarrow E \\ \downarrow & \downarrow & \downarrow & \\ S & F & C & S \\ A & D & E & E \end{array}$$

$$f(n) = 4 \cdot f(n-1)$$
$$= 4 \cdot 4 \cdot f(n-2)$$
$$= \underbrace{4 \cdot 4 \cdot \cdots\cdots}_{k}$$

k is the length of the target word.

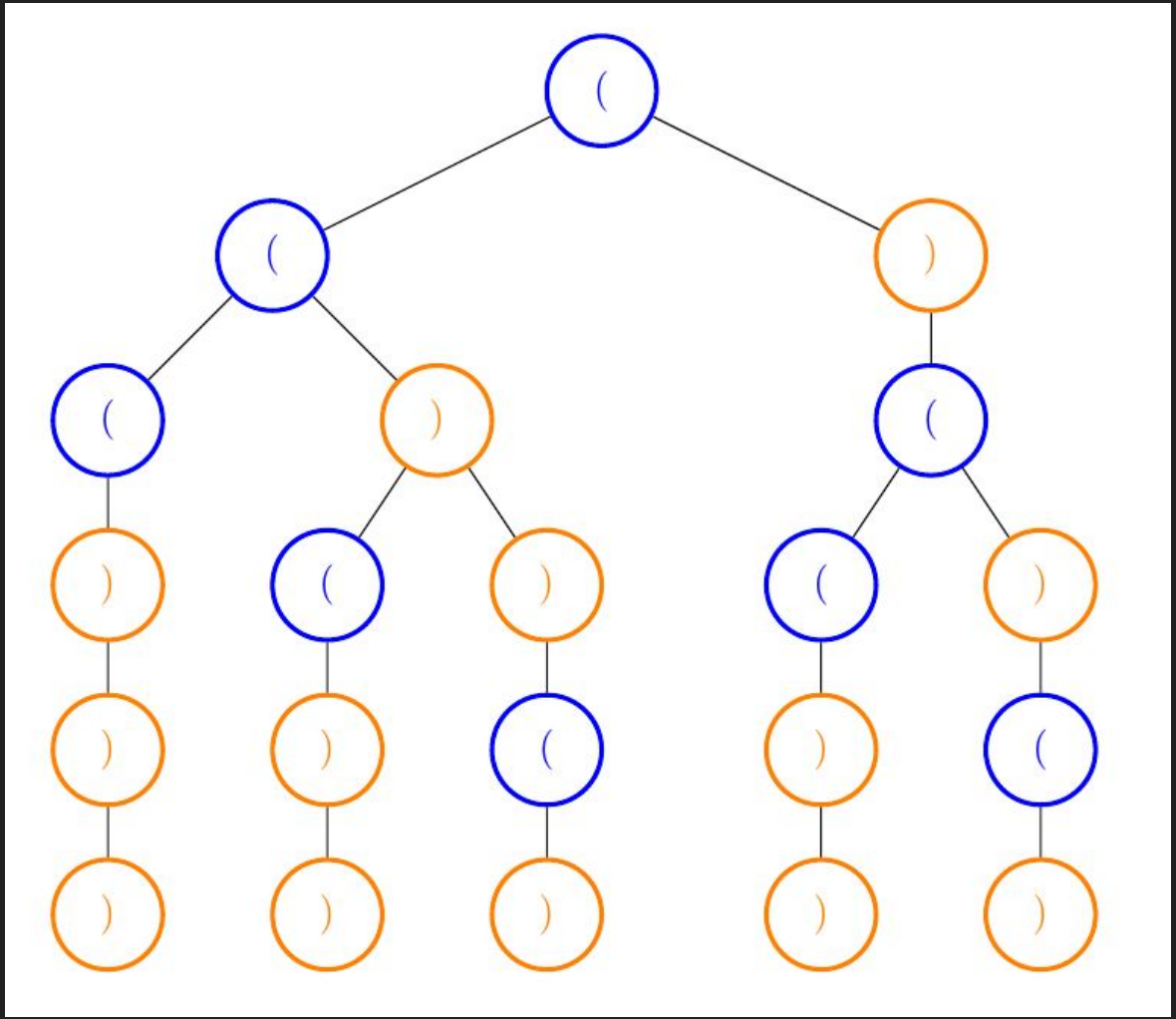
DFS is called $m \cdot n$

Therefore, overall time complexity is $O(m \cdot n \cdot 4^k)$

Question: Given n pairs of parentheses, write a function to generate all combinations of well-formed parentheses.

