AI VIETNAM
@aivietnam.edu.vn

Module 01 - Exercise

Word Suggestion

Code

Nguyen Quoc Thai
MSc in Computer Science

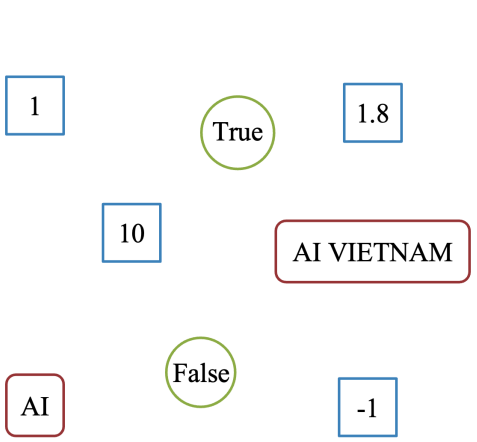
Objectives

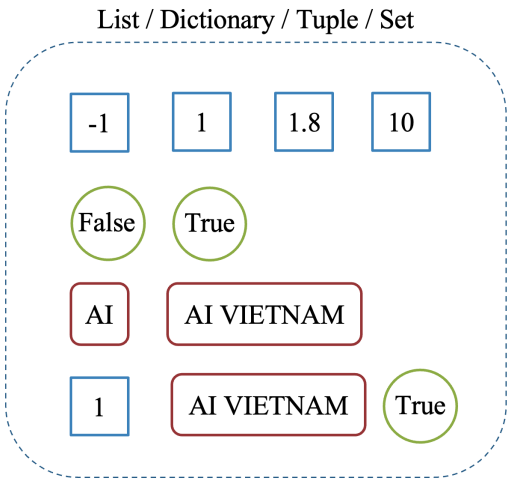
Data Structure in Python

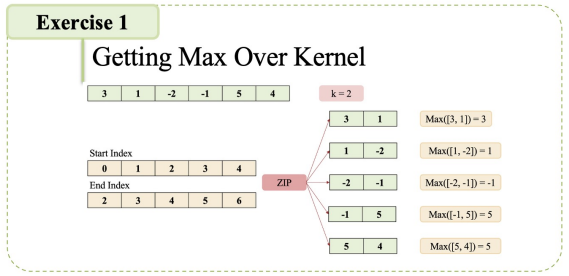
- List
- Dictionary
- Tuple
- Set

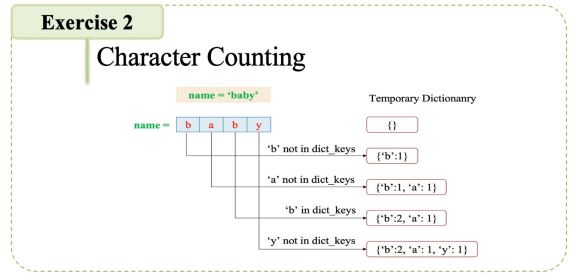
Practice

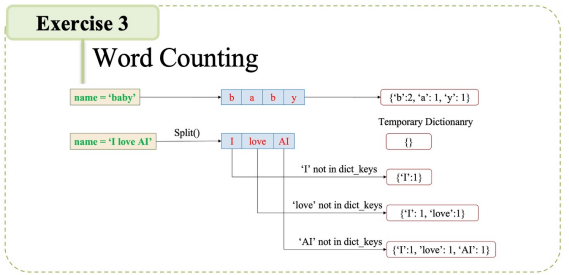
- Getting Max Over Kernel
- Character Counting
- Word Counting
- Levenshtein Distance

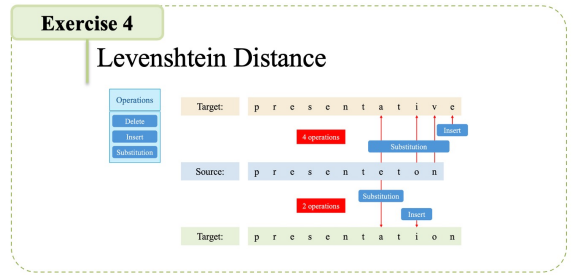








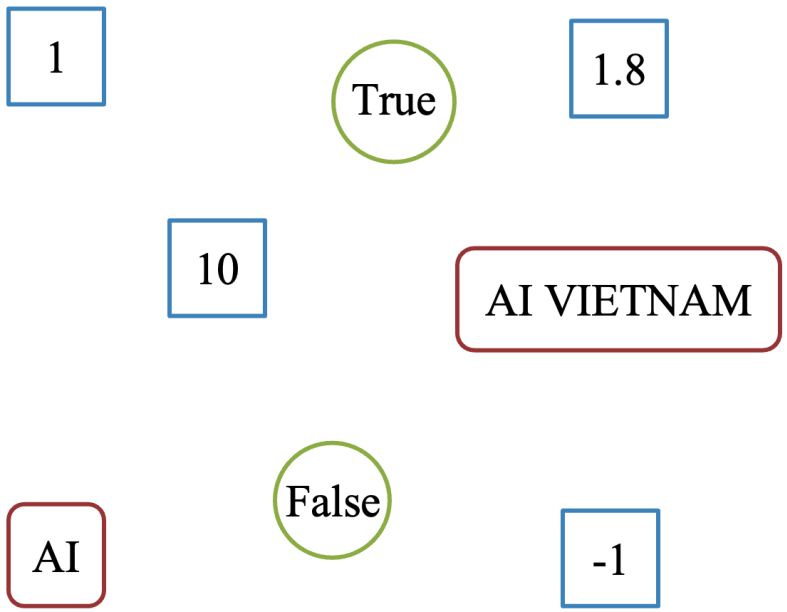




Outline

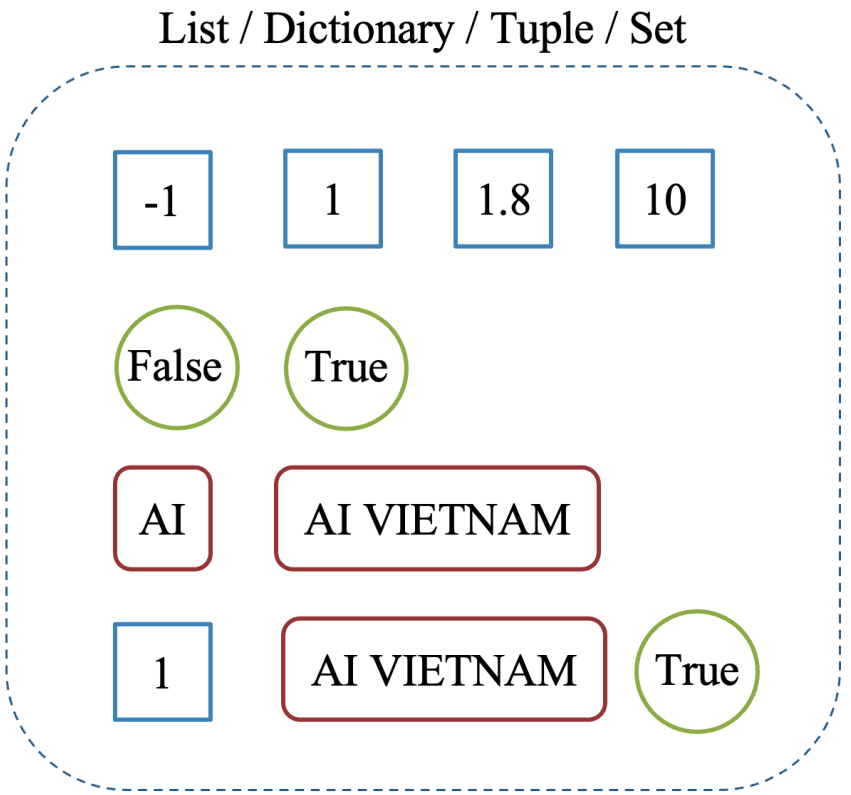
SECTION 1

Data Structure



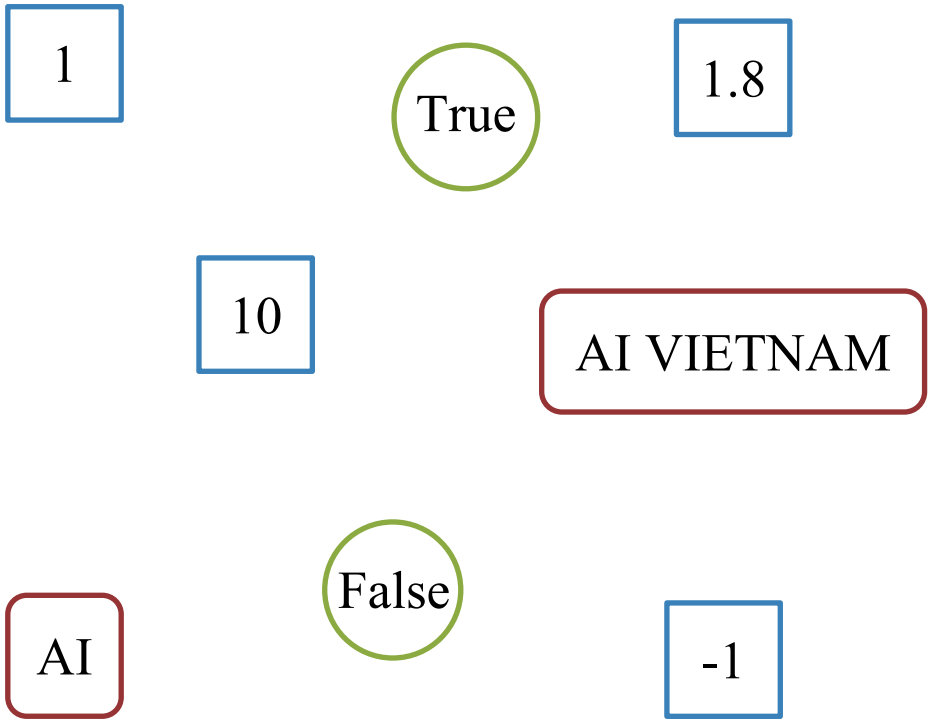
SECTION 2

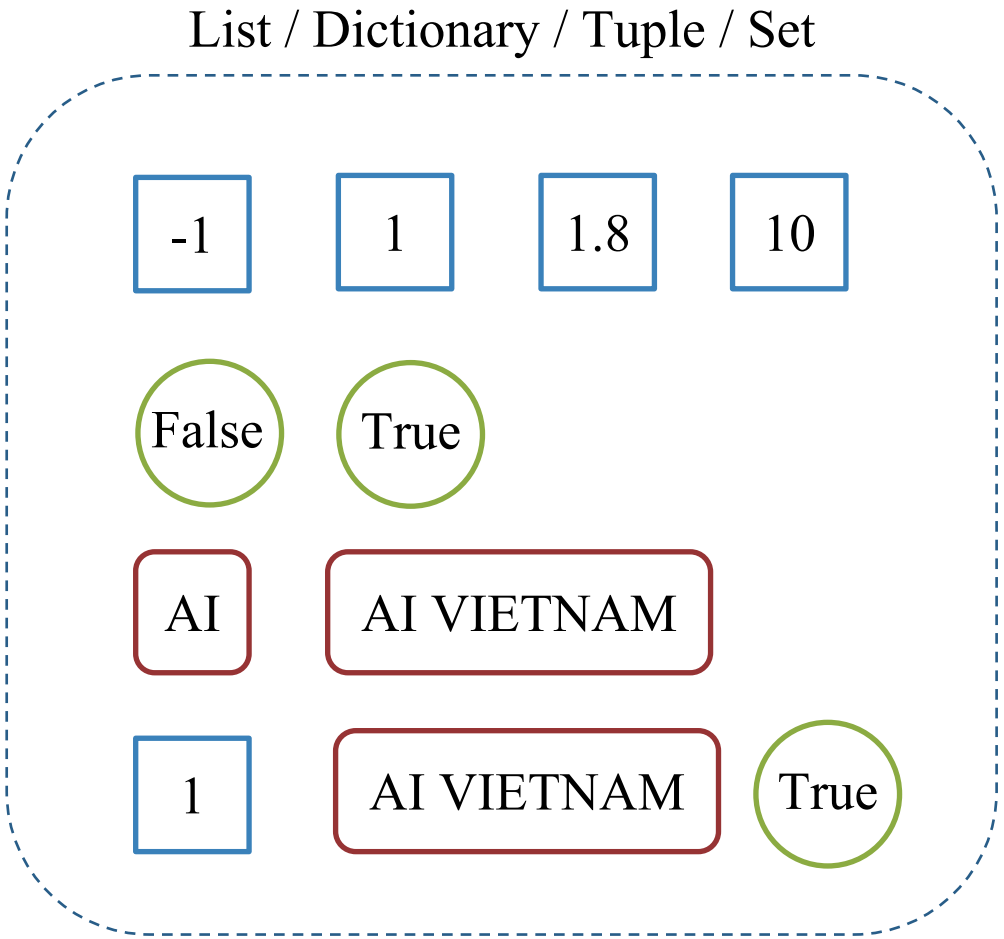
Practice



Data Structure

Data Structures





Data Structure

List

- Lists allow to store multiple items in a single variable
- List Characteristics
  - **Ordered**: Maintain the order of elements
  - **Mutable**: Items can be changed after creation
  - **Allow duplicates**: Lists can contain duplicate values

