1. Max, Min Path Sum

Choose min (max) path among all
before the current state, then add value for the current

```
routes[i] = min(routes[i-1], routes[i-2], ... ,
routes[i-k]) + cost[i]
```

120. Triangle

Given a triangle array, return *the minimum path sum from top to bottom*.

For each step, you may move to an adjacent number of the row below. More formally, if you are on index i on the current row, you may move to either index i or index i + 1 on the next row.

221. Maximal Square

Given an m x n binary matrix filled with 0's and 1's, *find the largest square containing only* 1's *and return its area*.

474. Ones and Zeroes

You are given an array of binary strings strs and two integers m and n.

Return *the size of the largest subset of strs such that there are at most m* 0*'s and* n 1*'s in the subset*.

A set `x` is a subset of a set `y` if all elements of `x` are also elements of `y`.

279. Perfect Square

A perfect square is an integer that is the square of an integer; in other words, it is the product of some integer with itself. For example, 1, 4, 9, and 16 are perfect squares while 3 and 11 are not

```
Input: n = 12
Output: 3
   Explanation: 12 = 4 + 4 + 4.
```

650. 2 Keys Keyboard

There is only one character `'A'` on the screen of a notepad. You can perform two operations on this notepad for each step:

- Copy All: You can copy all the characters present on the screen (a partial copy is not allowed).
- Paste: You can paste the characters which are copied last time.

Given an integer `n`, return *the minimum number of operations to get the character* `'A'` *exactly* `n` *times on the screen*.

Approach #1: Prime Factorization [Accepted]

We can break our moves into groups of (copy, paste, ..., paste). Let C denote copying and P denote pasting. Then for example, in the sequence of moves CPPCPPPPCP, the groups would be [CPP][CPPPP][CP]

Say these groups have lengths $g_1, g_2, \ldots$ After parsing the first group, there are g_1 'A's. After parsing the second group, there are g_1 * g_2 'A's, and so on. At the end, there are g_1 * g_2 * ... * g_n 'A's. $N = g_1 * g_2 * \ldots * g_n$

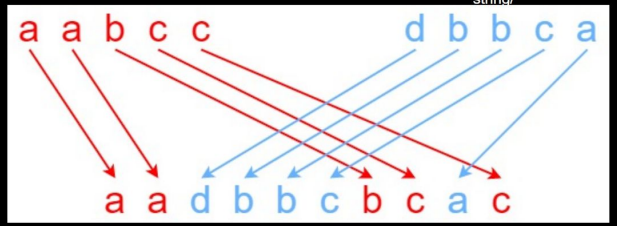
We want exactly g_i. If any of the $g_i = p * q$ are composite, say $g_i = p * q$, then we can split this group into two groups (the first of which has one copy followed by p-1 pastes, while the second group having one copy and q-1 pastes).

Such a split never uses more moves: we use p+q moves when splitting, and pq moves previously. As p+q <= pq is equivalent to 1 <= (p-1)(q-1), which is true as long as p >= 2 and q >= 2.

Algorithm By the above argument, we can suppose $g_1, g_2, \ldots$ is the prime factorization of N, and the answer is therefore the sum of these prime factors.

Interleaving Form

https://leetcode.com/problems/interleaving-string/



1. a a b
2. a x y

3.1 a a x a b y → T

3.2 a b a b x y → F

a a x a b y

	0	a	a	b
0	T	T	T	F
a				
x				
y				

```java
public boolean isInterleavedRecursive(char str1[], char str2[], char str3[],
                                      int pos1, int pos2, int pos3){
    if(pos1 == str1.length && pos2 == str2.length && pos3 == str3.length){
        return true;
    }

    if(pos3 == str3.length){
        return false;
    }

    return (pos1 < str1.length && str1[pos1] == str3[pos3]
            && isInterleavedRecursive(str1, str2, str3, pos1+1, pos2, pos3+1))
            || (pos2 < str2.length && str2[pos2] == str3[pos3]
            && isInterleavedRecursive(str1, str2, str3, pos1, pos2+1, pos3+1));

}

public boolean isInterleaved(char str1[], char str2[], char str3[]){
    boolean T[][] = new boolean[str1.length +1][str2.length +1];

    if(str1.length + str2.length != str3.length){
        return false;
    }

    for(int i=0; i < T.length; i++){
        for(int j=0; j < T[i].length; j++){
            int l = i + j -1;
            if(i == 0 && j == 0){
                T[i][j] = true;
            }
            else if(i == 0){
                if(str3[l] == str2[j-1]){
                    T[i][j] = T[i][j-1];
                }
            }
            else if(j == 0){
                if(str1[i-1] == str3[l]){
                    T[i][j] = T[i-1][j];
                }
            }
            else{
                T[i][j] = (str1[i-1] == str3[l] ? T[i-1][j] : false)
                          || (str2[j-1] == str3[l] ? T[i][j-1] : false);
            }
        }
    }
    return T[str1.length][str2.length];
}
```

Longest Zig Zag Path Binary Tree

	0	1	2	3	4	5	6	7
i								
j								
			2					

	0	1	2	3	4	5	6	7
i			i					
j		j						
			2					

	0	1	2	3	4	5	6	7
i								
j								
			2	3				

	0	1	2	3	4	5	6	7
i								
j								
dp[]			2	3	4			

```
dp[i] = dp[j] + (i/j);
dp[4] = dp[2] + (4/2)=4;
```

	0	1	2	3	4	5	6	7
i								
j								
dp[]			2	3	4			

	0	1	2	3	4	5	6	7
i								
j								
dp[]			2	3	4	5		

	0	1	2	3	4	5	6	7
i								
j								
dp[]			2	3	4	5		

	0	1	2	3	4	5	6	7
i								
j								
dp[]			2	3	4	5		

	0	1	2	3	4	5	6	7
i								
j								
dp[]			2	3	4	5		

```
for (int j = i-1; j > 1; j--)
```

	0	1	2	3	4	5	6	7
i								
j								
dp[]			2	3	4	5	6	

```
for (int j = i-1; j > 1; j--)
```

	0	1	2	3	4	5	6	7
i								
j								
dp[]			2	3	4	5	6	

```
for (int j = i-1; j > 1; j--)
```

```
dp[i] = dp[j] + (i/j);
dp[6] = dp[3] + (6/3)=5;
```

	0	1	2	3	4	5	6	7
i								
j								
dp[]			2	3	4	5	5	

```
for (int j = i-1; j > 1; j--)
```

Ones and Zeros

Dungeon Game

Ones and Zeros

```
dp[i][j] = Math.max(dp[i][j], dp[i - cntZero][j - cntOne] +
1);
```

i/j					
		0	1	2	3
	0				
	1				
	2				
	3				
	4				
	5				1

```
new String[]{"10", "0001", "111001", "1", "0"}
```

```
dp[i][j] = Math.max(dp[i][j], dp[i - cntZero][j - cntOne] +
1);
```

i/j					
		0	1	2	3
	0				
	1				
	2				
	3				
	4				
	5			1	1

```
new String[]{"10", "0001", "111001", "1", "0"}
```

```
dp[i][j] = Math.max(dp[i][j], dp[i - cntZero][j - cntOne] +
1);
```

i/j					
		0	1	2	3
	0				
	1				
	2				
	3				
	4				
	5		1		1

```
new String[]{"10", "0001", "111001", "1", "0"}
```

```
dp[i][j] = Math.max(dp[i][j], dp[i - cntZero][j - cntOne] +
1);
```

i/j					
		0	1	2	3
	0				
	1				
	2				
	3				
	4				1
	5		1	1	1

```
new String[]{"10", "0001", "111001", "1", "0"}
```

```
dp[i][j] = Math.max(dp[i][j], dp[i - cntZero][j - cntOne] +
1);
```

i/j					
		0	1	2	3
	0				
	1				
	2				
	3				
	4			1	1
	5		1	1	1

cntZero=1, cntOne=1

```
new String[]{"10", "0001", "111001", "1", "0"}
```

```
dp[i][j] = Math.max(dp[i][j], dp[i - cntZero][j - cntOne] +
1);
```

i/j					
		0	1	2	3
	0				
	1				
	2				
	3				
	4		1	1	1
	5		1	1	1

cntZero=1, cntOne=1

```
new String[]{"10", "0001", "111001", "1", "0"}
```

```
dp[i][j] = Math.max(dp[i][j], dp[i - cntZero][j - cntOne] +
1);
```

i/j					
		0	1	2	3
	0				
	1				
	2				
	3				1
	4		1	1	1
	5		1	1	1

cntZero=1, cntOne=1

```
new String[]{"10", "0001", "111001", "1", "0"}
```

```
dp[i][j] = Math.max(dp[i][j], dp[i - cntZero][j - cntOne] +
1);
```

i/j					
		0	1	2	3
	0				
	1				
	2				
	3			1	1
	4		1	1	1
	5		1	1	1

cntZero=1, cntOne=1

```
new String[]{"10", "0001", "111001", "1", "0"}
```

```
dp[i][j] = Math.max(dp[i][j], dp[i - cntZero][j - cntOne] +
1);
```

i/j					
		0	1	2	3
	0				
	1				
	2				
	3		1	1	1
	4		1	1	1
	5		1	1	1

cntZero=1, cntOne=1

```
new String[]{"10", "0001", "111001", "1", "0"}
```

```
dp[i][j] = Math.max(dp[i][j], dp[i - cntZero][j - cntOne] +
1);
```

i/j					
		0	1	2	3
	0				
	1				
	2				1
	3		1	1	1
	4		1	1	1
	5		1	1	1

cntZero=1, cntOne=1

```
new String[]{"10", "0001", "111001", "1", "0"}
```

```
dp[i][j] = Math.max(dp[i][j], dp[i - cntZero][j - cntOne] +
1);
```

i/j					
		0	1	2	3
	0				
	1				
	2			1	1
	3		1	1	1
	4		1	1	1
	5		1	1	1

cntZero=1, cntOne=1

```
new String[]{"10", "0001", "111001", "1", "0"}
```

```
dp[i][j] = Math.max(dp[i][j], dp[i - cntZero][j - cntOne] +
1);
```

i/j					
		0	1	2	3
	0				
	1				
	2		1	1	1
	3		1	1	1
	4		1	1	1
	5		1	1	1

cntZero=1, cntOne=1

```
new String[]{"10", "0001", "111001", "1", "0"}
```

```
dp[i][j] = Math.max(dp[i][j], dp[i - cntZero][j - cntOne] +
1);
```

i/j					
		0	1	2	3
	0				
	1				1
	2		1	1	1
	3		1	1	1
	4		1	1	1
	5		1	1	1

cntZero=1, cntOne=1

```
new String[]{"10", "0001", "111001", "1", "0"}
```

```
dp[i][j] = Math.max(dp[i][j], dp[i - cntZero][j - cntOne] +
1);
```

i/j					
		0	1	2	3
	0				
	1			1	1
	2		1	1	1
	3		1	1	1
	4		1	1	1
	5		1	1	1

cntZero=1, cntOne=1

```
new String[]{"10", "0001", "111001", "1", "0"}
```

```
dp[i][j] = Math.max(dp[i][j], dp[i - cntZero][j - cntOne] +
1);
```

i/j					
		0	1	2	3
	0				
	1		1	1	1
	2		1	1	1
	3		1	1	1
	4		1	1	1
	5		1	1	1

```
cntZero=1, cntOne=1
new String[]{"10", "0001", "111001", "1", "0"}
```

```
str="0001"
```

```
dp[i][j] = Math.max(dp[i][j], dp[i - cntZero][j - cntOne] +
1);
```

str="0001"

i/j					
		0	1	2	3
	0				
	1		1	1	1
	2		1	1	1
	3		1	1	1
	4		1	1	1
	5		1	2	2

cntZero=3, cntOne=1

new String[]{"10", "0001", "111001", "1", "0"}

```
dp[i][j] = Math.max(dp[i][j], dp[i - cntZero][j - cntOne] +
1);
```

str="0001"

i/j					
		0	1	2	3
	0				
	1		1	1	1
	2		1	1	1
	3		1	1	1
	4		1	1	1
	5		1	2	2

cntZero=3, cntOne=1

new String[]{"10", "0001", "111001", "1", "0"}

```
dp[i][j] = Math.max(dp[i][j], dp[i - cntZero][j - cntOne] +
1);
```

i/j					
		0	1	2	3
	0				
	1		1	1	1
	2		1	1	1
	3		1	1	1
	4		1	1	1
	5		1	2	2

str="0001"

cntZero=3, cntOne=1

new String[]{"10", "0001", "111001", "1", "0"}

```
dp[i][j] = Math.max(dp[i][j], dp[i - cntZero][j - cntOne] +
1);
```

str="0001"

i/j					
		0	1	2	3
	0				
	1		1	1	1
	2		1	1	1
	3		1	1	1
	4		1	1	2
	5		1	2	2

cntZero=3, cntOne=1

new String[]{"10", "0001", "111001", "1", "0"}

```
dp[i][j] = Math.max(dp[i][j], dp[i - cntZero][j - cntOne] +
1);
```

i/j					
		0	1	2	3
	0				
	1		1	1	1
	2		1	1	1
	3		1	1	1
	4		1	2	2
	5		1	2	2

str="0001"

cntZero=3, cntOne=1

new String[]{"10", "0001", "111001", "1", "0"}

```
dp[i][j] = Math.max(dp[i][j], dp[i - cntZero][j - cntOne] +
1);
```

str="0001"

i/j					
		0	1	2	3
	0				
	1		1	1	1
	2		1	1	1
	3		1	1	1
	4		1	2	2
	5		1	2	2

cntZero=3, cntOne=1

new String[]{"10", "0001", "111001", "1", "0"}

str="111001"

i=5

```
dp[i][j] = Math.max(dp[i][j], dp[i - cntZero][j - cntOne] +
1);
```

i/j					
		0	1	2	3
	0				
	1		1	1	1
	2		1	1	1
	3		1	1	1
	4		1	2	2
	5		1	2	3

str="111001"

dp[i][j]=
dp[i-cntZero][j-cntOne]

cntZero=2, cntOne=4

```
new String[]{"10", "0001", "111001", "1", "0"}
```

str="1"

i=5

```
dp[i][j] = Math.max(dp[i][j], dp[i - cntZero][j - cntOne] +
1);
```

i/j					
		0	1	2	3
	0				
	1		1	1	1
	2		1	1	1
	3		1	1	1
	4		1	2	2
	5		1	2	3

str="1"
cntZero=0
cntOne=1

```
new String[]{"10", "0001", "111001", "1", "0"}
```

```
dp[i][j] = Math.max(dp[i][j], dp[i - cntZero][j - cntOne] +
1);
```

i/j					
		0	1	2	3
	0				
	1		1	1	1
	2		1	1	1
	3		1	1	1
	4		1	2	2
	5		1	2	3

str="1"
cntZero=0
cntOne=1

```
new String[]{"10", "0001", "111001", "1", "0"}
```

```
dp[i][j] = Math.max(dp[i][j], dp[i - cntZero][j - cntOne] +
1);
```

