AI VIET NAM
@aivietnam.edu.vn

AI VIET NAM – AI COURSE 2025

Exercise: Word Suggestion

Quoc-Thai Nguyen, Quang-Vinh Dinh

I. Giới thiệu

Trong ngôn ngữ lập trình Python, có bốn kiểu cấu trúc dữ liệu cơ bản thường được sử dụng để lưu trữ và thao tác với tập hợp dữ liệu: **List**, **Tuple**, **Set**, và **Dictionary**. Mỗi loại có đặc điểm và cách sử dụng riêng, phù hợp với từng ngữ cảnh cụ thể trong lập trình.

List (Danh sách)

List là một dãy các phần tử có thứ tự và có thể thay đổi (mutable). Đây là một trong những kiểu dữ liệu linh hoạt và phổ biến nhất trong Python.

- Được khai báo bằng dấu ngoặc vuông `[]`.
- Các phần tử có thể thuộc nhiều kiểu dữ liệu khác nhau.
- Cho phép các phần tử trùng lặp.
- Hỗ trợ các thao tác như thêm, xoá, cập nhật, duyệt lặp.

Ví dụ sử dụng list:

```
fruits = ["apple", "banana", "cherry"]
print(fruits[1])       # In ra phần tử thứ 2: banana
fruits.append("mango") # Thêm phần tử vào cuối
fruits[0] = "grape"    # Thay đổi phần tử đầu tiên
print(fruits)
```

Tuple (Bộ giá trị bất biến)

Tuple là một dạng dãy phần tử giống như list, nhưng không thể thay đổi sau khi đã khởi tạo. Điều này giúp tuple hoạt động hiệu quả hơn và an toàn hơn trong một số tình huống.

- Được khai báo bằng dấu ngoặc tròn `()`.
- Không thể thay đổi nội dung sau khi tạo.
- Có thể chứa nhiều kiểu dữ liệu và trùng lặp phần tử.
- Thường dùng cho dữ liệu cố định như toạ độ, cấu hình.

Ví dụ sử dụng tuple:

```python
coordinates = (10, 20)
print(coordinates[0])  # In ra: 10
# coordinates[0] = 30  # Lỗi: không thể gán lại giá trị
```

Set (Tập hợp)

Set là một tập các phần tử không có thứ tự và không chứa phần tử trùng lặp. Đây là lựa chọn tốt khi muốn lưu trữ các giá trị duy nhất.

- Khai báo bằng dấu ngoặc nhọn `{}` hoặc hàm `set()`.
- Tự động loại bỏ phần tử trùng.
- Không thể truy cập bằng chỉ số.
- Hỗ trợ các phép toán tập hợp: hợp, giao, hiệu,...

Ví dụ sử dụng set:

```python
numbers = {1, 2, 3, 3, 4}
print(numbers)    # Kết quả: {1, 2, 3, 4}
numbers.add(5)    # Thêm phần tử mới
numbers.remove(2) # Xoá phần tử
```

Dictionary (Từ điển)

Dictionary là một cấu trúc ánh xạ giữa các **key** và **value**. Đây là kiểu dữ liệu rất mạnh trong Python khi cần truy cập nhanh theo khoá định danh.

- Khai báo bằng dấu ngoặc nhọn `{}` với các cặp `key: value`.
- Key là duy nhất và không thể thay đổi (immutable).
- Value có thể là bất kỳ kiểu dữ liệu nào.
- Hỗ trợ thêm, cập nhật, xoá, truy cập theo khoá.

Ví dụ sử dụng dictionary:

```python
person = {
    "name": "Alice",
    "age": 25,
    "city": "Hanoi"
}
print(person["name"])     # Truy cập theo khoá
person["age"] = 26        # Cập nhật giá trị
person["email"] = "a@gmail.com" # Thêm mới
```

Tóm tắt:

- **List**: Có thứ tự, thay đổi được, cho phép trùng lặp.
- **Tuple**: Có thứ tự, không thay đổi, cho phép trùng lặp.
- **Set**: Không thứ tự, không trùng lặp, không chỉ mục.
- **Dictionary**: Ánh xạ khóa – giá trị, key duy nhất.

