

# Data Structure – Exercise

Ngày 9 tháng 6 năm 2024

## I. Câu hỏi tự luận

1. Cho một list các số nguyên `num_list` và một sliding window có kích thước size `k` di chuyển từ trái sang phải. Mỗi lần dịch chuyển 1 vị trí sang phải có thể nhìn thấy được `k` số trong `num_list` và tìm số lớn nhất trong `k` số này sau mỗi lần trượt `k` phải lớn hơn hoặc bằng 1

Input: `num_list = [3, 4, 5, 1, -44, 5, 10, 12, 33, 1]` với `k=3`

Output: `[5, 5, 5, 5, 10, 12, 33, 33]`

Ví dụ:

`[3, 4, 5], 1, -44, 5, 10, 12, 33, 1 => max 5`  
`3, [4, 5, 1], -44, 5, 10, 12, 33, 1 => max 5`  
`3, 4, [5, 1, -44], 5, 10, 12, 33, 1 => max 5`  
`3, 4, 5, [1, -44, 5], 10, 12, 33, 1 => max 5`  
`3, 4, 5, 1, [-44, 5, 10], 12, 33, 1 => max 10`  
`3, 4, 5, 1, -44, [5, 10, 12], 33, 1 => max 12`  
`3, 4, 5, 1, -44, 5, [10, 12, 33], 1 => max 33`  
`3, 4, 5, 1, -44, 5, 10, [12, 33, 1] => max 33`

2. Thực hiện theo các yêu cầu sau.

Viết function trả về một dictionary đếm số lượng chữ xuất hiện trong một từ, với key là chữ cái và value là số lần xuất hiện

- **Input:** một từ
- **Output:** dictionary đếm số lần các chữ xuất hiện
- **Note:** Giả sử các từ nhập vào đều có các chữ cái thuộc `[a-z]` hoặc `[A-Z]`

```
1 # Examples
2 string = 'Happiness'
3 count_chars(string)
4 >> {'H': 1, 'a': 1, 'e': 1, 'i': 1, 'n': 1, 'p': 2, 's': 2}
5
6 string = 'smiles'
7 count_chars(string)
8 >> {'e': 1, 'i': 1, 'l': 1, 'm': 1, 's': 2}
```

### 3. Thực hiện theo các yêu cầu sau.

Viết function đọc các câu trong một file txt, đếm số lượng các từ xuất hiện và trả về một dictionary với key là từ và value là số lần từ đó xuất hiện.

- **Input:** Đường dẫn đến file txt
- **Output:** dictionary đếm số lần các từ xuất hiện
- **Note:**
  - Giả sử các từ trong file txt đều có các chữ cái thuộc [a-z] hoặc [A-Z]
  - Không cần các thao tác xử lý string phức tạp **nhưng cần xử lý các từ đều là viết thường**
  - Các bạn dùng lệnh này để download  
!gdown <https://drive.google.com/uc?id=1IBScGdW2xlNsc9v5zSAya548kNgiOrko>

```
1 # Examples
2 !gdown https://drive.google.com/uc?id=1IBScGdW2xlNsc9v5zSAya548kNgiOrko
3 file_path = '/content/P1_data.txt'
4 word_count(file_path)
5 >>{'a': 7,
6    'again': 1,
7    'and': 1,
8    'are': 1,
9    'at': 1,
10   'be': 1,
11   'become': 2,
12   ...}
```

### 4. Khoảng cách Levenshtein.

Viết chương trình tính khoảng cách chỉnh sửa tối thiểu Levenshtein. Khoảng cách Levenshtein thể hiện khoảng cách khác biệt giữa 2 chuỗi ký tự. Khoảng cách Levenshtein giữa chuỗi S và chuỗi T là số bước ít nhất biến chuỗi S thành chuỗi T thông qua 3 phép biến đổi là:

- Xoá một ký tự
- Thêm một ký tự
- Thay thế ký tự này bằng ký tự khác

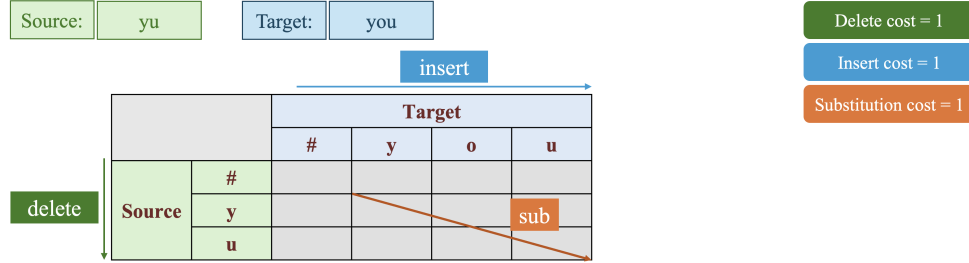
Khoảng cách này được sử dụng trong việc tính toán sự giống và khác nhau giữa 2 chuỗi, như chương trình kiểm tra lỗi chính tả của winword spellchecker. Ví dụ: Khoảng cách Levenshtein giữa 2 chuỗi "kitten" và "sitting" là 3, vì phải dùng ít nhất 3 lần biến đổi. Trong đó:

- kitten -> sitten (thay "k" bằng "s")
- sitten -> sittin (thay "e" bằng "i")
- sittin -> sitting (thêm ký tự "g")

Để hiểu rõ về thuật toán, chúng ta lấy ví dụ, khoảng cách cần tính giữa hai từ source: 'yu' và target: 'you'. Chi phí thực hiện cho các phép biến đổi bao gồm: xoá một ký tự, thêm một ký tự và thay thế ký tự này thành ký tự khác đều là 1 (Nếu 2 ký tự giống nhau thì chi phí thực hiện là 0).

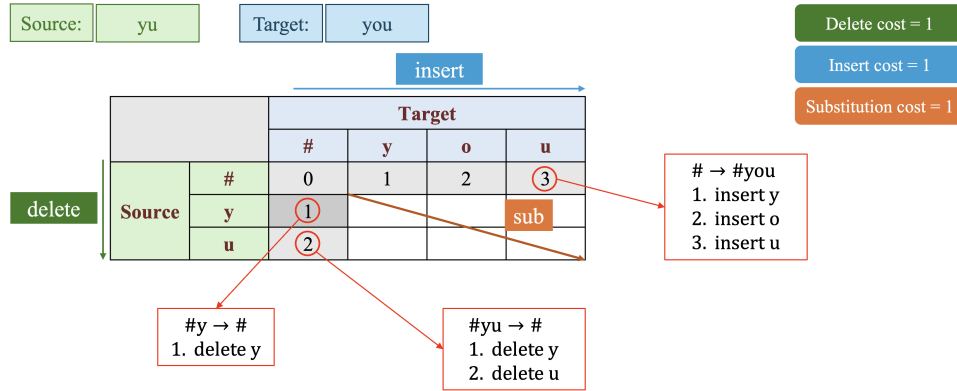
Các bước thực hiện như sau:

- Bước 1: Xây dựng ma trận lưu trữ có số hàng là  $M$  và số cột là  $N$ . Trong đó  $M$  là số lượng các ký tự trong từ source + 1,  $N$  là số lượng các ký tự trong từ target + 1. Vì vậy với ví dụ 'yu' và 'you', ta có ma trận được biểu diễn như hình 1. Ký hiệu '#' đại diện cho chuỗi rỗng. Gọi là ma trận  $D$ .



Hình 1: Khởi tạo ma trận  $D$

- Bước 2: Hoàn thiện hàng và cột đầu tiên. Với hàng đầu tiên, các giá trị đại diện cho chuỗi bắt đầu là chuỗi '#' và phép biến đổi là thêm (insert) từ chuỗi '#' thành '#y', '#yo', '#you' lần lượt là 0, 1, 2, 3 tương ứng với ô  $D[0, 0]$ ,  $D[0, 1]$ ,  $D[0, 2]$ ,  $D[0, 3]$ . Với cột đầu tiên, các giá trị đại diện cho chuỗi '#', '#y', '#yu' và phép biến đổi là xóa (delete) để thu được chuỗi '#' lần lượt là: 0, 1, 2 tương ứng với ô  $D[0, 0]$ ,  $D[1, 0]$ ,  $D[2, 0]$ . Ta được hình 2.