i/j					
		0	1	2	3
	0				
	1		1	1	1
	2		1	1	1
	3		1	1	1
	4		1	2	2
	5		1	2	3

str="1"
cntZero=0
cntOne=1

```
new String[]{"10", "0001", "111001", "1", "0"}
```

str="1"

i=4

```
dp[i][j] = Math.max(dp[i][j], dp[i - cntZero][j - cntOne] +
1);
```

i/j					
		0	1	2	3
	0				
	1		1	1	1
	2		1	1	1
	3		1	1	1
	4		1	2	3
	5		1	2	3

str="1"
cntZero=0
cntOne=1

dp[i][j]=
dp[i-cntZero][j-cntOne]

```
new String[]{"10", "0001", "111001", "1", "0"}
```

```
dp[i][j] = Math.max(dp[i][j], dp[i - cntZero][j - cntOne] +
1);
```

i/j					
		0	1	2	3
	0				
	1		1	1	1
	2		1	1	1
	3		1	1	1
	4		1	2	3
	5		1	2	3

str="1"
cntZero=0
cntOne=1

```
new String[]{"10", "0001", "111001", "1", "0"}
```

```
dp[i][j] = Math.max(dp[i][j], dp[i - cntZero][j - cntOne] +
1);
```

i/j					
		0	1	2	3
	0				
	1		1	1	1
	2		1	1	1
	3		1	1	1
	4		1	2	3
	5		1	2	3

str="1"
cntZero=0
cntOne=1

```
new String[]{"10", "0001", "111001", "1", "0"}
```

str="1"

i=3

```
dp[i][j] = Math.max(dp[i][j], dp[i - cntZero][j - cntOne] +
1);
```

i/j					
		0	1	2	3
	0				
	1		1	1	1
	2		1	1	1
	3		1	1	2
	4		1	2	3
	5		1	2	3

str="1"
cntZero=0
cntOne=1

dp[i][j]=
dp[i-cntZero][j-cntOne]

```
new String[]{"10", "0001", "111001", "1", "0"}
```

```
dp[i][j] = Math.max(dp[i][j], dp[i - cntZero][j - cntOne] +
1);
```

i/j					
		0	1	2	3
	0				
	1		1	1	1
	2		1	1	1
	3		1	2	2
	4		1	2	3
	5		1	2	3

str="1"
cntZero=0
cntOne=1

dp[i][j]=
dp[i-cntZero][j-cntOne]

```
new String[]{"10", "0001", "111001", "1", "0"}
```

```
dp[i][j] = Math.max(dp[i][j], dp[i - cntZero][j - cntOne] +
1);
```

i/j					
		0	1	2	3
	0				
	1		1	1	1
	2		1	1	1
	3		1	2	2
	4		1	2	3
	5		1	2	3

str="1"
cntZero=0
cntOne=1

```
new String[]{"10", "0001", "111001", "1", "0"}
```

str="1"

i=2

```
dp[i][j] = Math.max(dp[i][j], dp[i - cntZero][j - cntOne] +
1);
```

i/j					
		0	1	2	3
	0				
	1		1	1	1
	2		1	1	2
	3		1	2	2
	4		1	2	3
	5		1	2	3

str="1"
cntZero=0
cntOne=1

dp[i][j]=
dp[i-cntZero][j-cntOne]

```
new String[]{"10", "0001", "111001", "1", "0"}
```

```
dp[i][j] = Math.max(dp[i][j], dp[i - cntZero][j - cntOne] +
1);
```

i/j					
		0	1	2	3
	0				
	1		1	1	1
	2		1	2	2
	3		1	2	2
	4		1	2	3
	5		1	2	3

str="1"
cntZero=0
cntOne=1

dp[i][j]=
dp[i-cntZero][j-cntOne]

```
new String[]{"10", "0001", "111001", "1", "0"}
```

```
dp[i][j] = Math.max(dp[i][j], dp[i - cntZero][j - cntOne] +
1);
```

i/j					
		0	1	2	3
	0				
	1		1	1	1
	2		1	2	2
	3		1	2	2
	4		1	2	3
	5		1	2	3

str="1"
cntZero=0
cntOne=1

```
new String[]{"10", "0001", "111001", "1", "0"}
```

str="1"

i=1

```
dp[i][j] = Math.max(dp[i][j], dp[i - cntZero][j - cntOne] +
1);
```

i/j					
		0	1	2	3
	0				
	1		1	1	2
	2		1	2	2
	3		1	2	2
	4		1	2	3
	5		1	2	3

str="1"
cntZero=0
cntOne=1

dp[i][j]=
dp[i-cntZero][j-cntOne]

```
new String[]{"10", "0001", "111001", "1", "0"}
```

```
dp[i][j] = Math.max(dp[i][j], dp[i - cntZero][j - cntOne] +
1);
```

i/j					
		0	1	2	3
	0				
	1		1	2	2
	2		1	2	2
	3		1	2	2
	4		1	2	3
	5		1	2	3

str="1"
cntZero=0
cntOne=1

dp[i][j]=
dp[i-cntZero][j-cntOne]

```
new String[]{"10", "0001", "111001", "1", "0"}
```

```
dp[i][j] = Math.max(dp[i][j], dp[i - cntZero][j - cntOne] +
1);
```

i/j					
		0	1	2	3
	0				
	1		1	2	2
	2		1	2	2
	3		1	2	2
	4		1	2	3
	5		1	2	3

str="1"
cntZero=0
cntOne=1

```
new String[]{"10", "0001", "111001", "1", "0"}
```

str="1"

i=0

```
dp[i][j] = Math.max(dp[i][j], dp[i - cntZero][j - cntOne] +
1);
```

i/j					
		0	1	2	3
	0				1
	1		1	2	2
	2		1	2	2
	3		1	2	2
	4		1	2	3
	5		1	2	3

str="1"
cntZero=0
cntOne=1

dp[i][j]=
dp[i-cntZero][j-cntOne]

```
new String[]{"10", "0001", "111001", "1", "0"}
```

```
dp[i][j] = Math.max(dp[i][j], dp[i - cntZero][j - cntOne] +
1);
```

i/j					
		0	1	2	3
	0			1	1
	1		1	2	2
	2		1	2	2
	3		1	2	2
	4		1	2	3
	5		1	2	3

str="1"
cntZero=0
cntOne=1

dp[i][j]=
dp[i-cntZero][j-cntOne]

```
new String[]{"10", "0001", "111001", "1", "0"}
```

```
dp[i][j] = Math.max(dp[i][j], dp[i - cntZero][j - cntOne] +
1);
```

i/j					
		0	1	2	3
	0		1	1	1
	1		1	2	2
	2		1	2	2
	3		1	2	2
	4		1	2	3
	5		1	2	3

str="1"
cntZero=0
cntOne=1

dp[i][j]=
dp[i-cntZero][j-cntOne]

```
new String[]{"10", "0001", "111001", "1", "0"}
```

```
str="0"
i=5
```

```
dp[i][j] = Math.max(dp[i][j], dp[i - cntZero][j - cntOne] +
1);
```

i/j					
		0	1	2	3
	0		1	1	1
	1		1	2	2
	2		1	2	2
	3		1	2	2
	4		1	2	3
	5		1	2	4

str="1"
cntZero=1
cntOne=0

dp[i][j]=
dp[i-cntZero][j-cntOne]+
1

```
new String[]{"10", "0001", "111001", "1", "0"}
```

```
dp[i][j] = Math.max(dp[i][j], dp[i - cntZero][j - cntOne] +
1);
```

i/j					
		0	1	2	3
	0		1	1	1
	1		1	2	2
	2		1	2	2
	3		1	2	2
	4		1	2	3
	5		1	3	4

str="1"
cntZero=1
cntOne=0

dp[i][j]=
dp[i-cntZero][j-cntOne]+1

```
new String[]{"10", "0001", "111001", "1", "0"}
```

```
dp[i][j] = Math.max(dp[i][j], dp[i - cntZero][j - cntOne] +
1);
```

i/j					
		0	1	2	3
	0		1	1	1
	1		1	2	2
	2		1	2	2
	3		1	2	2
	4		1	2	3
	5		2	3	4

str="1"
cntZero=1
cntOne=0

dp[i][j]=
dp[i-cntZero][j-cntOne]+1

```
new String[]{"10", "0001", "111001", "1", "0"}
```

```
dp[i][j] = Math.max(dp[i][j], dp[i - cntZero][j - cntOne] +
1);
```

i/j					
		0	1	2	3
	0		1	1	1
	1		1	2	2
	2		1	2	2
	3		1	2	2
	4		1	2	3
	5	1	2	3	4

str="1"
cntZero=1
cntOne=0

dp[i][j]=
dp[i-cntZero][j-cntOne]+1

```
new String[]{"10", "0001", "111001", "1", "0"}
```

str="0"

i=4

```
dp[i][j] = Math.max(dp[i][j], dp[i - cntZero][j - cntOne] +
1);
```

i/j					
		0	1	2	3
	0		1	1	1
	1		1	2	2
	2		1	2	2
	3		1	2	2
	4		1	3	3
	5	1	2	3	4

str="1"
cntZero=1
cntOne=0

3==2+1

```
new String[]{"10", "0001", "111001", "1", "0"}
```

```
dp[i][j] = Math.max(dp[i][j], dp[i - cntZero][j - cntOne] +
1);
```

i/j					
		0	1	2	3
	0		1	1	1
	1		1	2	2
	2		1	2	2
	3		1	2	2
	4		1	3	3
	5	1	2	3	4

str="1"
cntZero=1
cntOne=0

3==2+1

```
new String[]{"10", "0001", "111001", "1", "0"}
```

```
dp[i][j] = Math.max(dp[i][j], dp[i - cntZero][j - cntOne] +
1);
```

i/j					
		0	1	2	3
	0		1	1	1
	1		1	2	2
	2		1	2	2
	3		1	2	2
	4		2	3	3
	5	1	2	3	4

str="1"
cntZero=1
cntOne=0

dp[i][j]=
dp[i-cntZero][j-cntOne]+1

```
new String[]{"10", "0001", "111001", "1", "0"}
```

```
dp[i][j] = Math.max(dp[i][j], dp[i - cntZero][j - cntOne] +
1);
```

i/j					
		0	1	2	3
	0		1	1	1
	1		1	2	2
	2		1	2	2
	3		1	2	2
	4	1	2	3	3
	5	1	2	3	4

str="1"
cntZero=1
cntOne=0

dp[i][j]=
dp[i-cntZero][j-cntOne]+1

```
new String[]{"10", "0001", "111001", "1", "0"}
```

str="0"

i=3

```
dp[i][j] = Math.max(dp[i][j], dp[i - cntZero][j - cntOne] +
1);
```

i/j					
		0	1	2	3
	0		1	1	1
	1		1	2	2
	2		1	2	2
	3		1	2	3
	4		2	3	3
	5		2	3	4

str="1"
cntZero=1
cntOne=0

2<2+1=3

dp[i][j]=
dp[i-cntZero][j-cntOne]+1

```
new String[]{"10", "0001", "111001", "1", "0"}
```

```
dp[i][j] = Math.max(dp[i][j], dp[i - cntZero][j - cntOne] +
1);
```

i/j					
		0	1	2	3
	0		1	1	1
	1		1	2	2
	2		1	2	2
	3		1	3	3
	4	1	2	3	3
	5	1	2	3	4

str="1"
cntZero=1
cntOne=0

2<2+1=3

dp[i][j]=
dp[i-cntZero][j-cntOne]+1

```
new String[]{"10", "0001", "111001", "1", "0"}
```

```
dp[i][j] = Math.max(dp[i][j], dp[i - cntZero][j - cntOne] +
1);
```

i/j					
		0	1	2	3
	0		1	1	1
	1		1	2	2
	2		1	2	2
	3		2	3	3
	4	1	2	3	3
	5	1	2	3	4

str="1"
cntZero=1
cntOne=0

1<1+1=2

dp[i][j]=
dp[i-cntZero][j-cntOne]+1

```
new String[]{"10", "0001", "111001", "1", "0"}
```

```
dp[i][j] = Math.max(dp[i][j], dp[i - cntZero][j - cntOne] +
1);
```

i/j					
		0	1	2	3
	0		1	1	1
	1		1	2	2
	2		1	2	2
	3	1	2	3	3
	4	1	2	3	3
	5	1	2	3	4

str="1"
cntZero=1
cntOne=0

1<1+1=2

dp[i][j]=
dp[i-cntZero][j-cntOne]+1

```
new String[]{"10", "0001", "111001", "1", "0"}
```

str="0"

i=2

```
dp[i][j] = Math.max(dp[i][j], dp[i - cntZero][j - cntOne] +
1);
```

i/j					
		0	1	2	3
	0		1	1	1
	1		1	2	2
	2		1	2	3
	3	1	2	3	3
	4	1	2	3	3
	5	1	2	3	4

str="1"
cntZero=1
cntOne=0

2<2+1=3

dp[i][j]=
dp[i-cntZero][j-cntOne]+1

```
new String[]{"10", "0001", "111001", "1", "0"}
```

```
dp[i][j] = Math.max(dp[i][j], dp[i - cntZero][j - cntOne] +
1);
```

i/j					
		0	1	2	3
	0		1	1	1
	1		1	2	2
	2		1	3	3
	3	1	2	3	3
	4	1	2	3	3
	5	1	2	3	4

str="1"
cntZero=1
cntOne=0

2<2+1=3

dp[i][j]=
dp[i-cntZero][j-cntOne]+1

```
new String[]{"10", "0001", "111001", "1", "0"}
```

```
dp[i][j] = Math.max(dp[i][j], dp[i - cntZero][j - cntOne] +
1);
```

i/j					
		0	1	2	3
	0		1	1	1
	1		1	2	2
	2		2	3	3
	3	1	2	3	3
	4	1	2	3	3
	5	1	2	3	4

str="1"
cntZero=1
cntOne=0

1<1+1=2

dp[i][j]=
dp[i-cntZero][j-cntOne]+1

```
new String[]{"10", "0001", "111001", "1", "0"}
```

```
dp[i][j] = Math.max(dp[i][j], dp[i - cntZero][j - cntOne] +
1);
```

i/j					
		0	1	2	3
	0		1	1	1
	1		1	2	2
	2	1	2	3	3
	3	1	2	3	3
	4	1	2	3	3
	5	1	2	3	4

str="1"
cntZero=1
cntOne=0

0<1+1=2

dp[i][j]=
dp[i-cntZero][j-cntOne]+1

```
new String[]{"10", "0001", "111001", "1", "0"}
```

```
str="0"
i=1
```

```
dp[i][j] = Math.max(dp[i][j], dp[i - cntZero][j - cntOne] +
1);
```

i/j					
		0	1	2	3
	0		1	1	1
	1	1	1	2	2
	2	1	2	3	3
	3	1	2	3	3
	4	1	2	3	3
	5	1	2	3	4

str="1"
cntZero=1
cntOne=0

2==1+1

```
new String[]{"10", "0001", "111001", "1", "0"}
```

```
dp[i][j] = Math.max(dp[i][j], dp[i - cntZero][j - cntOne] +
1);
```

i/j					
		0	1	2	3
	0		1	1	1
	1	1	1	2	2
	2	1	2	3	3
	3	1	2	3	3
	4	1	2	3	3
	5	1	2	3	4

str="1"
cntZero=1
cntOne=0

2==1+1

```
new String[]{"10", "0001", "111001", "1", "0"}
```

```
dp[i][j] = Math.max(dp[i][j], dp[i - cntZero][j - cntOne] +
1);
```

i/j					
		0	1	2	3
	0		1	1	1
	1	1	2	2	2
	2	1	2	3	3
	3	1	2	3	3
	4	1	2	3	3
	5	1	2	3	4

str="1"
cntZero=1
cntOne=0

1<1+1=2

dp[i][j]=
dp[i-cntZero][j-cntOne]+1

```
new String[]{"10", "0001", "111001", "1", "0"}
```