Các cấu trúc dữ liệu này là nền tảng quan trọng giúp lập trình hiệu quả với dữ liệu trong Python. Việc hiểu rõ đặc điểm và cách sử dụng của từng kiểu sẽ giúp bạn thiết kế thuật toán tối ưu và dễ bảo trì hơn.

Nội dung bài tập sẽ tập trung vào làm việc với một số cấu trúc dữ liệu và giải thuật đơn giản.

Mục lục

II. Câu hỏi tự luận

II.1. Getting Max Over Kernel

Tìm giá trị lớn nhất trong cửa sổ trượt trên danh sách cho trước

Bài toán 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 đươc 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

II.2. Character Counting

Đếm tần suất xuất hiện của các ký tự

Bài toán Viết hàm 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]

```
string = "Happiness"
count_character(string)
>> {"H": 1, "a": 1, "e": 1, "i": 1, "n": 1, "p": 2, "s": 2}

string = "smiles"
count_character(string)
>> {"e": 1, "i": 1, "l": 1, "m": 1, "s": 2}
```

II.3. Word Counting

Đếm tần suất xuất hiện của các từ trong đoạn văn bản

Bài toán Viết hàm 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

```
# Tải file từ Google Drive
!gdown https://drive.google.com/uc?id=1IBScGdW2xlNsc9v5zSAya548kNgiOrko

# Đường dẫn file
file_path = "./P1_data.txt"

# Gọi hàm xử lý
count_word(file_path)

>> {
    "a": 7,
    "again": 1,
    "and": 1,
    "are": 1,
    "at": 1,
    "be": 1,
    "become": 2,
    ...
}
```

II.4. Levenshtein Distance

Khoảng cách chỉnh sửa văn bản Levenshtein

Bài toán: 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

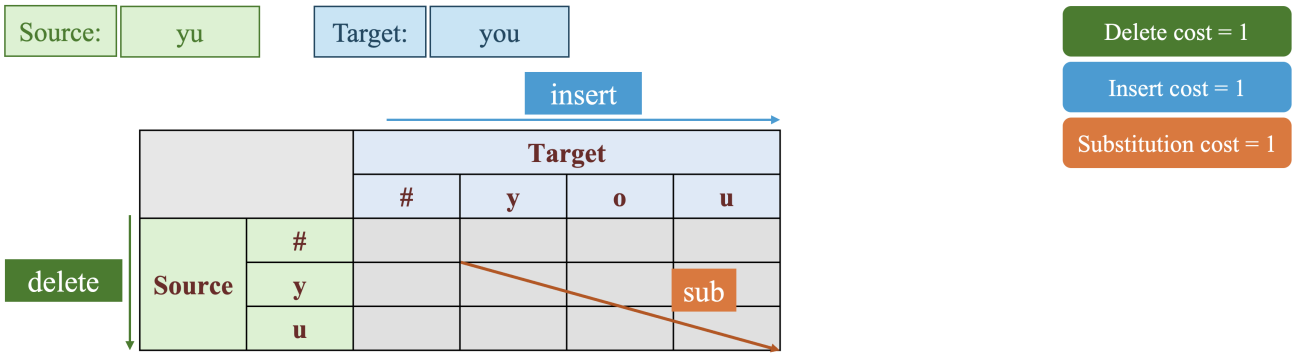
Khoảng cách này được sử dụng trong việc tính toán sự giống và khác nhau giữa 2 chuỗi, như chương trình kiểm tra lỗi chính tả của winword spellchecker. Ví dụ: Khoảng cách Levenshtein giữa 2 chuỗi "kitten" và "sitting" là 3, vì phải dùng ít nhất 3 lần biến đổi. Trong đó:

- 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:

