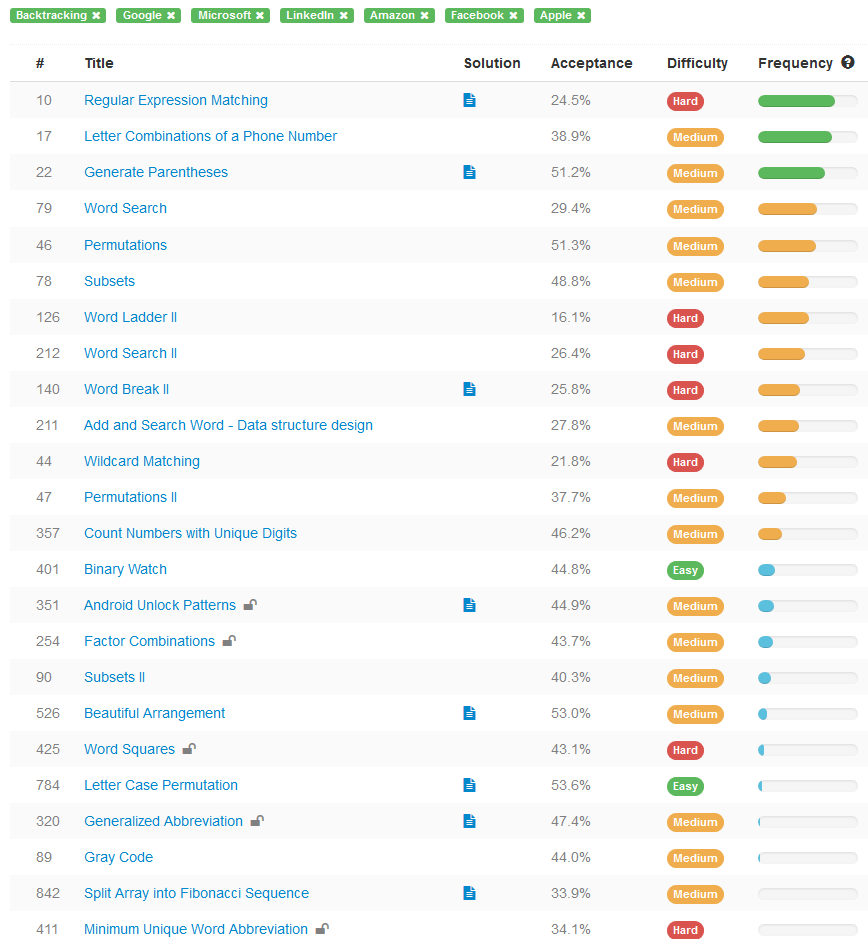
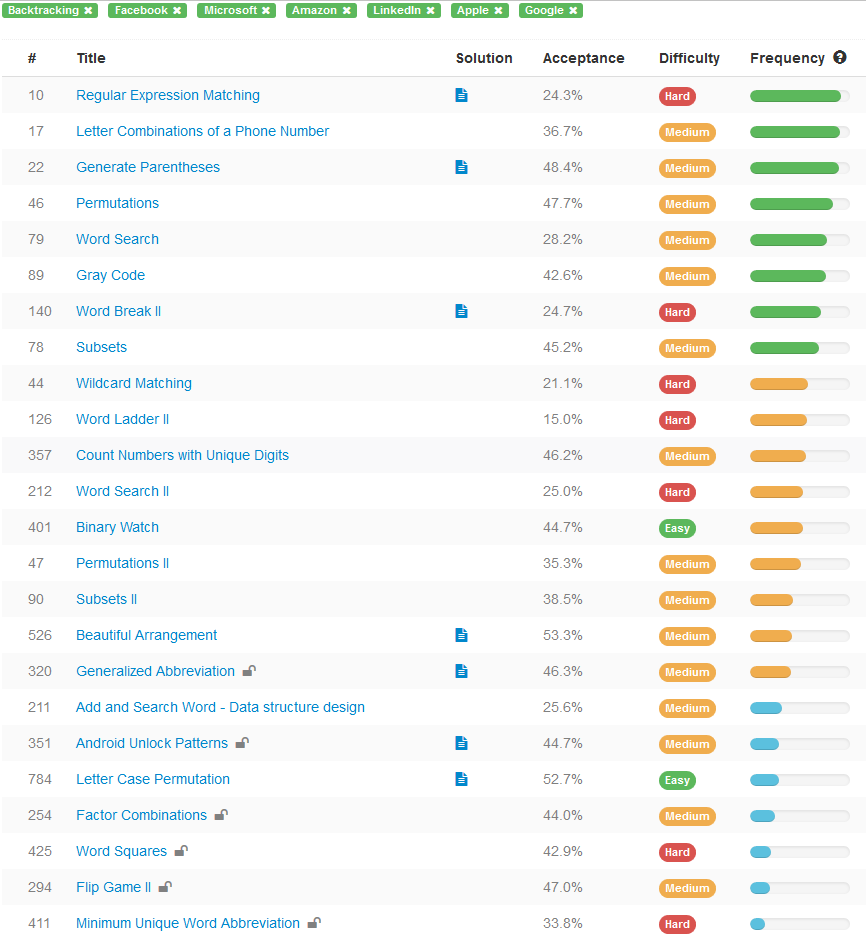
BACK TRACKING

