

# Visualization of the solution of

## 516. Longest Palindromic Subsequence

Medium

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

<https://leetcode.com/problems/longest-palindromic-subsequence/>

Slides credit to @BryanBo-Cao

Solution author @花花酱

<http://zxi.mytechroad.com/blog/dynamic-programming/leetcode-516-longest-palindromic-subsequence/>

Solution 1: DP

$dp[i][j]$  := sol of  $s[i..j]$

Stage: length of the substring

```
for len = 1 to n:
  for i = 0 to n - len:
    j = i + len - 1
    if s[i] == s[j]:
      dp[i][j] = dp[i + 1][j - 1] + 2
    else:
      dp[i][j] = max(dp[i + 1][j],
                     dp[i][j - 1])
```

Ans:  $dp[0][n-1]$

Time complexity:  $O(n^2)$

Space complexity:  $O(n^2)$  86 ms

->  $O(n)$  28 ms

Base case:

$a \rightarrow dp[i][i] = 1$

case 1:  $s[i] == s[j]$

$a*****a \rightarrow dp[i][j]$   
 $= dp[i+1][j-1] + 2$

case 2:  $s[i] != s[j]$

$ab*****b \quad dp[i][j] = dp[i+1][j]$

$a*****ab \quad dp[i][j] = dp[i][j-1]$

^ ^  
 i j

<http://zxi.mytechroad.com/blog/>

<http://zxi.mytechroad.com/blog/dynamic-programming/leetcode-516-longest-palindromic-subsequence/>

[https://www.youtube.com/watch?v=OZX1nqaQ\\_9M](https://www.youtube.com/watch?v=OZX1nqaQ_9M)

# Example:

c	b	b	s	a	c
0	1	2	3	4	5

return 4  
(len of "cbbc")

Initialize the table, note that blank means 0

j

	0	1	2	3	4	5
0						
1						
2						
3						
4						
5						

i

c	b	b	s	a	c
0	1	2	3	4	5
i,j					

len: from 1 to 6 (n)	crnt_v: 1
i: from 0 to 5 (n - len)	crnt_v: 0
j: i + len - 1	crnt_v: 0

Note: crnt\_v refers to current value

```

1  class Solution {
2  public:
3      int longestPalindromeSubseq(string s) {
4          const int n = s.length();
5          vector<vector<int>> dp(n, vector<int>(n, 0));
6          for (int len = 1; len <= n; ++len) {
7              for (int i = 0; i <= n - len; ++i) {
8                  int j = i + len - 1;
9                  if (i == j) {
10                     dp[i][j] = 1;
11                     continue;
12                 }
13                 dp[i][j] = max(dp[i + 1][j], dp[i][j - 1]);
14                 if (s[i] == s[j])
15                     dp[i][j] = dp[i + 1][j - 1] + 2;
16             }
17         }
18         return dp[0][n - 1];
19     }
20 };

```

	0	1	2	3	4	5
0						
1						
2						
3						
4						
5						

c    b    b    s    a    c  
 0    1    2    3    4    5  
 i,j

```

1  class Solution {
2  public:
3      int longestPalindromeSubseq(string s) {
4          const int n = s.length();
5          vector<vector<int>> dp(n, vector<int>(n, 0));
6          for (int len = 1; len <= n; ++len) {
7              for (int i = 0; i <= n - len; ++i) {
8                  int j = i + len - 1;
9                  if (i == j) {
10                     dp[i][j] = 1;
11                     continue;
12                 }
13                 dp[i][j] = max(dp[i + 1][j], dp[i][j - 1]);
14                 if (s[i] == s[j])
15                     dp[i][j] = dp[i + 1][j - 1] + 2;
16             }
17         }
18         return dp[0][n - 1];
19     }
20 };

```

len: from 1 to 6 (n)                      crnt\_v: 1  
 i: from 0 to 5 (n - len)                  crnt\_v: 0  
 j: i + len - 1                              crnt\_v: 0  
 since i == j

dp[0][0] = 1

	0	1	2	3	4	5
0	1					
1						
2						
3						
4						
5						

c	b	b	s	a	c
0	1	2	3	4	5
	i,j				

len: from 1 to 6 (n)	currnt_v: 1
i: from 0 to 5 (n - len)	currnt_v: 1
j: i + len - 1	currnt_v: 1

```

1  class Solution {
2  public:
3      int longestPalindromeSubseq(string s) {
4          const int n = s.length();
5          vector<vector<int>> dp(n, vector<int>(n, 0));
6          for (int len = 1; len <= n; ++len) {
7              for (int i = 0; i <= n - len; ++i) {
8                  int j = i + len - 1;
9                  if (i == j) {
10                     dp[i][j] = 1;
11                     continue;
12                 }
13                 dp[i][j] = max(dp[i + 1][j], dp[i][j - 1]);
14                 if (s[i] == s[j])
15                     dp[i][j] = dp[i + 1][j - 1] + 2;
16             }
17         }
18         return dp[0][n - 1];
19     }
20 };

```

	0	1	2	3	4	5
0	1					
1						
2						
3						
4						
5						



c	b	b	s	a	c
0	1	2	3	4	5
	i,j				

```

1  class Solution {
2  public:
3      int longestPalindromeSubseq(string s) {
4          const int n = s.length();
5          vector<vector<int>> dp(n, vector<int>(n, 0));
6          for (int len = 1; len <= n; ++len) {
7              for (int i = 0; i <= n - len; ++i) {
8                  int j = i + len - 1;
9                  if (i == j) {
10                     dp[i][j] = 1;
11                     continue;
12                 }
13                 dp[i][j] = max(dp[i + 1][j], dp[i][j - 1]);
14                 if (s[i] == s[j])
15                     dp[i][j] = dp[i + 1][j - 1] + 2;
16             }
17         }
18         return dp[0][n - 1];
19     }
20 };