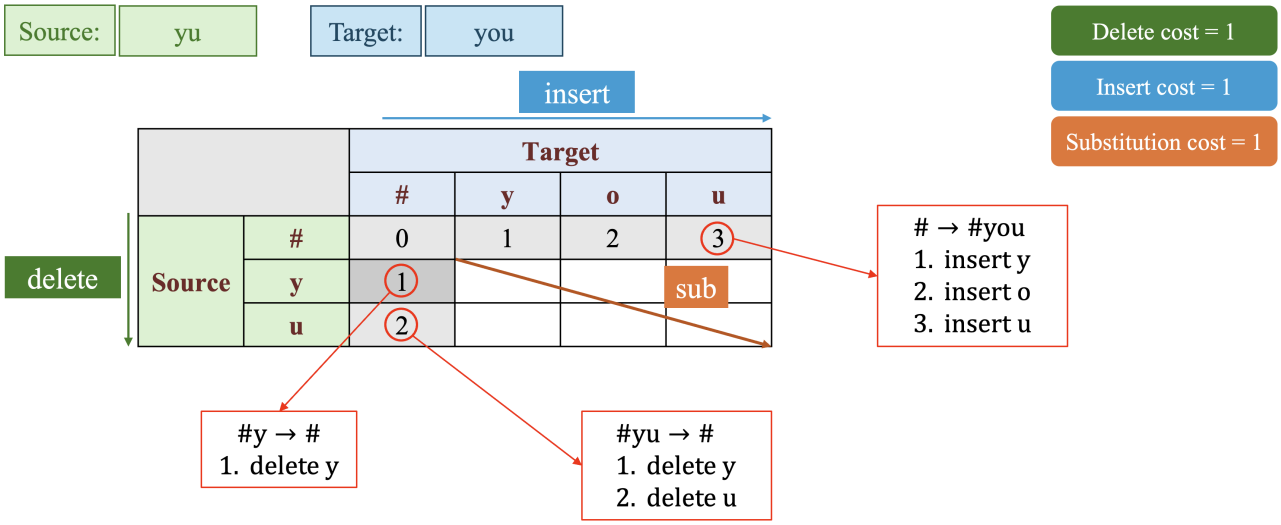
- Bước 1: Xây dựng ma trận lưu trữ có số hàng là M và số cột là N. Trong đó M là số lượng các ký tự trong từ source + 1, N là số lượng các ký tự trong từ target + 1. Vì vậy với ví dụ "yu" và "you", ta có ma trận được biểu diễn như hình 1. Ký hiệu "#" đại diện cho chuỗi rỗng. Gọi là ma trận D.



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 xoá (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.

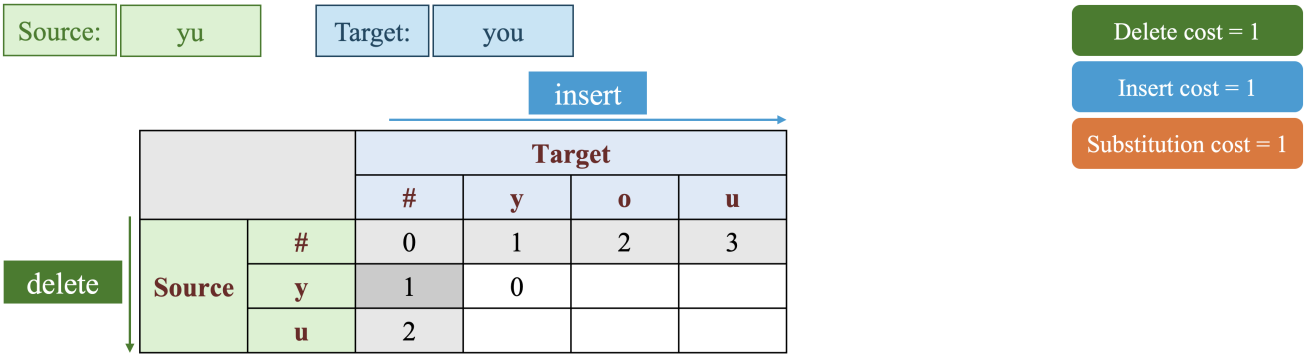


Hình 2: Số phép biến đổi cho hàng đầu tiên (thêm) và cột đầu tiên (xoá).