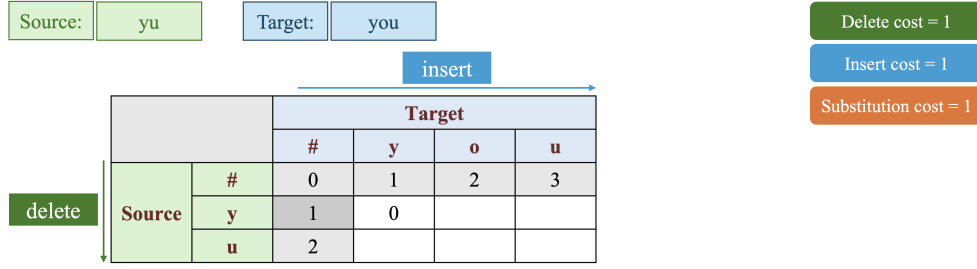


Hình 2: Số phép biến đổi cho hàng đầu tiên (thêm) và cột đầu tiên (xóa).

- Bước 3. Tính toán các giá trị với các ô còn lại trong ma trận. Bắt đầu từ  $D[1, 1]$  được tính dựa vào 3 ô phía trước là  $D[0, 1]$ ,  $D[1, 0]$ ,  $D[0, 0]$  như sau:

$$D[1, 1] = \min \begin{cases} D[0, 1] + del\_cost(source[1]) = 1 + 1 = 2 \\ D[1, 0] + ins\_cost(target[1]) = 1 + 1 = 2 \\ D[0, 0] + sub\_cost(source[1], target[1]) = 0 + 0 = 0 \end{cases} \quad (1)$$

Vì vậy  $D[1, 1] = 0$  ta được ma trận  $D$  như sau:

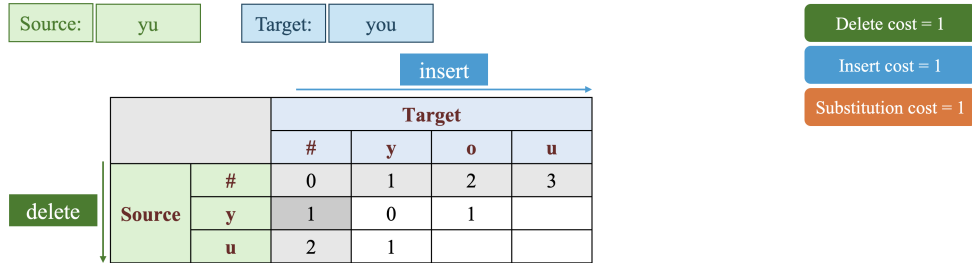
Hình 3: Giá trị tại  $D[1, 1]$ .

Tiếp theo chúng ta tính  $D[2, 1]$ ,  $D[1, 2]$ :

$$D[2, 1] = \begin{cases} D[1, 1] + del\_cost(source[2]) = 0 + 1 = 1 \\ D[2, 0] + ins\_cost(target[1]) = 2 + 1 = 3 \\ D[1, 0] + sub\_cost(source[2], target[1]) = 1 + 1 = 2 \end{cases} \quad (2)$$

$$D[1, 2] = \begin{cases} D[0, 2] + del\_cost(source[1]) = 2 + 1 = 3 \\ D[1, 1] + ins\_cost(target[2]) = 0 + 1 = 1 \\ D[0, 1] + sub\_cost(source[1], target[2]) = 1 + 1 = 2 \end{cases} \quad (3)$$

Vì vậy  $D[2, 1] = 1$ ,  $D[1, 2] = 1$  ta được ma trận  $D$  như sau:

Hình 4: Giá trị tại  $D[2, 1]$ ,  $D[1, 2]$ .

Cuối cùng, chúng ta tính  $D[1, 3]$ ,  $D[2, 2]$ ,  $D[2, 3]$ :

$$D[1, 3] = \begin{cases} D[0, 3] + del\_cost(source[1]) = 3 + 1 = 4 \\ D[1, 2] + ins\_cost(target[3]) = 1 + 1 = 2 \\ D[0, 2] + sub\_cost(source[1], target[3]) = 2 + 1 = 3 \end{cases} \quad (4)$$

$$D[2, 2] = \begin{cases} D[1, 2] + del\_cost(source[2]) = 1 + 1 = 2 \\ D[2, 1] + ins\_cost(target[2]) = 1 + 1 = 2 \\ D[1, 1] + sub\_cost(source[2], target[2]) = 0 + 1 = 1 \end{cases} \quad (5)$$

$$D[2, 3] = \begin{cases} D[1, 3] + del\_cost(source[2]) = 2 + 1 = 3 \\ D[2, 2] + ins\_cost(target[3]) = 1 + 1 = 2 \\ D[1, 2] + sub\_cost(source[2], target[3]) = 1 + 0 = 1 \end{cases} \quad (6)$$