[	"Python"	"C"	"C++"	"Java"	]
Index	0	1	2	3	
Negative Index	-4	-3	-2	-1	

Data Structure

List

```
1  # Create a list
2  nums = [12, 23, 60, 7, 7]
3  print(nums)
4
5  # Acesses elements
6  print(nums[0])
7  print(nums[1:3])
```

```
[12, 23, 60, 7, 7]
12
[23, 60]
```

```
1  # Update elements
2  nums = [12, 23, 60]
3  print(nums)
4  nums[0] = 20
5  print(nums)
6
7  for item in nums:
8      print(item)
```

✓ 0.0s

```
[12, 23, 60]
[20, 23, 60]
20
23
60
```

Data Structure

Useful List Methods

- **append():** Adds an item to the end of the list
- **extend():** Adds items of lists and other iterables to the end of the list
- **insert()**: Inserts an item at the specified index
- **remove():** Removes the specified value
- **clear**(): Removes all items from the list
- **index**(): Returns the index of the first matched item
- **count**(): Returns the count of the specified item in the list
- **sort**(): Sorts the list in ASC/DES order
- **reverse**(): Reverses the item of the list
- **copy**(): Returns the shallow copy of the list

```
1  # Update elements
2  nums = [12, 23, 60]
3  print(nums)
4  nums.append(50)
5  print(nums)
6  nums.extend([1, -1])
7  print(nums)
8  nums.remove(12)
9  print(nums)
10 nums.clear()
11 print(nums)
```

✓ 0.0s

```
[12, 23, 60]
[12, 23, 60, 50]
[12, 23, 60, 50, 1, -1]
[23, 60, 50, 1, -1]
[]
```

Data Structure

Tuple

- A tuple is a collection similar to a Python list, but cannot modify a tuple once it is created
- Tuple Characteristics
  - **Ordered**: Maintain the order of elements
  - **Immutable**: Cannot be changed after creation
  - **Allow duplicates**: Tuples can contain duplicate values

(	"Python"	"C"	"C++"	"Java"	)
Index	0	1	2	3	
Negative Index	-4	-3	-2	-1	

Data Structure

Tuple

```
1  # Create a tuple
2  t = (1, 3, -2.0, "Hello")
3  print(t)
4
5  # Access elements
6  print(t[0])
7  print(t[-1])
8  print(t[2:4])
```

✓ 0.0s

```
(1, 3, -2.0, 'Hello')
1
Hello
(-2.0, 'Hello')
```

```
1  # Iterate through a tuple
2  t = (1, -2.0, "Hello")
3  for item in t:
4      print(item)
5
6  # Modify
7  t[2] = "Hi"
```

⊗ 0.1s

```
1
-2.0
Hello

---------------------------------------
TypeError
Cell In[9], line 7
      4     print(item)
      6 # Modify
----> 7 t[2] = "Hi"
```

Data Structure

Set

- A set is a collection of unique data, meaning that elements within a set cannot be duplicated
- Set Characteristics
  - **Unordered**: Not maintain the order of elements
  - **Mutable**: Can be changed after creation
  - **Not allow duplicates**: Sets cannot contain duplicate values

{	"Python"	"C"	"C++"	"Java"	}

```python
# create a set
s = {1, 2, 3, 1}
print(s)

# Add an element
s.add("Hello")
print(s)

for item in s:
    print(item)
```

✓ 0.0s

```
{1, 2, 3}
{'Hello', 1, 2, 3}
Hello
1
2
3
```

Data Structure

Useful Set Methods

- **add():** Adds an item to the set
- **clear**(): Removes all items from the set
- **copy**(): Returns the shallow copy of the set
- **discard**(): Removes an element from the set
- **remove**(): Removes an element from the set
- **update**(): Updates the set
- **pop**(): Removes and returns an arbitrary set element
- **intersection**(): Returns the intersection of two sets
- **union**(): Returns the union of two sets
- **issubset**(): Check if another set contains this set
- **issuperset**(): Check if this set contains another set

```
1  s = {1, 2, 3, 1}
2  print(s)
3  s.add("Hello")
4  print(s)
5  s.update({1})
6  print(s)
7  print(s.issuperset({2,3}))
8  s.remove(2)
9  print(s)
10 s.discard(3)
11 print(s)
12 s.clear()
13 print(s)
```

✓ 0.0s

```
{1, 2, 3}
{'Hello', 1, 2, 3}
{1, 'Hello', 2, 3}
False
True
{1, 'Hello', 3}
{1, 'Hello'}
set()
```

Data Structure

Dictionary

- A dictionary is a collection of items, each item is a key-value pair
- Dictionary Characteristics
  - **Ordered**: Maintain the order of elements (>= Python 3.7)
  - **Mutable**: Can be changed after creation (**Key**: **Immutable**)
  - **Not allow duplicates**: Keys are unique, and values can be any data type

```
class_infor = {
    "name": "python",
    "num_students": 20
}
```

Element 1

Element 2

Key Value

Data Structure

Dictionary

```python
class_infor = {"name": "Python",
    "num_student": 20}
print(class_infor)
print(class_infor["name"])
class_infor["topic"] = "Dictionary"
print(class_infor)
class_infor["num_student"] = 30
print(class_infor)

for item in class_infor:
    print(item)
```

✓ 0.0s

```
{'name': 'Python', 'num_student': 20}
Python
{'name': 'Python', 'num_student': 20, 'topic': 'Dictionary'}
{'name': 'Python', 'num_student': 30, 'topic': 'Dictionary'}
name
num_student
topic
```

```python
for item in class_infor.keys():
    print(item)

for item in class_infor.values():
    print(item)

for item in class_infor.items():
    print(item)
```

✓ 0.0s

```
name
num_student
topic
Python
30
Dictionary
('name', 'Python')
('num_student', 30)
('topic', 'Dictionary')
```

Data Structure

Useful Dictionary Methods

- **pop():** Removes the item with the specified key
- **update**(): Adds or changes dictionary item
- **clear**(): Remove all the items
- **keys**(): Returns all the dictionary's keys
- **values**(): Returns all the dictionary's values
- **get**(): Returns the value of the specified key
- **popitem**(): Returns the last inserted key-value
- **copy**(): Returns a copy of the dictionary

```
1  person = {"name": "An"}
2  print(person)
3  print(person.get("name"))
4  print(person.get("age", 20))
5  person.update({"class": "Python"})
6  print(person)
7  print(person.pop("class"))
8  print(person)
9  print("name" in person)
```

✓ 0.0s

```
{'name': 'An'}
An
20
{'name': 'An', 'class': 'Python'}
Python
{'name': 'An'}
True
```

Outline

SECTION 1

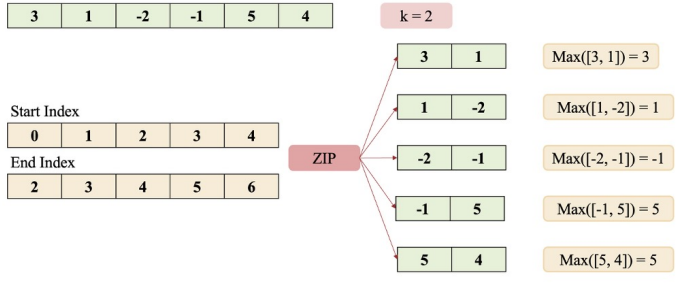
Data Structure

SECTION 2

Practice

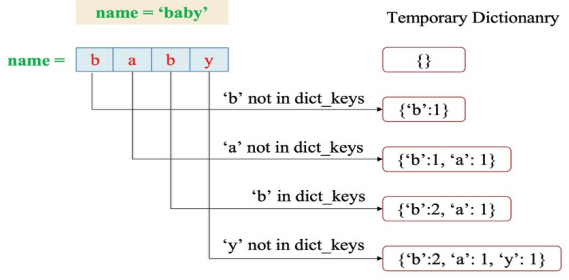
Exercise 1

Getting Max Over Kernel



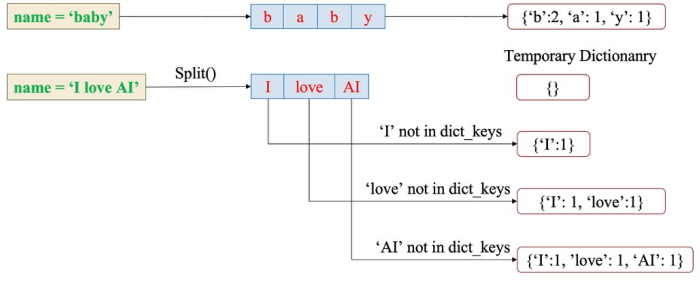
Exercise 2

Character Counting



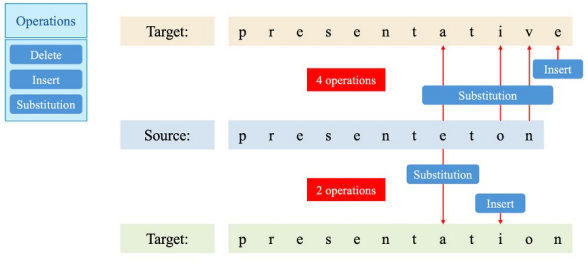
Exercise 3

Word Counting



Exercise 4

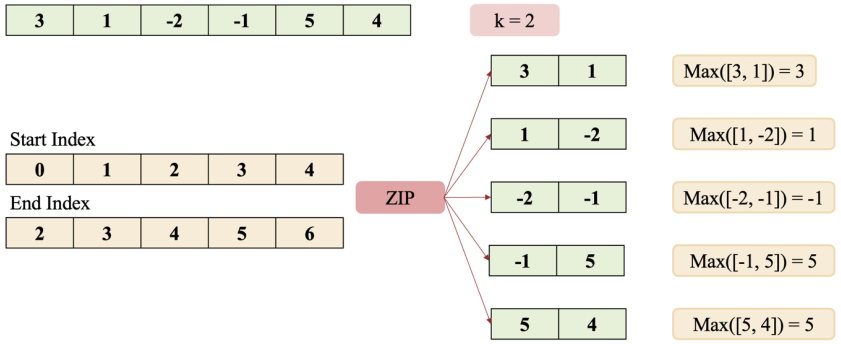
Levenshtein Distance



Practice

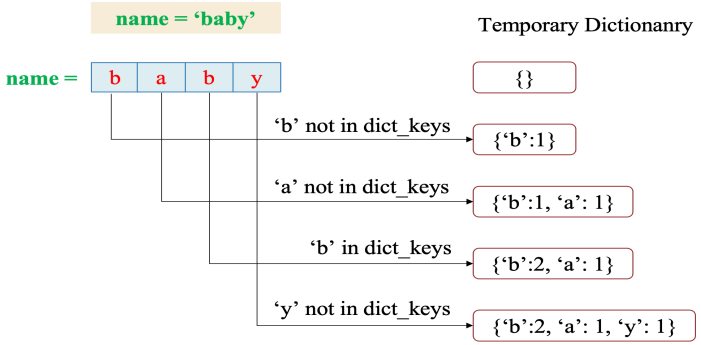
Exercise 1

Getting Max Over Kernel



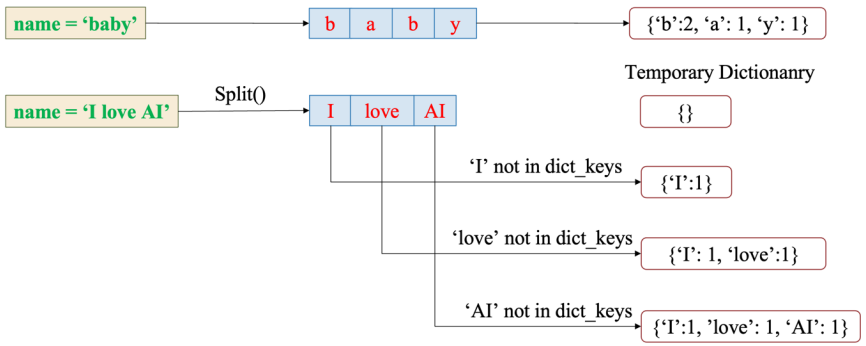
Exercise 2

Character Counting



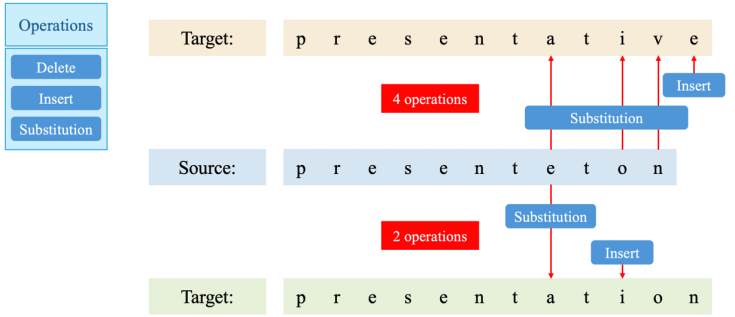
Exercise 3

Word Counting



Exercise 4

Levenshtein Distance



Getting Max Over Kernel

1 Description

Problem: Cho một list các số nguyên ***num_list*** và một sliding window (các bạn có thể tạm hiểu sliding window như là một list có kích thước nhỏ hơn ***num_list***) 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. Các bạn hãy viết chương trình Python giải quyết vấn đề trên.