- 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:



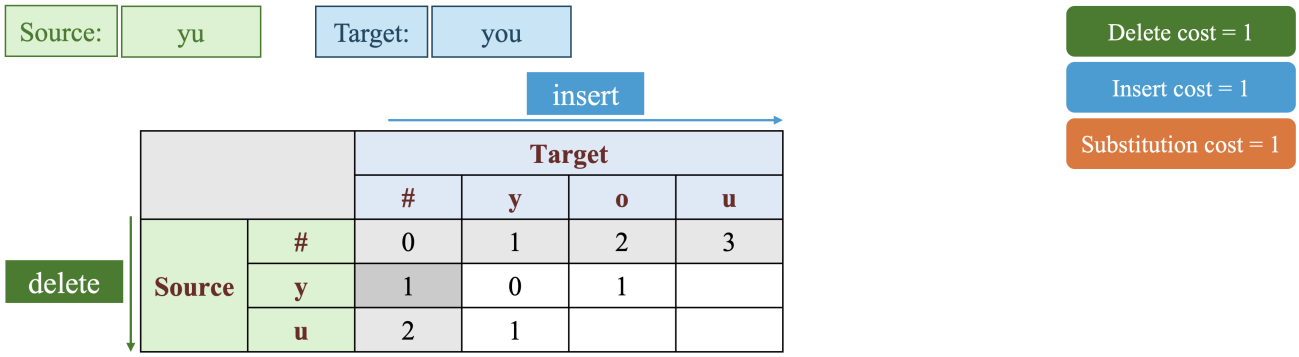
Hình 3: Giá trị tại $D[1,1]$.

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

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

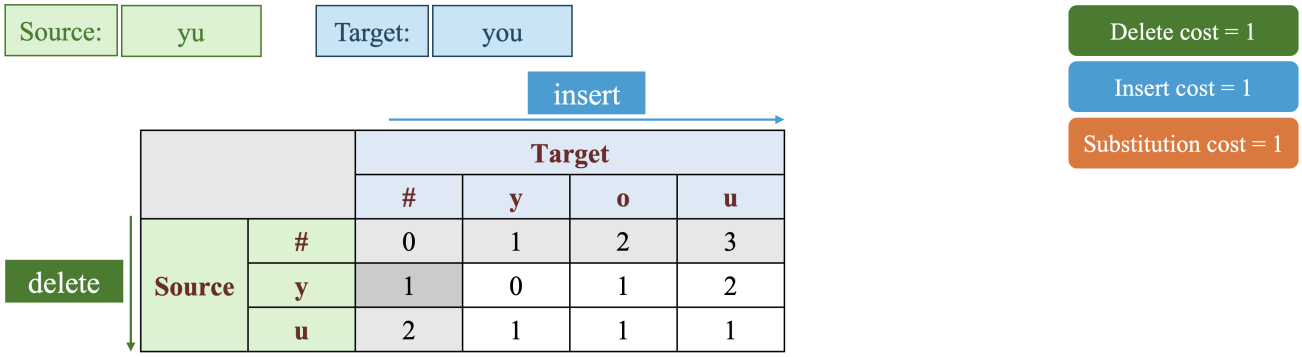
$$D[1,2] = \min \begin{cases} D[0,2] + del\_cost(source[1]) = 2 + 1 = 3 \\ D[1,1] + ins\_cost(target[2]) = 0 + 1 = 1 \\ D[0,1] + sub\_cost(source[1], target[2]) = 1 + 1 = 2 \end{cases} \quad (3)$$

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 $D[1,3], D[2,2], D[2,3]$:



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

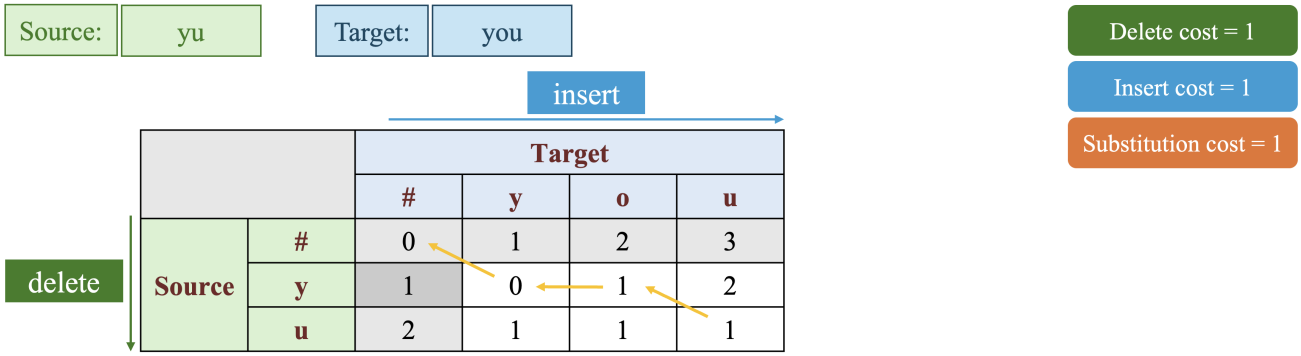
$$D[1,3] = \min \begin{cases} D[0,3] + del\_cost(source[1]) = 3 + 1 = 4 \\ D[1,2] + ins\_cost(target[3]) = 1 + 1 = 2 \\ D[0,2] + sub\_cost(source[1], target[3]) = 2 + 1 = 3 \end{cases} \quad (4)$$