Top Down Approach

```
for (int j = 0; j < ways.size(); ++j) {
    result = min(result, topDown(target - ways[j]) + cost/ path /
sum);
}
return memo[/*state parameters*/] = result;
```

746. Min Cost Climbing Stairs

```
int result = min(minCost(n-1, cost, memo), minCost(n-2, cost, memo))
+ (n == cost.size() ? 0 : cost[n]);
return memo[n] = result;
```

1. Max, Min

routes[i] = min(routes[i-1], routes[i-2], ... , routes[i-k]) + cost[i]

Top Down Approach

```
for (int i = 0; i < ways.size(); ++i) {
    result = min(result, topDown(target - ways[i]) + cost/ path / sum);
}
return memo[/*state parameters*/] = result;
```

Bottom Up Approach

```
for (int i = 1; i ≤ target; ++i) {
    for (int j = 0; j < ways.size(); ++j) {
        if (ways[j] ≤ i) {
            dp[i] = min(dp[i], dp[i - ways[j]] + cost / path / sum) ;
        }
    }
}
return dp[target]
```

routes[i] = min(routes[i-1], routes[i-2], … , routes[i-k]) + cost[i]

1. Max, Min

```
int result = min(minCost(n-1, cost, memo), minCost(n-2, cost,
memo)) + (n == cost.size() ? 0 : cost[n]);

return memo[n] = result;
```

Top Down Approach

```
for (int i = 2; i <= n; ++i) {
    dp[i] = min(dp[i-1], dp[i-2]) + (i == n ? 0 : cost[i]);
}
return dp[n]
```

Bottom Up Approach

1. Max, Min

746. Min Cost Climbing Stairs

Top Down Approach

```
int result = min(pathSum(i+1, j, grid, memo),
pathSum(i, j+1, grid, memo)) + grid[i][j];

return memo[i][j] = result;
```

No topDown[target-ways[j]]]

No Cost

Bottom Up Approach

```
for (int i = 1; i < n; ++i) {
    for (int j = 1; j < m; ++j) {
        grid[i][j] = min(grid[i-1][j], grid[i][j-1]) +
grid[i][j];
    }
}
return grid[n-1][m-1]
```

No dp[i-ways[j]]

1. Max, Min

64. Minimum Path Sum

Top Down Approach

```
for (int i = 0; i < coins.size(); ++i) {
    if (coins[i] ≤ target) { // check validity of a sub-problem
        result = min(ans, CoinChange(target - coins[i], coins) + 1);
    }
}
return memo[target] = result;
```

No Cost

Bottom Up Approach

```
for (int j = 1; j ≤ amount; ++j) {
    for (int i = 0; i < coins.size(); ++i) {
        if (coins[i] ≤ j) {
            dp[j] = min(dp[j], dp[j - coins[i]] + 1);
        }
    }
}
```

1. Max, Min

322. Coin Change

Given an integer n, return *the minimum number of operations to get the character 'A' exactly* n *times on the screen.*

```
public int minSteps(int n) {
    int ans = 0, d = 2;
    while (n > 1) {
        while (n % d == 0) {
            ans += d;
            n /= d;
        }
        d++;
    }
    return ans;
}
```

we find the next smaller length sequence (than the one under consideration) which can be copied and then pasted over multiple times to generate the desired sequence. The moment we find a length that divides our required sequence length perfectly, then we don't need to check for any smaller length sequences.

```
public int minSteps(int n) {
    int[] dp = new int[n+1];
    for (int i = 2; i ≤ n; i++) {
        dp[i] = i;
        for (int j = i-1; j > 1; j--) {
 // if sequence of length 'j' can be pasted
multiple times to get length 'i' sequence
            if (i % j == 0) {
 // we just need to paste sequence j (i/j - 1)
times, hence additional (i/j) times since we
need to copy it first as well.
                // we don't need checking any
smaller length sequences
                dp[i] = dp[j] + (i/j);
                break;
            }
        }
    }
    return dp[n];
}
```

xt = "aaabcxy

Prime Factorization

```
public int minSteps(int n) {
    int ans = 0, d = 2;
    while (n > 1) {
        while (n % d == 0) {
            ans += d;
            n /= d;
        }
        d++;
    }
    return ans;
}
```

We can break our moves into groups of `(copy, paste, ..., paste)`. Let `C` denote copying and `P` denote pasting. Then for example, in the sequence of moves `CPPCPPPPCP`, the groups would be `[CPP][CPPPP][CP]`.

Say these groups have lengths `g_1, g_2, ....` After parsing the first group, there are `g_1 'A'`s. After parsing the second group, there are `g_1 * g_2 'A'`s, and so on. At the end, there are `g_1 * g_2 * ... * g_n 'A'`s.

We want exactly `N = g_1 * g_2 * ... * g_n`.
If any of the `g_i` are composite, say `g_i = p * q`, then we can split this group into two groups (the first of which has one copy followed by `p-1` pastes, while the second group having one copy and `q-1` pastes).
Such a split never uses more moves: we use `p+q` moves when splitting, and `pq` moves previously.

As `p+q ⩽ pq` is equivalent to `1 ⩽ (p-1)(q-1)`, which is true as long as `p ⩾ 2` and `q ⩾ 2`.

Algorithm By the above argument, we can suppose `g_1, g_2, ...` is the prime factorization of `N`, and the answer is therefore the sum of these prime factors.

1	2	3	4
4	-1	3	-2
-2	5	-3	2
7	-6	9	8

dp

1	2	3	4
5			

1	2	3	4
4	-1	3	-2
-2	5	-3	2
7	-6	9	8

dp

1	2	3	4
5	0		

1	2	3	4
4	-1	3	-2
-2	5	-3	2
7	-6	9	8

dp

1	2	3	4
5	0	5	

```
int minFallingPathSum(vector<vector<int>>& a) {
    int n = a.size();
    vvi dp(n, vi(n));
    dp[0] = a[0];
    FORI (1, n) {
        FORJ (0, n) {
            dp[i][j] = a[i][j] + dp[i - 1][j];
            if (j > 0)      dp[i][j] = min(dp[i][j], a[i][j] + dp[i - 1][j - 1]);
            if (j < n - 1) dp[i][j] = min(dp[i][j], a[i][j] + dp[i - 1][j + 1]);
        }
    }
    return *min_element(dp[n - 1].begin(), dp[n - 1].end());
}
```

1	2	3	4
4	-1	3	-2
-2	5	-3	2
7	-6	9	8

dp

1	2	3	4
5	0	5	1

Min Falling Path Sum

```
public int minFallingPathSum(int[][] A) {
   for (int i = 1; i < A.length; ++i)
       for (int j = 0; j < A.length; ++j)
           A[i][j] += Math.min(A[i - 1][j], Math.min(A[i - 1][Math.max(0, j - 1)], A[i -
1][Math.min(A.length - 1, j + 1)]));
   return Arrays.stream(A[A.length - 1]).min().getAsInt();
}
```

The min path to get to element `A[i][j]` is the min of `A[i - 1][j - 1]`, `A[i - 1][j]` and `A[i - 1][j + 1]`.

Starting from row 1, we add the min path to each element. The smallest number in the last row is the miminum path sum.

Example:

[1, 2, 3]

[4, 5, 6] => [5, 6, 8]

[7, 8, 9] => [7, 8, 9] => [12, 13, 15]

2. Distinct Ways

Paths, Stairs, Knights, Soup, Dominos, LIS, Dice

```
routes[i] = routes[i-1] + routes[i-2], ... ,   + routes[i-k]
```

Num of ways to stay in same place after steps

https://leetcode.com/problems/number-of-ways-to-stay-in-the-same-place-after-some-steps/

i/j							
		0	1	2	3	4	5
	0						
	1	1	1				
	2						
	3						
	4						

```
//第i步的情况由第i-1步情况得到，三种情况：静止在原地 + 从j+1往回跳一步的情况 + 从j-1向前跳一步的情况
dp[i][j] = (dp[i-1][j] + dp[i-1][j+1] + (j>0?dp[i-1][j-1]:0))%1000000007;
```

i/j						
		0	1	2	3	4
	0	1				
	1	1				
	2					
	3					
	4					

```
//第i步的情况由第i-1步情况得到，三种情况：静止在原地 + 从j+1往回跳一步的情况 + 从j-1向前跳一步的情况
dp[i][j] = (dp[i-1][j] + dp[i-1][j+1] + (j>0?dp[i-1][j-1]:0))%1000000007;
```

i/j						
		0	1	2	3	4
	0	1				
	1	1	1			
	2					
	3					
	4					

$i=2$

```
//第i步的情况由第i-1步情况得到，三种情况：静止在原地 + 从j+1往回跳一步的情况 + 从j-1向前跳一步的情况
dp[i][j] = (dp[i-1][j] + dp[i-1][j+1] + (j>0?dp[i-1][j-1]:0))%1000000007;
```

i/j						
		0	1	2	3	4
	0	1				
	1	1	1			
	2	2	2			
	3					
	4					

```
//第i步的情况由第i-1步情况得到，三种情况：静止在原地 + 从j+1往回跳一步的情况 + 从j-1向前跳一步的情况
dp[i][j] = (dp[i-1][j] + dp[i-1][j+1] + (j>0?dp[i-1][j-1]:0))%1000000007;
```

i/j						
		0	1	2	3	4
	0	1				
	1	1	1			
	2	2	2	2		
	3					
	4					

```
//第i步的情况由第i-1步情况得到，三种情况：静止在原地 + 从j+1往回跳一步的情况 + 从j-1向前跳一步的情况
dp[i][j] = (dp[i-1][j] + dp[i-1][j+1] + (j>0?dp[i-1][j-1]:0))%1000000007;
```

i/j						
		0	1	2	3	4
	0	1				
	1	1	1			
	2	2	2	1		
	3					
	4					

i=3

```
//第i步的情况由第i-1步情况得到，三种情况：静止在原地+ 从j+1往回跳一步的情况 + 从j-1向前跳一步的情况
dp[i][j] = (dp[i-1][j] + dp[i-1][j+1] + (j>0?dp[i-1][j-1]:0))%1000000007;
```

i/j						
		0	1	2	3	4
	0	1				
	1	1	1			
	2	2	2	1		
	3	4				
	4					

```
//第i步的情况由第i-1步情况得到，三种情况：静止在原地 + 从j+1往回跳一步的情况 + 从j-1向前跳一步的情况
dp[i][j] = (dp[i-1][j] + dp[i-1][j+1] + (j>0?dp[i-1][j-1]:0))%1000000007;
```

i/j						
		0	1	2	3	4
	0	1				
	1	1	1			
	2	2	2	1		
	3	4	5			
	4					

```
//第i步的情况由第i-1步情况得到，三种情况：静止在原地 + 从j+1往回跳一步的情况 + 从j-1向前跳一步的情况
dp[i][j] = (dp[i-1][j] + dp[i-1][j+1] + (j>0?dp[i-1][j-1]:0))%1000000007;
```

i/j						
		0	1	2	3	4
	0	1				
	1	1	1			
	2	2	2	1		
	3	4	5	3		
	4					

```
//第i步的情况由第i-1步情况得到，三种情况：静止在原地 + 从j+1往回跳一步的情况 + 从j-1向前跳一步的情况
dp[i][j] = (dp[i-1][j] + dp[i-1][j+1] + (j>0?dp[i-1][j-1]:0))%1000000007;
```

i/j						
		0	1	2	3	4
	0	1				
	1	1	1			
	2	2	2	1		
	3	4	5	3	1	
	4					

i=4

```
//第i步的情况由第i-1步情况得到，三种情况：静止在原地 + 从j+1往回跳一步的情况 + 从j-1向前跳一步的情况
dp[i][j] = (dp[i-1][j] + dp[i-1][j+1] + (j>0?dp[i-1][j-1]:0))%1000000007;
```

i/j						
		0	1	2	3	4
	0	1				
	1	1	1			
	2	2	2	1		
	3	4	5	3	1	
	4	9				

```
//第i步的情况由第i-1步情况得到，三种情况：静止在原地 + 从j+1往回跳一步的情况 + 从j-1向前跳一步的情况
dp[i][j] = (dp[i-1][j] + dp[i-1][j+1] + (j>0?dp[i-1][j-1]:0))%1000000007;
```

i/j						
		0	1	2	3	4
	0	1				
	1	1	1			
	2	2	2	1		
	3	4	5	3	1	
	4	9	12			

```
//第i步的情况由第i-1步情况得到，三种情况：静止在原地 + 从j+1往回跳一步的情况 + 从j-1向前跳一步的情况
dp[i][j] = (dp[i-1][j] + dp[i-1][j+1] + (j>0?dp[i-1][j-1]:0))%1000000007;
```

i/j						
		0	1	2	3	4
	0	1				
	1	1	1			
	2	2	2	1		
	3	4	5	3	1	
	4	9	12	9	4	

```
for (int j = 0; j < ways.size(); ++j) {
    result += topDown(target - ways[j]);
}
return memo[/*state parameters*/] = result;
```

Top Down Approach

```
for (int i = 1; i ≤ target; ++i) {
    for (int j = 0; j < ways.size(); ++j) {
        if (ways[j] ≤ i) {
            dp[i] += dp[i - ways[j]];
        }
    }
}
return dp[target]
```

Bottom Up Approach

2. Distinct Way

```
int result = climbStairs(n-1, memo) + climbStairs(n-2, memo);

return memo[n] = result;
```

Top Down Approach

```
for (int stair = 2; stair ≤ n; ++stair) {
    for (int step = 1; step ≤ 2; ++step) {
        dp[stair] += dp[stair-step];
    }
}
```

