DP = 子问题空间(分类讨论) + TopDown/BottomUp

单变量 f(n)

Climbing Stairs `f(n) = f(n-1) + f(n-2)`
New 21 Game `dp[n]: prob of getting n points`
Unique Binary Search Trees I `f(n) = sum(f(i)*f(n-1-i) for i in range(n))`
Unique Binary Search Trees II `map k to list of nodes`

双变量 f(m, n)

Unique Paths I `f(m, n) = f(m-1, n) + f(m, n-1)`
Unique Paths II (obstacles) `f(m, n) = f(m-1, n) + f(m, n-1), or 0 (obstacle)`

单序列

Decode Ways I `dp[i] for s[..@i]` `分类讨论 0,1-9`
Decode Ways II (wildcard) `dp[i] for s[..@i]` `分类讨论 *,0,1-9`

Word Break I (is possible) `dp[i] for s[..@i]` `dp[i,j]: break s[@i..@j]`
Word Break II (all results) `记录 split points, backtracking`

Best Time to Buy and Sell Stock I (at most 1 transaction) `价格差: max subarray sum, DP`
Best Time to Buy and Sell Stock II (infinite transactions) `累加正差`

Min Cost Climbing Stairs `dp[i] for s[..@i]`
House Robber I `dp[i]: max money from A[..i], dp[i] = max(dp[i-1], dp[i-2]+ A[i])`
`dp[i]: max money from A[..@i], dp[i] = A[i] + max(dp[i-2], dp[i-3])`
House Robber II (circular) `掐头 or 去尾`

Burst Balloons `DP[i][j], 遍历 which last`

单序列 + 单变量

Paint Fence (number of ways) `dp[i][same/dif]: paint i with same/dif color`
Paint House I `dp[i][color]: paint i with color`
Paint House II `dp[i][color]: paint i with color`

Best Time to Buy and Sell Stock III (k transactions) `dp[k][t] for k transactions on s[..t]`

```
def buyStock(prices, n_trans):
    n_days = len(prices)
    if n_days <= 1 or n_trans <= 0: return 0
    dp = [[0] * n_days for _ in range(n_trans + 1)]
    for k in range(1, n_trans+1):
        for t in range(1, n_days):
            dp[k][t] = dp[k][t-1]  # 不交易
            for i in range(t):     # 交易
                dp[k][t] = max(dp[k][t], dp[k-1][i] + prices[t] - prices[i])
    return dp[-1][-1]
```

双序列 `dp[i][j] for s[..@i] and t[..@j]`

Is Subsequence `双指针`
Longest Common Subsequence `max(左,上,斜(+1/0))`

Edit Distance `dp[i][j] for s[..@i] and t[..@j]`
Is One Edit Distance `greedy`
Delete Operation for Two Strings `longest common subsequence`
Regular Expression Matching
Wildcard Matching

```
def number_Of_Edit(w1, w2):
    m, n = len(w1), len(w2)
    dp = [ [0] * (n + 1) for _ in range(m + 1)]
    for i in range(m + 1): dp[i][0] = i
    for j in range(n + 1): dp[0][j] = j
    for i in range(1, m + 1):
        for j in range(1, n + 1):
            dp[i][j] = min(dp[i-1][j] + 1, dp[i][j-1] + 1,
                           dp[i-1][j-1] + (word1[i-1] != word2[j-1]))
    return dp[m][n]

def longest_Common_Subsequence(w1, w2):
    m, n = len(w1), len(w2)
    dp = [[0] * (n + 1) for _ in range(m + 1)]
    for i in range(1, m+1):
        for j in range(1, n+1):
            top, left, topleft = dp[i-1][j], dp[i][j-1], dp[i-1][j-1]
            dp[i][j] = max(top, left, topleft + (w1[i-1] == w2[j-1]))
    return dp[m][n]

def numberOfDelete(w1, w2):
    return len(w1) + len(w2) - 2 * longest_Common_Subsequence(w1, w2)

def is_One_Edit_Distance(s, t):
    if s == t: return False
    m, n = len(s), len(t)
    if m > n: return is_One_Edit_Distance(t, s) # force m <= n
    if n - m > 1: return False
    for i in range(l1):
        if s[i] != t[i]:
            if m == n: s = s[:i]+t[i]+s[i+1:]  # replacement
            else: s = s[:i]+t[i]+s[i:]  # insertion
            break
    return s == t or s == t[:-1]

def regular_Expression_Matching(s, p):
    m, n = len(s), len(p)
    dp = [[False] * (n + 1) for _ in range(m + 1)]
    dp[0][0] = True
    for i, c in enumerate(p):
        if c == '*': dp[0][i+1] = dp[0][i-1]
    for i in range(1, m+1):
        for j in range(1, n+1):
            if p[j-1] == s[i-1] or p[j-1]=='.':  # char matched
                dp[i][j] = dp[i-1][j-1]
            elif p[j-1] == '*':
                dp[i][j] = dp[i][j-2]   # repeat 0 time
                if p[j-2] == s[i-1] or p[j-2]=='.': dp[i][j] |= dp[i-1][j]
            # else: dp[i][j] = False
    return dp[m][n]

def wildcard_Matching(s, p):
    m, n = len(s), len(p)
    dp = [[False] * (n + 1) for _ in range(m + 1)]
    dp[0][0] = True
    for i, c in enumerate(p):
        if c == '*': dp[0][i+1] = dp[0][i]
    for i in range(1, m+1):
        for j in range(1, n+1):
            if p[j-1] == s[i-1] or p[j-1] == '?': # char matched
                dp[i][j] = dp[i-1][j-1]
            elif p[j-1] == '*':
                dp[i][j] = dp[i][j-1] or dp[i-1][j]
            # else: dp[i][j] = False
    return dp[m][n]
```