```

len: from 1 to 6 (n)  
i: from 0 to 5 (n - len)  
j: i + len - 1  
since i == j

dp[1][1] = 1

	0	1	2	3	4	5
0	1					
1		1				
2						
3						
4						
5						

c	b	b	s	a	c
0	1	2	3	4	5
		i,j			

len: from 1 to 6 (n)	currnt_v: 1
i: from 0 to 5 (n - len)	currnt_v: 2
j: i + len - 1	currnt_v: 2

```

1  class Solution {
2  public:
3      int longestPalindromeSubseq(string s) {
4          const int n = s.length();
5          vector<vector<int>> dp(n, vector<int>(n, 0));
6          for (int len = 1; len <= n; ++len) {
7              for (int i = 0; i <= n - len; ++i) {
8                  int j = i + len - 1;
9                  if (i == j) {
10                     dp[i][j] = 1;
11                     continue;
12                 }
13                 dp[i][j] = max(dp[i + 1][j], dp[i][j - 1]);
14                 if (s[i] == s[j])
15                     dp[i][j] = dp[i + 1][j - 1] + 2;
16             }
17         }
18         return dp[0][n - 1];
19     }
20 };

```

	0	1	2	3	4	5
0	1					
1		1				
2						
3						
4						
5						

c	b	b	s	a	c
0	1	2	3	4	5
		i,j			

```

1  class Solution {
2  public:
3      int longestPalindromeSubseq(string s) {
4          const int n = s.length();
5          vector<vector<int>> dp(n, vector<int>(n, 0));
6          for (int len = 1; len <= n; ++len) {
7              for (int i = 0; i <= n - len; ++i) {
8                  int j = i + len - 1;
9                  if (i == j) {
10                     dp[i][j] = 1;
11                     continue;
12                 }
13                 dp[i][j] = max(dp[i + 1][j], dp[i][j - 1]);
14                 if (s[i] == s[j])
15                     dp[i][j] = dp[i + 1][j - 1] + 2;
16             }
17         }
18         return dp[0][n - 1];
19     }
20 };

```

len: from 1 to 6 (n)  
i: from 0 to 5 (n - len)  
j: i + len - 1  
since i == j  
dp[2][2] = 1

	0	1	2	3	4	5
0	1					
1		1				
2			1			
3						
4						
5						

c	b	b	s	a	c
0	1	2	3	4	5
			i,j		

len: from 1 to 6 (n)  
i: from 0 to 5 (n - len)  
j: i + len - 1

currnt\_v: 1  
currnt\_v: 3  
currnt\_v: 3

```

1  class Solution {
2  public:
3      int longestPalindromeSubseq(string s) {
4          const int n = s.length();
5          vector<vector<int>> dp(n, vector<int>(n, 0));
6          for (int len = 1; len <= n; ++len) {
7              for (int i = 0; i <= n - len; ++i) {
8                  int j = i + len - 1;
9                  if (i == j) {
10                     dp[i][j] = 1;
11                     continue;
12                 }
13                 dp[i][j] = max(dp[i + 1][j], dp[i][j - 1]);
14                 if (s[i] == s[j])
15                     dp[i][j] = dp[i + 1][j - 1] + 2;
16             }
17         }
18         return dp[0][n - 1];
19     }
20 };

```

	0	1	2	3	4	5
0	1					
1		1				
2			1			
3						
4						
5						

c	b	b	s	a	c
0	1	2	3	4	5
			i,j		

```

1  class Solution {
2  public:
3      int longestPalindromeSubseq(string s) {
4          const int n = s.length();
5          vector<vector<int>> dp(n, vector<int>(n, 0));
6          for (int len = 1; len <= n; ++len) {
7              for (int i = 0; i <= n - len; ++i) {
8                  int j = i + len - 1;
9                  if (i == j) {
10                     dp[i][j] = 1;
11                     continue;
12                 }
13                 dp[i][j] = max(dp[i + 1][j], dp[i][j - 1]);
14                 if (s[i] == s[j])
15                     dp[i][j] = dp[i + 1][j - 1] + 2;
16             }
17         }
18         return dp[0][n - 1];
19     }
20 };

```

len: from 1 to 6 (n)  
i: from 0 to 5 (n - len)  
j: i + len - 1  
since i == j

dp[3][3] = 1

	0	1	2	3	4	5
0	1					
1		1				
2			1			
3				1		
4						
5						

c	b	b	s	a	c
0	1	2	3	4	5
				i,j	

len: from 1 to 6 (n)	currnt_v: 1
i: from 0 to 5 (n - len)	currnt_v: 4
j: i + len - 1	currnt_v: 4

```

1  class Solution {
2  public:
3      int longestPalindromeSubseq(string s) {
4          const int n = s.length();
5          vector<vector<int>> dp(n, vector<int>(n, 0));
6          for (int len = 1; len <= n; ++len) {
7              for (int i = 0; i <= n - len; ++i) {
8                  int j = i + len - 1;
9                  if (i == j) {
10                     dp[i][j] = 1;
11                     continue;
12                 }
13                 dp[i][j] = max(dp[i + 1][j], dp[i][j - 1]);
14                 if (s[i] == s[j])
15                     dp[i][j] = dp[i + 1][j - 1] + 2;
16             }
17         }
18         return dp[0][n - 1];
19     }
20 };

```

	0	1	2	3	4	5
0	1					
1		1				
2			1			
3				1		
4						
5						

c	b	b	s	a	c
0	1	2	3	4	5
				i,j	

```

1  class Solution {
2  public:
3      int longestPalindromeSubseq(string s) {
4          const int n = s.length();
5          vector<vector<int>> dp(n, vector<int>(n, 0));
6          for (int len = 1; len <= n; ++len) {
7              for (int i = 0; i <= n - len; ++i) {
8                  int j = i + len - 1;
9                  if (i == j) {
10                     dp[i][j] = 1;
11                     continue;
12                 }
13                 dp[i][j] = max(dp[i + 1][j], dp[i][j - 1]);
14                 if (s[i] == s[j])
15                     dp[i][j] = dp[i + 1][j - 1] + 2;
16             }
17         }
18         return dp[0][n - 1];
19     }
20 };

```

len: from 1 to 6 (n)  
i: from 0 to 5 (n - len)  
j: i + len - 1  
since i == j

dp[4][4] = 1

	0	1	2	3	4	5
0	1					
1		1				
2			1			
3				1		
4					1	
5						

c    b    b    s    a    c  
 0    1    2    3    4    5  
                   i,j

len: from 1 to 6 (n)            crnt\_v: 1  
 i: from 0 to 5 (n - len)      crnt\_v: 5  
 j: i + len - 1                crnt\_v: 5

```

1  class Solution {
2  public:
3      int longestPalindromeSubseq(string s) {
4          const int n = s.length();
5          vector<vector<int>> dp(n, vector<int>(n, 0));
6          for (int len = 1; len <= n; ++len) {
7              for (int i = 0; i <= n - len; ++i) {
8                  int j = i + len - 1;
9                  if (i == j) {
10                     dp[i][j] = 1;
11                     continue;
12                 }
13                 dp[i][j] = max(dp[i + 1][j], dp[i][j - 1]);
14                 if (s[i] == s[j])
15                     dp[i][j] = dp[i + 1][j - 1] + 2;
16             }
17         }
18         return dp[0][n - 1];
19     }
20 };