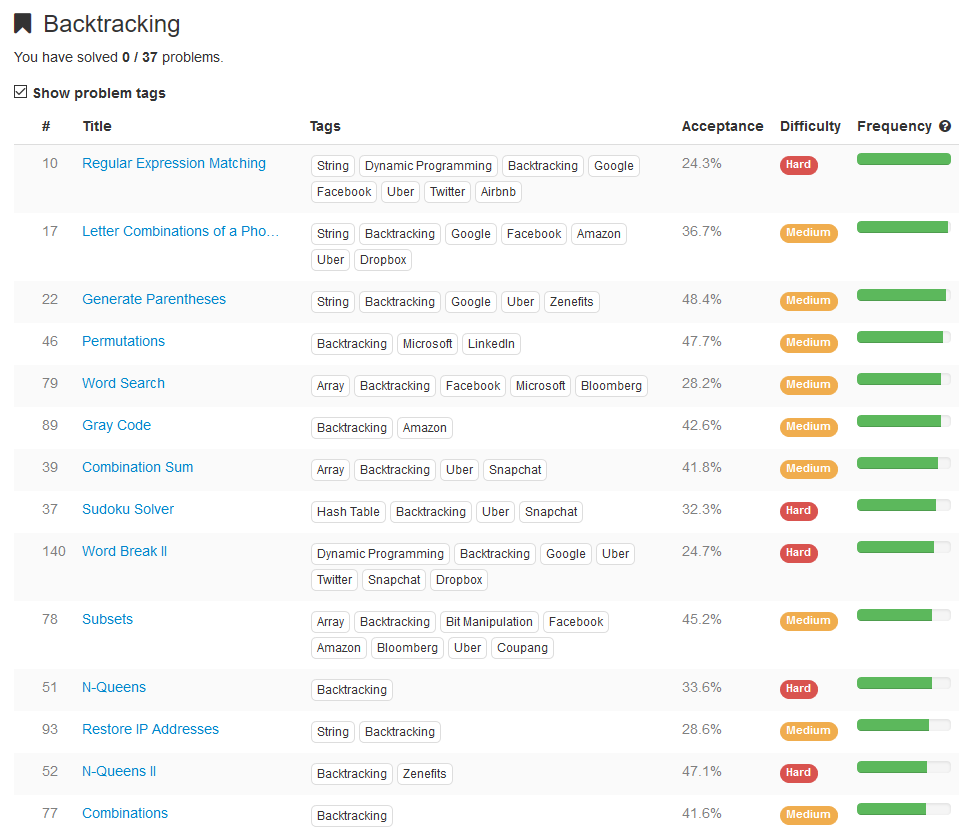
Updated: Nov 19th 2018

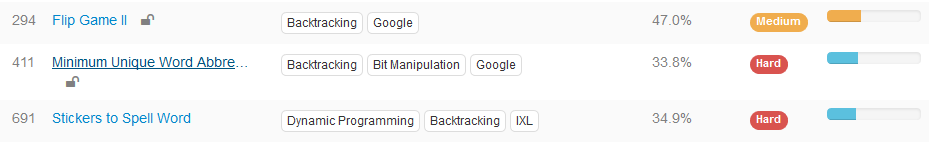
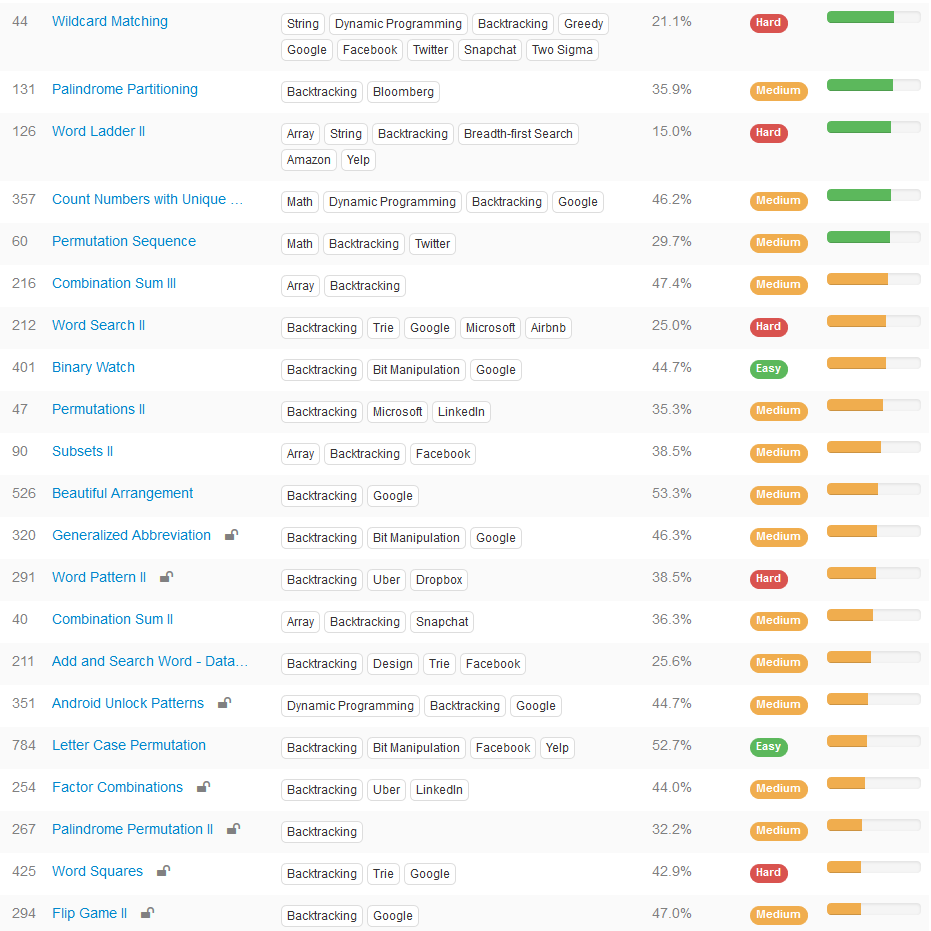