Vì vậy  $D[1, 3] = 2, D[2, 2] = 1, D[2, 3] = 1$  ta được ma trận như sau:

Source:

yu

Target:

you

insert

delete

		Target			
		#	y	o	u
Source	#	0	1	2	3
	y	1	0	1	2
	u	2	1	1	1

Delete cost = 1

Insert cost = 1

Substitution cost = 1

Hình 5: Giá trị tại  $D[1, 3], D[2, 2], D[2, 3]$ .

- Bước 4: Sau khi hoàn thành ma trận, chúng ta đi tìm đường đi từ ô cuối cùng  $D[2, 3]$  có giá trị là 1. Vì vậy khoảng cách chỉnh sửa từ source: 'yu' sang thành target: 'you' là 1. Đầu tiên ký tự 'y' giữ nguyên sau đó thực hiện 1 phép thêm ký tự 'o' vào sau ký tự 'y' và cuối cùng ký tự 'u' được giữ nguyên. Minh họa các bước quay lui để tìm đường đi ngắn nhất tương ứng mũi tên vàng trong hình sau:

Source:

yu

Target:

you

insert

delete

		Target			
		#	y	o	u
Source	#	0	1	2	3
	y	1	0	1	2
	u	2	1	1	1

Delete cost = 1

Insert cost = 1

Substitution cost = 1

Hình 6: Quay lui, tìm các bước thực hiện chỉnh sửa từ source 'yu' sang target: 'you'.

## II. Câu hỏi trắc nghiệm

- Đọc tự luận trước để nắm được idea tổng quát (sẽ không yêu cầu nhưng khuyến khích các bạn tự làm tự luận) và các bài này sẽ được giải trong buổi TA.
- Các bạn phải làm phần trắc nghiệm
  - Các câu hỏi có ký hiệu **(Code)**: là câu hỏi yêu cầu các bạn phải trực tiếp code vào phần bị khuyết để có thể chọn được đáp án đúng
  - **Lưu ý**: Đối với dạng câu hỏi **(Code)** trong file hint luôn có 1 test casse bắt đầu với từ khóa assert nếu các bạn chạy không báo lỗi có nghĩa các bạn đã vượt qua được test case này và chạy lệnh tiếp theo để trả lời câu hỏi trắc nghiệm
  - **Lưu ý**: Đọc kỹ các code gợi ý và code ví dụ mẫu ở tự luận có thể sẽ có ích cho các bạn khi làm trắc nghiệm

**Câu hỏi 1:**(Code) Hoàn thành chương trình sau với mô tả bài toán từ câu I.1. Đầu ra của chương trình dưới đây là gì?

```

1 def max_kernel(num_list, k):
2     result = []
3
4     # Your Code Here
5
6     # End Code Here
7
8     return result
9
10 assert max_kernel([3, 4, 5, 1, -44], 3) == [5, 5, 5]
11 num_list = [3, 4, 5, 1, -44, 5, 10, 12, 33, 1]
12 k = 3
13 print(max_kernel(num_list, k))

```

- a) [5, 5, 5, 5, 10, 12, 33, 33]
- b) [2, 5, 3, 4, 1, 10, 3, 3]
- c) [0, 9, 5, 1, 0, 12, 3, 33]
- d) Raise an Error

**Câu hỏi 2:**(Code) Hoàn thành chương trình sau với mô tả bài toán từ câu I.2. Đầu ra của chương trình dưới đây là gì?

```

1 def character_count(word):
2     character_statistic = {}
3
4     # Your Code Here
5
6     # End Code Here
7     return character_statistic
8
9 assert character_count("Baby") == {'B': 1, 'a': 1, 'b': 1, 'y': 1}
10 print(character_count('smiles'))

```

- a) 's': 2, 'm': 1, 'i': 1, 'l': 1, 'e': 1
- b) 's': 0, 'm': 1, 'i': 1, 'l': 1, 'e': 8
- c) 's': 4, 'm': 1, 'i': 2, 'l': 1, 'e': 1
- d) Raise a Error