```

	0	1	2	3	4	5
0	1					
1		1				
2			1			
3				1		
4					1	
5						



c    b    b    s    a    c  
 0    1    2    3    4    5  
                   i,j

```

1  class Solution {
2  public:
3      int longestPalindromeSubseq(string s) {
4          const int n = s.length();
5          vector<vector<int>> dp(n, vector<int>(n, 0));
6          for (int len = 1; len <= n; ++len) {
7              for (int i = 0; i <= n - len; ++i) {
8                  int j = i + len - 1;
9                  if (i == j) {
10                     dp[i][j] = 1;
11                     continue;
12                 }
13                 dp[i][j] = max(dp[i + 1][j], dp[i][j - 1]);
14                 if (s[i] == s[j])
15                     dp[i][j] = dp[i + 1][j - 1] + 2;
16             }
17         }
18         return dp[0][n - 1];
19     }
20 };

```

len: from 1 to 6 (n)                      crnt\_v: 1  
 i: from 0 to 5 (n - len)                  crnt\_v: 5  
 j: i + len - 1                              crnt\_v: 5  
 since i == j

dp[5][5] = 1

	0	1	2	3	4	5
0	1					
1		1				
2			1			
3				1		
4					1	
5						1

c	b	b	s	a	c
0	1	2	3	4	5
i	j				

len: from 1 to 6 (n)	currnt_v: 2
i: from 0 to 4 (n - len)	currnt_v: 0
j: i + len - 1	currnt_v: 1

```

1  class Solution {
2  public:
3      int longestPalindromeSubseq(string s) {
4          const int n = s.length();
5          vector<vector<int>> dp(n, vector<int>(n, 0));
6          for (int len = 1; len <= n; ++len) {
7              for (int i = 0; i <= n - len; ++i) {
8                  int j = i + len - 1;
9                  if (i == j) {
10                     dp[i][j] = 1;
11                     continue;
12                 }
13                 dp[i][j] = max(dp[i + 1][j], dp[i][j - 1]);
14                 if (s[i] == s[j])
15                     dp[i][j] = dp[i + 1][j - 1] + 2;
16             }
17         }
18         return dp[0][n - 1];
19     }
20 };

```

	0	1	2	3	4	5
0	1					
1		1				
2			1			
3				1		
4					1	
5						1

c	b	b	s	a	c
0	1	2	3	4	5
i	j				

```

1  class Solution {
2  public:
3      int longestPalindromeSubseq(string s) {
4          const int n = s.length();
5          vector<vector<int>> dp(n, vector<int>(n, 0));
6          for (int len = 1; len <= n; ++len) {
7              for (int i = 0; i <= n - len; ++i) {
8                  int j = i + len - 1;
9                  if (i == j) {
10                     dp[i][j] = 1;
11                     continue;
12                 }
13                 dp[i][j] = max(dp[i + 1][j], dp[i][j - 1]);
14                 if (s[i] == s[j])
15                     dp[i][j] = dp[i + 1][j - 1] + 2;
16             }
17         }
18         return dp[0][n - 1];
19     }
20 };

```

len: from 1 to 6 (n)      crnt\_v: 2  
i: from 0 to 4 (n - len)      crnt\_v: 0  
j: i + len - 1      crnt\_v: 1

since  $i \neq j$

$$dp[0][1] = \max(dp[1][1], dp[0][0]) \\ = \max(1, 1) = 1$$

	0	1	2	3	4	5
0	1	1				
1		1				
2			1			
3				1		
4					1	
5						1

c	b	b	s	a	c
0	1	2	3	4	5
	i	j			

len: from 1 to 6 (n)	currnt_v: 2
i: from 0 to 4 (n - len)	currnt_v: 1
j: i + len - 1	currnt_v: 2

```

1  class Solution {
2  public:
3      int longestPalindromeSubseq(string s) {
4          const int n = s.length();
5          vector<vector<int>> dp(n, vector<int>(n, 0));
6          for (int len = 1; len <= n; ++len) {
7              for (int i = 0; i <= n - len; ++i) {
8                  int j = i + len - 1;
9                  if (i == j) {
10                     dp[i][j] = 1;
11                     continue;
12                 }
13                 dp[i][j] = max(dp[i + 1][j], dp[i][j - 1]);
14                 if (s[i] == s[j])
15                     dp[i][j] = dp[i + 1][j - 1] + 2;
16             }
17         }
18         return dp[0][n - 1];
19     }
20 };

```

	0	1	2	3	4	5
0	1	1				
1		1				
2			1			
3				1		
4					1	
5						1

c	b	b	s	a	c
0	1	2	3	4	5
	i	j			

```

1  class Solution {
2  public:
3      int longestPalindromeSubseq(string s) {
4          const int n = s.length();
5          vector<vector<int>> dp(n, vector<int>(n, 0));
6          for (int len = 1; len <= n; ++len) {
7              for (int i = 0; i <= n - len; ++i) {
8                  int j = i + len - 1;
9                  if (i == j) {
10                     dp[i][j] = 1;
11                     continue;
12                 }
13                 dp[i][j] = max(dp[i + 1][j], dp[i][j - 1]);
14                 if (s[i] == s[j])
15                     dp[i][j] = dp[i + 1][j - 1] + 2;
16             }
17         }
18         return dp[0][n - 1];
19     }
20 };