Bottom Up Approach

70. Climbing Stairs

2. Distinct Way

```
int result = UniquePaths(x-1, y) + UniquePaths(x, y-1);

return memo[x][y] = result;
```

Top Down Approach

```
for (int i = 1; i < m; ++i) {
    for (int j = 1; j < n; ++j) {
        dp[i][j] = dp[i][j-1] + dp[i-1][j];
    }
}
```

Bottom Up Approach

2. Distinct Way

62. Unique Paths

```
for (int rep = 1; rep ≤ d; ++rep) {
    vector<int> new_ways(target+1);
    for (int already = 0; already ≤ target; ++already) {
        for (int pipe = 1; pipe ≤ f; ++pipe) {
            if (already - pipe ≥ 0) {
                new_ways[already] += ways[already - pipe];
                new_ways[already] %= mod;
            }
        }
    }
    ways = new_ways;
}
```

1155. Number of Dice Rolls With Target Sum

2. Distinct Way

DP (Distinct Ways)

17

3. Merge Intervals

Paths, Stairs, Knights, Soup, Dominos, LIS, Dice

```
dp[i][j] = dp[i][k] + result[k] + dp[k+1][j]
```

```
for (int k = i; k ≤ j; ++k) {
   result = max(result, topDown(nums, i, k-1) + result[k] + topDown(nums,
k+1, j));
}
return memo[/*state parameters*/] = result;
```

Top Down Approach

```
for(int l = 1; l<n; l++) {
   for(int i = 0; i<n-l; i++) {
       int j = i+l;
       for(int k = i; k<j; k++) {
           dp[i][j] = max(dp[i][j], dp[i][k] + result[k] + dp[k+1][j]);
       }
   }
}
return dp[0][n-1];
```

Bottom Up Approach

3. Merge Intervals

```
for (int l = 1; l < n; ++l) {
    for (int i = 0; i < n - l; ++i) {
        int j = i + l;
        dp[i][j] = INT_MAX;
        for (int k = i; k < j; ++k) {
            dp[i][j] = min(dp[i][j], dp[i][k] + dp[k+1][j] +
maxs[i][k] * maxs[k+1][j]);
        }
    }
}
```

Bottom Up Approach

1130. Minimum Cost Tree From Leaf Values

3. Merging Intervals

```
for (int k = i; k ≤ j; ++k) {
   result = max(result, topDown(nums, i, k-1, memo) + (i-1 ≥ 0 ? nums[i-1] :
1) * nums[k] * (j+1 < nums.size() ? nums[j+1] : 1) + topDown(nums, k+1, j,
memo));
}
return memo[i][j] = result;
```

Top Down Approach

```
for(int l = 1; l < n; l++) {
   for(int i = 0; i < n-l; i++) {
       int j = i+l;
       for(int k = i; k ≤ j; k++) {
           dp[i][j] = max(dp[i][j], (((k>i && k>0) ? dp[i][k-1] : 0) + (i>0 ? nums[i-1] :
1) * nums[k] * (j<n-1 ? nums[j+1] : 1) + ((k<j && k<n-1) ? dp[k+1][j] : 0)));
       }
   }
}
return dp[0][n-1];
```

Bottom Up Approach

312. Burst Balloons

3. Merging Intervals

```
// i - indexing string s1 // j - indexing string
s2
for (int i = 1; i <= n; ++i) {
 for (int j = 1; j <= m; ++j) {
   if (s1[i-1] == s2[j-1]) {
    dp[i][j] = /*code*/;
   } else {
    p[i][j] = /*code*/;
   }
  }
}
```

```
for (int l = 1; l < n; ++l) {
 for (int i = 0; i < n-l; ++i) {
   int j = i + l;
   if (s[i] == s[j]) {
     dp[i][j] = /*code*/;
   } else {
     dp[i][j] = /*code*/;
   }
 }
}
```

4. DP on Strings

LPS2, ED, DS, ASCII Delete Sum, SCS, LCS

Longest Common Subsquence

Longest Common Subsequence

		a	b	c	d	a	a
		0	0	0	0	0	0
a	0						
c	0						
b	0						
c	0						
f	0						

Longest Common Subsequence

		a	b	c	d	a	a
		0	0	0	0	0	0
a	0	1	1	1	1	1	1
c	0						
b	0						
c	0						
f	0						

Longest Common Subsequence

		a	b	c	d	a	a
		0	0	0	0	0	0
a	0	1	1	1	1	1	1
c	0	1					
b	0						
c	0						
f	0						

Longest Common Subsequence

		a	b	c	d	a	a
		0	0	0	0	0	0
a	0	1	1	1	1	1	1
c	0	1	1				
b	0						
c	0						
f	0						

Longest Common Subsequence

		a	b	c	d	a	a
		0	0	0	0	0	0
a	0	1	1	1	1	1	1
c	0	1	1	2			
b	0						
c	0						
f	0						

1+1=2

Longest Common Subsequence

		a	b	c	d	a	a
		0	0	0	0	0	0
a	0	1	1	1	1	1	1
c	0	1	1	2	2		
b	0						
c	0						
f	0						

Longest Common Subsequence

		a	b	c	d	a	a
		0	0	0	0	0	0
a	0	1	1	1	1	1	1
c	0	1	1	2	2	2	2
b	0						
c	0						
f	0						

		a	b	c	d	a	a
		0	0	0	0	0	0
a	0	1	1	1	1	1	1
c	0	1	1	2	2	2	2
b	0						
c	0						
f	0						

Longest Common Subsequence

		a	b	c	d	a	a
		0	0	0	0	0	0
a	0	1	1	1	1	1	1
c	0	1	1	2	2	2	2
b	0	1					
c	0						
f	0						

		a	b	c	d	a	a
		0	0	0	0	0	0
a	0	1	1	1	1	1	1
c	0	1	1	2	2	2	2
b	0	1	2				
c	0						
f	0						

Longest Common Subsequence

		a	b	c	d	a	a
		0	0	0	0	0	0
a	0	1	1	1	1	1	1
c	0	1	1	2	2	2	2
b	0	1	2	2			
c	0						
f	0						

		a	b	c	d	a	a
		0	0	0	0	0	0
a	0	1	1	1	1	1	1
c	0	1	1	2	2	2	2
b	0	1	2	2	2		
c	0						
f	0						

Longest Common Subsequence

		a	b	c	d	a	a
		0	0	0	0	0	0
a	0	1	1	1	1	1	1
c	0	1	1	2	2	2	2
b	0	1	2	2	2	2	
c	0						
f	0						

		a	b	c	d	a	a
		0	0	0	0	0	0
a	0	1	1	1	1	1	1
c	0	1	1	2	2	2	2
b	0	1	2	2	2	2	2
c	0						
f	0						

$i=3$

Longest Common Subsequence

Longest Common Subsequence

		a	b	c	d	a	a
		0	0	0	0	0	0
a	0	1	1	1	1	1	1
c	0	1	1	2	2	2	2
b	0	1	2	2	2	2	2
c	0	1					
f	0						

		a	b	c	d	a	a
		0	0	0	0	0	0
a	0	1	1	1	1	1	1
c	0	1	1	2	2	2	2
b	0	1	2	2	2	2	2
c	0	1	2				
f	0						

Longest Common Subsequence

		a	b	c	d	a	a
		0	0	0	0	0	0
a	0	1	1	1	1	1	1
c	0	1	1	2	2	2	2
b	0	1	2	2	2	2	2
c	0	1	2	3			
f	0						

		a	b	c	d	a	a
		0	0	0	0	0	0
a	0	1	1	1	1	1	1
c	0	1	1	2	2	2	2
b	0	1	2	2	2	2	2
c	0	1	2	3	3		
f	0						

Longest Common Subsequence

		a	b	c	d	a	a
		0	0	0	0	0	0
a	0	1	1	1	1	1	1
c	0	1	1	2	2	2	2
b	0	1	2	2	2	2	2
c	0	1	2	3	3	3	
f	0						

		a	b	c	d	a	a
		0	0	0	0	0	0
a	0	1	1	1	1	1	1
c	0	1	1	2	2	2	2
b	0	1	2	2	2	2	2
c	0	1	2	3	3	3	3
f	0						

$i=4$

Longest Common Subsequence

Longest Common Subsequence

		a	b	c	d	a	a
		0	0	0	0	0	0
a	0	1	1	1	1	1	1
c	0	1	1	2	2	2	2
b	0	1	2	2	2	2	2
c	0	1	2	3	3	3	3
f	0	1	2	3	3	3	

		a	b	c	d	a	f
		0	0	0	0	0	0
a	0	1	1	1	1	1	1
c	0	1	1	2	2	2	2
b	0	1	2	2	2	2	2
c	0	1	2	3	3	3	3
f	0	1	2	3	4	3	4

Distinct Subsequence 115

	0	1	2	3	4	5	6	7
0	1							
1								
2								
3								
4								
5								
6								

```java
// filling the first row: with 1s
for(int j=0; j≤S.length(); j++) {
   mem[0][j] = 1;
}

// the first column is 0 by default in every other
rows but the first, which we need.

for(int i=0; i<T.length(); i++) {
   for(int j=0; j<S.length(); j++) {
       if(T.charAt(i) == S.charAt(j)) {
           mem[i+1][j+1] = mem[i][j] + mem[i+1][j];
       } else {
           mem[i+1][j+1] = mem[i+1][j];
       }
   }
}
```

	0	1	2	3	4	5	6	7
0	1	1	1	1	1	1	1	1
1								
2								
3								
4								
5								
6								

```java
// filling the first row: with 1s
for(int j=0; j⩽S.length(); j++) {
    mem[0][j] = 1;
}

// the first column is 0 by default in every other
rows but the first, which we need.

for(int i=0; i<T.length(); i++) {
    for(int j=0; j<S.length(); j++) {
        if(T.charAt(i) == S.charAt(j)) {
            mem[i+1][j+1] = mem[i][j] + mem[i+1][j];
        } else {
            mem[i+1][j+1] = mem[i+1][j];
        }
    }
}
```

	0	1	2	3	4	5	6	7
0	1	1	1	1	1	1	1	1
1		1	1	1	1	1	1	1
2								
3								
4								
5								
6								

```java
// filling the first row: with 1s
for(int j=0; j≤S.length(); j++) {
    mem[0][j] = 1;
}

// the first column is 0 by default in every other
rows but the first, which we need.

for(int i=0; i<T.length(); i++) {
    for(int j=0; j<S.length(); j++) {
        if(T.charAt(i) == S.charAt(j)) {
            mem[i+1][j+1] = mem[i][j] + mem[i+1][j];
        } else {
            mem[i+1][j+1] = mem[i+1][j];
        }
    }
}
```

		r	a	b	b	b	i	t	
		0	1	2	3	4	5	6	7
r	0	1	1	1	1	1	1	1	1
a	1		1	1	1	1	1	1	1
b	2			1	1	1	1	1	1
b	3								
i	4								
t	5								
	6								

```java
// filling the first row: with 1s
for(int j=0; j≤S.length(); j++) {
   mem[0][j] = 1;
}

// the first column is 0 by default in every other
rows but the first, which we need.

for(int i=0; i<T.length(); i++) {
   for(int j=0; j<S.length(); j++) {
       if(T.charAt(i) == S.charAt(j)) {
           mem[i+1][j+1] = mem[i][j] + mem[i+1][j];
       } else {
           mem[i+1][j+1] = mem[i+1][j];
       }
   }
}
```

		r	a	b	b	b	i	t	
		0	1	2	3	4	5	6	7
r	0	1	1	1	1	1	1	1	1
a	1		1	1	1	1	1	1	1
b	2			1	1	1	1	1	1
b	3				1				
i	4								
t	5								
	6								

```java
// filling the first row: with 1s
for(int j=0; j≤S.length(); j++) {
   mem[0][j] = 1;
}

// the first column is 0 by default in every other
rows but the first, which we need.

for(int i=0; i<T.length(); i++) {
   for(int j=0; j<S.length(); j++) {
       if(T.charAt(i) == S.charAt(j)) {
           mem[i+1][j+1] = mem[i][j] + mem[i+1][j];
       } else {
           mem[i+1][j+1] = mem[i+1][j];
       }
   }
}
```

		r	a	b	b	b	i	t	
		0	1	2	3	4	5	6	7
r	0	1	1	1	1	1	1	1	1
a	1		1	1	1	1	1	1	1
b	2			1	1	1	1	1	1
b	3				1	2			
i	4								
t	5								
	6								

```java
// filling the first row: with 1s
for(int j=0; j≤S.length(); j++) {
   mem[0][j] = 1;
}

// the first column is 0 by default in every other
rows but the first, which we need.

for(int i=0; i<T.length(); i++) {
   for(int j=0; j<S.length(); j++) {
       if(T.charAt(i) == S.charAt(j)) {
           mem[i+1][j+1] = mem[i][j] + mem[i+1][j];
       } else {
           mem[i+1][j+1] = mem[i+1][j];
       }
   }
}
```

		r	a	b	b	b	i	t	
		0	1	2	3	4	5	6	7
r	0	1	1	1	1	1	1	1	1
a	1		1	1	1	1	1	1	1
b	2			1	1	1	1	1	1
b	3				1	2	3		
i	4								
t	5								
	6								

```java
// filling the first row: with 1s
for(int j=0; j≤S.length(); j++) {
   mem[0][j] = 1;
}

// the first column is 0 by default in every other
rows but the first, which we need.

for(int i=0; i<T.length(); i++) {
   for(int j=0; j<S.length(); j++) {
       if(T.charAt(i) == S.charAt(j)) {
           mem[i+1][j+1] = mem[i][j] + mem[i+1][j];
       } else {
           mem[i+1][j+1] = mem[i+1][j];
       }
   }
}
```

		r	a	b	b	b	i	t	
		0	1	2	3	4	5	6	7
r	0	1	1	1	1	1	1	1	1
a	1		1	1	1	1	1	1	1
b	2			1	1	1	1	1	1
b	3				1	2	3	3	
i	4								
t	5								
	6								

```java
// filling the first row: with 1s
for(int j=0; j≤S.length(); j++) {
   mem[0][j] = 1;
}

// the first column is 0 by default in every other
rows but the first, which we need.

for(int i=0; i<T.length(); i++) {
   for(int j=0; j<S.length(); j++) {
       if(T.charAt(i) == S.charAt(j)) {
           mem[i+1][j+1] = mem[i][j] + mem[i+1][j];
       } else {
           mem[i+1][j+1] = mem[i+1][j];
       }
   }
}
```

		r	a	b	b	b	i	t	
		0	1	2	3	4	5	6	7
r	0	1	1	1	1	1	1	1	1
a	1		1	1	1	1	1	1	1
b	2			1	1	1	1	1	1
b	3				1	2	3	3	
i	4								
t	5								
	6								

```java
// filling the first row: with 1s
for(int j=0; j≤S.length(); j++) {
   mem[0][j] = 1;
}

// the first column is 0 by default in every other
rows but the first, which we need.

for(int i=0; i<T.length(); i++) {
   for(int j=0; j<S.length(); j++) {
       if(T.charAt(i) == S.charAt(j)) {
           mem[i+1][j+1] = mem[i][j] + mem[i+1][j];
       } else {
           mem[i+1][j+1] = mem[i+1][j];
       }
   }
}
```

		r	a	b	b	b	i	t	
		0	1	2	3	4	5	6	7
r	0	1	1	1	1	1	1	1	1
a	1		1	1	1	1	1	1	1
b	2			1	1	1	1	1	1
b	3				1	2	3	3	3
i	4								
t	5								
	6								

```java
// filling the first row: with 1s
for(int j=0; j≤S.length(); j++) {
   mem[0][j] = 1;
}

// the first column is 0 by default in every other
rows but the first, which we need.

for(int i=0; i<T.length(); i++) {
   for(int j=0; j<S.length(); j++) {
       if(T.charAt(i) == S.charAt(j)) {
           mem[i+1][j+1] = mem[i][j] + mem[i+1][j];
       } else {
           mem[i+1][j+1] = mem[i+1][j];
       }
   }
}
```