YOU HAVE 15 MINUTES

https://leetcode.com/problems/generate-parentheses/



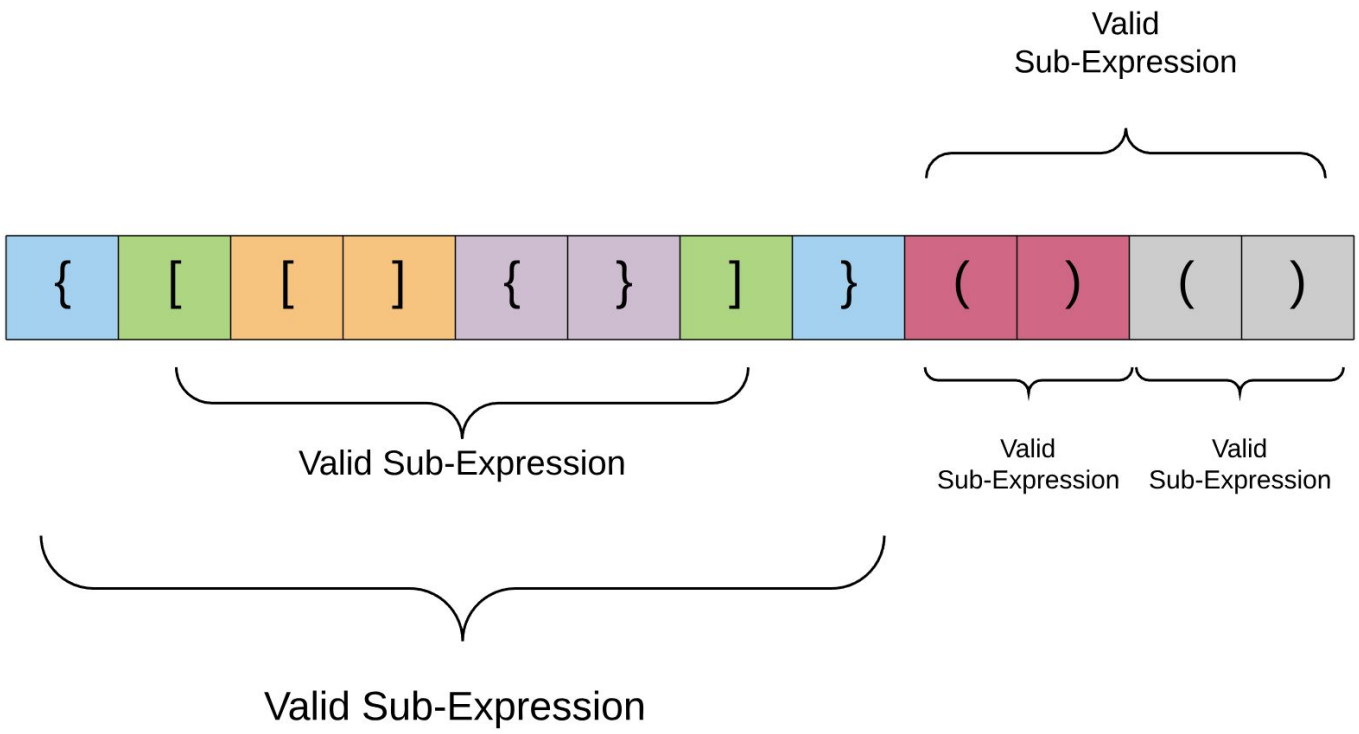
```python
# Generate Paranthesis
class Solution(object):
    def backtrack(self,N,ans, S = '',left = 0, right = 0):
            if len(S) == 2*N:
                ans.append(S)
                return
            if left < N:
                self.backtrack(N,ans,S+'(', left+1, right)
            if right < left:
                self.backtrack(N,ans,S+')', left, right+1)

    def generateParenthesis(self, N):
        ans = []
        self.backtrack(N,ans)
        return ans
```