Example:

- **Input:**

  num_list = [3, 4, 5, 1, −44, 5, 10, 12, 33, 1]

  k = 3

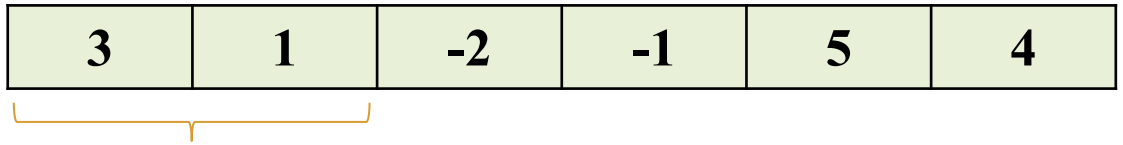
- **Output:** [5, 5, 5, 5, 10, 12, 33, 33]

Getting Max Over Kernel

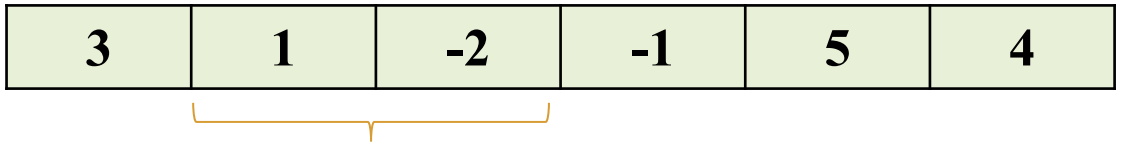
1 Solution

Use *slicing* and *max()* function

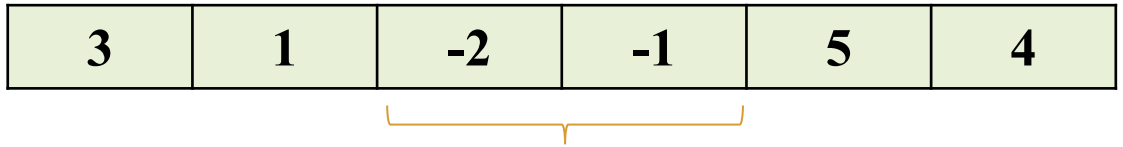
k = 2



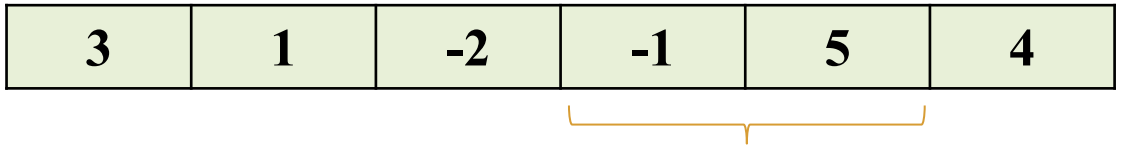
Max([3, 1]) = 3



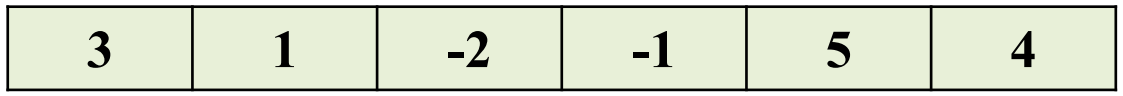
Max([1, -2]) = 1



Max([-2, -1]) = -1



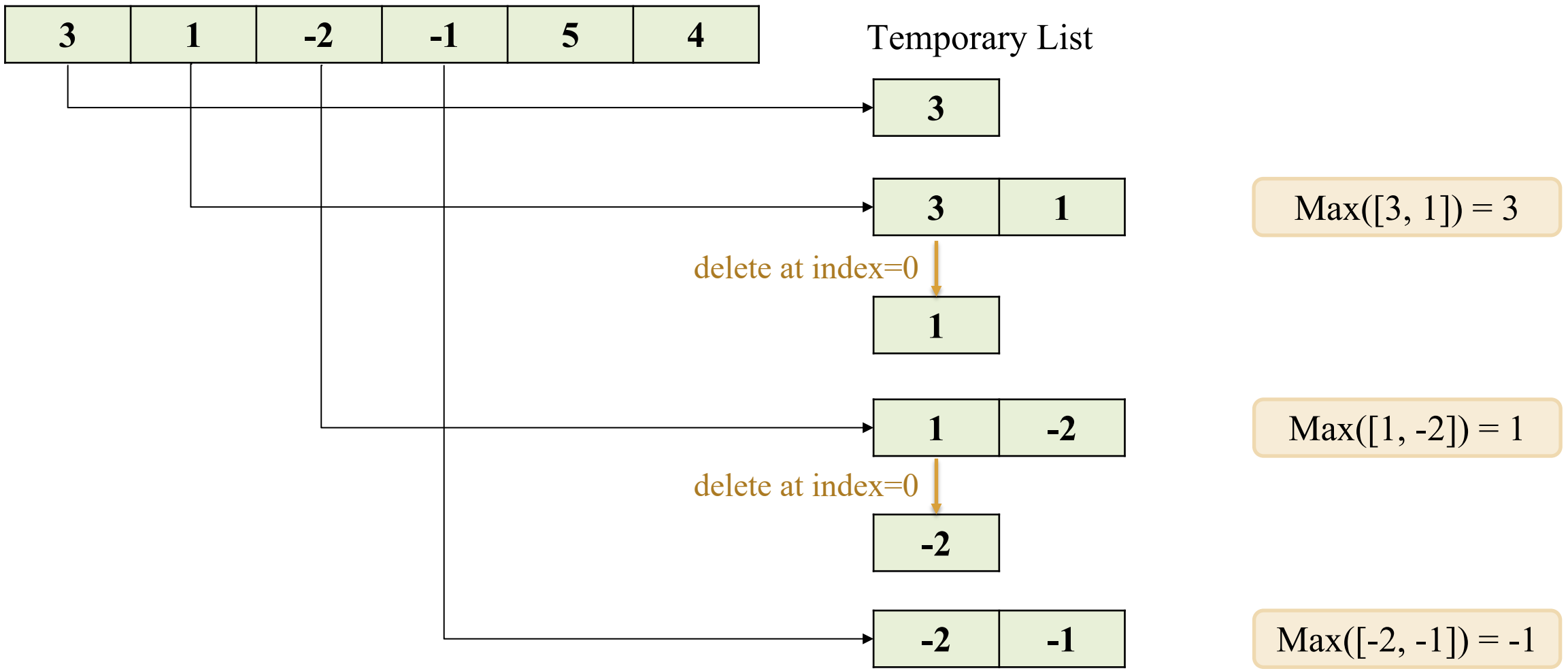
Max([-1, 5]) = 5



Max([5, 4]) = 5

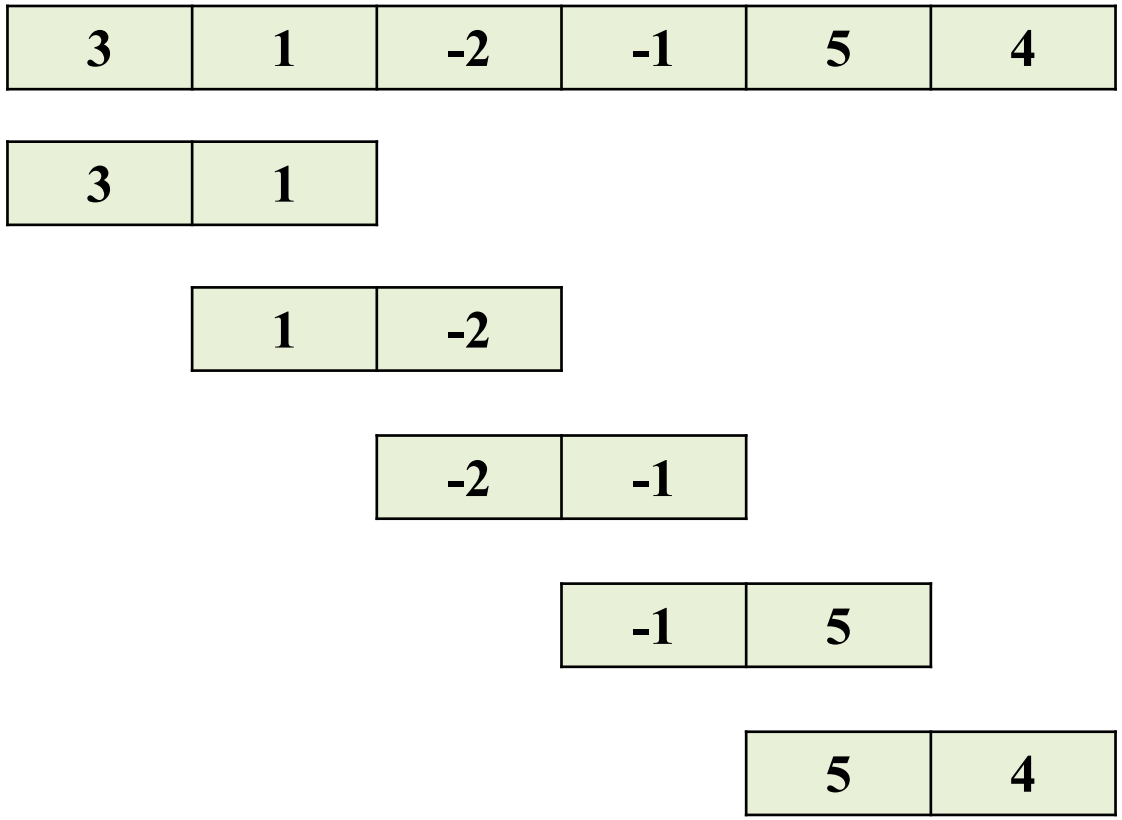
Getting Max Over Kernel

Get Kernels



Getting Max Over Kernel

Get Kernels



k = 2

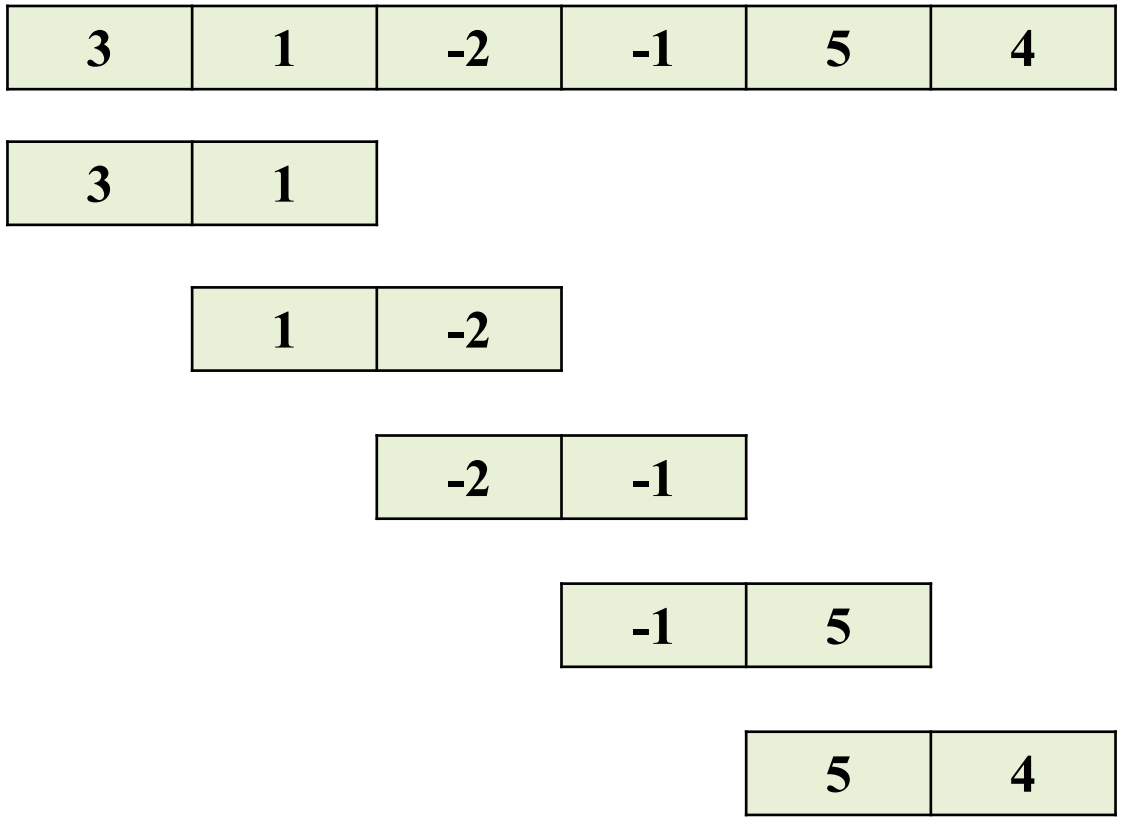
```
1  num_list = [3 , 1, -2, -1, 5, 4]
2  k = 2
3  sub_list = []
4
5  for element in num_list:
6      sub_list.append(element)
7
8      if len(sub_list) == k:
9          print(sub_list)
10         del sub_list[0]
```