Distinct Subsequence 115

$i = 4$

		r	a	b	b	b	i	t	
		0	1	2	3	4	5	6	7
r	0	1	1	1	1	1	1	1	1
a	1		1	1	1	1	1	1	1
b	2			1	1	1	1	1	1
b	3				1	2	3	3	3
i	4					1	3	3	
t	5								
	6								

```java
// filling the first row: with 1s
for(int j=0; j≤S.length(); j++) {
   mem[0][j] = 1;
}

// the first column is 0 by default in every other
rows but the first, which we need.

for(int i=0; i<T.length(); i++) {
   for(int j=0; j<S.length(); j++) {
       if(T.charAt(i) == S.charAt(j)) {
           mem[i+1][j+1] = mem[i][j] + mem[i+1][j];
       } else {
           mem[i+1][j+1] = mem[i+1][j];
       }
   }
}
```

		r	a	b	b	b	i	t	
		0	1	2	3	4	5	6	7
r	0	1	1	1	1	1	1	1	1
a	1		1	1	1	1	1	1	1
b	2			1	1	1	1	1	1
b	3				1	2	3	3	3
i	4					1	3	3	
t	5								
	6								

```java
// filling the first row: with 1s
for(int j=0; j≤S.length(); j++) {
   mem[0][j] = 1;
}

// the first column is 0 by default in every other
rows but the first, which we need.

for(int i=0; i<T.length(); i++) {
   for(int j=0; j<S.length(); j++) {
       if(T.charAt(i) == S.charAt(j)) {
           mem[i+1][j+1] = mem[i][j] + mem[i+1][j];
       } else {
           mem[i+1][j+1] = mem[i+1][j];
       }
   }
}
```

		r	a	b	b	b	i	t	
		0	1	2	3	4	5	6	7
r	0	1	1	1	1	1	1	1	1
a	1		1	1	1	1	1	1	1
b	2			1	1	1	1	1	1
b	3				1	2	3	3	3
i	4					1	3	3	3
t	5								
	6								

```java
// filling the first row: with 1s
for(int j=0; j≤S.length(); j++) {
   mem[0][j] = 1;
}

// the first column is 0 by default in every other
rows but the first, which we need.

for(int i=0; i<T.length(); i++) {
   for(int j=0; j<S.length(); j++) {
       if(T.charAt(i) == S.charAt(j)) {
           mem[i+1][j+1] = mem[i][j] + mem[i+1][j];
       } else {
           mem[i+1][j+1] = mem[i+1][j];
       }
   }
}
```

Distinct Subsequence 115

$i = 3$

		r	a	b	b	b	i	t	
		0	1	2	3	4	5	6	7
r	0	1	1	1	1	1	1	1	1
a	1		1	1	1	1	1	1	1
b	2			1	1	1	1	1	1
b	3				1	2	3	3	3
i	4					1	3	3	3
t	5							3	3
	6								

```java
// filling the first row: with 1s
for(int j=0; j≤S.length(); j++) {
   mem[0][j] = 1;
}

// the first column is 0 by default in every other
rows but the first, which we need.

for(int i=0; i<T.length(); i++) {
   for(int j=0; j<S.length(); j++) {
       if(T.charAt(i) == S.charAt(j)) {
           mem[i+1][j+1] = mem[i][j] + mem[i+1][j];
       } else {
           mem[i+1][j+1] = mem[i+1][j];
       }
   }
}
```

		r	a	b	b	b	i	t	
		0	1	2	3	4	5	6	7
r	0	1	1	1	1	1	1	1	1
a	1		1	1	1	1	1	1	1
b	2			1	1	1	1	1	1
b	3				1	2	3	3	3
i	4					1	3	3	3
t	5							3	3
	6								

```java
// filling the first row: with 1s
for(int j=0; j≤S.length(); j++) {
   mem[0][j] = 1;
}

// the first column is 0 by default in every other
rows but the first, which we need.

for(int i=0; i<T.length(); i++) {
   for(int j=0; j<S.length(); j++) {
       if(T.charAt(i) == S.charAt(j)) {
           mem[i+1][j+1] = mem[i][j] + mem[i+1][j];
       } else {
           mem[i+1][j+1] = mem[i+1][j];
       }
   }
}
```

Distinct Subsequence 115

`i`=2

		r	a	b	b	b	i	t	
		0	1	2	3	4	5	6	7
r	0	1	1	1	1	1	1	1	1
a	1		1	1	1	1	1	1	1
b	2			1	1	1	1	1	1
b	3				1	2	3	3	3
i	4					1	3	3	3
t	5							3	3
	6								3

```java
// filling the first row: with 1s
for(int j=0; j≤S.length(); j++) {
   mem[0][j] = 1;
}

// the first column is 0 by default in every other
rows but the first, which we need.

for(int i=0; i<T.length(); i++) {
   for(int j=0; j<S.length(); j++) {
       if(T.charAt(i) == S.charAt(j)) {
           mem[i+1][j+1] = mem[i][j] + mem[i+1][j];
       } else {
           mem[i+1][j+1] = mem[i+1][j];
       }
   }
}
```

Edit Distance 72

		e	x	e	c	u	t	i	o	n	
		0	1	2	3	4	5	6	7	8	9
i	0	0	1	2	3	4	5	6	7	8	9
n	1	1	1								
t	2	2									
e	3	3									
n	4	4									
t	5	5									
i	6	6									
o	7	7									
n	8	8									
	9	9									

```java
for(int i = 0; i < m; i++) {
    for(int j = 0; j < n; j++) {
        if(word1.charAt(i) == word2.charAt(j))
            cost[i + 1][j + 1] = cost[i][j];
        else {
            int replace = cost[i][j]
            int delete = cost[i][j + 1];
            int insert = cost[i + 1][j];
            cost[i + 1][j + 1] =
Math.min(replace, Math.min(delete, insert)) +
1;
            cost[i + 1][j + 1]++;
        }
    }
```

		e	x	e	c	u	t	i	o	n	
		0	1	2	3	4	5	6	7	8	9
i	0	0	1	2	3	4	5	6	7	8	9
n	1	1	2								
t	2	2									
e	3	3									
n	4	4									
t	5	5									
i	6	6									
o	7	7									
n	8	8									
	9	9									

```java
for(int i = 0; i < m; i++) {
    for(int j = 0; j < n; j++) {
        if(word1.charAt(i) == word2.charAt(j))
            cost[i + 1][j + 1] = cost[i][j];
        else {
            int replace = cost[i][j]
            int delete = cost[i][j + 1];
            int insert = cost[i + 1][j];
            cost[i + 1][j + 1] =
Math.min(replace, Math.min(delete, insert)) +
1;
            cost[i + 1][j + 1]++;
        }
    }
```

		e	x	e	c	u	t	i	o	n	
		0	1	2	3	4	5	6	7	8	9
i	0	0	1	2	3	4	5	6	7	8	9
n	1	1	2	2							
t	2	2									
e	3	3									
n	4	4									
t	5	5									
i	6	6									
o	7	7									
n	8	8									
	9	9									

```java
for(int i = 0; i < m; i++) {
    for(int j = 0; j < n; j++) {
        if(word1.charAt(i) == word2.charAt(j))
            cost[i + 1][j + 1] = cost[i][j];
        else {
            int replace = cost[i][j]
            int delete = cost[i][j + 1];
            int insert = cost[i + 1][j];
            cost[i + 1][j + 1] =
Math.min(replace, Math.min(delete, insert)) +
1;
            cost[i + 1][j + 1]++;
        }
    }
```

		e	x	e	c	u	t	i	o	n	
		0	1	2	3	4	5	6	7	8	9
i	0	0	1	2	3	4	5	6	7	8	9
n	1	1	2	3							
t	2	2									
e	3	3									
n	4	4									
t	5	5									
i	6	6									
o	7	7									
n	8	8									
	9	9									

```java
for(int i = 0; i < m; i++) {
    for(int j = 0; j < n; j++) {
        if(word1.charAt(i) == word2.charAt(j))
            cost[i + 1][j + 1] = cost[i][j];
        else {
            int replace = cost[i][j]
            int delete = cost[i][j + 1];
            int insert = cost[i + 1][j];
            cost[i + 1][j + 1] =
Math.min(replace, Math.min(delete, insert)) +
1;
            cost[i + 1][j + 1]++;
        }
    }
```

		e	x	e	c	u	t	i	o	n	
		0	1	2	3	4	5	6	7	8	9
i	0	0	1	2	3	4	5	6	7	8	9
n	1	1	2	3	3						
t	2	2									
e	3	3									
n	4	4									
t	5	5									
i	6	6									
o	7	7									
n	8	8									
	9	9									

```java
for(int i = 0; i < m; i++) {
    for(int j = 0; j < n; j++) {
        if(word1.charAt(i) == word2.charAt(j))
            cost[i + 1][j + 1] = cost[i][j];
        else {
            int replace = cost[i][j]
            int delete = cost[i][j + 1];
            int insert = cost[i + 1][j];
            cost[i + 1][j + 1] =
Math.min(replace, Math.min(delete, insert)) +
1;
            cost[i + 1][j + 1]++;
        }
    }
```

		e	x	e	c	u	t	i	o	n	
		0	1	2	3	4	5	6	7	8	9
i	0	0	1	2	3	4	5	6	7	8	9
n	1	1	2	3	4						
t	2	2									
e	3	3									
n	4	4									
t	5	5									
i	6	6									
o	7	7									
n	8	8									
	9	9									

```java
for(int i = 0; i < m; i++) {
    for(int j = 0; j < n; j++) {
        if(word1.charAt(i) == word2.charAt(j))
            cost[i + 1][j + 1] = cost[i][j];
        else {
            int replace = cost[i][j]
            int delete = cost[i][j + 1];
            int insert = cost[i + 1][j];
            cost[i + 1][j + 1] =
Math.min(replace, Math.min(delete, insert)) +
1;
            cost[i + 1][j + 1]++;
        }
    }
```

		e	x	e	c	u	t	i	o	n	
		0	1	2	3	4	5	6	7	8	9
i	0	0	1	2	3	4	5	6	7	8	9
n	1	1	2	3	4						
t	2	2									
e	3	3									
n	4	4									
t	5	5									
i	6	6									
o	7	7									
n	8	8									
	9	9									

```java
for(int i = 0; i < m; i++) {
    for(int j = 0; j < n; j++) {
        if(word1.charAt(i) == word2.charAt(j))
            cost[i + 1][j + 1] = cost[i][j];
        else {
            int replace = cost[i][j]
            int delete = cost[i][j + 1];
            int insert = cost[i + 1][j];
            cost[i + 1][j + 1] =
Math.min(replace, Math.min(delete, insert)) +
1;
            cost[i + 1][j + 1]++;
        }
    }
```

Distinct Subsequence 115

`i`=2

```cpp
// i - indexing string s1 // j - indexing string s2
for (int i = 1; i ≤ n; ++i) {
    for (int j = 1; j ≤ m; ++j) {
        if (s1[i-1] == s2[j-1]) {
            dp[i][j] = /*code*/;
        } else {
            dp[i][j] = /*code*/;
        }
    }
}
```

```cpp
for (int l = 1; l < n; ++l) {
    for (int i = 0; i < n-l; ++i) {
        int j = i + l;
        if (s[i] == s[j]) {
            dp[i][j] = /*code*/;
        } else {
            dp[i][j] = /*code*/;
        }
    }
}
```

4. DP on Strings

```
for (int i = 1; i ≤ n; ++i) {
    for (int j = 1; j ≤ m; ++j) {
        if (text1[i-1] == text2[j-1]) {
            dp[i][j] = dp[i-1][j-1] + 1;
        } else {
            dp[i][j] = max(dp[i-1][j], dp[i][j-1]);
        }
    }
}
```

1143. Longest Common Subsequence

```
for (int l = 1; l < n; ++l) {
    for (int i = 0; i < n-l; ++i) {
        int j = i + l;
        if (s[i] == s[j] && dp[i+1][j-1] == j-i-1) {
            dp[i][j] = dp[i+1][j-1] + 2;
        } else {
            dp[i][j] = 0;
        }
    }
}
```

647. Palindromic Substrings

4. DP on Strings

```
for (int i = 1; i ≤ n; ++i) {
    for (int j = 1; j ≤ m; ++j) {
        if (text1[i-1] == text2[j-1]) {
            dp[i][j] = dp[i-1][j-1] + 1;
        } else {
            dp[i][j] = max(dp[i-1][j], dp[i][j-1]);
        }
    }
}
```

1143. Longest Common Subsequence

```
for (int l = 1; l < n; ++l) {
    for (int i = 0; i < n-l; ++i) {
        int j = i + l;
        if (s[i] == s[j] && dp[i+1][j-1] == j-i-1) {
            dp[i][j] = dp[i+1][j-1] + 2;
        } else {
            dp[i][j] = 0;
        }
    }
}
```

4. DP on Strings

647. Palindromic Substrings

```
// i - indexing a set of values   // j - options to ignore j values
for (int i = 1; i < n; ++i) {
    for (int j = 1; j <= k; ++j) {
      dp[i][j] = max({dp[i][j], dp[i-1][j] + arr[i], dp[i-1][j-1]});
       dp[i][j-1] = max({dp[i][j-1], dp[i-1][j-1] + arr[i], arr[i]});
    }
 }
```

```
// i - indexing a set of values // j - options to ignore j values
for (int i = 1; i < n; ++i) {
 for (int j = 1; j <= k; ++j) {
   dp[i][j] = max({dp[i][j], dp[i-1][j] + arr[i], dp[i-1][j-1]});
   dp[i][j-1] = max({dp[i][j-1], dp[i-1][j-1] + arr[i], arr[i]});
 }
}
```

5. Decision Making

Robber, Stock, Cooldown, Transaction Fee

O(n^2)

```
// i - indexing a set of values
// j - options to ignore j values
for (int i = 1; i < n; ++i) {
    for (int j = 1; j ≤ k; ++j) {
        dp[i][j] = max({dp[i][j], dp[i-1][j] + arr[i],
dp[i-1][j-1]});
        dp[i][j-1] = max({dp[i][j-1], dp[i-1][j-1] +
arr[i], arr[i]});
    }
}
```

5. Decision Making

```
for (int i = 1; i < n; ++i) {
    dp[i][1] = max(dp[i-1][0] + nums[i],
dp[i-1][1]);
    dp[i][0] = dp[i-1][1];
}
```


198. House Robber

	Statement	Approach
Min Path to Reach Target routes[i] = min(routes[i-1], routes[i-2], ... , routes[i-k]) + cost[i]	Given a target find minimum (maximum) cost / path / sum to reach the target.	Choose min (maximum) path among all possible paths before the current state, then add value for the current state.
Distinct Ways routes[i] = routes[i-1] + routes[i-2], ... , + routes[i-k]	Given a target find a number of distinct ways to reach the target.	Sum all possible ways to reach the current state.
Merge Intervals dp[i][j] = dp[i][k] + result[k] + dp[k+1][j] Get the best from the left and right sides and add a solution for the current position.	considering the current number and the best you can get from the left and right sides.	Find all optimal solutions for every interval and return the best possible answer.
DP on Strings Two strings where lengths of those strings are not big	Given two strings s1 and s2, return some result.	can be accepted in O(n^2) complexity.

