Python Programming - Tư Duy Thuật Toán

> **Mục tiêu học tập**: Phát triển tư duy giải quyết vấn đề bằng thuật toán - từ phân tích bài toán đến viết code hiệu quả.

1. Giới Thiệu Tư Duy Thuật Toán

1.1. Thuật Toán là gì?

🤔 Định nghĩa

Thuật toán (Algorithm) là một chuỗi các bước hữu hạn, rõ ràng để giải quyết một bài toán.

Đặc điểm của thuật toán tốt:

- ☑ **Correct** (đúng): Cho kết quả chính xác
- ☑ **Clear** (rõ ràng): Dễ hiểu, dễ implement
- ☑ **Efficient** (hiệu quả): Nhanh và tiết kiệm bộ nhớ
- ☑ **General** (tổng quát): Áp dụng được cho nhiều trường hợp

1.2. Quy Trình Giải Quyết Vấn Đề

```
1. HIỂU bài toán
   - Input là gì?
   - Output mong muốn?
   - Điều kiện, ràng buộc?

2. PHÂN TÍCH
   - Bài toán quen thuộc nào?
   - Cần dữ liệu gì?
   - Edge cases?

3. THIẾT KẾ thuật toán
   - Các bước cần thực hiện?
   - Vẽ sơ đồ, viết pseudocode

4. IMPLEMENT
   - Viết code từng bước
   - Test với ví dụ nhỏ

5. KIỂM TRA
   - Test cases thông thường
   - Edge cases
   - Tối ưu nếu cần
```

2. Tìm Kiếm Tuyến Tính (Linear Search)

2.1. Khái Niệm

Linear Search là duyệt qua **từng phần tử** cho đến khi tìm thấy hoặc hết list.

Time Complexity: O(n) - worst case phải duyệt hết n phần tử

2.2. Pattern Cơ Bản

```
def linear_search(items, target):
    """
    Tìm vị trí đầu tiên của target trong items

    Args:
        items (list): Danh sách cần tìm
        target: Giá trị cần tìm

    Returns:
        int: Index của target, hoặc -1 nếu không tìm thấy
    """
    for i in range(len(items)):
        if items[i] == target:
            return i  # Tìm thấy
    return -1  # Không tìm thấy

# Test
numbers = [3, 7, 2, 9, 1, 5]
print(linear_search(numbers, 9))  # 3
print(linear_search(numbers, 10)) # -1
```

2.3. Tìm Tất Cả Vị Trí

```
def find_all_occurrences(items, target):
    """
    Tìm tất cả vị trí xuất hiện của target

    Returns:
        list: Danh sách các indices
    """
    indices = []
    for i in range(len(items)):
        if items[i] == target:
            indices.append(i)
    return indices

# Hoặc dùng list comprehension
def find_all_occurrences_v2(items, target):
    return [i for i, x in enumerate(items) if x == target]

# Test
```

```
numbers = [1, 2, 3, 2, 4, 2, 5]
print(find_all_occurrences(numbers, 2))  # [1, 3, 5]
```

2.4. Tìm Kiếm Với Điều Kiện

```
# Tìm số chẵn đầu tiên
def find_first_even(numbers):
    """Tìm số chẵn đầu tiên"""
    for num in numbers:
        if num % 2 == 0:
            return num
    return None

# Tìm string bắt đầu bằng 'A'
def find_first_starting_with_a(words):
    """Tìm từ đầu tiên bắt đầu bằng 'A'"""
    for word in words:
        if word.startswith('A'):
            return word
    return None

# Test
numbers = [1, 3, 5, 8, 9]
print(find_first_even(numbers))  # 8

words = ["banana", "Apple", "cherry"]
print(find_first_starting_with_a(words))  # Apple
```

2.5. Kiểm Tra Tồn Tại

```
def contains(items, target):
    """Kiểm tra target có trong items không"""
    for item in items:
        if item == target:
            return True
    return False

# Pythonic hơn - dùng 'in'
def contains_v2(items, target):
    return target in items

# Test
numbers = [1, 2, 3, 4, 5]
print(contains(numbers, 3))   # True
print(contains(numbers, 10))  # False
```

---

3. Tìm Min/Max Với Điều Kiện

3.1. Tìm Min/Max Đơn Giản

```
# Pattern cơ bản
def find_max(numbers):
    """Tìm số lớn nhất"""
    if not numbers:
        return None

    max_num = numbers[0]  # Giả sử phần tử đầu là max

    for num in numbers[1:]:
        if num > max_num:
            max_num = num

    return max_num

# Hoặc dùng built-in
def find_max_v2(numbers):
    return max(numbers) if numbers else None

# Test
numbers = [3, 7, 2, 9, 1, 5]
print(find_max(numbers))  # 9
```