**Older**



TAGS:





Backtrack Summary: General Solution for 10 Questions!!!!!!!! Python (Combination Sum, Subsets, Permutation, Palindrome)

<https://leetcode.com/problems/subsets-ii/discuss/30196/Backtrack-Summary:-General-Solution-for-10-Questions!!!!!!!!-Python-(Combination-Sum-Subsets-Permutation-Palindrome)>

**39. Combination Sum**  
<https://leetcode.com/problems/combination-sum/>

defcombinationSum(self, candidates, target):

def backtrack(tmp, start, end, target):

if target == 0:

ans.append(tmp[:])

elif target > 0:

fori in range(start, end):

tmp.append(candidates[i])

backtrack(tmp, i, end, target - candidates[i])

tmp.pop()

ans = []

candidates.sort(reverse= True)

backtrack([], 0, len(candidates), target)

returnans

**40. Combination Sum II**  
<https://leetcode.com/problems/combination-sum-ii/>

def combinationSum2(self, candidates, target):

def backtrack(start, end, tmp, target):

if target == 0:

ans.append(tmp[:])

elif target > 0:

fori in range(start, end):

ifi> start and candidates[i] == candidates[i-1]:

continue

tmp.append(candidates[i])

backtrack(i+1, end, tmp, target - candidates[i])

tmp.pop()

ans = []

candidates.sort(reverse= True)

backtrack(0, len(candidates), [], target)

returnans

**78. Subsets**  
<https://leetcode.com/problems/subsets/>

def subsets(self, nums):

def backtrack(start, end, tmp):

ans.append(tmp[:])

fori in range(start, end):

tmp.append(nums[i])

backtrack(i+1, end, tmp)

tmp.pop()

ans = []

backtrack(0, len(nums), [])

returnans

**90. Subsets II**  
<https://leetcode.com/problems/subsets-ii/>

defsubsetsWithDup(self, nums):

def backtrack(start, end, tmp):

ans.append(tmp[:])

fori in range(start, end):

ifi> start and nums[i] == nums[i-1]:

continue

tmp.append(nums[i])

backtrack(i+1, end, tmp)

tmp.pop()

ans = []

nums.sort()

backtrack(0, len(nums), [])

returnans

**46. Permutations**  
<https://leetcode.com/problems/permutations/>

def permute(self, nums):

def backtrack(start, end):

if start == end:

ans.append(nums[:])

fori in range(start, end):

nums[start], nums[i] = nums[i], nums[start]

backtrack(start+1, end)

nums[start], nums[i] = nums[i], nums[start]

ans = []

backtrack(0, len(nums))

returnans

**47. Permutations II**  
<https://leetcode.com/problems/permutations-ii/>

defpermuteUnique(self, nums):

def backtrack(tmp, size):

iflen(tmp) == size:

ans.append(tmp[:])

else:

fori in range(size):

if visited[i] or (i> 0 and nums[i-1] == nums[i] and not visited[i-1]):

continue

visited[i] = True

tmp.append(nums[i])

backtrack(tmp, size)

tmp.pop()

visited[i] = False

ans = []

visited = [False] \* len(nums)

nums.sort()

backtrack([], len(nums))

returnans

**60. Permutation Sequence**  
<https://leetcode.com/problems/permutation-sequence/>

defgetPermutation(self, n, k):

nums = [str(i) for i in range(1, n+1)]

fact = [1] \* n

fori in range(1,n):

fact[i] = i\*fact[i-1]

k -= 1

ans = []

fori in range(n, 0, -1):

id = k / fact[i-1]

k %= fact[i-1]

ans.append(nums[id])

nums.pop(id)

return ''.join(ans)