Decision Making
dp[i][j] = max({dp[i][j], dp[i-1][j] + arr[i], dp[i-1][j-1]});
dp[i][j-1] = max({dp[i][j-1], dp[i-1][j-1] + arr[i], arr[i]});
	Given a set of values find an answer with an option to choose or ignore the current value.	If choose the current ignore previous vice-versa, If choose the previous ignore current

```
for (int j = 0; j < ways.size(); ++j) {
  result += topDown(target - ways[j]);
}
return memo[/*state parameters*/] = result;
```

```
for (int i = 1; i <= target; ++i) {
  for (int j = 0; j < ways.size(); ++j) {
    if (ways[j] <= i) {
      dp[i] += dp[i - ways[j]];
    }
  }
}
return dp[target]
```

```
for (int k = i; k ≤ j; ++k) {
   result = max(result, topDown(nums, i, k-1) +
result[k] + topDown(nums, k+1, j));
}
return memo[/*state parameters*/] = result;
```

```
dp[i][j] = dp[i][k] + result[k] +
dp[k+1][j]
```

1. Max, Min

1. Max, Min Path Sum

Triangle, Dungeon, 4Square, Coin, Key, Refueling, Ticket

```
routes[i] = min(routes[i-1], routes[i-2], ... , routes[i-k]) + cost[i]
```

14 Max-Min

1. Minimum Path Sum
2. Triangle
3. Dungeon Game
4. Maximal Square
5. Perfect Squares
6. Coin Change
7. Ones and Zeroes
8. 2 Keys Keyboard
9. Min Cost Climbing Stairs
10. Minimum Number of Refueling Stops
11. Minimum Falling Path Sum
12. Minimum Cost For Tickets
13. Last Stone Weight II
14. Tiling a Rectangle with the Fewest Squares

DP(Max-Min) 14

Statement

> Given a target find minimum (maximum) cost / path / sum to reach the target.

Approach

> Choose minimum (maximum) path among all possible paths before the current state, then add value for the current state.

```
routes[i] = min(routes[i-1], routes[i-2], ... , routes[i-k]) + cost[i]
```

Generate optimal solutions for all values in the target and return the value for the target.

Top-Down

```
for (int j = 0; j < ways.size(); ++j) {
    result = min(result, topDown(target - ways[j]) + cost/ path / sum);
}
return memo[/*state parameters*/] = result;
```

Bottom-Up

```
for (int i = 1; i <= target; ++i) {
   for (int j = 0; j < ways.size(); ++j) {
       if (ways[j] <= i) {
           dp[i] = min(dp[i], dp[i - ways[j]] + cost / path / sum) ;
       }
   }
}

return dp[target]
```

Approach

> Choose minimum (maximum) path among all possible paths before the current state, then add value for the current state.

```
routes[i] = min(routes[i-1], routes[i-2], ... , routes[i-k]) + cost[i]
```

Generate optimal solutions for all values in the target and return the value for the target.

Top-Down

```
for (int j = 0; j < ways.size(); ++j) {
    result = min(result, topDown(target - ways[j]) + cost/ path / sum);
}
return memo[/*state parameters*/] = result;
```

Bottom-Up

```
for (int i = 1; i <= target; ++i) {
   for (int j = 0; j < ways.size(); ++j) {
       if (ways[j] <= i) {
           dp[i] = min(dp[i], dp[i - ways[j]] + cost / path / sum) ;
       }
   }
}

return dp[target]
```

```python
class Solution:
    def numSquares(self, n: int) -> int:
        dp = [n] * (n + 1)
        dp[0] = 0

        for target in range(1, n + 1):
            for s in range(1, target + 1):
                square = s * s
                if target - square < 0:
                    break
                dp[target] = min(dp[target], 1 + dp[target - square])

        return dp[n]

        O(n * n^1/2)
```

64. Minimum Path Sum

Medium 6144 90 Add to List Share

Given a `m x n` `grid` filled with non-negative numbers, find a path from top left to bottom right, which minimizes the sum of all numbers along its path.

Note: You can only move either down or right at any point in time.

Example 1:

1	3	1
1	5	1
4	2	1

```
Input: grid = [[1,3,1],[1,5,1],[4,2,1]]
Output: 7
Explanation: Because the path 1 → 3 → 1 → 1 → 1 minimizes the sum.
```

Example 2:

```
Input: grid = [[1,2,3],[4,5,6]]
Output: 12
```

Solution 1: Recursion with memoization

```
minSum(x, y)
  = grid[y][x] + min(minSum(x - 1, y),
                     minSum(x, y - 1))
```

Time complexity: O(mn)

Space complexity: O(mn)

Solution 2: DP

dp[i][j]:= min path sum to (j, i)

```
dp[i][j] = grid[i][j] + min(dp[i - 1][j],
                            dp[i][j - 1])
```

```
dp[0][0] = grid[0][0]
dp[0][j] = grid[0][j] + dp[0][j - 1]
dp[i][0] = grid[i][0] + dp[i - 1][0]
```

Time complexity: O(mn)

Space complexity: O(mn) -> O(n) -> O(1)

5. Perfect Squares

6. Coin Change

[1, 3, 4, 5]

Amount = 7

MinCoin = ~~3~~ 2

DP-Bottom-up

min value not distinct way

DP[0] = 0
DP[1] = 1
DP[2] = 2
DP[3] = 1

DP[7] = 1 + DP[6]
1+2 = 3
1 + DP[4]
1+1 = 2
1+DP[3] = 2
+DP[2] = 3

DP[0] = 0
DP[1] = 1
DP[2] = 2

```python
class Solution:
    def coinChange(self, coins: List[int], amount: int) -> int:
        dp = [amount + 1] * (amount + 1)
        dp[0] = 0

        for a in range(1, amount + 1):
            for c in coins:
                if a - c >= 0:
                    dp[a] = min(dp[a], 1 + dp[a - c])

        return dp[amount] if dp[amount] != amount + 1 else -1
```

120. Triangle

Medium 4011 360 Add to List Share

Given a `triangle` array, return *the minimum path sum from top to bottom*.

For each step, you may move to an adjacent number of the row below. More formally, if you are on index `i` on the current row, you may move to either index `i` or index `i + 1` on the next row.

Example 1:

```
Input: triangle = [[2],[3,4],[6,5,7],[4,1,8,3]]
Output: 11
Explanation: The triangle looks like:
   2
  3 4
 6 5 7
4 1 8 3
The minimum path sum from top to bottom is 2 + 3 + 5 + 1 =
11 (underlined above).
```

Example 2:

```
Input: triangle = [[-10]]
Output: -10
```

174. Dungeon Game

Hard 3520 70 Add to List Share

The demons had captured the princess and imprisoned her in **the bottom-right corner** of a `dungeon`. The `dungeon` consists of `m x n` rooms laid out in a 2D grid. Our valiant knight was initially positioned in **the top-left room** and must fight his way through `dungeon` to rescue the princess.

The knight has an initial health point represented by a positive integer. If at any point his health point drops to `0` or below, he dies immediately.

Some of the rooms are guarded by demons (represented by negative integers), so the knight loses health upon entering these rooms; other rooms are either empty (represented as 0) or contain magic orbs that increase the knight's health (represented by positive integers).

To reach the princess as quickly as possible, the knight decides to move only **rightward** or **downward** in each step.

Return *the knight's minimum initial health so that he can rescue the princess*.

Note that any room can contain threats or power-ups, even the first room the knight enters and the bottom-right room where the princess is imprisoned.

Example 1:

-2 →	-3 →	3 ↓
-5	-10	1 ↓
10	30	-5

```
Input: dungeon = [[-2,-3,3],[-5,-10,1],[10,30,-5]]
Output: 7
Explanation: The initial health of the knight must be at least 7 if he follows the
optimal path: RIGHT-> RIGHT -> DOWN -> DOWN.
```

Example 2:

```
Input: dungeon = [[0]]
Output: 1
```

```
class Solution {
public:
    int calculateMinimumHP(vector<vector<int> > &dungeon) {
        const vector<vector<int>>& d = dungeon;
        const int m = d.size();
        const int n = d[0].size();

        // hp[y][x]: min hp required to reach bottom right (P).
        vector<vector<int>> hp(m+1, vector<int>(n+1, INT_MAX));
        hp[m][n-1] = hp[m-1][n] = 1;

        for(int y=m-1;y>=0;--y)
            for(int x=n-1;x>=0;--x)
                hp[y][x] = max(1, min(hp[y+1][x], hp[y][x+1]) - d[y][x]);

        return hp[0][0];
    }
};
```

-2 (K)	-3	3
-5	-10	1
10	30	-5 (P)

7 (K)	5	2	MAX
6	11	5	MAX
1	1	6 (P)	1
MAX	MAX	1	MAX

Example 1:

```
Input: coins = [1,2,5], amount = 11
Output: 3
Explanation: 11 = 5 + 5 + 1
```

Example 2:

```
Input: coins = [2], amount = 3
Output: -1
```

Example 3:

```
Input: coins = [1], amount = 0
Output: 0
```

Example 4:

```
Input: coins = [1], amount = 1
Output: 1
```

Example 5:

```
Input: coins = [1], amount = 2
Output: 2
```

```
int coinChange(vector<int>& coins, int amount) {
    int Max = amount + 1;
    vector<int> dp(amount + 1, Max);
    dp[0] = 0;
    for (int i = 1; i <= amount; i++) {
        for (int j = 0; j < coins.size(); j++) {
            if (coins[j] <= i) {
                dp[i] = min(dp[i], dp[i - coins[j]] + 1);
            }
        }
    }
    return dp[amount] > amount ? -1 : dp[amount];
}
```

```
var coinChange = function(coins, amount) {
    let minCoins = new Array(amount + 1).fill(Infinity);
    minCoins[0] = 0;
    for(let i = 0; i < coins.length; i++){
        for(let j =  0; j < minCoins.length; j++){
            if(coins[i] <= j){
                minCoins[j] = Math.min(minCoins[j], 1 + minCoins[j-coins[i]]
            }
        }
    };
    return minCoins[amount]!= Infinity ? minCoins[amount] : -1;
```

17 Distinct Way

1. Unique Paths
2. Unique Paths II
3. Climbing Stairs
4. Combination Sum IV
5. Partition Equal Subset Sum
6. Target Sum
7. Out of Boundary Paths
8. Number of Longest Increasing Subsequence
9. Knight Probability in Chessboard
10. Domino and Tromino Tiling
11. Minimum Swaps To Make Sequences Increasing
12. Soup Servings
13. Knight Dialer
14. Number of Dice Rolls With Target Sum
15. Count Vowels Permutation
16. Dice Roll Simulation
17. Number of Ways to Stay in the Same Place After Some Steps

"To develop Midshipmen morally, mentally and physically and to imbue them with the highest ideals of duty, honor and loyalty in order to graduate leaders who are dedicated to a career of naval service and have potential for future development in mind and character to assume the highest responsibilities of command, citizenship and government."

Unique Path-II

Follow up for "Unique Paths":

Now consider if some obstacles are added to the grids. How many unique paths would there be

An obstacle and empty space is marked as `1` and `0` respectively in the grid.

For example,

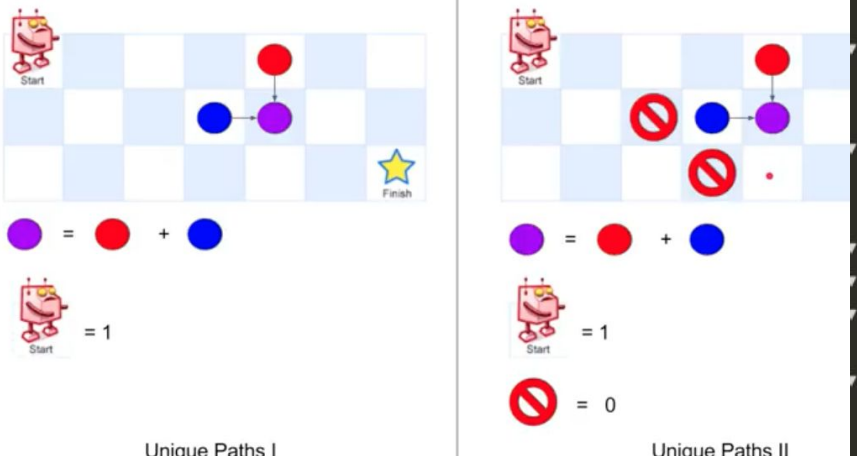
There is one obstacle in the middle of a 3x3 grid as illustrated below.

```
[
  [0,0,0],
  [0,1,0],
  [0,0,0]
]
```

The total number of unique paths is `2`.

Note: *m* and *n* will be at most 100.

LeetCode 63. Unique Paths II



Unique Paths I

Unique Paths II

```
public:
    int uniquePathsWithObstacles(vector<vector<int>>& obstacleGrid) {
        int n = obstacleGrid.size();
        if (n == 0) return 0;
        int m = obstacleGrid[0].size();

        // f[i][j] = paths(i, j)
        // INT_MIN -> not solved yet, solution unknown
        f_ = vector<vector<int>>(n + 1, vector<int>(m + 1, INT_MIN));
        return paths(m, n, obstacleGrid);
    }
private:
    vector<vector<int>> f_;

    int paths(int x, int y, const vector<vector<int>>& o) {
        // Out of bound, return 0.
        if (x <= 0 || y <= 0) return 0;

        // Reaching the starting point.
        // Note, there might be an obstacle here as well.
        if (x == 1 && y == 1) return 1 - o[0][0];

        // Already solved, return the answer.
        if (f_[y][x] != INT_MIN) return f_[y][x];

        // There is an obstacle on current block, no path
        if (o[y - 1][x - 1] == 1) {
            f_[y][x] = 0;
        } else {
            // Recursively find paths.
            f_[y][x] = paths(x - 1, y, o) + paths(x, y - 1, o);
        }
```

You are climbing a stair case. It takes *n* steps to reach to the top.