3.2. Tìm Element Có Thuộc Tính Max

```
# Tìm student có điểm cao nhất
def find_best_student(students):
    """
    Args:
        students (list): [("name", score), ...]

    Returns:
        tuple: Student có điểm cao nhất
    """
    if not students:
        return None

    best_student = students[0]
    max_score = students[0][1]

    for student in students[1:]:
        name, score = student
        if score > max_score:
            max_score = score
            best_student = student

    return best_student

# Hoặc dùng max() với key
```

```python
def find_best_student_v2(students):
    if not students:
        return None
    return max(students, key=lambda x: x[1])

# Test
students = [("An", 8), ("Bình", 9), ("Chi", 7)]
print(find_best_student(students))  # ('Bình', 9)
```

3.3. Tìm Min/Max Với Điều Kiện

Pattern: Khởi tạo None, update khi thỏa điều kiện

```python
def find_max_even(numbers):
    """
    Tìm số chẵn lớn nhất

    Returns:
        int: Số chẵn lớn nhất, hoặc None nếu không có
    """
    max_even = None

    for num in numbers:
        if num % 2 == 0:  # Điều kiện: số chẵn
            if max_even is None or num > max_even:
                max_even = num

    return max_even

# Test
numbers = [1, 3, 8, 5, 2, 9, 6]
print(find_max_even(numbers))  # 8

numbers = [1, 3, 5, 7]
print(find_max_even(numbers))  # None (không có số chẵn)
```

Ví dụ thực tế: Tìm string dài nhất bắt đầu bằng 'A'

```python
def longest_word_starting_with_a(words):
    """
    Tìm từ dài nhất bắt đầu bằng 'A'

    Returns:
        str: Từ dài nhất, hoặc None
    """
    longest = None
    max_length = 0
```

```
    for word in words:
        if word.startswith('A'):  # Điều kiện
            if len(word) > max_length:
                max_length = len(word)
                longest = word

    return longest

# Test
words = ["Apple", "banana", "Apricot", "cherry", "Avocado"]
print(longest_word_starting_with_a(words))  # Avocado
```

3.4. Tìm Phần Tử Thỏa Nhiều Điều Kiện

```
def find_best_product(products):
    """
    Tìm sản phẩm tốt nhất:
    - Giá <= 100
    - Rating cao nhất

    Args:
        products (list): [(name, price, rating), ...]

    Returns:
        tuple: Sản phẩm tốt nhất
    """
    best_product = None
    max_rating = 0

    for product in products:
        name, price, rating = product

        # Điều kiện 1: giá <= 100
        if price <= 100:
            # Điều kiện 2: rating cao nhất
            if rating > max_rating:
                max_rating = rating
                best_product = product

    return best_product

# Test
products = [
    ("Laptop", 1200, 4.5),
    ("Mouse", 30, 4.8),
    ("Keyboard", 80, 4.9),
    ("Monitor", 300, 4.7)
]

print(find_best_product(products))
# ('Keyboard', 80, 4.9)
```

3.5. Top K Elements

```
def find_top_k_students(students, k):
    """
    Tìm k học sinh có điểm cao nhất

    Args:
        students (list): [(name, score), ...]
        k (int): Số lượng

    Returns:
        list: Top k students
    """
    # Cách 1: Sort và lấy k đầu
    sorted_students = sorted(students, key=lambda x: x[1], reverse=True)
    return sorted_students[:k]

# Test
students = [
    ("An", 8.5),
    ("Bình", 9.0),
    ("Chi", 7.5),
    ("Dũng", 8.8),
    ("Em", 9.2)
]

top_3 = find_top_k_students(students, 3)
print(top_3)
# [('Em', 9.2), ('Bình', 9.0), ('Dũng', 8.8)]
```

---

4. So Sánh và Lọc Dữ Liệu

4.1. Lọc Với Một Điều Kiện

```
# Pattern cơ bản
def filter_even_numbers(numbers):
    """Lọc số chẵn"""
    result = []
    for num in numbers:
        if num % 2 == 0:
            result.append(num)
    return result

# List comprehension (Pythonic)
def filter_even_numbers_v2(numbers):
    return [num for num in numbers if num % 2 == 0]

# Test
```

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8]
print(filter_even_numbers(numbers))  # [2, 4, 6, 8]
```

4.2. Lọc Với Nhiều Điều Kiện

```
def filter_students(students, min_score=8.0, min_attendance=0.8):
    """
    Lọc học sinh thỏa:
    - Điểm >= min_score
    - Attendance >= min_attendance

    Args:
        students (list): [(name, score, attendance), ...]
    """
    result = []

    for student in students:
        name, score, attendance = student

        # Kiểm tra tất cả điều kiện
        if score >= min_score and attendance >= min_attendance:
            result.append(student)

    return result

# List comprehension
def filter_students_v2(students, min_score=8.0, min_attendance=0.8):
    return [s for s in students
            if s[1] >= min_score and s[2] >= min_attendance]

# Test
students = [
    ("An", 8.5, 0.9),
    ("Bình", 7.0, 0.95),
    ("Chi", 9.0, 0.7),
    ("Dũng", 8.2, 0.85)
]

eligible = filter_students(students)
print(eligible)
# [('An', 8.5, 0.9), ('Dũng', 8.2, 0.85)]
```

4.3. Lọc và Transform

```
def get_high_scorers_names(students, threshold=8.0):
    """
    Lấy tên học sinh có điểm >= threshold

    Args:
```

```
        students (list): [(name, score), ...]