**131. Palindrome Partitioning**  
<https://leetcode.com/problems/palindrome-partitioning/>

def partition(self, s):

def backtrack(start, end, tmp):

if start == end:

ans.append(tmp[:])

fori in range(start, end):

cur = s[start:i+1]

if cur == cur[::-1]:

tmp.append(cur)

backtrack(i+1, end, tmp)

tmp.pop()

ans = []

backtrack(0, len(s), [])

returnans

**267. Palindrome Permutation II**  
<https://leetcode.com/problems/palindrome-permutation-ii/>  
Related to this two:  
31. Next Permutation: <https://leetcode.com/problems/next-permutation/>  
266. Palindrome Permutation: <https://leetcode.com/problems/palindrome-permutation/>

defgeneratePalindromes(self, s):

kv = collections.Counter(s)

mid = [k for k, v in kv.iteritems() if v%2]

iflen(mid) > 1:

return []

mid = '' if mid == [] else mid[0]

half = ''.join([k \* (v/2) for k, v in kv.iteritems()])

half = [c for c in half]

def backtrack(end, tmp):

iflen(tmp) == end:

cur = ''.join(tmp)

ans.append(cur + mid + cur[::-1])

else:

fori in range(end):

if visited[i] or (i>0 and half[i] == half[i-1] and not visited[i-1]):

continue

visited[i] = True

tmp.append(half[i])

backtrack(end, tmp)

visited[i] = False

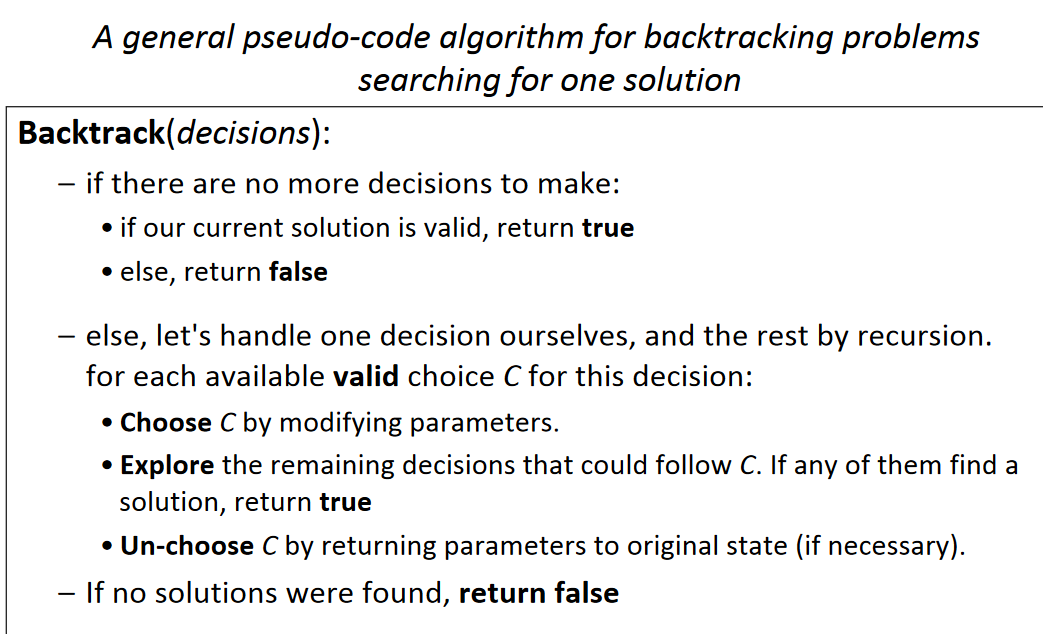
tmp.pop()

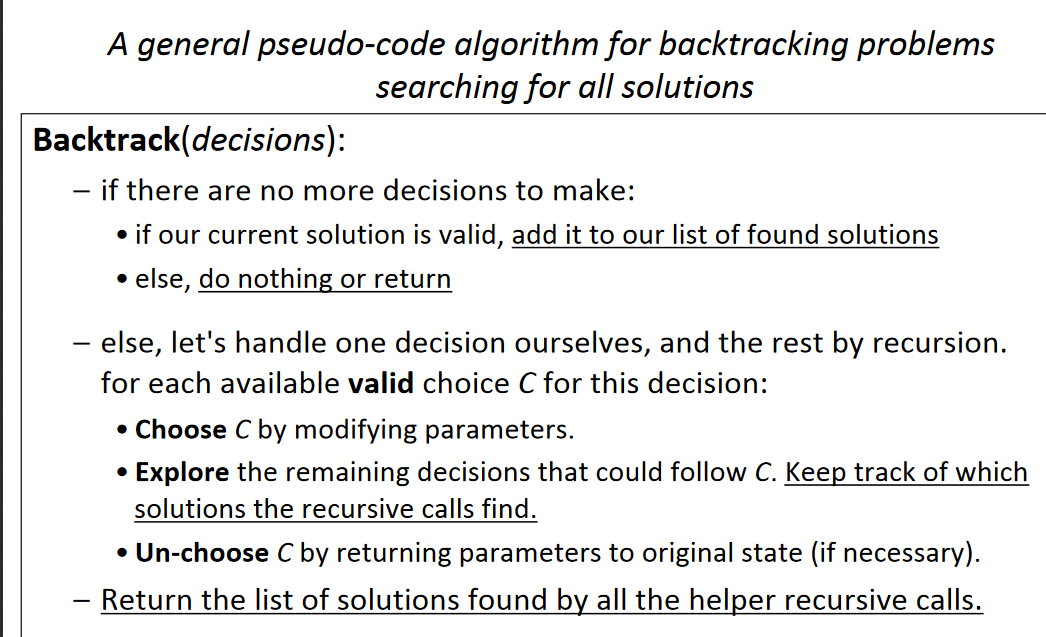
ans = []