✓ 0.0s

```
[3, 1]
[1, -2]
[-2, -1]
[-1, 5]
[5, 4]
```

Getting Max Over Kernel

Get Kernels – Solution #1



k = 2

```python
def sliding_maximum(num_list, k):
    result = []
    sub_list = []

    for element in num_list:
        sub_list.append(element)

        if len(sub_list) == k:
            result.append(max(sub_list))
            del sub_list[0]

    return result


# Kiểm tra hàm
num_list = [3, 1, -2, -1, 5, 4]
k = 2
print(sliding_maximum(num_list, k))
```

✓ 0.0s

```
[3, 1, -1, 5, 5]
```

Getting Max Over Kernel

Slicing – Solution #2

3	1	-2	-1	5	4

k = 2

list[start:end]

list[0:2]	list[1:3]	list[2:4]	list[3:5]	list[4:6]
3 1	1 -2	-2 -1	-1 5	5 4

Start Index	0	1	2	3	4	0 => len(list) - k
End Index	2	3	4	5	6	k => len(list)

Getting Max Over Kernel

Slicing – Solution #2

3	1	-2	-1	5	4

k = 2

Start Index	0	1	2	3	4

0 => len(list) - k

End Index	2	3	4	5	6

k => len(list)

```
1  num_list = [3 , 1 , -2, -1, 5, 4]
2  k = 2
3  start_indexs = list(range(0, len(num_list)-k+1))
4  end_indexs = list(range(k, len(num_list)+1))
5  print(start_indexs)
6  print(end_indexs)
```

✓ 0.0s

```
[0, 1, 2, 3, 4]
[2, 3, 4, 5, 6]
```

Getting Max Over Kernel

Slicing – Solution #2

3	1	-2	-1	5	4

k = 2

Start Index

0	1	2	3	4

End Index

2	3	4	5	6

ZIP

3	1	Max([3, 1]) = 3
1	-2	Max([1, -2]) = 1
-2	-1	Max([-2, -1]) = -1
-1	5	Max([-1, 5]) = 5
5	4	Max([5, 4]) = 5

Getting Max Over Kernel

Slicing

3	1	-2	-1	5	4

k = 2

```python
def max_kernel(num_list, k):
    start_indices = list(range(0, len(num_list) - k + 1))
    end_indices = list(range(k, len(num_list) + 1))

    result = []

    for start_index, end_index in zip(start_indices, end_indices):
        sub_list = num_list[start_index:end_index]
        result.append(max(sub_list))

    return result


# Kiểm tra hàm
num_list = [3, 4, 5, 1, -44, 5, 10, 12, 33, 1]
k = 3
print(max_kernel(num_list, k))
```

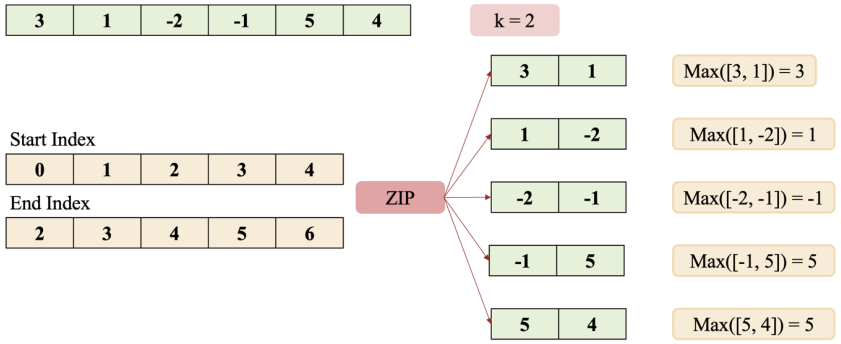
✓ 0.0s

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

Practice

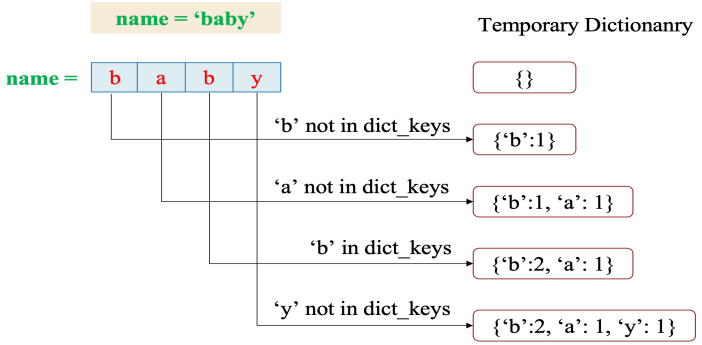
Exercise 1

Getting Max Over Kernel



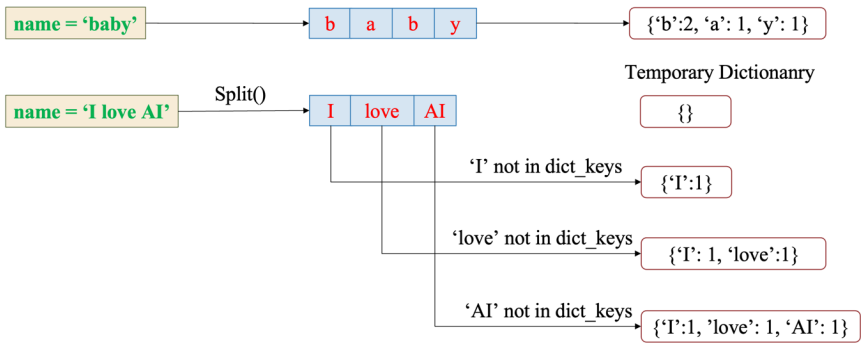
Exercise 2

Character Counting



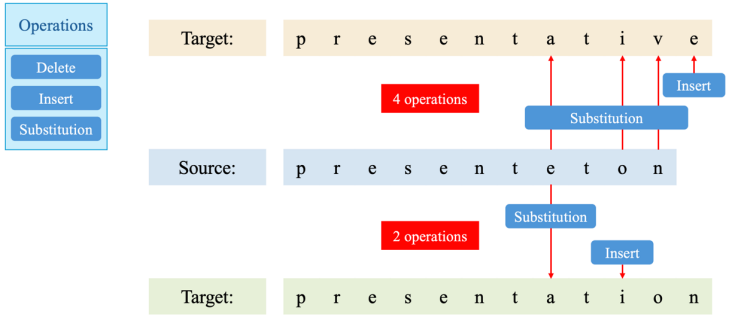
Exercise 3

Word Counting



Exercise 4

Levenshtein Distance



Character Counting

Description

Problem: Viết thuật toán 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]

Example:

- **Input:**

  word = 'baby'

- **Output:**

```
{'b': 2, 'a': 1, 'y': 1}
```

Character Counting

Solution

name = 'baby'

index	0	1	2	3
negative index	-4	-3	-2	-1

name =

b	a	b	y

```
1 word = 'baby'
2
3 for character in word:
4     print(character)
```

✓ 0.0s

```
b
a
b
y
```

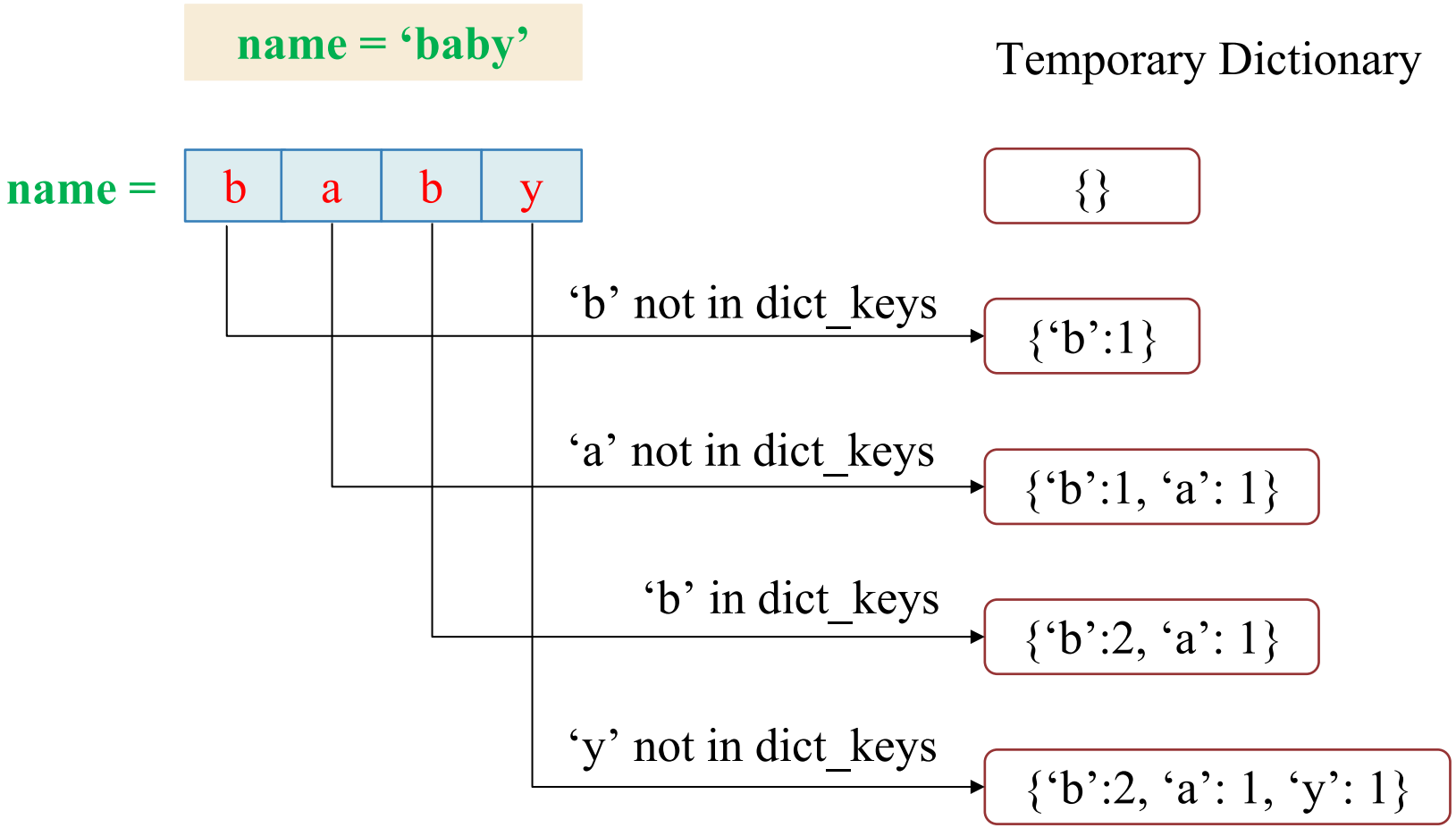
```
1 word = "Baby"
2 print(word)
3 print(word[0])
4 print(word[1:3])
5 word[2] = "H"
```