```

len: from 1 to 6 (n)                  crrent\_v: 2  
i: from 0 to 4 (n - len)              crrent\_v: 1  
j: i + len - 1                          crrent\_v: 2

since  $s[i] == s[j]$  ( $== 'b'$ )

$$dp[1][2] = dp[2][1] + 2 = 2$$

	0	1	2	3	4	5
0	1	1				
1		1	2			
2		0	1			
3				1		
4					1	
5						1

c	b	b	s	a	c
0	1	2	3	4	5
		i	j		

len: from 1 to 6 (n)	currnt_v: 2
i: from 0 to 4 (n - len)	currnt_v: 2
j: i + len - 1	currnt_v: 3

```

1  class Solution {
2  public:
3      int longestPalindromeSubseq(string s) {
4          const int n = s.length();
5          vector<vector<int>> dp(n, vector<int>(n, 0));
6          for (int len = 1; len <= n; ++len) {
7              for (int i = 0; i <= n - len; ++i) {
8                  int j = i + len - 1;
9                  if (i == j) {
10                     dp[i][j] = 1;
11                     continue;
12                 }
13                 dp[i][j] = max(dp[i + 1][j], dp[i][j - 1]);
14                 if (s[i] == s[j])
15                     dp[i][j] = dp[i + 1][j - 1] + 2;
16             }
17         }
18         return dp[0][n - 1];
19     }
20 };

```

	0	1	2	3	4	5
0	1	1				
1		1	2			
2			1			
3				1		
4					1	
5						1

c	b	b	s	a	c
0	1	2	3	4	5
		i	j		

```

1  class Solution {
2  public:
3      int longestPalindromeSubseq(string s) {
4          const int n = s.length();
5          vector<vector<int>> dp(n, vector<int>(n, 0));
6          for (int len = 1; len <= n; ++len) {
7              for (int i = 0; i <= n - len; ++i) {
8                  int j = i + len - 1;
9                  if (i == j) {
10                     dp[i][j] = 1;
11                     continue;
12                 }
13                 dp[i][j] = max(dp[i + 1][j], dp[i][j - 1]);
14                 if (s[i] == s[j])
15                     dp[i][j] = dp[i + 1][j - 1] + 2;
16             }
17         }
18         return dp[0][n - 1];
19     }
20 };

```

len: from 1 to 6 (n)      crnt\_v: 2  
i: from 0 to 4 (n - len)      crnt\_v: 2  
j: i + len - 1      crnt\_v: 3

since  $i \neq j$

$$dp[2][3] = \max(dp[3][3], dp[2][2]) \\ = \max(1, 1) = 1$$

	0	1	2	3	4	5
0	1	1				
1		1	2			
2			1	1		
3				1		
4					1	
5						1

c	b	b	s	a	c
0	1	2	3	4	5
			i	j	

len: from 1 to 6 (n)	currnt_v: 2
i: from 0 to 4 (n - len)	currnt_v: 3
j: i + len - 1	currnt_v: 4

```

1  class Solution {
2  public:
3      int longestPalindromeSubseq(string s) {
4          const int n = s.length();
5          vector<vector<int>> dp(n, vector<int>(n, 0));
6          for (int len = 1; len <= n; ++len) {
7              for (int i = 0; i <= n - len; ++i) {
8                  int j = i + len - 1;
9                  if (i == j) {
10                     dp[i][j] = 1;
11                     continue;
12                 }
13                 dp[i][j] = max(dp[i + 1][j], dp[i][j - 1]);
14                 if (s[i] == s[j])
15                     dp[i][j] = dp[i + 1][j - 1] + 2;
16             }
17         }
18         return dp[0][n - 1];
19     }
20 };

```

	0	1	2	3	4	5
0	1	1				
1		1	2			
2			1	1		
3				1		
4					1	
5						1



c	b	b	s	a	c
0	1	2	3	4	5
			i	j	

```

1  class Solution {
2  public:
3      int longestPalindromeSubseq(string s) {
4          const int n = s.length();
5          vector<vector<int>> dp(n, vector<int>(n, 0));
6          for (int len = 1; len <= n; ++len) {
7              for (int i = 0; i <= n - len; ++i) {
8                  int j = i + len - 1;
9                  if (i == j) {
10                     dp[i][j] = 1;
11                     continue;
12                 }
13                 dp[i][j] = max(dp[i + 1][j], dp[i][j - 1]);
14                 if (s[i] == s[j])
15                     dp[i][j] = dp[i + 1][j - 1] + 2;
16             }
17         }
18         return dp[0][n - 1];
19     }
20 };

```

len: from 1 to 6 (n)      crnt\_v: 2  
i: from 0 to 4 (n - len)      crnt\_v: 3  
j: i + len - 1      crnt\_v: 4

since  $i \neq j$

$$dp[3][4] = \max(dp[4][4], dp[3][3]) \\ = \max(1, 1) = 1$$

	0	1	2	3	4	5
0	1	1				
1		1	2			
2			1	1		
3				1	1	
4					1	
5						1

c	b	b	s	a	c
0	1	2	3	4	5
				i	j

len: from 1 to 6 (n)	currnt_v: 2
i: from 0 to 4 (n - len)	currnt_v: 4
j: i + len - 1	currnt_v: 5

```

1  class Solution {
2  public:
3      int longestPalindromeSubseq(string s) {
4          const int n = s.length();
5          vector<vector<int>> dp(n, vector<int>(n, 0));
6          for (int len = 1; len <= n; ++len) {
7              for (int i = 0; i <= n - len; ++i) {
8                  int j = i + len - 1;
9                  if (i == j) {
10                     dp[i][j] = 1;
11                     continue;
12                 }
13                 dp[i][j] = max(dp[i + 1][j], dp[i][j - 1]);
14                 if (s[i] == s[j])
15                     dp[i][j] = dp[i + 1][j - 1] + 2;
16             }
17         }
18         return dp[0][n - 1];
19     }
20 };

```

	0	1	2	3	4	5
0	1	1				
1		1	2			
2			1	1		
3				1	1	
4					1	
5						1

c	b	b	s	a	c
0	1	2	3	4	5
				i	j

```

1  class Solution {
2  public:
3      int longestPalindromeSubseq(string s) {
4          const int n = s.length();
5          vector<vector<int>> dp(n, vector<int>(n, 0));
6          for (int len = 1; len <= n; ++len) {
7              for (int i = 0; i <= n - len; ++i) {
8                  int j = i + len - 1;
9                  if (i == j) {
10                     dp[i][j] = 1;
11                     continue;
12                 }
13                 dp[i][j] = max(dp[i + 1][j], dp[i][j - 1]);
14                 if (s[i] == s[j])
15                     dp[i][j] = dp[i + 1][j - 1] + 2;
16             }
17         }
18         return dp[0][n - 1];
19     }
20 };