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

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

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

- 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 hoạ các bước quay lui để tìm đường đi ngắn nhất tương ứng mũi tên vàng trong hình sau:



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

III. Câu hỏi trắc nghiệm

Phần trắc nghiệm dựa trên các nội dung của câu hỏi tự luận, vì vậy học viên cần xem kỹ các yêu cầu mô tả, hoặc code ví dụ trong phần tự luận II. để hoàn thiện tốt nhất.

1. Hoàn thành chương trình sau với mô tả bài toán tìm giá trị lớn nhất trong cửa sổ trượt II.1. và cho biết đầu ra của chương trình dưới đây là gì?

```python
def max_kernel(num_list, k):
    """
    Trả về danh sách các giá trị lớn nhất trong mỗi cửa sổ con (window) kích thước k
    chạy trượt trên danh sách num_list.

    Parameters:
        num_list (list of numbers): Danh sách các số đầu vào.
        k (int): Kích thước của cửa sổ trượt.

    Returns:
        list: Danh sách các giá trị lớn nhất trong từng cửa sổ con liên tiếp.
    """
    result = []
    # Your Code Here

    # End Code Here

    return result

assert max_kernel([3 , 4 , 5 , 1 , -44], 3) == [5, 5, 5]
num_list = [3, 4, 5, 1, -44 , 5 ,10, 12 ,33, 1]
k = 3
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

2. Hoàn thành chương trình sau với mô tả bài toán đếm tần suất xuất hiện các ký tự II.2. và cho biết đầu ra của chương trình dưới đây là gì?

```python
def count_character(word):
    """
    Đếm số lần xuất hiện của từng ký tự trong chuỗi đầu vào.

```

```
        Parameters:
            word (str): Chuỗi ký tự cần thống kê.

        Returns:
            dict: Từ điển với mỗi ký tự là một khóa, giá trị là số lần xuất hiện.
        """
        character_statistic = {}

        # Your Code Here

        # End Code Here
        return character_statistic

    assert count_character("Baby") == {"B": 1, "a": 1, "b": 1, "y": 1}
    print(count_character("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 an Error

3. Hoàn thành chương trình sau với mô tả bài toán đếm tần suất xuất hiện các từ II.3. và cho biết đầu ra của chương trình dưới đây là gì?

```
!gdown https://drive.google.com/uc?id=1IBScGdW2xlNsc9v5zSAya548kNgiOrko

def count_word(file_path):
    """
    Đếm số lần xuất hiện của từng từ trong văn bản đầu vào sau khi tiền xử lý.

    Quá trình tiền xử lý gồm:
    - Chuyển văn bản thành chữ thường
    - Loại bỏ dấu chấm (.) và dấu phẩy (,)
    - Tách văn bản thành danh sách từ

    Parameters:
        file_path (str): Đường dẫn đến file.

    Returns:
        dict: Từ điển đếm số lần xuất hiện của từng từ.
    """
    counter = {}

    # Your Code Here

    # End Code Here

    return counter
file_path = "./P1_data.txt"
```

```
result = count_word(file_path)
assert result["who"] == 3
print(result["man"])
```

(a) 4

(b) 5

(c) 6

(d) 9

4. Hoàn thành chương trình sau với mô tả bài toán tính khoảng cách Levenshtein II.4. và cho biết đầu ra của chương trình dưới đây là gì?

```
def levenshtein_distance(token1, token2):
    """
    Tính khoảng cách Levenshtein (edit distance) giữa hai chuỗi.

    Khoảng cách Levenshtein là số lần chỉnh sửa nhỏ nhất (chèn, xóa, thay thế)
    để biến chuỗi token1 thành token2.

    Parameters:
        token1 (str): Chuỗi thứ nhất.
        token2 (str): Chuỗi thứ hai.

    Returns:
        int: Khoảng cách Levenshtein giữa hai chuỗi.
    """
    # Your Code Here

    # End Code Here

    return distance

assert levenshtein_distance("hi", "hello") == 4
print(levenshtein_distance("hola", "hello"))
```