⊗ 0.1s

```
Baby
B
ab

---------------------------------------------------------------
TypeError                                 Traceback (most
Cell In[8], line 5
      3 print(word[0])
      4 print(word[1:3])
----> 5 word[2] = "H"

TypeError: 'str' object does not support item assignment
```

Character Counting

Solution



Character Counting

Solution

```
character_statistic = {}

word = 'baby'

for character in word:
    if character in character_statistic:
        character_statistic[character] += 1
    else:
        character_statistic[character] = 1

print(character_statistic)
```

✓ 0.0s

```
{'b': 2, 'a': 1, 'y': 1}
```

```
character_statistic = {}

word = 'Baby'

for character in word:
    if character in character_statistic:
        character_statistic[character] += 1
    else:
        character_statistic[character] = 1

print(character_statistic)
```

✓ 0.0s

```
{'B': 1, 'a': 1, 'b': 1, 'y': 1}
```

Character Counting

Extension

‘Baby’ `{'B': 1, 'a': 1, 'b': 1, 'y': 1}`

≠

‘baby’ `{'b': 2, 'a': 1, 'y': 1}`

‘Baby’ and ‘baby’: the same meaning in text

Text Preprocessing

‘Baby’ → ‘baby’

Lowercasing

```
1  character_statistic = {}
2
3  word = 'Baby'
4  word = word.lower()
5
6  for character in word:
7      if character in character_statistic:
8          character_statistic[character] += 1
9      else:
10         character_statistic[character] = 1
11
12 print(character_statistic)
```

✓ 0.0s

```
{'b': 2, 'a': 1, 'y': 1}
```

Character Counting

Extension

name = 'baby'

name = | b | a | b | y |

name = 'Baby'

Lowercasing ↓

name = 'baby'

name = | b | a | b | y |

```python
def count_character(word):
    character_statistic = {}

    for character in word.lower():
        if character in character_statistic:
            character_statistic[character] += 1
        else:
            character_statistic[character] = 1

    return character_statistic


# Kiểm tra hàm
print(count_character('smiles'))
```

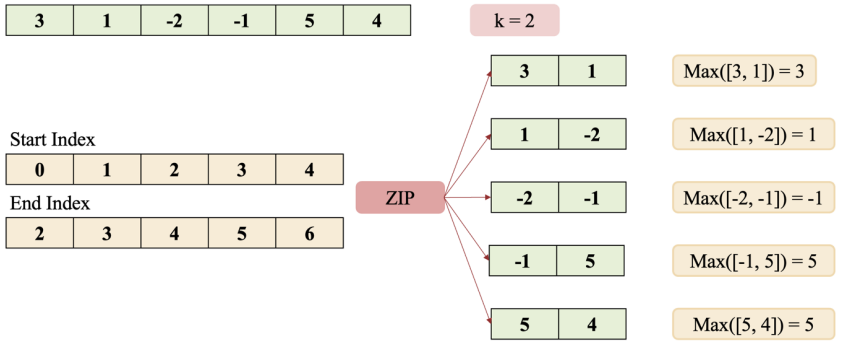
✓ 0.0s

```
{'s': 2, 'm': 1, 'i': 1, 'l': 1, 'e': 1}
```

Practice

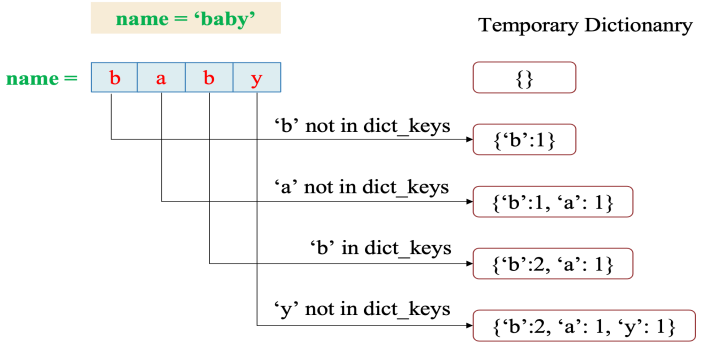
Exercise 1

Getting Max Over Kernel



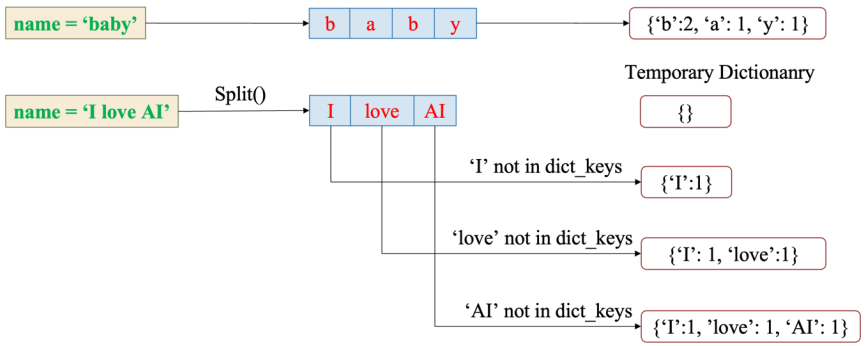
Exercise 2

Character Counting



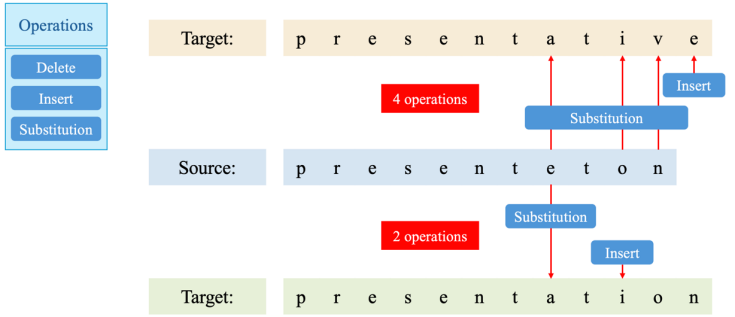
Exercise 3

Word Counting



Exercise 4

Levenshtein Distance



Word Counting

Description

Problem: Viết thuật toán 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

File: https://drive.google.com/uc?id=1IBScGdW2xlNsc9v5zSAya548kNgiOrko

Word Counting

Read File

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

```
Downloading...
From: https://drive.google.com/uc?id=1IBScGdW2xlNsc9v5zSAya548kNgiOrko
To: /content/P1_data.txt

  0% 0.00/747 [00:00<?, ?B/s]
100% 747/747 [00:00<00:00, 2.96MB/s]
```

```
with open('/content/P1_data.txt', 'r') as f:
    document = f.read()
```

```
document
```

```
'He who conquers himself is the mightiest warrior\nTry not
```

```
with open('/content/P1_data.txt', 'r') as f:
    sentences = f.readlines()
type(sentences)
```

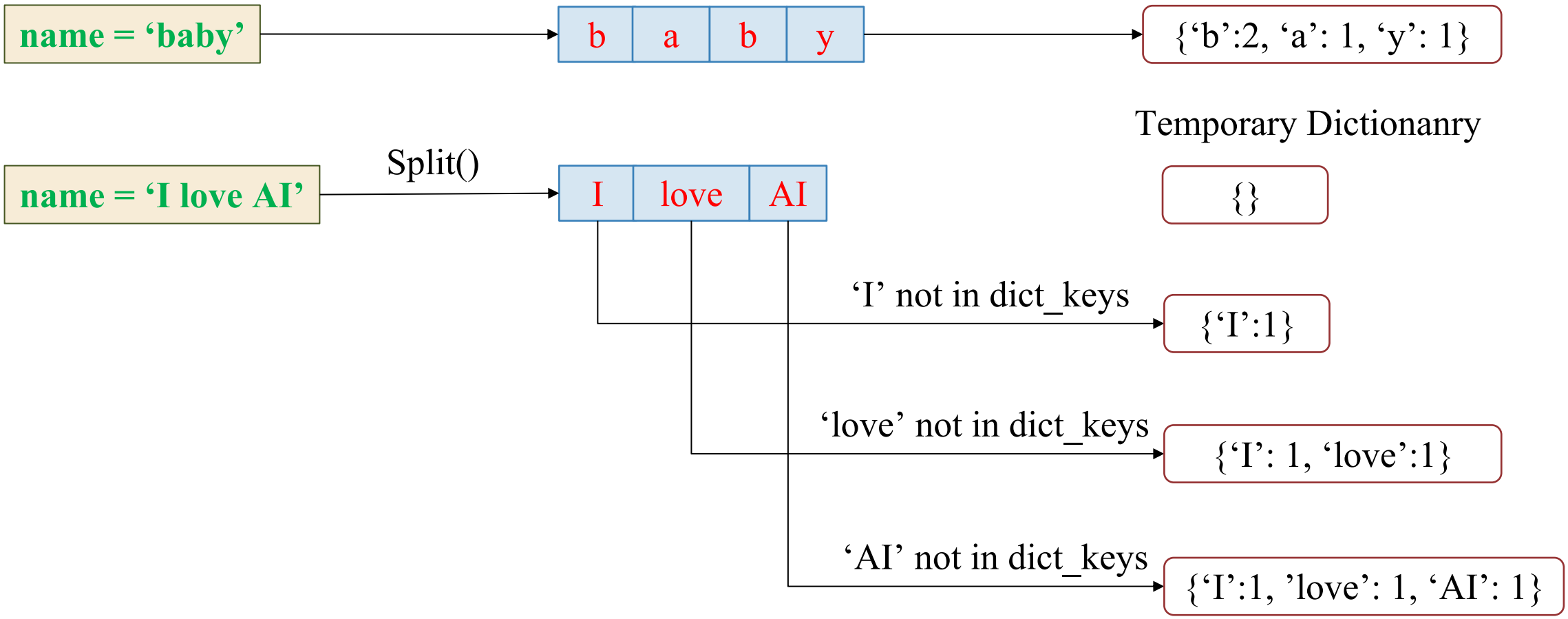
```
list
```

```
sentences[:2]
```

```
['He who conquers himself is the mightiest warrior\n',
 'Try not to become a man of success but rather become a man
```