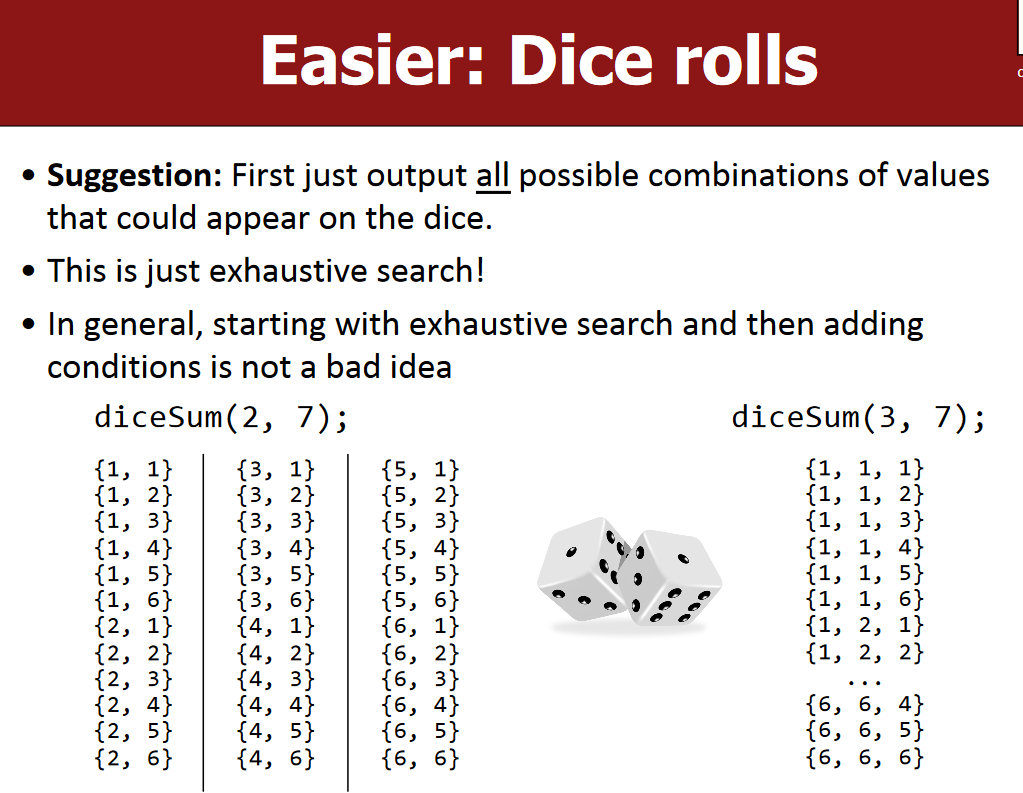
visited = [False] \* len(half)

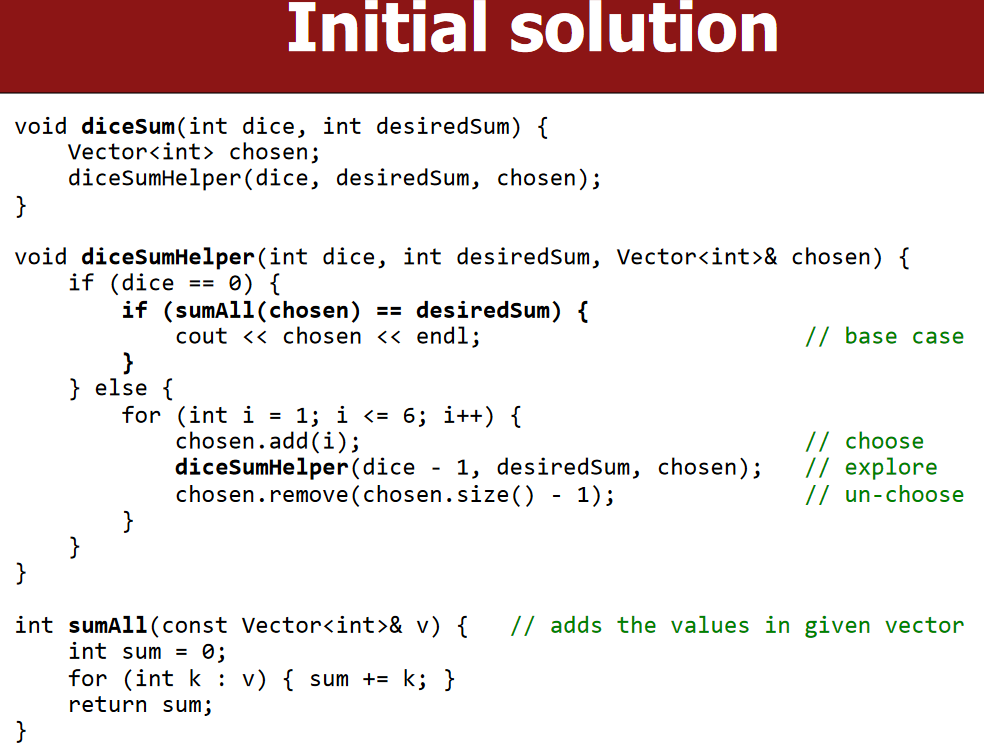
backtrack(len(half), [])