(a) 1

(b) 2

(c) 3

(d) 4

5. Hoàn thành chương trình sau. Đầu ra của chương trình dưới đây là gì?

```
def check_the_number(N):
  list_of_numbers = []
  result = ""
  for i in range(1, 5):
    #Your code here
    # Su dung append them i vao trong list_of_number
  if N in list_of_numbers:
    results = "True"
  if N not in list_of_numbers:
    results = "False"
  return results

N = 7
assert check_the_number(N) == "False"

N = 2
results = check_the_number(N)
print(results)

```

(a) True

(b) False

(c) None

(d) Raise an Error

6. Hoàn thành chương trình sau. Đầu ra của chương trình dưới đây là gì?

```
def my_function(data, max_value, min_value):
    result = []
    for i in data:
        # Your code here
        # Neu i < min_value thi them min vao result
        elif i > max_value:
            result.append(max_value)
        else:
            result.append(i)
    return result

# test
my_list = [5, 2, 5, 0, 1]
max_value = 1
min_value = 0
assert my_function(my_list, max_value, min_value) == [1, 1, 1, 0, 1]
my_list = [10, 2, 5, 0, 1]
max = 2
min = 1
print(my_function(my_list, max_value, min_value))

```

(a) $[10, 2, 5, 1, 1]$

(b) $[0, 2, 2, 0, 0]$

(c) $[2, 2, 2, 1, 1]$

(d) Raise an Error

7. Hoàn thành chương trình sau. Đầu ra của chương trình dưới đây là gì?

```
def my_function(n):
    #Your code here

my_list = [1, 22, 93, -100]
assert my_function(my_list) == -100

my_list = [1, 2, 3, -1]
print(my_function(my_list))

```

(a) None

(b) Raise an Error

(c) -1

(d) 3

8. Hoàn thành chương trình sau. Đầu ra của chương trình dưới đây là gì?

```
def my_function(n):
    #Your code here

my_list = [1001, 9, 100, 0]
assert my_function(my_list) == 1001

my_list = [1, 9, 9, 0]
print(my_function(my_list))

```

(a) None

(b) Raise an Error

(c) 0

(d) 9

9. Hoàn thành chương trình sau. Đầu ra của chương trình dưới đây là gì?

```
    def my_function(integers, number = 1):
        return any(#Your code here: Thuc hien duyet tung phan tu trong integers,
                   #so sanh tung phan tu voi number, neu bang nhau tra ve True,
                   #khac nhau tra ve False
                   #vi du: integers = [1, 2, 3], number = 2,
                   #ban se tao ra list [False, True, False])

    my_list = [1, 3, 9, 4]
    assert my_function(my_list, -1) == False

    my_list = [1, 2, 3, 4]
    print(my_function(my_list, 2))

```

(a) 1

(b) 4

(c) True

(d) False

10. Hoàn thành chương trình sau. Đầu ra của chương trình dưới đây là gì?

```
    def my_function(list_nums = [0, 1, 2]):
        var = 0
        for i in list_nums:
            var += i
        return #Your code here: Tra ve gia tri trung binh cua list
               #bang cach chia var cho so luong phan tu trong list_nums

    assert my_function([4, 6, 8]) == 6
    print(my_function())

```

(a) 1.0

(b) 2.0

(c) Raise an Error

(d) A and C

11. Hoàn thành chương trình sau. Đầu ra của chương trình dưới đây là gì?

```
    def my_function(data):
      var = []
      for i in data:
        #Your code here
```

```
        #Neu i chia het cho 3 thi them i vao list var
      return var

    assert my_function([3, 9, 4, 5]) == [3, 9]
    print(my_function([1, 2, 3, 5, 6]))