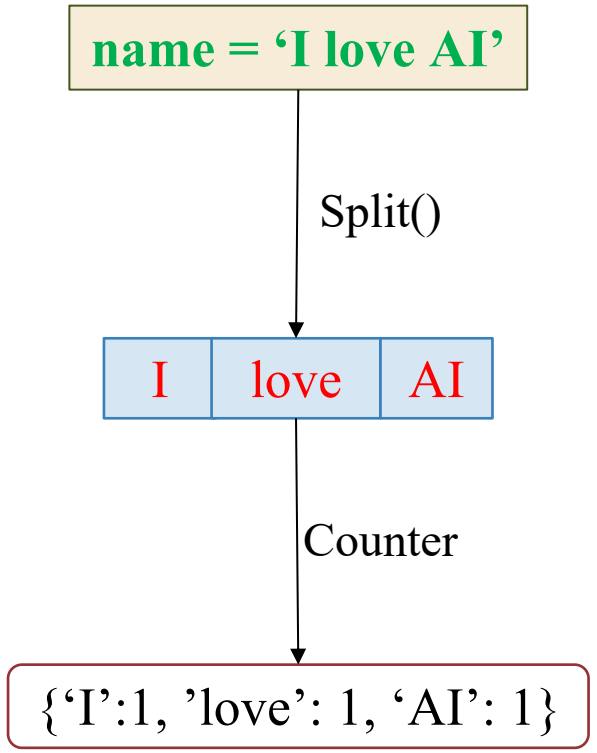
Word Counting

Counting



Word Counting

Counting



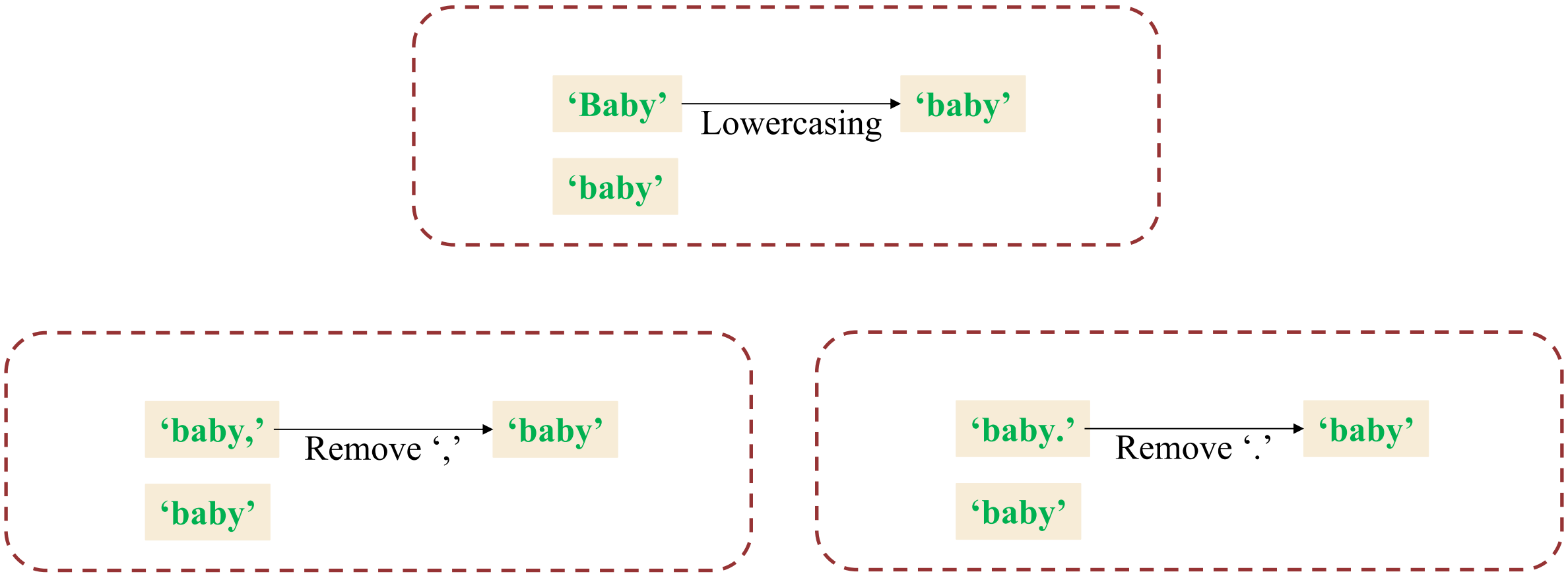

```
1  sentence = 'I love AI'
2  words = sentence.split()
3
4  counter = {}
5  for word in words:
6      if word in counter:
7          counter[word] += 1
8      else:
9          counter[word] = 1
10
11 print(counter)
```

✓ 0.0s

```
{'I': 1, 'love': 1, 'AI': 1}
```

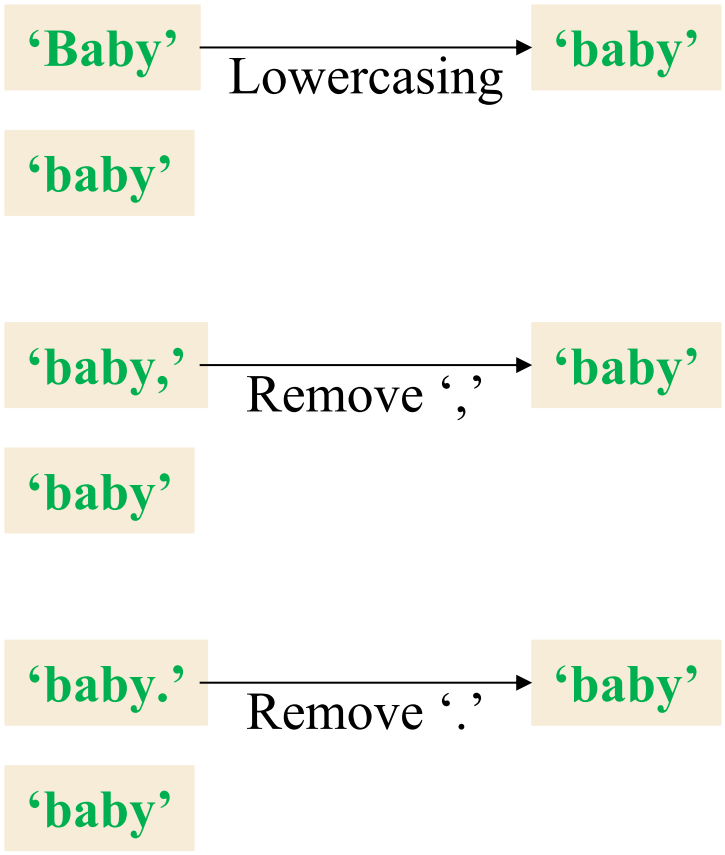
Word Counting

Text Preprocessing



Word Counting

Text Preprocessing




```python
def preprocess_text(sentence):
    """
    Tiền xử lý một câu bằng cách:
    - Chuyển tất cả các ký tự thành chữ thường
    - Loại bỏ dấu chấm (.) và dấu phẩy (,)
    - Tách câu thành danh sách các từ
    """
    sentence = sentence.lower()
    sentence = sentence.replace('.', '').replace(',', '')
    words = sentence.split()
    return words

# Kiểm tra hàm
sentence = 'I love AI. AI is not easy'
print(preprocess_text(sentence))
# ['i', 'love', 'ai', 'ai', 'is', 'not', 'easy']
```

✓ 0.0s

```
['i', 'love', 'ai', 'ai', 'is', 'not', 'easy']
```

Word Counting

Word Counting

```
for sentence in sentences:
    words = preprocess_text(sentence)
    for word in words:
        if word in counter:
            counter[word] += 1
        else:
            counter[word] = 1

return counter
```

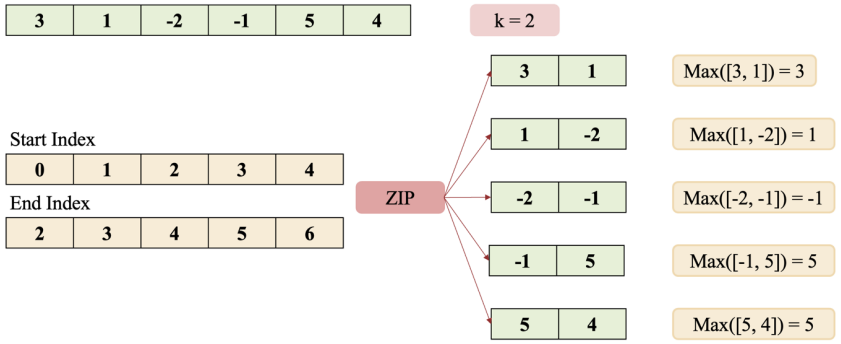
```
1  def count_word(data_path):
2      with open(data_path, 'r') as f:
3          document = f.read()
4      words = preprocess_text(document)
5
6      counter = {}
7      for word in words:
8          if word in counter:
9              counter[word] += 1
10         else:
11             counter[word] = 1
12
13     return counter
14
15 data_path = './P1_data.txt'
16 result = count_word(data_path)
17 print(result.get('man', 0))
```


QUIZ TIME

Practice

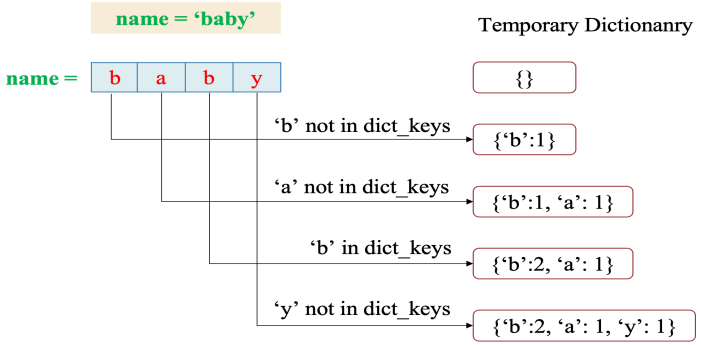
Exercise 1

Getting Max Over Kernel



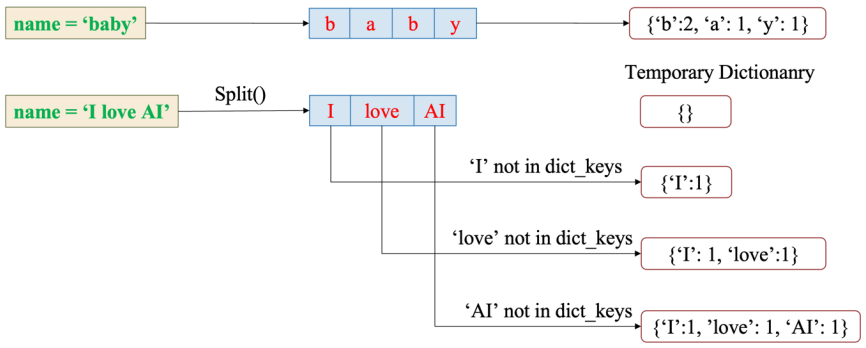
Exercise 2

Character Counting



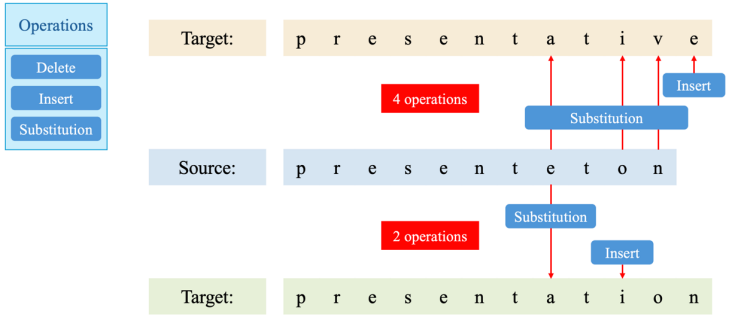
Exercise 3

Word Counting



Exercise 4

Levenshtein Distance



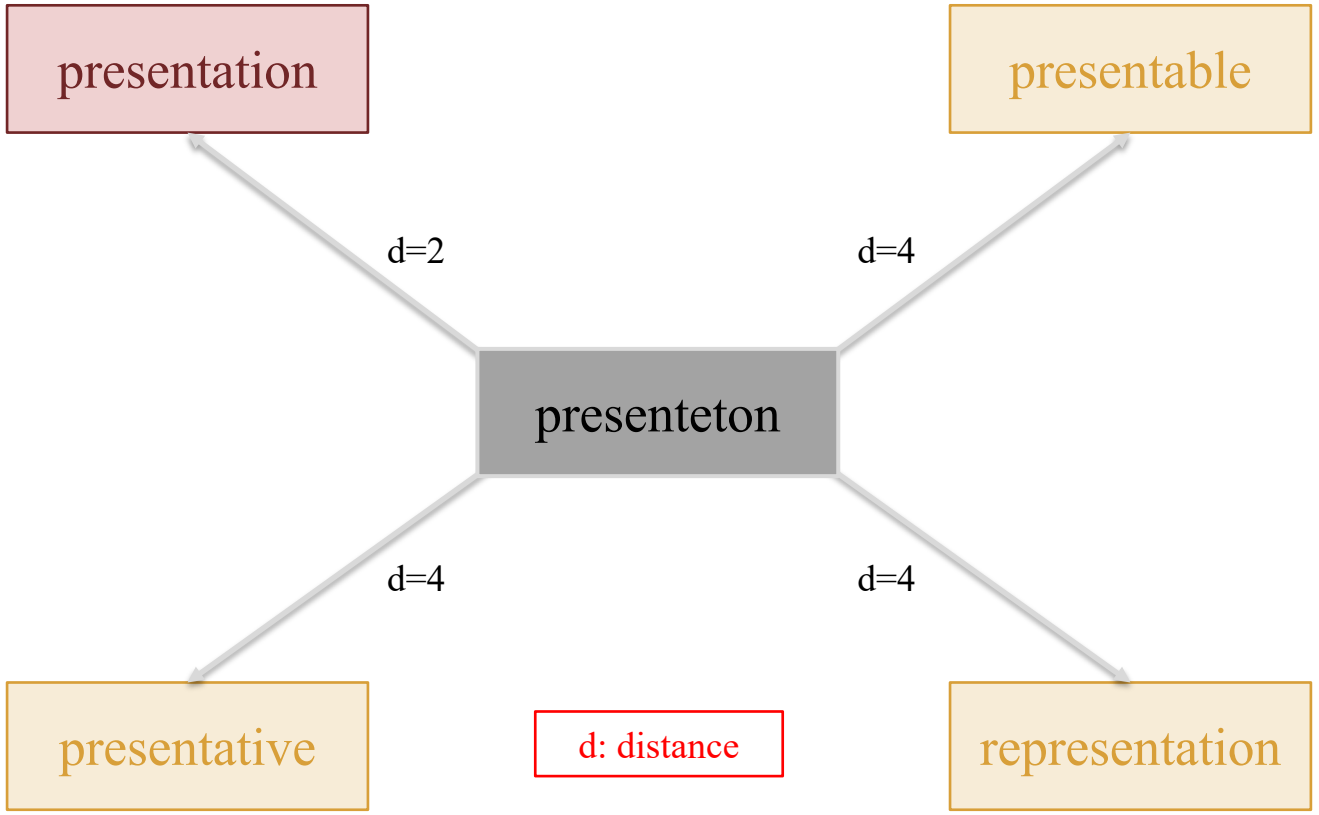
Levenshtein Distance

Getting Started

presenteton

Có phải ý bạn là: ***presentation***

How to measure the similarity or gap between two strings ?