**Câu hỏi 3:**(Code) Hoàn thành chương trình sau với mô tả bài toán từ câu I.3. Đầu ra của chương trình dưới đây là gì?

```
1 !gdown https://drive.google.com/uc?id=1IBScGdW2x1Nsc9v5zSAya548kNgi0rko
2
3 def count_word(file_path):
4     counter = {}
5
6     # Your Code Here
7
8     # End Code Here
9
10    return counter
11 file_path = '/content/P1_data.txt'
12 result = count_word(file_path)
13 assert result['who'] == 3
14 print(result['man'])
```

- a) 4
- b) 5
- c) 6
- d) 9

**Câu hỏi 4:**(Code) Hoàn thành chương trình sau với mô tả bài toán từ câu I.4. Đầu ra của chương trình dưới đây là gì?

```
1 def levenshtein_distance(token1, token2):
2     # Your Code Here
3
4     # End Code Here
5
6     return distance
7
8 assert levenshtein_distance("hi", "hello") == 4.0
9 print(levenshtein_distance("hola", "hello"))
```

- a) 1.0
- b) 2.0
- c) 3.0
- d) 4.0

**Câu hỏi 5:**(Code) Hoàn thành chương trình sau. Đầu ra của chương trình dưới đây là gì?

```
1 def check_the_number(N):
2     list_of_numbers = []
3     result = ""
4     for i in range(1, 5):
5         #Your code here
6         #Su dung append them i vao trong list_of_number
7     if N in list_of_numbers:
8         results = "True"
9     if N not in list_of_numbers:
10        results = "False"
11    return results
12
13 N = 7
14 assert check_the_number(N) == "False"
15
```

```
16 N = 2
17 results = check_the_number(N)
18 print(results)
```

- a) True
- b) False
- c) None
- d) Raise an Error

**Câu hỏi 6:**(Code) Hãy hoàn thành chương trình dưới đây. Đầu ra của chương trình là gì?

```
1 def my_function(data, max, min):
2     result = []
3     for i in data:
4         #Your code here
5         #Neu i < min thi them min vao result
6         elif i > max:
7             result.append(max)
8         else:
9             result.append(i)
10    return result
11 my_list = [5, 2, 5, 0, 1]
12 max = 1
13 min = 0
14 assert my_function(max = max, min = min, data = my_list) == [1, 1, 1, 0, 1]
15 my_list = [10, 2, 5, 0, 1]
16 max = 2
17 min = 1
18 print(my_function(max = max, min = min, data = my_list))
```

- a) [10, 2, 5, 1, 1]
- b) [0, 2, 2, 0, 0]
- c) [2, 2, 2, 1, 1]
- d) Raise an Error



**Câu hỏi 7:(code)** Hãy hoàn thành chương trình dưới đây. Đầu ra của chương trình là gì?

```
1 def my_function(x, y):
2     #Your code here
3     #Su dung extend de noi y vao x
4     #return x
5
6 list_num1 = ['a', 2, 5]
7 list_num2 = [1, 1]
8 list_num3 = [0, 0]
9
10 assert my_function(list_num1, my_function(list_num2, list_num3)) == ['a', 2, 5, 1, 1,
    0, 0]
11
12 list_num1 = [1, 2]
13 list_num2 = [3, 4]
14 list_num3 = [0, 0]
15
16 print(my_function(list_num1, my_function(list_num2, list_num3)))
```

- a) [1, 2, 3, 4, 0, 0]
- b) [1, 2, [3, 4, 0, 0]]
- c) [[1, 2, 3, 4, 0, 0]]
- d) Raise an Error

**Câu hỏi 8:(code)** Hãy hoàn thành chương trình tìm phần tử có giá trị nhỏ nhất trong một list dưới đây. Đầu ra của chương trình là gì?

```
1 def my_function(n):
2     #Your code here
3
4 my_list = [1, 22, 93, -100]
5 assert my_function(my_list) == -100
6
7 my_list = [1, 2, 3, -1]
8 print(my_function(my_list))
```

- a) None
- b) Raise an Error
- c) -1
- d) 3

**Câu hỏi 9:(code)** Hãy hoàn thành chương trình tìm phần tử có giá trị lớn nhất trong một list dưới đây. Đầu ra của chương trình là gì?

```
1 def my_function(n):
2     #Your code here
3
4 my_list = [1001, 9, 100, 0]
5 assert my_function(my_list) == 1001
6
7 my_list = [1, 9, 9, 0]
8 print(my_function(my_list))
```