矩阵

Bomb Enemy `遇墙后，重新计算rowKill, colKills`
Longest Line of Consecutive One in Matrix `dp[i][j][0/1/2/3]`
Maximal Square `通用 O(n^3): 2D前缀和` `更快 O(n^2): min(左,上,斜)+1`
Maximal Rectangle `dp[i] for M[..@i]` `转化为📊最大矩形`

背包问题

Integer Break `3a + 2b`
Target Sum `dp(nums, target)` `top down`

不用DP，用BFS更快

Perfect Squares `BFS`
Coin Change `BFS`

subarray, substring

Maximum Subarray `dp[i]: maximum subarray ending with N[i]`
Maximum Product Subarray `负负得正` `maximum, minimum`
Max Sum of Rectangle `left,right,max subarray sum`
Max Sum of Rectangle No Larger Than K `left,right,max subarray sum <= k` `cumsum[j]-x<=k`
Maximum Length of Repeated Subarray
Maximum Sum of 3 Non-Overlapping Subarrays
Bitwise ORs of Subarrays
Continuous Subarray Sum
Longest Palindromic Substring `dp[i][j]: s[@i..@j] is panlindromic`
Longest Valid Parentheses `dp[i] for s[..@i]` `前左 ?()` `前右 ??(--))`
Palindromic Substrings
Palindrome Partitioning II
Unique Substrings in Wraparound String

subsequence

Distinct Subsequences

Wiggle Subsequence

Count Different Palindromic Subsequences

Minimum Window Subsequence

Longest Increasing Continuous Subsequence
Longest Increasing Subsequence
Number of Longest Increasing Subsequence
Minimum Swaps To Make Sequences Increasing

Longest Fibonacci Subsequence

Longest Palindromic Subsequence `dp[i][j]: longest in s[i..j]`

没分类

Sentence Screen Fitting `指针: abc_de_f_`

还没做

Counting Bits
Number Of Corner Rectangles
Stone Game
Arithmetic Slices
Arithmetic Slices II - Subsequence

2 Keys Keyboard
4 Keys Keyboard

Maximum Length of Pair Chain
Count Numbers with Unique Digits
Shopping Offers

Predict the Winner

Android Unlock Patterns

Delete and Earn

Encode String with Shortest Length

Combination Sum IV

Shortest Path Visiting All Nodes
Minimum Path Sum

Largest Sum of Averages

Push Dominoes
Largest Plus Sign

Knight Probability in Chessboard

Split Array Largest Sum
Freedom Trail

Valid Permutations for DI Sequence

Partition Equal Subset Sum
Ones and Zeroes
Partition to K Equal Sum Subsets
Remove Boxes

Triangle
Super Washing Machines
Guess Number Higher or Lower II

Maximum Vacation Days
Stickers to Spell Word

Strange Printer
Soup Servings

Ugly Number II
Range Sum Query - Immutable
Largest Divisible Subset
Domino and Tromino Tiling
Frog Jump

Russian Doll Envelopes

Concatenated Words
Profitable Schemes
Non-negative Integers without Consecutive Ones

Student Attendance Record II

Out of Boundary Paths
Scramble String
Cheapest Flights Within K Stops
Race Car

K Inverse Pairs Array

Count The Repetitions
Interleaving String
Can I Win

Minimum Number of Refueling Stops
Coin Path

Dungeon Game
Numbers At Most N Given Digit Set
Create Maximum Number
Cherry Pickup

Super Egg Drop