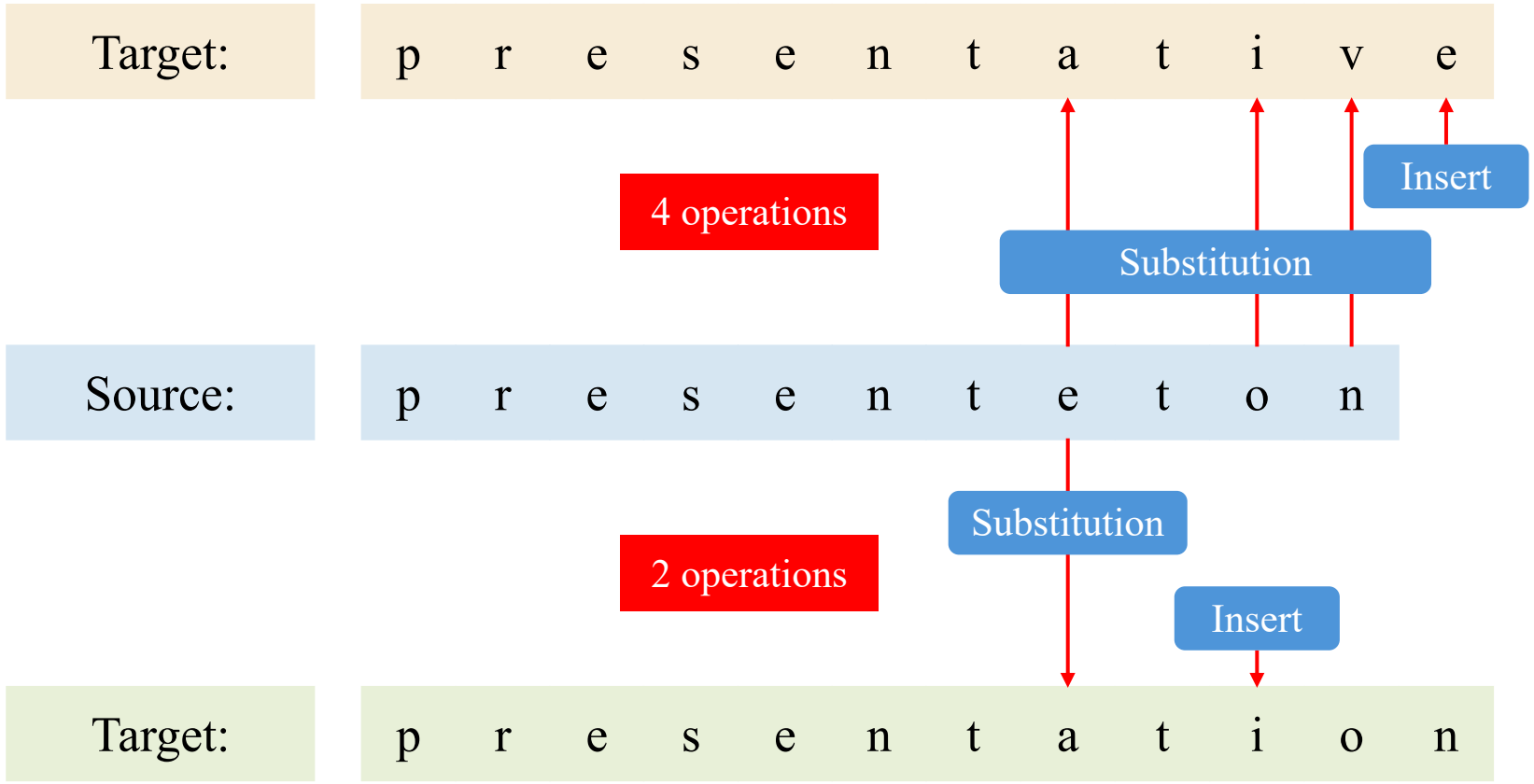


Levenshtein Distance

Minimum number of editing operations

Operations

- Delete
- Insert
- Substitution



Levenshtein Distance

The Minimum Edit Distance Algorithm

Source: hola

Target: hello

Delete cost = 1

Insert cost = 1

Substitution cost = 1

insert

delete

		Target					
		#	h	e	l	l	o
Source	#	0	1	2	3	4	5
	h	1					
	o	2					
	l	3					
	a	4					

sub

→ #hel
1. insert h
2. insert e
3. insert l

#h → #
1. delete h

#hol → #
1. delete h
2. delete o
3. delete l

Levenshtein Distance

Algorithm

$$D[i,j] = min \begin{cases} D[i-1,j] + delcost(source[i]) \\ D[i,j-1] + inscost(target[j]) \\ D[i-1,j-1] + subcost(source[i], target[j]) \end{cases}$$

insert

delete

i \ j		Target #	h	e	l	l	o
Source	#	0	1	2	3	4	5
	h	1					
	o	2					
	l	3					
	a	4					

Delete cost = 1

Insert cost = 1

Substitution cost = 1

$$D[1,1] = min \begin{cases} D[0,1] + delcost(source[1]) = 1 + 1 = 2 \\ D[1,0] + inscost(target[1]) = 1 + 1 = 2 \\ D[0,0] + subcost(source[1], target[1]) = 0 + 0 = 0 \end{cases}$$

Note:
If source[i] == target[j]:
subcost(source[i] ,target[j])=0

Levenshtein Distance

Algorithm

insert

i \ j		Target #	h	e	l	l	o
	#	0	1	2	3	4	5
Source	h	1	0				
	o	2					
	l	3					
	a	4					

delete

Delete cost = 1

Insert cost = 1

Substitution cost = 1

$$D[2,1] = min \begin{cases} D[1,1] + delcost(source[2]) = 0 + 1 = 1 \\ D[2,0] + inscost(target[1]) = 2 + 1 = 3 \\ D[1,0] + subcost(source[2], target[1]) = 1 + 1 = 2 \end{cases}$$

Note:
If source[i] == target[j]:
subcost(source[i] ,target[j])=0

$$D[1,2] = min \begin{cases} D[0,2] + delcost(source[1]) = 2 + 1 = 3 \\ D[1,1] + inscost(target[2]) = 0 + 1 = 1 \\ D[0,1] + subcost(source[1], target[2]) = 1 + 1 = 2 \end{cases}$$

Levenshtein Distance

Algorithm

insert

i \ j		Target #	h	e	l	l	o
Source	#	0	1	2	3	4	5
	h	1	0	1			
	o	2	1				
	l	3					
	a	4					

delete

Delete cost = 1

Insert cost = 1

Substitution cost = 1

$$D[2,2] = min \begin{cases} D[1,2] + \text{delcost}(source[2]) = 1 + 1 = 2 \\ D[2,1] + \text{inscost}(target[2]) = 1 + 1 = 2 \\ D[1,1] + \text{subcost}(source[2], target[2]) = 0 + 1 = 1 \end{cases}$$

Note:
If source[i] == target[j]:
subcost(source[i] ,target[j])=0

Levenshtein Distance

Algorithm

insert

delete

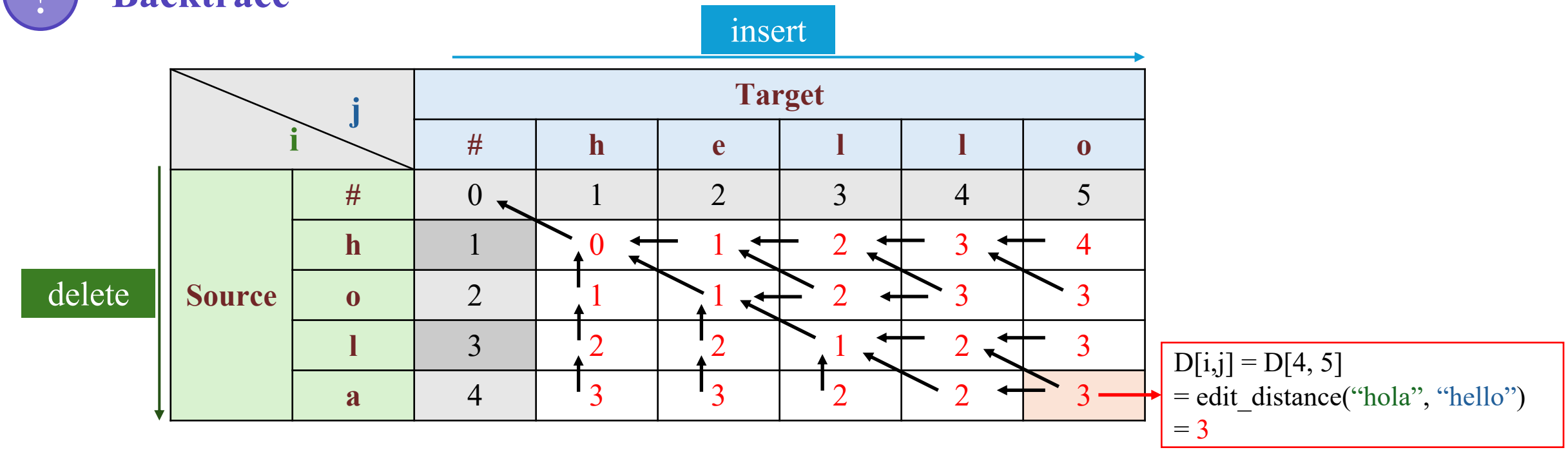
i \ j		Target #	h	e	l	l	o
Source	#	0	1	2	3	4	5
	h	1	0	1	2	3	4
	o	2	1	1	2	3	3
	l	3	2	2	1	2	3
	a	4	3	3	2	2	3

D[i,j] = D[4, 5]
= edit_distance("hola","hello")
= 3

When going down each step, store back pointers in each cell to serve for the backtrace phase.

Levenshtein Distance

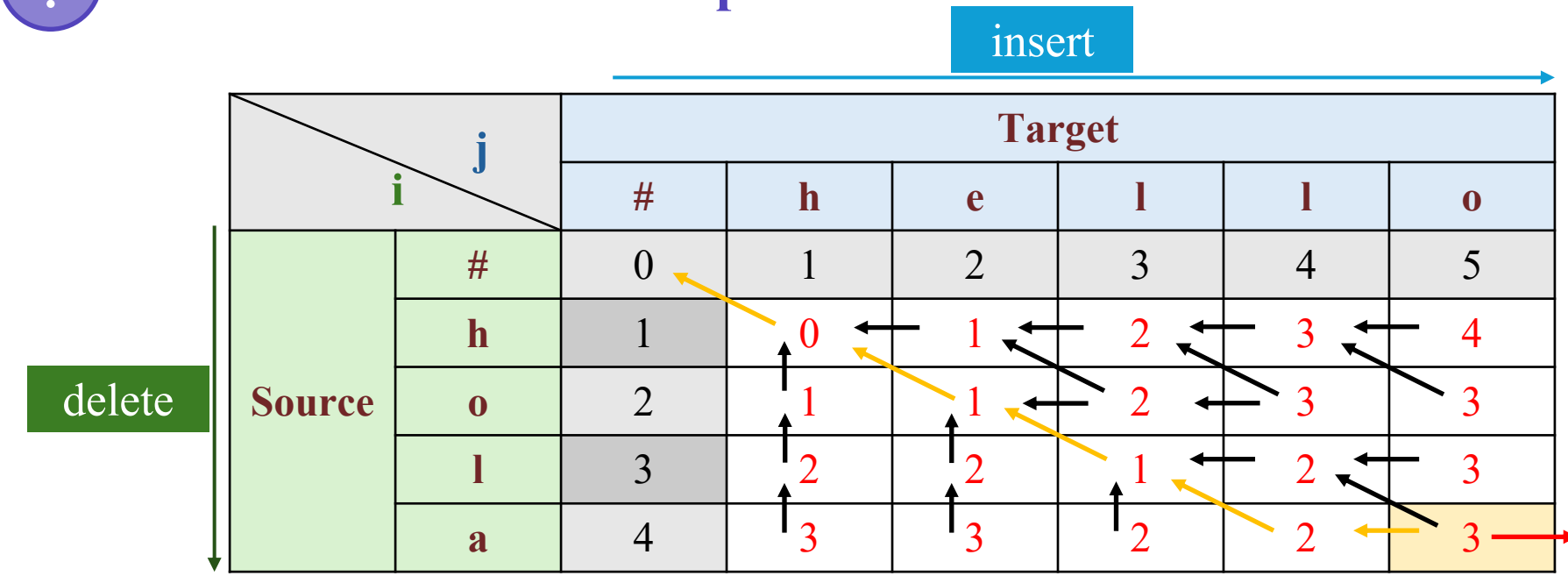
Backtrace



- Starting from the last cell and returning based on choosing the minimum cell value
- Each cell may have multiple to return to because they have the same minimum value.