```

len: from 1 to 6 (n)      crnt\_v: 2  
i: from 0 to 4 (n - len)      crnt\_v: 4  
j: i + len - 1      crnt\_v: 5  
since i != j

$dp[4][5] = \max(dp[5][5], dp[4][4])$   
 $= \max(1, 1) = 1$

	0	1	2	3	4	5
0	1	1				
1		1	2			
2			1	1		
3				1	1	
4					1	1
5						1

c	b	b	s	a	c
0	1	2	3	4	5
i		j			

len: from 1 to 6 (n)	currnt_v: 3
i: from 0 to 3 (n - len)	currnt_v: 0
j: i + len - 1	currnt_v: 2

```

1  class Solution {
2  public:
3      int longestPalindromeSubseq(string s) {
4          const int n = s.length();
5          vector<vector<int>> dp(n, vector<int>(n, 0));
6          for (int len = 1; len <= n; ++len) {
7              for (int i = 0; i <= n - len; ++i) {
8                  int j = i + len - 1;
9                  if (i == j) {
10                     dp[i][j] = 1;
11                     continue;
12                 }
13                 dp[i][j] = max(dp[i + 1][j], dp[i][j - 1]);
14                 if (s[i] == s[j])
15                     dp[i][j] = dp[i + 1][j - 1] + 2;
16             }
17         }
18         return dp[0][n - 1];
19     }
20 };

```

	0	1	2	3	4	5
0	1	1				
1		1	2			
2			1	1		
3				1	1	
4					1	1
5						1

c	b	b	s	a	c
0	1	2	3	4	5
i		j			

```

1  class Solution {
2  public:
3      int longestPalindromeSubseq(string s) {
4          const int n = s.length();
5          vector<vector<int>> dp(n, vector<int>(n, 0));
6          for (int len = 1; len <= n; ++len) {
7              for (int i = 0; i <= n - len; ++i) {
8                  int j = i + len - 1;
9                  if (i == j) {
10                     dp[i][j] = 1;
11                     continue;
12                 }
13                 dp[i][j] = max(dp[i + 1][j], dp[i][j - 1]);
14                 if (s[i] == s[j])
15                     dp[i][j] = dp[i + 1][j - 1] + 2;
16             }
17         }
18         return dp[0][n - 1];
19     }
20 };

```

len: from 1 to 6 (n)      crnt\_v: 3  
i: from 0 to 3 (n - len)      crnt\_v: 0  
j: i + len - 1      crnt\_v: 2

since  $i \neq j$

$$dp[0][2] = \max(dp[1][2], dp[0][1]) \\ = \max(2, 1) = 2$$

	0	1	2	3	4	5
0	1	1	2			
1		1	2			
2			1	1		
3				1	1	
4					1	1
5						1

c	b	b	s	a	c
0	1	2	3	4	5
	i		j		

len: from 1 to 6 (n)	currnt_v: 3
i: from 0 to 3 (n - len)	currnt_v: 1
j: i + len - 1	currnt_v: 3

```

1  class Solution {
2  public:
3      int longestPalindromeSubseq(string s) {
4          const int n = s.length();
5          vector<vector<int>> dp(n, vector<int>(n, 0));
6          for (int len = 1; len <= n; ++len) {
7              for (int i = 0; i <= n - len; ++i) {
8                  int j = i + len - 1;
9                  if (i == j) {
10                     dp[i][j] = 1;
11                     continue;
12                 }
13                 dp[i][j] = max(dp[i + 1][j], dp[i][j - 1]);
14                 if (s[i] == s[j])
15                     dp[i][j] = dp[i + 1][j - 1] + 2;
16             }
17         }
18         return dp[0][n - 1];
19     }
20 };

```

	0	1	2	3	4	5
0	1	1	2			
1		1	2			
2			1	1		
3				1	1	
4					1	1
5						1

c	b	b	s	a	c
0	1	2	3	4	5
	i		j		

```

1  class Solution {
2  public:
3      int longestPalindromeSubseq(string s) {
4          const int n = s.length();
5          vector<vector<int>> dp(n, vector<int>(n, 0));
6          for (int len = 1; len <= n; ++len) {
7              for (int i = 0; i <= n - len; ++i) {
8                  int j = i + len - 1;
9                  if (i == j) {
10                     dp[i][j] = 1;
11                     continue;
12                 }
13                 dp[i][j] = max(dp[i + 1][j], dp[i][j - 1]);
14                 if (s[i] == s[j])
15                     dp[i][j] = dp[i + 1][j - 1] + 2;
16             }
17         }
18         return dp[0][n - 1];
19     }
20 };

```

len: from 1 to 6 (n)      crnt\_v: 3  
i: from 0 to 3 (n - len)      crnt\_v: 1  
j: i + len - 1      crnt\_v: 3

since  $i \neq j$

$$dp[1][3] = \max(dp[2][3], dp[1][2]) \\ = \max(1, 2) = 2$$

	0	1	2	3	4	5
0	1	1	2			
1		1	2	2		
2			1	1		
3				1	1	
4					1	1
5						1

c	b	b	s	a	c
0	1	2	3	4	5
		i		j	

len: from 1 to 6 (n)	currnt_v: 3
i: from 0 to 3 (n - len)	currnt_v: 2
j: i + len - 1	currnt_v: 4

```

1  class Solution {
2  public:
3      int longestPalindromeSubseq(string s) {
4          const int n = s.length();
5          vector<vector<int>> dp(n, vector<int>(n, 0));
6          for (int len = 1; len <= n; ++len) {
7              for (int i = 0; i <= n - len; ++i) {
8                  int j = i + len - 1;
9                  if (i == j) {
10                     dp[i][j] = 1;
11                     continue;
12                 }
13                 dp[i][j] = max(dp[i + 1][j], dp[i][j - 1]);
14                 if (s[i] == s[j])
15                     dp[i][j] = dp[i + 1][j - 1] + 2;
16             }
17         }
18         return dp[0][n - 1];
19     }
20 };