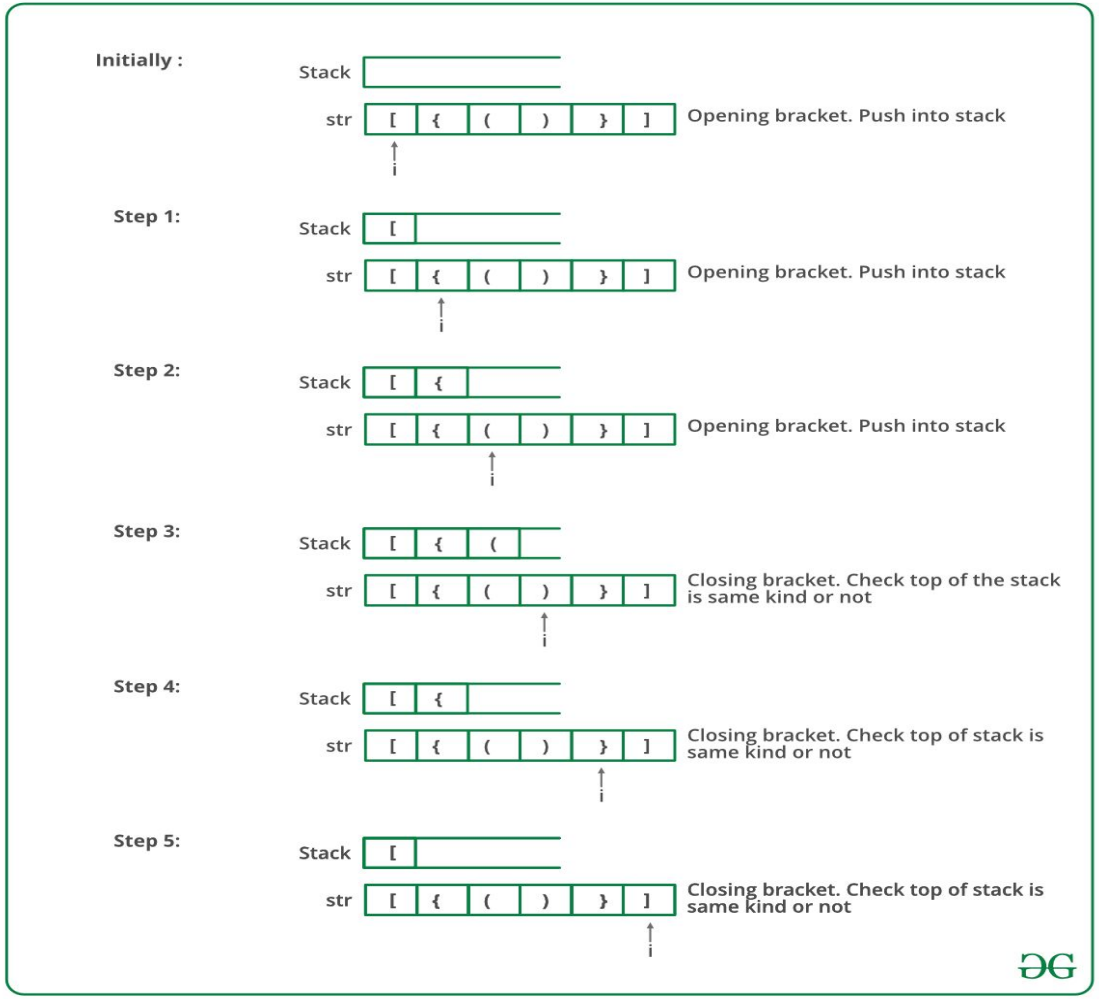
TIME COMPLEXITY and SPACE COMPLEXITY - $O(4^n/n^{1/2})$

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

YOU HAVE 15 MINUTES

https://leetcode.com/problems/valid-parentheses/


Valid
Sub-Expression
{ [[ ] { }] } ( ) ( )
Valid Sub-Expression
Valid
Sub-Expression
Valid
Sub-Expression
Valid Sub-Expression


Initially :
Stack
str [{ ( ) }]
Opening bracket. Push into stack
i
Step 1:
Stack [
str [{ ( ) }]
Opening bracket. Push into stack
i
Step 2:
Stack [{
str [{ ( ) }]
Opening bracket. Push into stack
i
Step 3:
Stack [{ (
str [{ ( ) }]
Closing bracket. Check top of the stack is same kind or not
i
Step 4:
Stack [{
str [{ ( ) }]
Closing bracket. Check top of stack is same kind or not
i
Step 5:
Stack [
str [{ ( ) }]
Closing bracket. Check top of stack is same kind or not
i