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

Note: Given *n* will be a positive integer.

```
Solution {
:
t climbStairs(int n) {
  f_ = vector<int>(n+1, 0);
  return numOfSolutions(n);

e:
t numOfSolutions(int n) {
  if(n<=1) return 1;
  return numOfSolutions(n-1) + numOfSolutions(n-2);

ctor<int> f_;
```

overtime

```
class Solution {
public:
    int climbStairs(int n) {
        // f[i] = climbStairs(i)
        vector<int> f(n+1, 0);

        f[0] = 1;
        f[1] = 1;

        // f[i] = f[i-1] + f[i-2]
        for(int i=2;i<=n;++i)
            f[i] = f[i-1] + f[i-2];

        return f[n];
    }
};
```

```
class Solution {
public:
    int climbStairs(int n) {
        int two=1;
        int one=1;
        int curr=1;
        // curr = two + one
        for(int i=2;i<=n;++i) {
            curr = two + one;
            two = one;
            one = curr;
        }

        return curr;
    }
};
```

```
class Solution {
public:
    int climbStairs(int n) {
        f_ = vector<int>(n+1, 0);
        return numOfSolutions(n);
    }
private:
    int numOfSolutions(int n) {
        if(n<=1) return 1;
        if(f_[n]>0) return f_[n];
        f_[n] = numOfSolutions(n-1) + numOfSolutions(n-2);
        return f_[n];
    }
    vector<int> f_;
};
```

memorized recursive

Combination Sum IV

```
nums = [1, 2, 3]
target = 4

The possible combination ways are
(1, 1, 1, 1)
(1, 1, 2)
(1, 2, 1)
(1, 3)
(2, 1, 1)
(2, 2)
(3, 1)

Note that different sequences are

Therefore the output is 7.
```

```
Solution: Recursion with Memoization
dp(nums, target): => all combinations that uses nums to sumup to target
  if target == 0: return {}
  return Union{{num_i} x dp(nums, target - num_i)} (num_i <= target)
Time complexity: O(sum{target/num_i}), Space complexity: O(target)
```

[1] 10

```
dp([1,2,3], 4) #
 = U({1} x dp([1,2,3], 4-1),  # {1,1,1,1}, {1,1,2},...
     {2} x dp([1,2,3], 4-2),  # {2,1,1}, {2,2}
     {3} x dp([1,2,3], 4-3))  # {3,1}
```

```
dp([1,2,3], 3)
 = U({1} x dp([1, 2, 3], 3-1), # {1,1,1}, {1,2}
     {2} x dp([1, 2, 3], 3-2), # {2,1}
     {3} x dp([1, 2, 3], 3-3)) # {3}
```

```
dp([1,2,3], 2)
 = U({1} x dp([1, 2, 3], 2-1), # {1,1}
     {2} x dp([1, 2, 3], 2-2)) # {2}
```

```
dp([1,2,3], 1) # {1}
 = U({1} x dp([1, 2, 3], 1-1)) # {1}
```

```
dp([1,2,3], 0) = {}
```

```
Convert to dp that only count the sols.
Time complexity: O(target*|nums|)
dp[0] = 1 # {}

for i from 1 to target:
  dp[i] = sum(dp[i - num_j])

return dp[target]
```

```
class Solution {
public:
    int combinationSum4(vector<int>& nums, int target) {
        m_ = vector<int>(target + 1, -1);
        m_[0] = 1;
        return dp(nums, target);
    }
private:
    int dp(const vector<int>& nums, int target) {
        if (target < 0) return 0;
        if (m_[target] != -1) return m_[target];
        int ans = 0;
        for (const int num : nums)
            ans += dp(nums, target - num);
        return m_[target] = ans;
    }
    vector<in
};
```

```
int combinationSum4(vector<int>& nums, int target) {
    vector<int> dp(target + 1, 0);
    dp[0] = 1;
    for (int i = 1; i <= target; ++i)
        for (const int num : nums)
            if (i - num >=0)
                dp[i] += dp[i - num];
    return dp[target];
}
```

初始化默认无解

子问题

Given a **non-empty** array containing **only positive integers**, find if the array can be partitioned into two subsets such that the sum of elements in both subsets is equal.

Note:

1. Each of the array element will not exceed 100.
2. The array size will not exceed 200.

Example 1:

```
Input: [1, 5, 11, 5]

Output: true

Explanation: The array can be partitioned as [1, 5, 5] and [11].
```

Example 2:

```
Input: [1, 2, 3, 5]

Output: false

Explanation: The array cannot be partitioned into equal sum subsets.
```

Solution 1: Brute force $O(2^n)$ TLE

Solution2: DP

dp[i][j]: whether we can sum to j using first i numbers

dp[i][j] = true if dp[i-1][j - num]

check dp[n-1][sum / 2]

init dp[-1][0] = true

Time complexity: O(n*sum)
Space complexity: O(n*sum) -> O(sum)

num	dp[i-1] == true	dp[i] == true
1	{0}	{0, 1}
5	{0,1}	{0, 5, 6}
11	{0, 5, 6}	{0,5,6,11,16,17}
5	{0,5,6,11,16,17}	{0,5,6,10,11,16,21,22}

Odd never possible
TLE

花花酱 LeetCode
www.mytechroad.com/blog/

bool canPartition(vector<int>& nums) {

11, 5]

Target = 11

Set = {0, 5, 11, 16, 10, 21}

Given a **non-empty** array containing **only positive integers**, find if the array can be partitioned into two subsets such that the sum of elements in both subsets is equal.

Note:

1. Each of the array element will not exceed 100.
2. The array size will not exceed 200.

Example 1:

```
Input: [1, 5, 11, 5]

Output: true

Explanation: The array can be partitioned as [1, 5, 5] and [11].
```

Example 2:

```
Input: [1, 2, 3, 5]

Output: false
```

Given a **non-empty** array containing **only positive integers**, find if the array can be partitioned into two subsets such that the sum of elements in both subsets is equal.

Note:

1. Each of the array element will not exceed 100.
2. The array size will not exceed 200.

Example 1:

```
Input: [1, 5, 11, 5]

Output: true

Explanation: The array can be partitioned as [1, 5, 5] and [11].
```

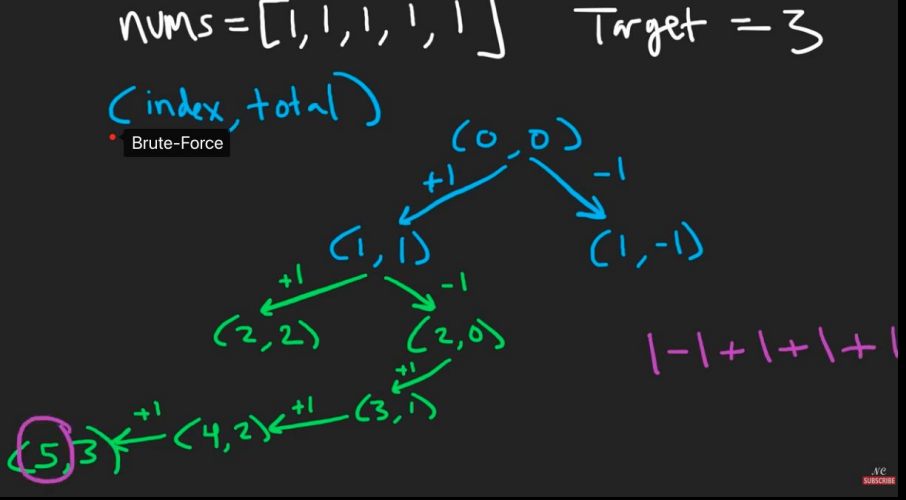
Example 2:

```
Input: [1, 2, 3, 5]

Output: false

Explanation: The array cannot be partitioned into equal sum subsets.
```

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

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

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

```python
class Solut
    def canPartition(self, nums: List[int]) -> bool:
        if sum(nums) % 2:
            return False

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

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

```cpp
ol canPartition(vector<int>& nums) {
  const int sum = std::accumulate(nums.begin(), r
  if (sum % 2 != 0) return false;
  vector<int> dp(sum + 1, 0);
  dp[0] = 1;
  for (const int num : nums) {
      for (int i = sum; i >= 0; --i)
          if (dp[i]) dp[i + num] = 1;
      if (dp[sum / 2]) return true;
  }
  return false;
```




```python
def findTargetSumWays(self, nums: List[int], target: int) -> int:
    dp = {} # (index, total) -> # of ways

    def backtrack(i, total):
        if i == len(nums):
            return 1 if total == target else 0
        if (i, total) in dp:
            return dp[(i, total)]

        dp[(i, total)] = (backtrack(i + 1, total + nums[i]) +
                          backtrack(i + 1, total - nums[i]))
        return dp[(i, total)]
    return backtrack(0, 0)
```

Observation:
Number of paths start from (i, j) to out of boundary ⇔
Number of paths start from out of boundary to (i, j)

Related Problems:

- 62. Unique Paths
- 63. Unique Paths II
- 688. Knight Probability in Chessboard

Solution: DP

dp[N][i][j]:= Number of paths start from out of boundary to (i, j) by moving N steps

dp[*][y][x] = 1, if (x, y) are out of boundary

```
dp[s][i][j] =   dp[s-1][i + 1][j]
              + dp[s-1][i - 1][j]
              + dp[s-1][i][j + 1]
              + dp[s-1][i][j - 1]
```

Ans: dp[N][i][j]

Time complexity: O(N*m*n)

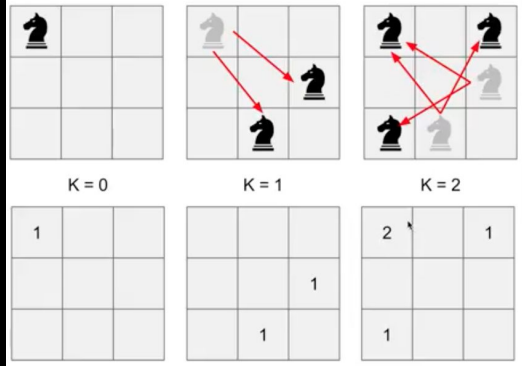
Space complexity: O(N*m*n) -> O(m*n)

High Performance Data Grids

```cpp
class Solution {
public:
  int findPaths(int m, int n, int N, int i, int j) {
    constexpr int kMod = 1000000007;
    vector<vector<vector<int>>> dp(N + 1, vector<vector<int>>(m, vector<int>(n,
0)));
    vector<int> dirs{-1, 0, 1, 0, -1};
    for (int s = 1; s <= N; ++s)
      for (int y = 0; y < m; ++y)
        for (int x = 0; x < n; ++x)
          for (int d = 0; d < 4; ++d) {
            int tx = x + dirs[d];
            int ty = y + dirs[d + 1];
            if (tx < 0 || ty < 0 || tx >= n || ty >= m)
              dp[s][y][x] += 1;
            else
              dp[s][y][x] = (dp[s][y][x] + dp[s - 1][ty][tx]) % kMod;
          }
    return dp[N][i][j];
  }
};
```

N=3,K=2,r=0,c=0

(2+1+1) / (8^2) = 0.0625



Solution: DP

Count ways to move to (j, i) after K moves.

dp[k][i][j] := number of ways to move to (j, i) after k steps

dp[k][i][j] = sum(dp[k-1][y][x]), for all (x,y) can move to (j, i)

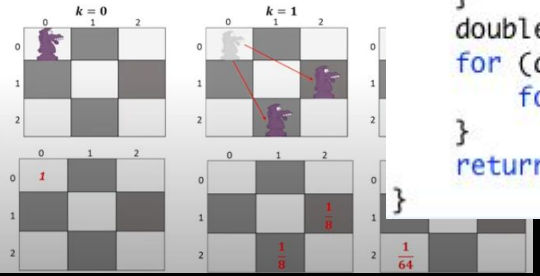
ans = sum(dp[k][i][j]) / (8^K)

Time complexity: O(k*N^2)
Space complexity: O(N^3) -> O(N^2)

Knight Probability on a Chessboard

Solution Algorithm:

1. Create 2 matrices A_1, A_2 of size $n \times n$
2. Initialize $A_1[r][c] = 1$ and the rest of the entries (in both matrices) with zeros.
3. Preform k times:
   3.1 Update A_2 entries (according the recursive function)
   which can be reached by a knight's move for the entries modified in A_1
   3.2 Switch between A_1 & A_2
4. Summarize all the entries in A_1 and return the result.



Runtime Complexity Analysis:

Our algorithm will run in $O(n^2 \cdot k)$

This is because we are executing k times $O(1)$ computations on $O(n^2)$ entries.

Space Complexity Analysis:

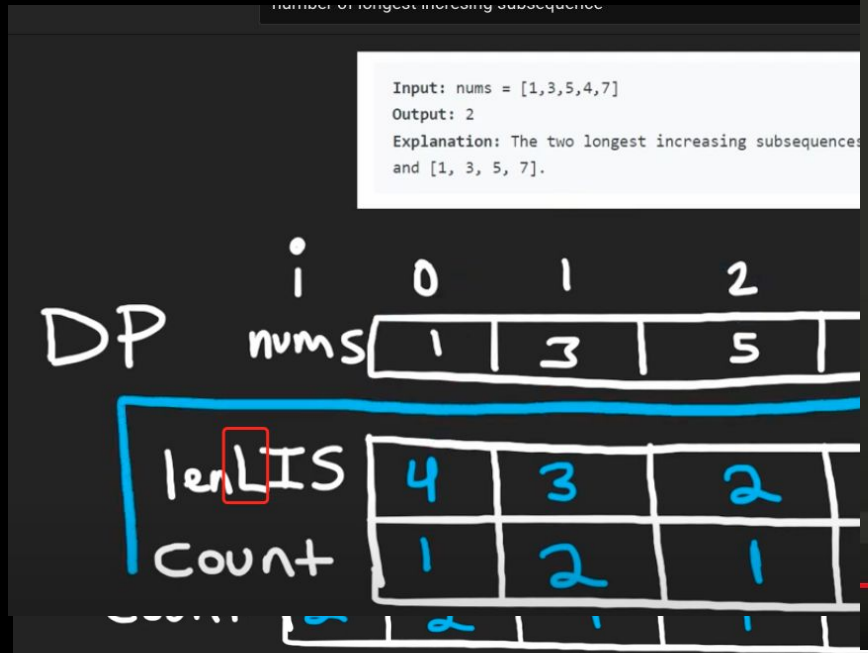
$O(n^2)$ - A constant number of matrices of size $n \times n$.