```

(a) $[3, 6]$

(b) $[1, 2, 3, 5, 6]$

(c) $[1, 3, 6]$

(d) $[5, 1]$

12. Hoàn thành chương trình sau đây thực hiện tính giai thừa của 1 số. Đầu ra của chương trình dưới đây là gì?

```
    def my_function(y):
      var = 1
      while(y > 1):
        #Your code here
      return var

    assert my_function(8) == 40320
    print(my_function(4))

```

(a) 0

(b) 20

(c) 24

(d) Raise an Error

13. Hoàn thành chương trình đảo ngược chuỗi dưới đây. Đầu ra của chương trình là gì?

```
    def my_function(x):
      #your code here

    x = "I can do it"
    assert my_function(x) == "ti od nac I"

    x = "apricot"
    print(my_function(x))

```

(a) apricot

(b) tocirpa

(c) Raise an Error

(d) None

14. Hoàn thành chương trình dưới đây. Đầu ra của chương trình là gì?

```
def function_helper(x):
  #Your code here
  #Neu x > 0 tra ve "T", nguoc lai tra ve "N"

def my_function(data):
  res = [function_helper(x) for x in data]
  return res

data = [10, 0, -10, -1]
assert my_function(data) == ["T", "N", "N", "N"]

data = [2, 3, 5, -1]
print(my_function(data))

```

(a) ["N", "T", "T", "N"]

(b) ["T", "N", "T", "N"]

(c) ["T", "T", "T", "N"]

(d) Raise an Error

15. Hoàn thành chương trình dưới đây để loại bỏ những phần tử trùng nhau. Đầu ra của chương trình là gì?

```
def function_helper(x, data):
  for i in data:
    #Your code here
    #Neu x == i thi return 0
  return 1

def my_function(data):
  res = []
  for i in data:
    if function_helper(i, res):
      res.append(i)
  return res

lst = [10, 10, 9, 7, 7]
assert my_function(lst) == [10, 9, 7]

lst = [9, 9, 8, 1, 1]
print(my_function(lst))

```

(a) $[9, 8, 1]$

(b) $[1, 1, 1]$

(c) $[9, 9, 8, 1, 1]$

(d) Raise an Error

Phụ lục

1. **Hint:** Các file code gợi ý có thể được tải tại đây.
2. **Solution:** Các file code cài đặt hoàn chỉnh và phần trả lời nội dung trắc nghiệm có thể được tải tại đây (Lưu ý: Sáng thứ 3 khi hết deadline phần project, ad mới copy các nội dung bài giải nêu trên vào đường dẫn).
3. **Rubric:**

Word Suggestion - Rubric		
Câu	**Kiến Thức**	**Đánh Giá**
1	- Sử dụng dictionary data để đếm số lần chữ xuất hiện trong 1 từ - Sử dụng dictionary data để đếm số lần từ xuất hiện trong 1 các câu của 1 file - Việc đếm số từ và chữ là 1 trong những bước cơ bản để học NLP - Đọc file và lấy từng line trong file - Xử lý string cơ bản viết thường, tách từ - Sử dụng dictionary kiểm tra key có tồn tại hay không, và cách cập nhật value của 1 key khi đã tồn tại trong dictionarry	- Bước đầu biết kết hợp xử lý string và dictionary để lưu data trong dictionary cho task NLP - Biết thao tác với dictionary cơ bản và quan trọng nhất, kiểm tra key và cập nhật value khi key tồn tại - Biết thao tác với file và lấy nội dung từng line trong file - Làm quen với xử lý string tách từ, và convert sang viết thường
2	- Sử dụng vòng lặp for để duyệt qua list. - Sử dụng một số phương thức của List trong Python: len(), append(). - Sử dụng hàm có sẵn max() trong Python để tìm phần tử lớn nhất. - Kĩ thuật slicing trong List. - Lý thuyết cơ bản về sliding window	- Vận dụng được các hàm, phương thức có sẵn liên quan đến List trong Python để giải quyết bài toán.
3	- Sử dụng các từ khóa điều kiện in, not trong Python để tạo câu điều kiện với List.	Biết sử dụng các toán tử điều kiện trong Python để tương tác với List.
4	- Kĩ thuật list comprehension. Sử dụng các câu điều kiện để tạo điều kiện ràng buộc cho người dùng.	- Vận dụng kĩ thuật list comprehension hoặc vòng lặp for để tạo một List mới dựa trên List ban đầu.