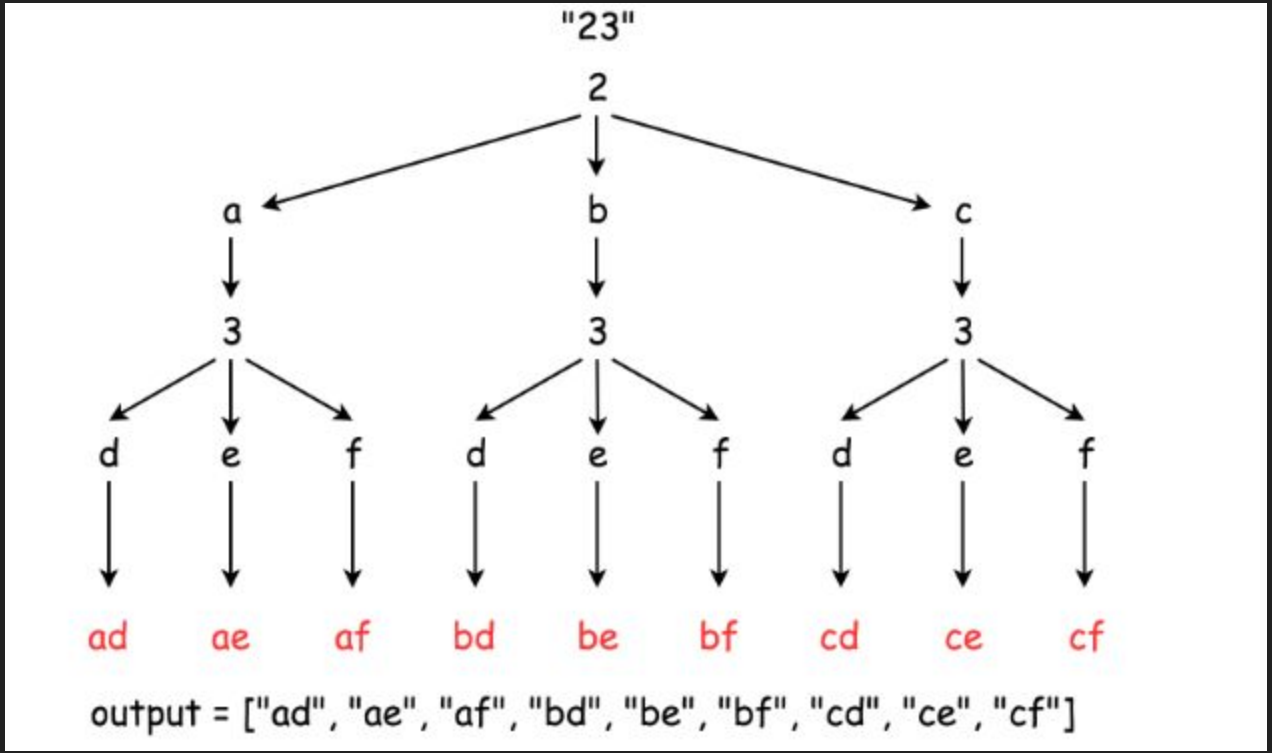
```python
# Balance Paranthesis
class Solution:
    def isValid(self, s: str) -> bool:
        stack, match = [], {'{':'}', '[':']', '(':')'}

        # print(type(match))
        # print(type(stack))
        for c in s:
            if c in match:
                stack.append(c)
            elif not stack or c != match[stack.pop()]:
                return False
        return not stack
```

TIME AND SPACE COMPLEXITY - O(N)

Question: Given a string containing digits from 2-9 inclusive, return all possible letter combinations that the number could represent.

YOU HAVE 15 MINUTES

https://leetcode.com/problems/letter-combinations-of-a-phone-number/


"23"
2
a
b
c
3
3
3
d
e
f
d
e
f
d
e
f
ad
ae
af
bd
be
bf
cd
ce
cf
output = ["ad", "ae", "af", "bd", "be", "bf", "cd", "ce", "cf"]

```python
# Letter combinations in a phone number
class Solution:
    def backtrack(self,combination, next_digits,output):
        if len(next_digits) == 0:
            output.append(combination)
        else:
            for letter in self.phone[next_digits[0]]:
                self.backtrack(combination + letter, next_digits[1:],output)
    def letterCombinations(self, digits):
        self.phone = {'2': ['a', 'b', 'c'],
                      '3': ['d', 'e', 'f'],
                      '4': ['g', 'h', 'i'],
                      '5': ['j', 'k', 'l'],
                      '6': ['m', 'n', 'o'],
                      '7': ['p', 'q', 'r', 's'],
                      '8': ['t', 'u', 'v'],
                      '9': ['w', 'x', 'y', 'z']}

        output = []
        if digits:
            self.backtrack("", digits,output)
        return output
```