- a) None
- b) Raise an Error
- c) 0
- d) 9

**Câu hỏi 10:**(code) Hãy hoàn thành chương trình dưới đây. Đầu ra của chương trình là gì?

```
1 def My_function(integers, number = 1):
2     return any(#Your code here: Thực hiện duyệt từng phần tử trong integers, so sánh
3               #từng phần tử với number, nếu bằng nhau trả về True, khác nhau trả về False
4               #ví dụ: integers = [1, 2, 3], number = 2, bạn sẽ tạo ra list [False,
5               True, False] )
6
7 my_list = [1, 3, 9, 4]
8 assert My_function(my_list, -1) == False
9
10 my_list = [1, 2, 3, 4]
11 print(My_function(my_list, 2))
```

- a) 1
- b) 4
- c) True
- d) False

**Câu hỏi 11:**(code) Hãy hoàn thành chương trình dưới đây. Đầu ra của chương trình là gì?

```
1 def my_function(list_nums = [0, 1, 2]):
2     var = 0
3     for i in list_nums:
4         var += i
5     return #Your code here: Trả về giá trị trung bình của list bằng cách chia var cho
6         #số lượng phần tử trong list_nums
7
8 assert my_function([4, 6, 8]) == 6
9 print(my_function())
```

- a) 1.0
- b) 2.0
- c) Raise an Error
- d) A and C

**Câu hỏi 12:**(code) Hãy hoàn thành chương trình dưới đây. Đầu ra của chương trình dưới đây là gì?

```
1 def my_function(data):
2     var = []
3     for i in data:
4         #Your code here
5         #Nếu i chia hết cho 3 thì thêm i vào list var
6     return var
7
8 assert my_function([3, 9, 4, 5]) == [3, 9]
9 print(my_function([1, 2, 3, 5, 6]))
```

- a) [3, 6]
- b) [1, 2, 3, 5, 6]
- c) a and d
- d) [5, 1]

**Câu hỏi 13:**(code) Hãy hoàn thành chương trình sau đây thực hiện tính giai thừa của 1 số. Đầu ra của chương trình dưới đây là gì?

```
1 def my_function(y):
2     var = 1
```

```
3 while(y > 1):
4     #Your code here
5     return var
6 assert my_function(8) == 40320
7 print(my_function(4))
```

- a) 0
- b) 20
- c) 24
- d) Raise an Error

**Câu hỏi 14:**(code) Hãy hoàn thành chương trình đảo ngược chuỗi dưới đây. Đầu ra của chương trình là gì?

```
1 def my_function(x):
2     #your code here
3
4 x = 'I can do it'
5 assert my_function(x)=="ti od nac I"
6
7 x = 'apricot'
8 print(my_function(x))
```

- a) apricot
- b) tocirpa
- c) Raise a Error
- d) None

**Câu hỏi 15:**(code) Hãy hoàn thành chương trình dưới đây. Đầu ra của chương trình là gì?

```
1 def function_helper(x):
2     #Your code here
3     #Neu x>0 tra ve 'T', nguoc lai tra ve 'N'
4
5 def my_function(data):
6     res = [function_helper(x) for x in data]
7     return res
8
9 data = [10, 0, -10, -1]
10 assert my_function(data) == ['T', 'N', 'N', 'N']
11
12 data = [2, 3, 5, -1]
13 print(my_function(data))
```

- a) ['N', 'T', 'T', 'N']
- b) ['T', 'N', 'T', 'N']
- c) ['T', 'T', 'T', 'N']
- d) Raise an Error

**Câu hỏi 16:**(code) Hãy hoàn thành chương trình dưới đây để loại bỏ những phần tử trùng nhau. Đầu ra của chương trình là gì?

```
1 def function_helper(x, data):
2     for i in data:
3         #Your code here
4         #Neu x == i thi return 0
5     return 1
6
7 def my_function(data):
```

```
8     res = []
9     for i in data:
10         if function_helper(i, res):
11             res.append(i)
12
13     return res
14
15 lst = [10, 10, 9, 7, 7]
16 assert my_function(lst)==[10, 9, 7]
17
18 lst = [9, 9, 8, 1, 1]
19 print(my_function(lst))
```

- a) [9, 8, 1]
- b) [1, 1, 1]
- c) [9, 9, 8, 1, 1]
- d) Raise an Error