```

	0	1	2	3	4	5
0	1	1	2			
1		1	2	2		
2			1	1		
3				1	1	
4					1	1
5						1



c	b	b	s	a	c
0	1	2	3	4	5
		i		j	

```

1  class Solution {
2  public:
3      int longestPalindromeSubseq(string s) {
4          const int n = s.length();
5          vector<vector<int>> dp(n, vector<int>(n, 0));
6          for (int len = 1; len <= n; ++len) {
7              for (int i = 0; i <= n - len; ++i) {
8                  int j = i + len - 1;
9                  if (i == j) {
10                     dp[i][j] = 1;
11                     continue;
12                 }
13                 dp[i][j] = max(dp[i + 1][j], dp[i][j - 1]);
14                 if (s[i] == s[j])
15                     dp[i][j] = dp[i + 1][j - 1] + 2;
16             }
17         }
18         return dp[0][n - 1];
19     }
20 };

```

len: from 1 to 6 (n)      crnt\_v: 3  
i: from 0 to 3 (n - len)      crnt\_v: 2  
j: i + len - 1      crnt\_v: 4

since  $i \neq j$

$$dp[2][4] = \max(dp[3][4], dp[2][3]) \\ = \max(1, 1) = 1$$

	0	1	2	3	4	5
0	1	1	2			
1		1	2	2		
2			1	1	1	
3				1	1	
4					1	1
5						1

c	b	b	s	a	c
0	1	2	3	4	5
			i		j

len: from 1 to 6 (n)	currnt_v: 3
i: from 0 to 3 (n - len)	currnt_v: 3
j: i + len - 1	currnt_v: 5

```

1  class Solution {
2  public:
3      int longestPalindromeSubseq(string s) {
4          const int n = s.length();
5          vector<vector<int>> dp(n, vector<int>(n, 0));
6          for (int len = 1; len <= n; ++len) {
7              for (int i = 0; i <= n - len; ++i) {
8                  int j = i + len - 1;
9                  if (i == j) {
10                     dp[i][j] = 1;
11                     continue;
12                 }
13                 dp[i][j] = max(dp[i + 1][j], dp[i][j - 1]);
14                 if (s[i] == s[j])
15                     dp[i][j] = dp[i + 1][j - 1] + 2;
16             }
17         }
18         return dp[0][n - 1];
19     }
20 };

```

	0	1	2	3	4	5
0	1	1	2			
1		1	2	2		
2			1	1	1	
3				1	1	
4					1	1
5						1

c	b	b	s	a	c
0	1	2	3	4	5
			i		j

```

1  class Solution {
2  public:
3      int longestPalindromeSubseq(string s) {
4          const int n = s.length();
5          vector<vector<int>> dp(n, vector<int>(n, 0));
6          for (int len = 1; len <= n; ++len) {
7              for (int i = 0; i <= n - len; ++i) {
8                  int j = i + len - 1;
9                  if (i == j) {
10                     dp[i][j] = 1;
11                     continue;
12                 }
13                 dp[i][j] = max(dp[i + 1][j], dp[i][j - 1]);
14                 if (s[i] == s[j])
15                     dp[i][j] = dp[i + 1][j - 1] + 2;
16             }
17         }
18         return dp[0][n - 1];
19     }
20 };

```

len: from 1 to 6 (n)      crnt\_v: 3  
i: from 0 to 3 (n - len)      crnt\_v: 3  
j: i + len - 1      crnt\_v: 5

since  $i \neq j$

$$dp[3][5] = \max(dp[4][5], dp[3][4]) \\ = \max(1, 1) = 1$$

	0	1	2	3	4	5
0	1	1	2			
1		1	2	2		
2			1	1	1	
3				1	1	1
4					1	1
5						1

c	b	b	s	a	c
0	1	2	3	4	5
i			j		

len: from 1 to 6 (n)	currnt_v: 4
i: from 0 to 2 (n - len)	currnt_v: 0
j: i + len - 1	currnt_v: 3

```

1  class Solution {
2  public:
3      int longestPalindromeSubseq(string s) {
4          const int n = s.length();
5          vector<vector<int>> dp(n, vector<int>(n, 0));
6          for (int len = 1; len <= n; ++len) {
7              for (int i = 0; i <= n - len; ++i) {
8                  int j = i + len - 1;
9                  if (i == j) {
10                     dp[i][j] = 1;
11                     continue;
12                 }
13                 dp[i][j] = max(dp[i + 1][j], dp[i][j - 1]);
14                 if (s[i] == s[j])
15                     dp[i][j] = dp[i + 1][j - 1] + 2;
16             }
17         }
18         return dp[0][n - 1];
19     }
20 };

```

	0	1	2	3	4	5
0	1	1	2			
1		1	2	2		
2			1	1	1	
3				1	1	1
4					1	1
5						1

c	b	b	s	a	c
0	1	2	3	4	5
i			j		

```

1  class Solution {
2  public:
3      int longestPalindromeSubseq(string s) {
4          const int n = s.length();
5          vector<vector<int>> dp(n, vector<int>(n, 0));
6          for (int len = 1; len <= n; ++len) {
7              for (int i = 0; i <= n - len; ++i) {
8                  int j = i + len - 1;
9                  if (i == j) {
10                     dp[i][j] = 1;
11                     continue;
12                 }
13                 dp[i][j] = max(dp[i + 1][j], dp[i][j - 1]);
14                 if (s[i] == s[j])
15                     dp[i][j] = dp[i + 1][j - 1] + 2;
16             }
17         }
18         return dp[0][n - 1];
19     }
20 };

```

len: from 1 to 6 (n)      crnt\_v: 4  
i: from 0 to 2 (n - len)      crnt\_v: 0  
j: i + len - 1      crnt\_v: 3

since  $i \neq j$

$$dp[0][3] = \max(dp[1][3], dp[0][2]) \\ = \max(2, 2) = 2$$

	0	1	2	3	4	5
0	1	1	2	2		
1		1	2	2		
2			1	1	1	
3				1	1	1
4					1	1
5						1

c	b	b	s	a	c
0	1	2	3	4	5
	i			j	

len: from 1 to 6 (n)	currnt_v: 4
i: from 0 to 2 (n - len)	currnt_v: 1
j: i + len - 1	currnt_v: 4

```

1  class Solution {
2  public:
3      int longestPalindromeSubseq(string s) {
4          const int n = s.length();
5          vector<vector<int>> dp(n, vector<int>(n, 0));
6          for (int len = 1; len <= n; ++len) {
7              for (int i = 0; i <= n - len; ++i) {
8                  int j = i + len - 1;
9                  if (i == j) {
10                     dp[i][j] = 1;
11                     continue;
12                 }
13                 dp[i][j] = max(dp[i + 1][j], dp[i][j - 1]);
14                 if (s[i] == s[j])
15                     dp[i][j] = dp[i + 1][j - 1] + 2;
16             }
17         }
18         return dp[0][n - 1];
19     }
20 };