```java
public double knightProbability(int N, int K, int sr, int sc) {
    double[][] dp = new double[N][N];
    int[] dr = new int[]{2, 2, 1, 1, -1, -1, -2, -2};
    int[] dc = new int[]{1, -1, 2, -2, 2, -2, 1, -1};

    dp[sr][sc] = 1;
    for (; K > 0; K--) {
        double[][] dp2 = new double[N][N];
        for (int r = 0; r < N; r++) {
            for (int c = 0; c < N; c++) {
                for (int k = 0; k < 8; k++) {
                    int cr = r + dr[k];
                    int cc = c + dc[k];
                    if (0 <= cr && cr < N && 0 <= cc && cc < N)
                        dp2[cr][cc] += dp[r][c] / 8.0;
                    }
                }
            }
        }
        dp = dp2;
    }
    double ans = 0.0;
    for (double[] row: dp) {
        for (double x: row) ans += x;
    }
    return ans;
}
```

Input: nums = [1,3,5,4,7]

Output: 2

Explanation: The two longest increasing subsequences are [1, 3, 4, 7] and [1, 3, 5, 7].

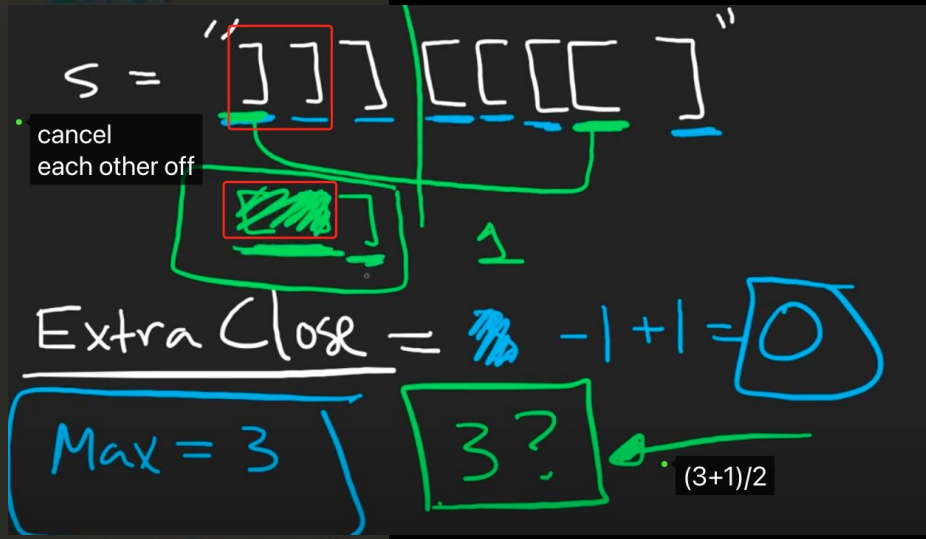



```python
def findNumberOfLIS(self, nums: List[int]) -> int:
    # Dynamic Programming

    dp = {} # key = index, value = [length of LIS, count]
    lenLIS, res = 0, 0 # length of LIS, count of LIS

    # i = start of subseq
    for i in range(len(nums) - 1, -1, -1):
        maxLen, maxCnt = 1, 1 # len, cnt of LIS start from i

        for j in range(i + 1, len(nums)):
            if nums[j] > nums[i]: # make sure increasing order
                length, count = dp[j] # len, cnt of LIS start from j
                if length + 1 > maxLen:
                    maxLen, maxCnt = length + 1, count
                elif length + 1 == maxLen:
                    maxCnt += count

        if maxLen > lenLIS:
            lenLIS, res = maxLen, maxCnt
        elif maxLen == lenLIS:
            res += maxCnt

        dp[i] = [maxLen, maxCnt]

    return res
```

10. Domino and Tromino Tiling

11. Minimum Swaps To Make Sequences Increasing

```
s Solution:
def minSwaps(self, s: str) -> int:
    close, maxClose = 0, 0

    for c in s:
        if c == "[":
            close -= 1
        else:
            close += 1
        maxClose = max(close, maxClose)

    return (maxClose + 1) // 2
```



12. Soup Servings

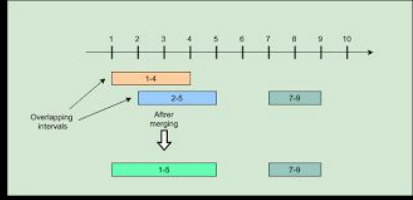
13. Knight Dialer

14. Number of Dice Rolls With Target Sum

15. Count Vowels Permutation

16. Dice Roll Simulation

Number of Ways to Stay in the Same Place After Some Steps



DP (Merge Intervals)

7

7 Intervals

Unique Binary Search Trees

Burst Balloons

Guess Number Higher or Lower II

Remove Boxes

Minimum Cost to Merge Stones

Minimum Score Triangulation of Polygon

Minimum Cost Tree From Leaf Values

1. Unique Binary Search Trees

```python
def numTrees(self, n: int) -> int:
    # numTree[4] = numTree[0] * numTree[3] +
    #              numTree[1] * numTree[2] +
    #              numTree[2] * numTree[1] +
    #              numTree[3] * numTree[1]
    numTree = [1] * (n + 1)

    # 0 nodes = 1 tree
    # 1 nodes = 1 tree
    for nodes in range(2, n + 1):
        total = 0
        for root in range(1, nodes + 1):
            left = root - 1
            right = nodes - root
            total += numTree[left] * numTree[right]
        numTree[nodes] = total
    return numTree[n]
```

2. Burst Balloons

312. Burst Balloons

Hard 3572 108 Add to List Share

DP – $O(n^3)$

$O(n^2)$

You are given n balloons, indexed from 0 to n - 1. Each balloon is pai
with a number on it represented by an array nums. You are asked to burst

Change to pop which value last

312. Burst Balloons

Hard 3572 108 Add to List Share

You are given n balloons, indexed from 0 to n - 1. Each balloon is painted with a number on it represented by an array nums. You are asked to burst all the balloons.

If you burst the ith balloon, you will get nums[i - 1] * nums[i] * nums[i + 1] coins. If i - 1 or i + 1 goes out of bounds of the array, then treat it as if there is a balloon with a 1 painted on it.

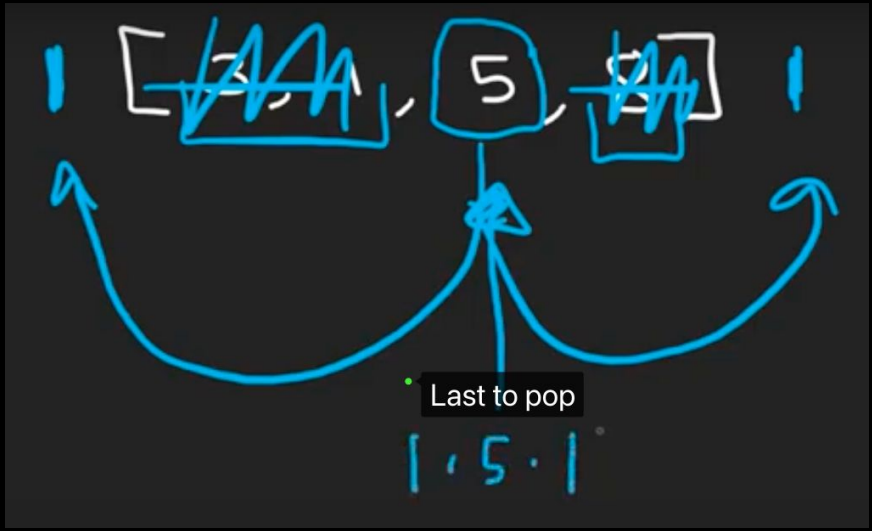
Return *the maximum coins you can collect by bursting the balloons wisely.*

n^n brute force

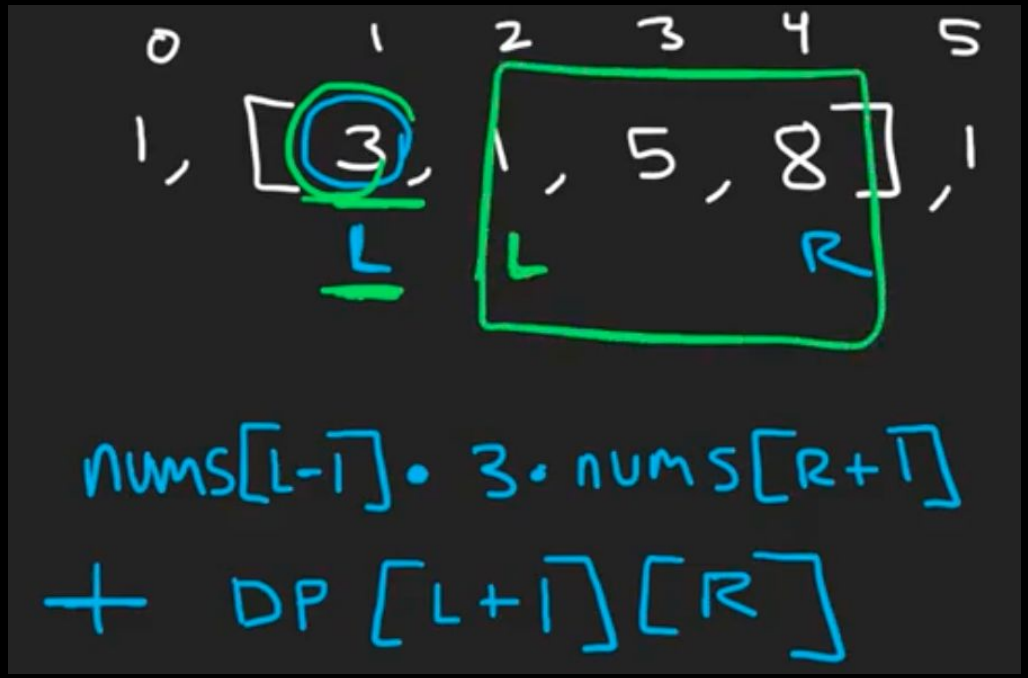
3 1 5 8

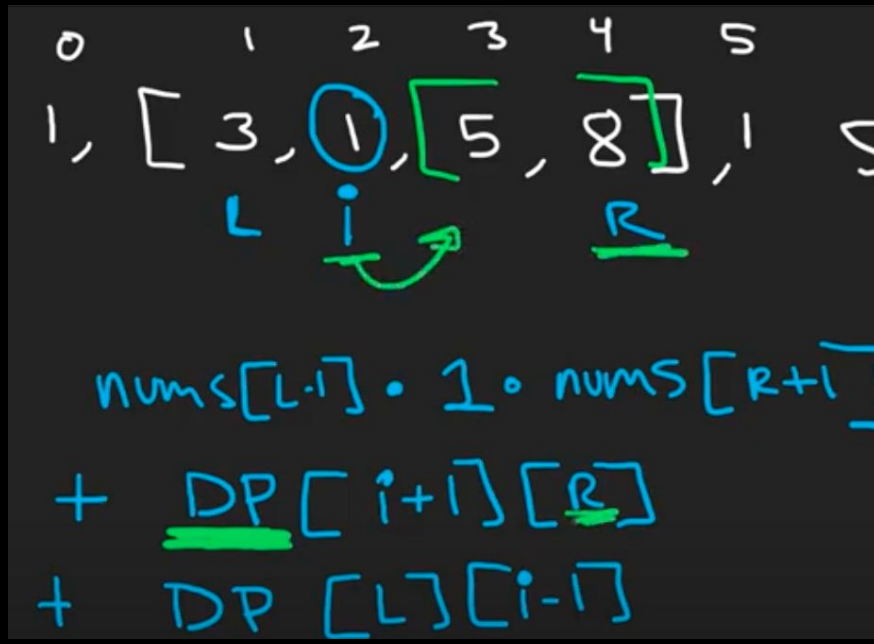
Example 1:

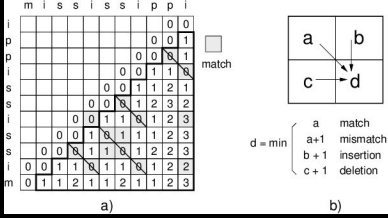
```
Input: nums = [3,1,5,8]
Output: 167
Explanation:
nums = [3,1,5,8] --> [3,5,8] --> [3,8] --> [8] --> []
coins =  3*1*5    +   3*5*8   +  1*3*8  + 1*8*1 = 167
```



2. Burst Balloons






m i s s i s s i p p i
i
p
p
i
s
s
i
s
s
i
m
match
a
b
c
d
d = min
a match
a+1 mismatch
b + 1 insertion
c + 1 deletion
a)
b)

DP (Strings)

8 Strings

- Longest Palindromic Substring
- Edit Distance
- Distinct Subsequences
- Longest Palindromic Subsequence
- Palindromic Substrings
- Minimum ASCII Delete Sum for Two Strings
- Shortest Common Supersequence
- Longest Common Subsequence

1. Longest Palindromic Substring

5. Longest Palindromic Substring

Medium 7384 555 Add to List Share

Given a string **s**, find the longest palindromic substring in **s**. You may assume that the maximum length of **s** is 1000.

Example 1:

```
Input: "babad"
Output: "bab"
Note: "aba" is also a valid answer.
```

Example 2:

```
Input: "cbbd"
Output: "bb"
```

b a b a d

b a b a d

$n \cdot n \rightarrow O(n^2)$

```python
class Solution:
    def longestPalindrome(self, s: str) -> str:
        res = ""
        resLen = 0

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

            # even length
            l, r = i, i + 1
            while l >= 0 and r < len(s) and s[l] == s[r]:
                if (r - l + 1) > resLen:
                    res = s[l:r+1]
                    resLen = r - l + 1
                l -= 1
                r += 1

        return res
```

2. Edit Distance

72. Edit Distance

Hard 5539 66 Add to List Share

Given two strings `word1` and `word2`, return *the minimum number of operations required to convert* `word1` *to* `word2`.

You have the following three operations permitted on a word:

- Insert a character
- Delete a character
- Replace a character

Example 1:

```
Input: word1 = "horse", word2 = "ros"
Output: 3
Explanation:
horse -> rorse (replace 'h' with 'r')
rorse -> rose (remove 'r')
rose -> ros (remove 'e')
```

Read the problem

DP (Decision-Making)

6

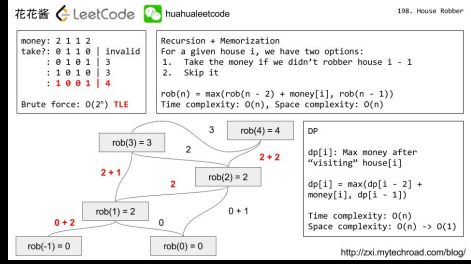
6. Decision-Making

Best Time to Buy and Sell Stock

Best Time to Buy and Sell Stock III

Best Time to Buy and Sell Stock IV

House Robber

Best Time to Buy and Sell Stock with Cooldown

Best Time to Buy and Sell Stock with Transaction Fee