Levenshtein Distance

Minimum edit distance path

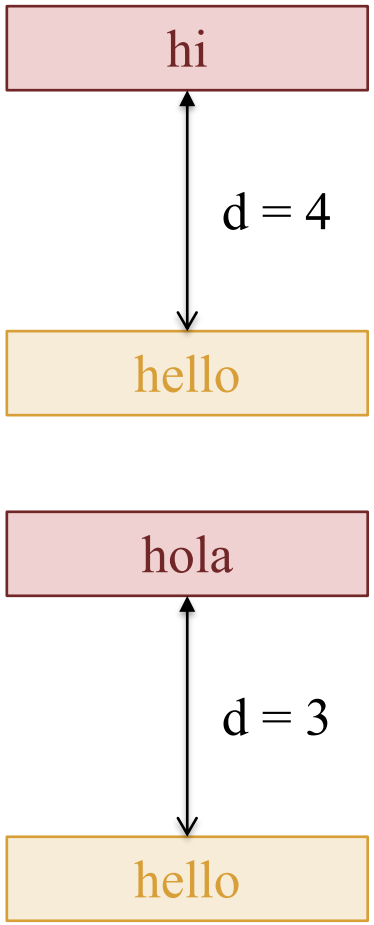


D[i,j] = D[4, 5]
= edit_distance("hola", "hello")
= 3

- This is one of the minimum edit distance paths.
- Modify steps (going in the reverse direction with the backtrace path):
  - sub(h, h) => hola; cost = 0
  - sub(o, e) => hela; cost = 1
  - sub(l, l) => hela; cost = 0
  - sub(a, l) => hell; cost = 1
  - insert(o) => hello; cost = 1

Levenshtein Distance

Solution

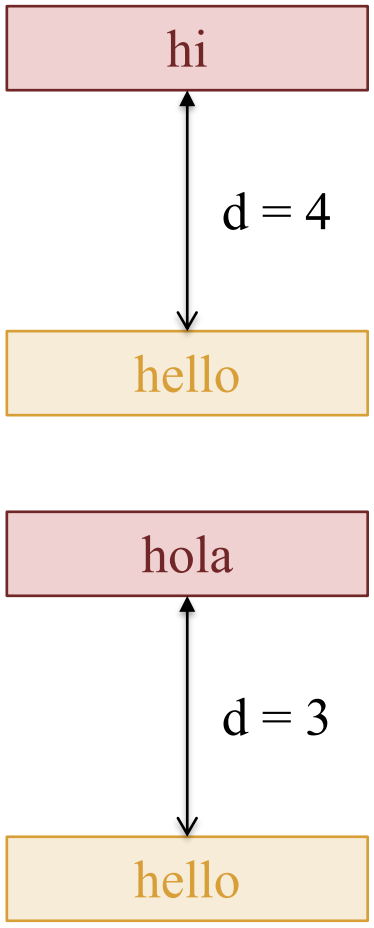



```python
def levenshtein_distance(token1, token2):
    distances = [[0] * (len(token2) + 1) for _ in range(len(token1) + 1)]

    for t1 in range(len(token1) + 1):
        distances[t1][0] = t1

    for t2 in range(len(token2) + 1):
        distances[0][t2] = t2

    for t1 in range(1, len(token1) + 1):
        for t2 in range(1, len(token2) + 1):
            if token1[t1 - 1] == token2[t2 - 1]:
                distances[t1][t2] = distances[t1 - 1][t2 - 1]
            else:
                a = distances[t1][t2 - 1]      # insert
                b = distances[t1 - 1][t2]      # delete
                c = distances[t1 - 1][t2 - 1]  # replace
                distances[t1][t2] = min(a, b, c) + 1

    return distances[len(token1)][len(token2)]
```

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

Levenshtein Distance

Solution

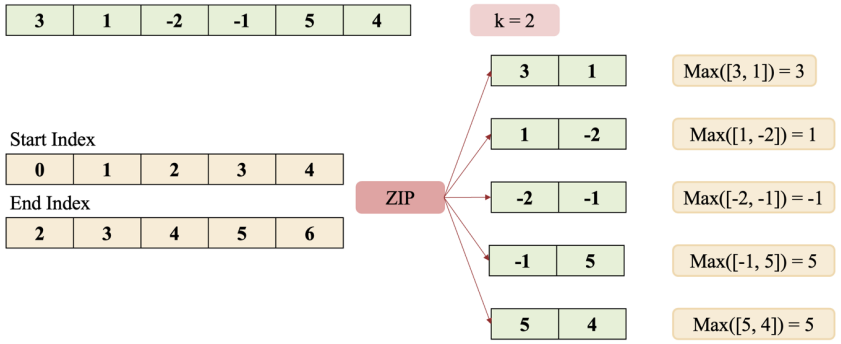



```python
def levenshtein_distance_dp(token1, token2):
    distances = [[0] * (len(token2) + 1) for _ in range(len(token1) + 1)]

    for t1 in range(len(token1) + 1):
        distances[t1][0] = t1

    for t2 in range(len(token2) + 1):
        distances[0][t2] = t2

    for t1 in range(1, len(token1) + 1):
        for t2 in range(1, len(token2) + 1):
            if token1[t1 - 1] == token2[t2 - 1]:
                distances[t1][t2] = distances[t1 - 1][t2 - 1]
            else:
                a = distances[t1][t2 - 1]      # Insert
                b = distances[t1 - 1][t2]      # Delete
                c = distances[t1 - 1][t2 - 1]  # Replace

                distances[t1][t2] = min(a, b, c) + 1

    print_distances(distances, len(token1), len(token2))
    return distances[len(token1)][len(token2)]

```

Practice

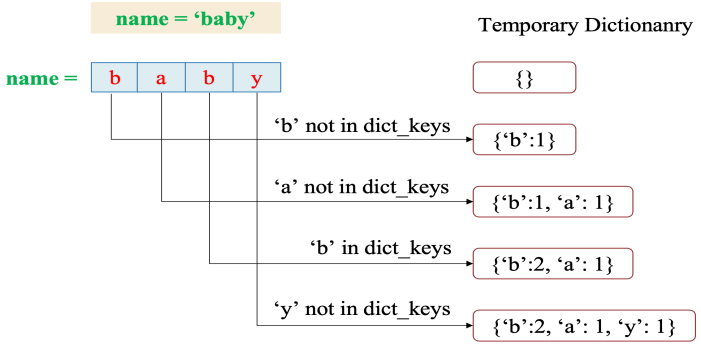
Exercise 1

Getting Max Over Kernel



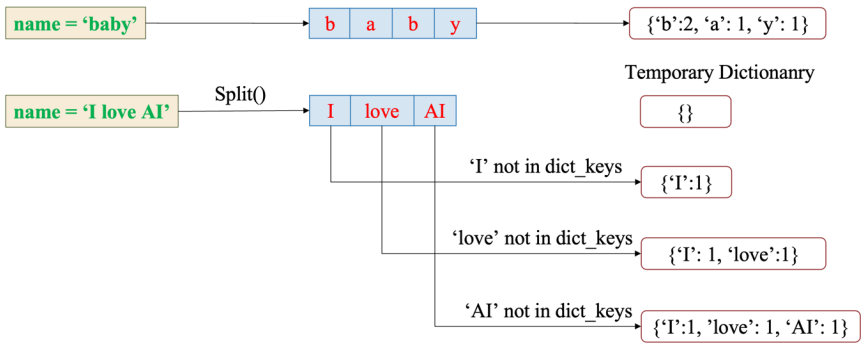
Exercise 2

Character Counting



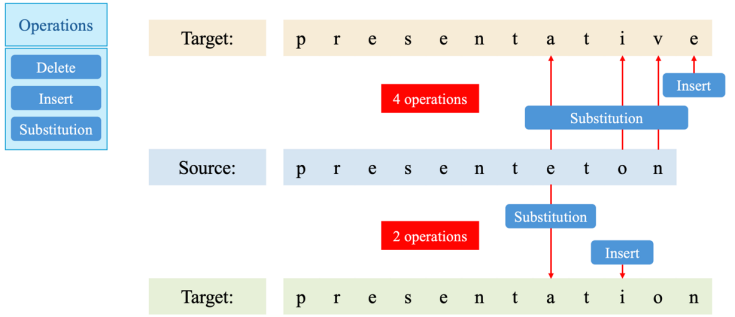
Exercise 3

Word Counting



Exercise 4

Levenshtein Distance



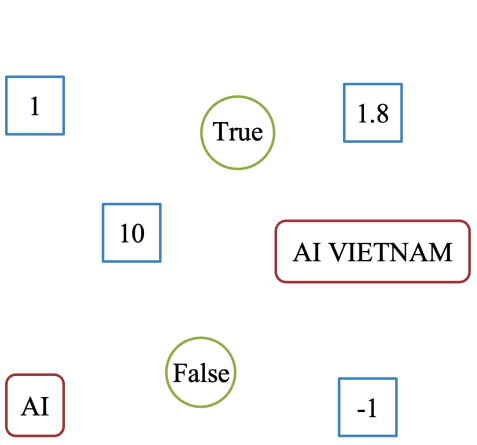
Objectives

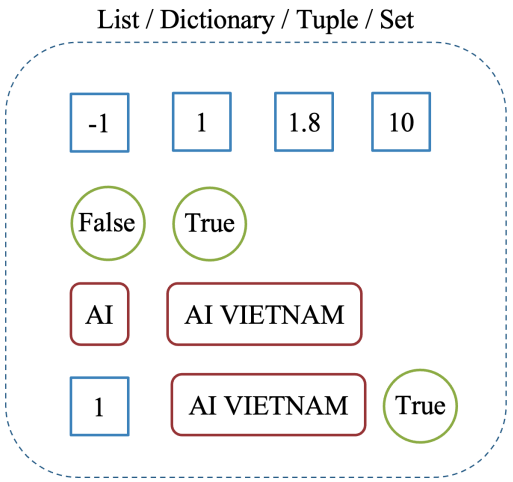
Data Structure in Python

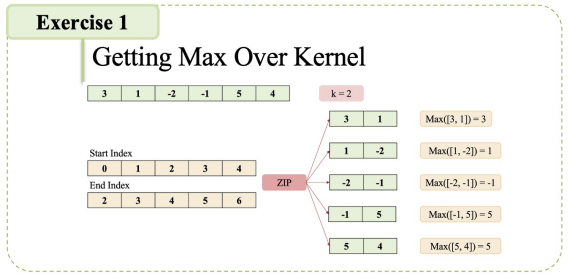
- List
- Dictionary
- Tuple
- Set

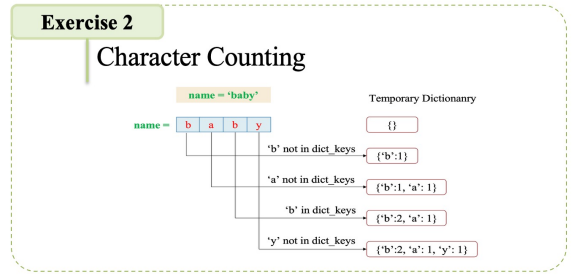
Practice

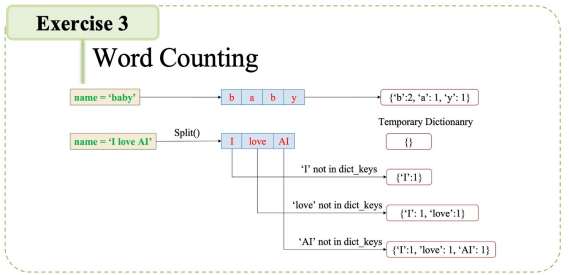
- Getting Max Over Kernel
- Character Counting
- Word Counting
- Levenshtein Distance

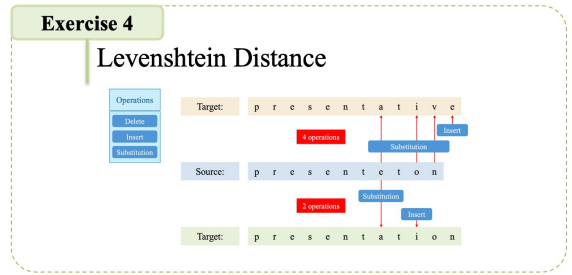












Thanks!

Any questions?