```

	0	1	2	3	4	5
0	1	1	2	2		
1		1	2	2		
2			1	1	1	
3				1	1	1
4					1	1
5						1

c	b	b	s	a	c
0	1	2	3	4	5
	i			j	

```

1  class Solution {
2  public:
3      int longestPalindromeSubseq(string s) {
4          const int n = s.length();
5          vector<vector<int>> dp(n, vector<int>(n, 0));
6          for (int len = 1; len <= n; ++len) {
7              for (int i = 0; i <= n - len; ++i) {
8                  int j = i + len - 1;
9                  if (i == j) {
10                     dp[i][j] = 1;
11                     continue;
12                 }
13                 dp[i][j] = max(dp[i + 1][j], dp[i][j - 1]);
14                 if (s[i] == s[j])
15                     dp[i][j] = dp[i + 1][j - 1] + 2;
16             }
17         }
18         return dp[0][n - 1];
19     }
20 };

```

len: from 1 to 6 (n)      crnt\_v: 4  
i: from 0 to 2 (n - len)      crnt\_v: 1  
j: i + len - 1      crnt\_v: 4

since  $i \neq j$

$$dp[1][4] = \max(dp[2][4], dp[1][3]) \\ = \max(1, 2) = 2$$

	0	1	2	3	4	5
0	1	1	2	2		
1		1	2	2	2	
2			1	1	1	
3				1	1	1
4					1	1
5						1

c	b	b	s	a	c
0	1	2	3	4	5
		i			j

len: from 1 to 6 (n)	currnt_v: 4
i: from 0 to 2 (n - len)	currnt_v: 2
j: i + len - 1	currnt_v: 5

```

1  class Solution {
2  public:
3      int longestPalindromeSubseq(string s) {
4          const int n = s.length();
5          vector<vector<int>> dp(n, vector<int>(n, 0));
6          for (int len = 1; len <= n; ++len) {
7              for (int i = 0; i <= n - len; ++i) {
8                  int j = i + len - 1;
9                  if (i == j) {
10                     dp[i][j] = 1;
11                     continue;
12                 }
13                 dp[i][j] = max(dp[i + 1][j], dp[i][j - 1]);
14                 if (s[i] == s[j])
15                     dp[i][j] = dp[i + 1][j - 1] + 2;
16             }
17         }
18         return dp[0][n - 1];
19     }
20 };

```

	0	1	2	3	4	5
0	1	1	2	2		
1		1	2	2	2	
2			1	1	1	
3				1	1	1
4					1	1
5						1



c	b	b	s	a	c
0	1	2	3	4	5
		i			j

```

1  class Solution {
2  public:
3      int longestPalindromeSubseq(string s) {
4          const int n = s.length();
5          vector<vector<int>> dp(n, vector<int>(n, 0));
6          for (int len = 1; len <= n; ++len) {
7              for (int i = 0; i <= n - len; ++i) {
8                  int j = i + len - 1;
9                  if (i == j) {
10                     dp[i][j] = 1;
11                     continue;
12                 }
13                 dp[i][j] = max(dp[i + 1][j], dp[i][j - 1]);
14                 if (s[i] == s[j])
15                     dp[i][j] = dp[i + 1][j - 1] + 2;
16             }
17         }
18         return dp[0][n - 1];
19     }
20 };

```

len: from 1 to 6 (n)      crnt\_v: 4  
i: from 0 to 2 (n - len)      crnt\_v: 2  
j: i + len - 1      crnt\_v: 5

since  $i \neq j$

$$dp[2][5] = \max(dp[3][5], dp[2][4]) \\ = \max(1, 2) = 2$$

	0	1	2	3	4	5
0	1	1	2	2		
1		1	2	2	2	
2			1	1	1	1
3				1	1	1
4					1	1
5						1

c	b	b	s	a	c
0	1	2	3	4	5
i				j	

len: from 1 to 6 (n)	currnt_v: 5
i: from 0 to 1 (n - len)	currnt_v: 0
j: i + len - 1	currnt_v: 4

```

1  class Solution {
2  public:
3      int longestPalindromeSubseq(string s) {
4          const int n = s.length();
5          vector<vector<int>> dp(n, vector<int>(n, 0));
6          for (int len = 1; len <= n; ++len) {
7              for (int i = 0; i <= n - len; ++i) {
8                  int j = i + len - 1;
9                  if (i == j) {
10                     dp[i][j] = 1;
11                     continue;
12                 }
13                 dp[i][j] = max(dp[i + 1][j], dp[i][j - 1]);
14                 if (s[i] == s[j])
15                     dp[i][j] = dp[i + 1][j - 1] + 2;
16             }
17         }
18         return dp[0][n - 1];
19     }
20 };

```

	0	1	2	3	4	5
0	1	1	2	2		
1		1	2	2	2	
2			1	1	1	1
3				1	1	1
4					1	1
5						1

c	b	b	s	a	c
0	1	2	3	4	5
i				j	

```

1  class Solution {
2  public:
3      int longestPalindromeSubseq(string s) {
4          const int n = s.length();
5          vector<vector<int>> dp(n, vector<int>(n, 0));
6          for (int len = 1; len <= n; ++len) {
7              for (int i = 0; i <= n - len; ++i) {
8                  int j = i + len - 1;
9                  if (i == j) {
10                     dp[i][j] = 1;
11                     continue;
12                 }
13                 dp[i][j] = max(dp[i + 1][j], dp[i][j - 1]);
14                 if (s[i] == s[j])
15                     dp[i][j] = dp[i + 1][j - 1] + 2;
16             }
17         }
18         return dp[0][n - 1];
19     }
20 };