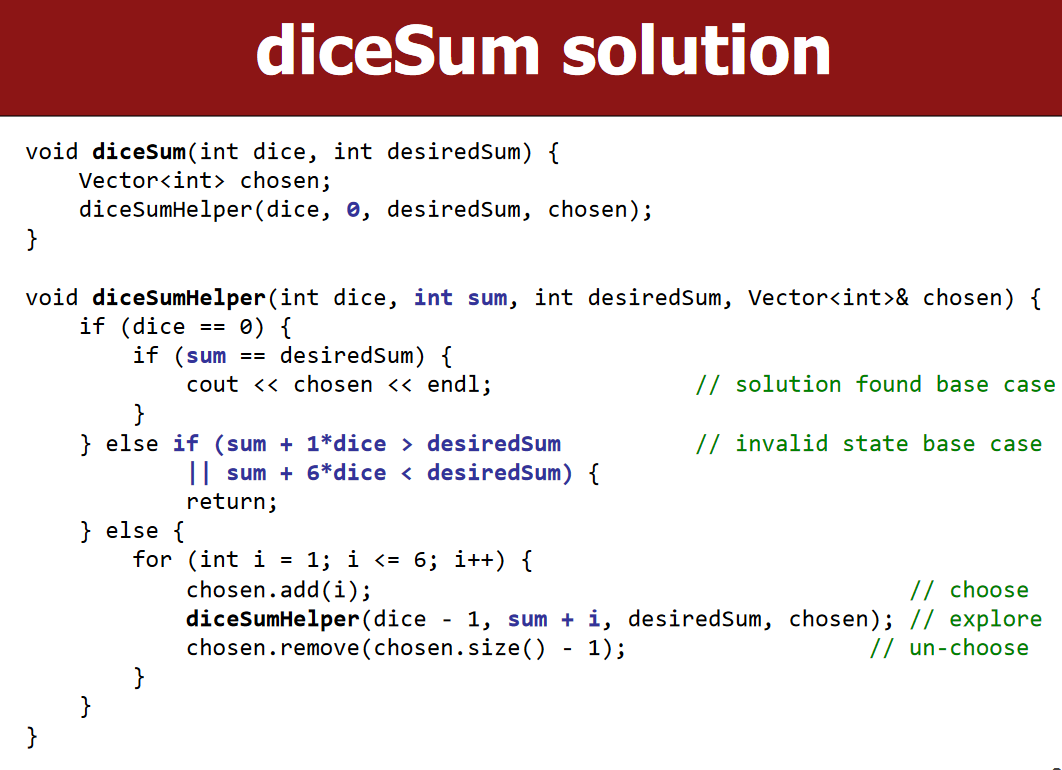
returnans

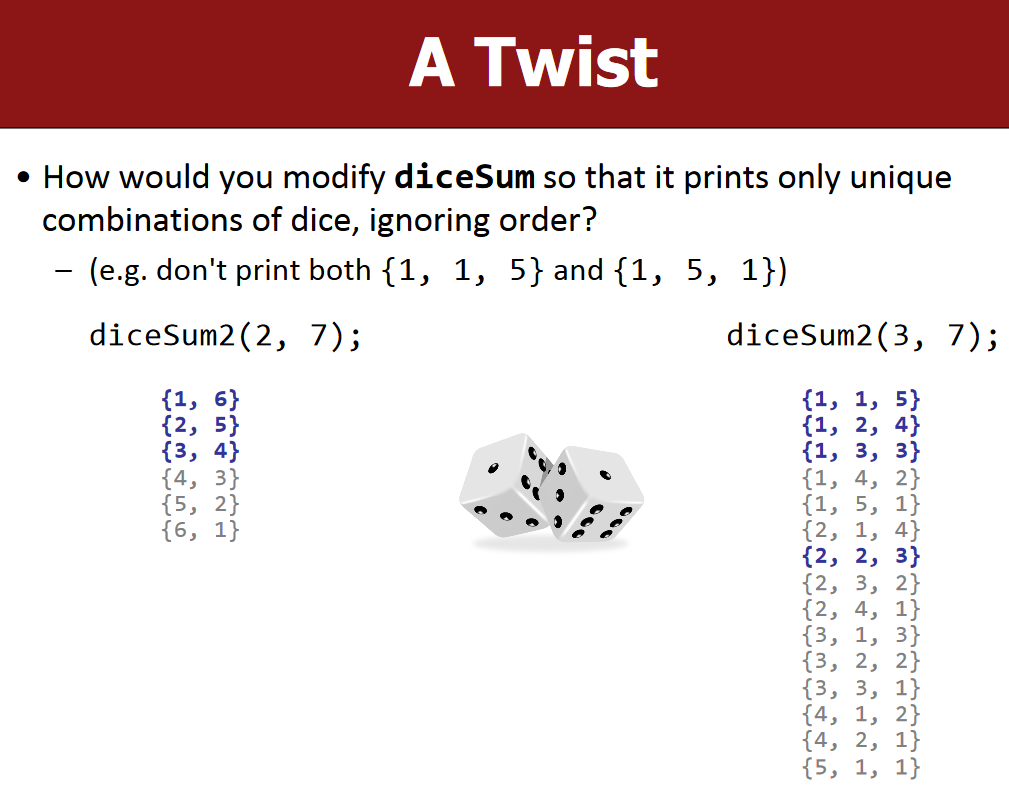




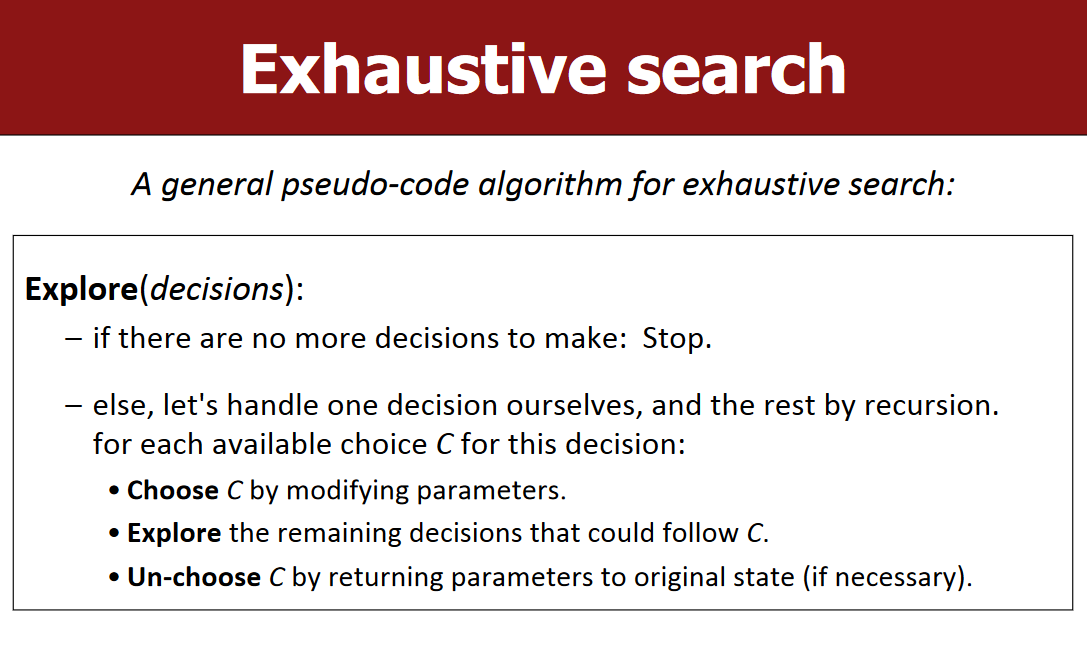