```

len: from 1 to 6 (n)      crnt\_v: 5  
i: from 0 to 1 (n - len)      crnt\_v: 0  
j: i + len - 1      crnt\_v: 4

since  $i \neq j$

$$dp[0][4] = \max(dp[1][4], dp[0][3]) \\ = \max(2, 2) = 2$$

	0	1	2	3	4	5
0	1	1	2	2	2	
1		1	2	2	2	
2			1	1	1	1
3				1	1	1
4					1	1
5						1

c	b	b	s	a	c
0	1	2	3	4	5
	i				j

len: from 1 to 6 (n)	crrent_v: 5
i: from 0 to 1 (n - len)	crrent_v: 1
j: i + len - 1	crrent_v: 5

```

1  class Solution {
2  public:
3      int longestPalindromeSubseq(string s) {
4          const int n = s.length();
5          vector<vector<int>> dp(n, vector<int>(n, 0));
6          for (int len = 1; len <= n; ++len) {
7              for (int i = 0; i <= n - len; ++i) {
8                  int j = i + len - 1;
9                  if (i == j) {
10                     dp[i][j] = 1;
11                     continue;
12                 }
13                 dp[i][j] = max(dp[i + 1][j], dp[i][j - 1]);
14                 if (s[i] == s[j])
15                     dp[i][j] = dp[i + 1][j - 1] + 2;
16             }
17         }
18         return dp[0][n - 1];
19     }
20 };

```

	0	1	2	3	4	5
0	1	1	2	2	2	
1		1	2	2	2	
2			1	1	1	1
3				1	1	1
4					1	1
5						1

c	b	b	s	a	c
0	1	2	3	4	5
	i				j

```

1  class Solution {
2  public:
3      int longestPalindromeSubseq(string s) {
4          const int n = s.length();
5          vector<vector<int>> dp(n, vector<int>(n, 0));
6          for (int len = 1; len <= n; ++len) {
7              for (int i = 0; i <= n - len; ++i) {
8                  int j = i + len - 1;
9                  if (i == j) {
10                     dp[i][j] = 1;
11                     continue;
12                 }
13                 dp[i][j] = max(dp[i + 1][j], dp[i][j - 1]);
14                 if (s[i] == s[j])
15                     dp[i][j] = dp[i + 1][j - 1] + 2;
16             }
17         }
18         return dp[0][n - 1];
19     }
20 };

```

len: from 1 to 6 (n)      crnt\_v: 5  
i: from 0 to 1 (n - len)      crnt\_v: 1  
j: i + len - 1      crnt\_v: 5

since  $i \neq j$

$$dp[1][5] = \max(dp[2][5], dp[1][4]) \\ = \max(1, 2) = 2$$

	0	1	2	3	4	5
0	1	1	2	2	2	
1		1	2	2	2	2
2			1	1	1	1
3				1	1	1
4					1	1
5						1

c	b	b	s	a	c
0	1	2	3	4	5
i					j

len: from 1 to 6 (n)	currnt_v: 6
i: from 0 to 0 (n - len)	currnt_v: 0
j: i + len - 1	currnt_v: 5

```

1  class Solution {
2  public:
3      int longestPalindromeSubseq(string s) {
4          const int n = s.length();
5          vector<vector<int>> dp(n, vector<int>(n, 0));
6          for (int len = 1; len <= n; ++len) {
7              for (int i = 0; i <= n - len; ++i) {
8                  int j = i + len - 1;
9                  if (i == j) {
10                     dp[i][j] = 1;
11                     continue;
12                 }
13                 dp[i][j] = max(dp[i + 1][j], dp[i][j - 1]);
14                 if (s[i] == s[j])
15                     dp[i][j] = dp[i + 1][j - 1] + 2;
16             }
17         }
18         return dp[0][n - 1];
19     }
20 };

```

	0	1	2	3	4	5
0	1	1	2	2	2	
1		1	2	2	2	2
2			1	1	1	1
3				1	1	1
4					1	1
5						1

c	b	b	s	a	c
0	1	2	3	4	5
i					j

```

1  class Solution {
2  public:
3      int longestPalindromeSubseq(string s) {
4          const int n = s.length();
5          vector<vector<int>> dp(n, vector<int>(n, 0));
6          for (int len = 1; len <= n; ++len) {
7              for (int i = 0; i <= n - len; ++i) {
8                  int j = i + len - 1;
9                  if (i == j) {
10                     dp[i][j] = 1;
11                     continue;
12                 }
13                 dp[i][j] = max(dp[i + 1][j], dp[i][j - 1]);
14                 if (s[i] == s[j])
15                     dp[i][j] = dp[i + 1][j - 1] + 2;
16             }
17         }
18         return dp[0][n - 1];
19     }
20 };

```

len: from 1 to 6 (n)                  crnt\_v: 6  
i: from 0 to 0 (n - len)              crnt\_v: 0  
j: i + len - 1                          crnt\_v: 5

since  $s[i] == s[j]$  ( $== 'c'$ )

$$dp[0][5] = dp[1][4] + 2 = 2 + 2 = 4$$

	0	1	2	3	4	5
0	1	1	2	2	2	4
1		1	2	2	2	2
2			1	1	1	1
3				1	1	1
4					1	1
5						1

c	b	b	s	a	c
0	1	2	3	4	5
i					j

len: from 1 to 6 (n)	currnt_v: 6
i: from 0 to 0 (n - len)	currnt_v: 0
j: i + len - 1	currnt_v: 5

```

1  class Solution {
2  public:
3      int longestPalindromeSubseq(string s) {
4          const int n = s.length();
5          vector<vector<int>> dp(n, vector<int>(n, 0));
6          for (int len = 1; len <= n; ++len) {
7              for (int i = 0; i <= n - len; ++i) {
8                  int j = i + len - 1;
9                  if (i == j) {
10                     dp[i][j] = 1;
11                     continue;
12                 }
13                 dp[i][j] = max(dp[i + 1][j], dp[i][j - 1]);
14                 if (s[i] == s[j])
15                     dp[i][j] = dp[i + 1][j - 1] + 2;
16             }
17         }
18         return dp[0][n - 1];
19     }
20 };

```

return 4 (dp[0][5])

	0	1	2	3	4	5
0	1	1	2	2	2	4
1		1	2	2	2	2
2			1	1	1	1
3				1	1	1
4					1	1
5						1



# Reference

<https://docs.google.com/presentation/d/1KhxVVgl8jzc-g7unDNKFiHY6XDNVSK6LNsadxB14K3U/edit?usp=sharing>

<https://leetcode.com/problems/longest-palindromic-subsequence/>

<http://zxi.mytechroad.com/blog/dynamic-programming/leetcode-516-longest-palindromic-subsequence/>

<https://www.youtube.com/watch?v=jOkE4X-PWOI&feature=youtu.be>