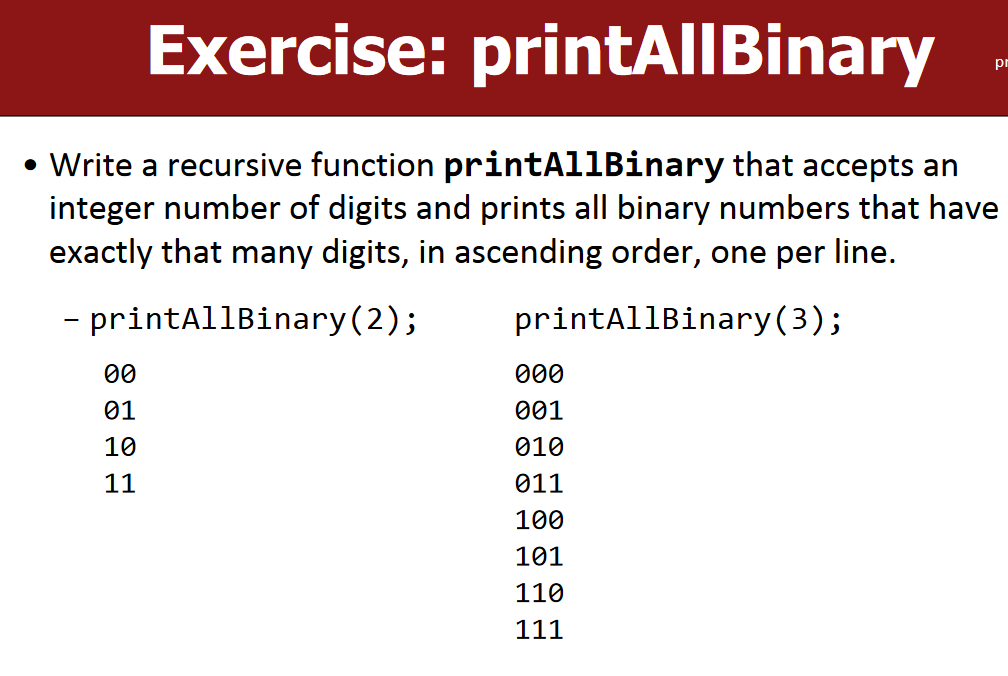


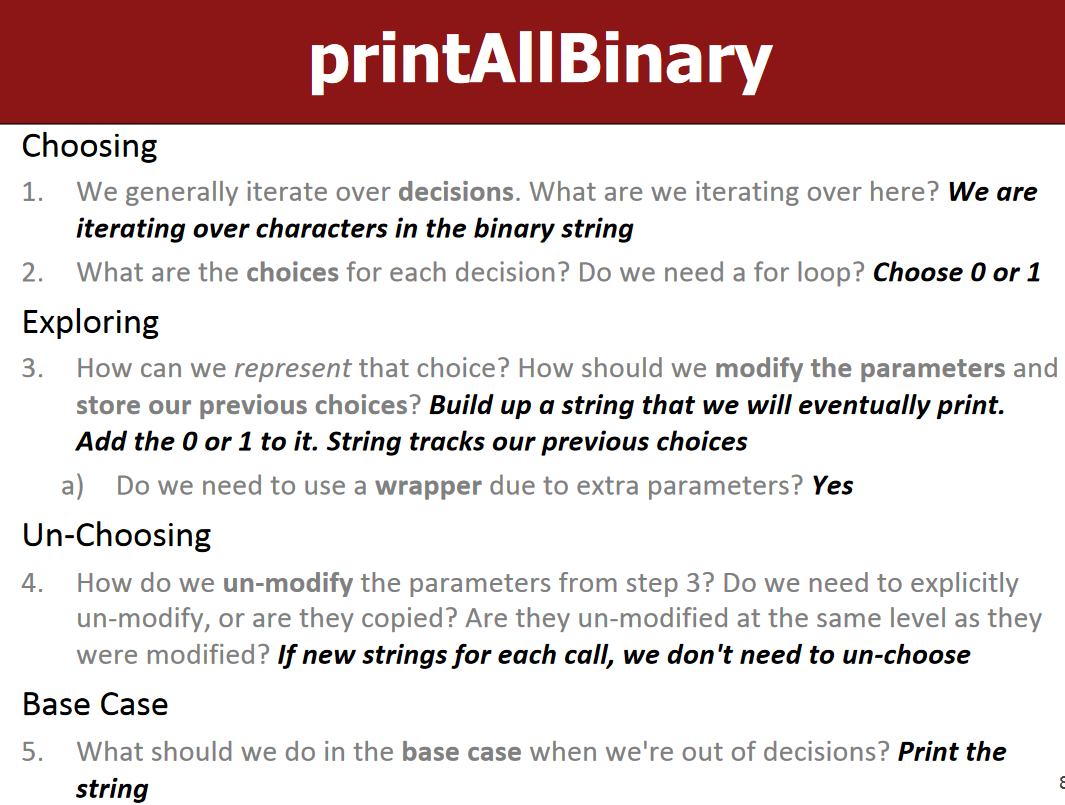


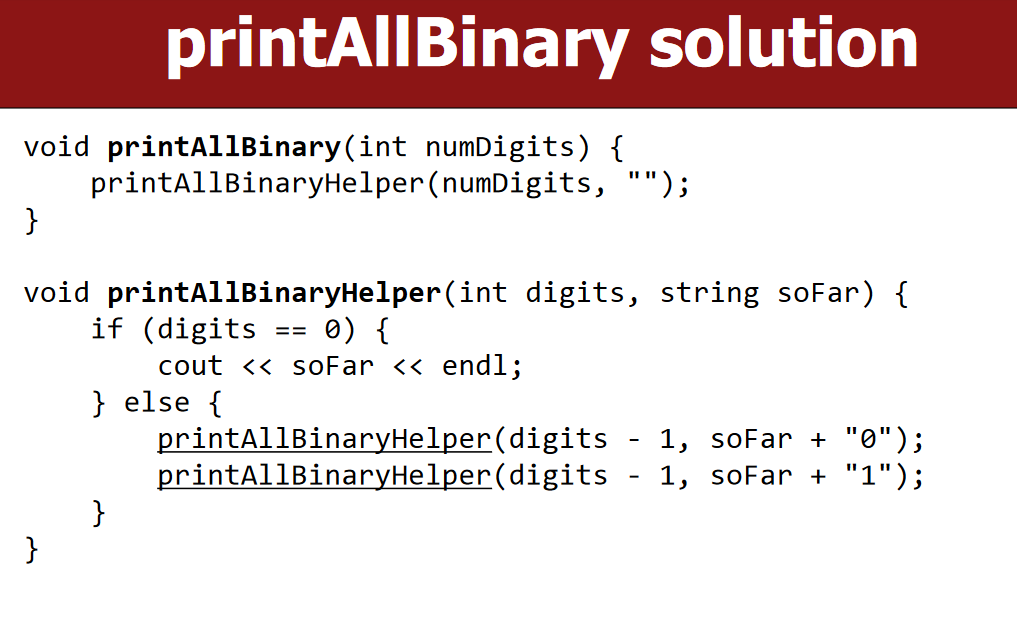


EXHAUSTIVE SEARCH:









LC-46: Backtracking techniques with Permutation problems …. TYPE-1 ( Swap and Track)