    Returns:
        list: Danh sách tên
    """
    names = []
    for name, score in students:
        if score >= threshold:
            names.append(name)
    return names

# List comprehension
def get_high_scorers_names_v2(students, threshold=8.0):
    return [name for name, score in students if score >= threshold]

# Test
students = [("An", 8.5), ("Bình", 7.0), ("Chi", 9.0)]
print(get_high_scorers_names(students))  # ['An', 'Chi']
```

4.4. Phân Loại (Categorize)

```
def categorize_by_score(students):
    """
    Phân loại học sinh theo điểm

    Returns:
        dict: {"A": [...], "B": [...], "C": [...], "D": [...]}
    """
    categories = {"A": [], "B": [], "C": [], "D": []}

    for name, score in students:
        if score >= 9:
            categories["A"].append(name)
        elif score >= 8:
            categories["B"].append(name)
        elif score >= 6.5:
            categories["C"].append(name)
        else:
            categories["D"].append(name)

    return categories

# Test
students = [
    ("An", 8.5),
    ("Bình", 9.2),
    ("Chi", 7.0),
    ("Dũng", 6.0)
]

result = categorize_by_score(students)
```

```
print(result)
# {'A': ['Bình'], 'B': ['An'], 'C': ['Chi'], 'D': ['Dũng']}
```

4.5. So Sánh Hai Tập Dữ Liệu

```
def compare_lists(list1, list2):
    """
    So sánh 2 lists

    Returns:
        dict: {
            "in_both": [...],
            "only_in_first": [...],
            "only_in_second": [...]
        }
    """
    set1 = set(list1)
    set2 = set(list2)

    return {
        "in_both": list(set1 & set2),
        "only_in_first": list(set1 - set2),
        "only_in_second": list(set2 - set1)
    }

# Test
list1 = [1, 2, 3, 4, 5]
list2 = [3, 4, 5, 6, 7]

result = compare_lists(list1, list2)
print(result)
# {
#   'in_both': [3, 4, 5],
#   'only_in_first': [1, 2],
#   'only_in_second': [6, 7]
# }
```

---

5. Xử Lý Chuỗi Con (Substring/Prefix Matching)

5.1. Kiểm Tra Substring

```
# Built-in: 'in' operator
text = "Python Programming"
print("Python" in text)  # True
print("Java" in text)    # False

# Tìm vị trí substring
pos = text.find("Programming")
```

```
print(pos)  # 7

# Đếm số lần xuất hiện
text = "hello hello world"
count = text.count("hello")
print(count)  # 2
```

5.2. Prefix Matching (Bài 2 & 5 Assessment)

```
def starts_with_prefix(text, prefix):
    """Kiểm tra text bắt đầu bằng prefix"""
    return text.startswith(prefix)

# Hoặc dùng slicing
def starts_with_prefix_v2(text, prefix):
    return text[:len(prefix)] == prefix

# Test
print(starts_with_prefix("Python", "Py"))    # True
print(starts_with_prefix("Python", "Java"))  # False
```

5.3. Tìm Tất Cả Strings Có Prefix

```
def find_words_with_prefix(words, prefix):
    """
    Tìm tất cả từ bắt đầu bằng prefix

    Returns:
        list: Danh sách từ thỏa mãn
    """
    result = []
    for word in words:
        if word.startswith(prefix):
            result.append(word)
    return result

# List comprehension
def find_words_with_prefix_v2(words, prefix):
    return [word for word in words if word.startswith(prefix)]

# Test
words = ["apple", "application", "banana", "apricot"]
print(find_words_with_prefix(words, "app"))
# ['apple', 'application']
```

5.4. Longest Common Prefix

```python
def longest_common_prefix(strings):
    """
    Tìm prefix chung dài nhất

    Args:
        strings (list): Danh sách strings

    Returns:
        str: Prefix chung dài nhất
    """
    if not strings:
        return ""

    # Lấy string ngắn nhất làm base
    shortest = min(strings, key=len)

    for i in range(len(shortest)):
        char = shortest[i]

        # Kiểm tra tất cả strings có char này ở vị trí i không
        for string in strings:
            if string[i] != char:
                return shortest[:i]

    return shortest

# Test
strings = ["flower", "flow", "flight"]
print(longest_common_prefix(strings))  # "fl"

strings = ["dog", "racecar", "car"]
print(longest_common_prefix(strings))  # "" (không có prefix chung)
```

5.5. Substring Matching (Pattern)

```python
def find_all_substrings(text, pattern):
    """
    Tìm tất cả vị trí xuất hiện của pattern trong text

    Returns:
        list: Danh sách indices
    """
    indices = []
    start = 0

    while True:
        pos = text.find(pattern, start)
        if pos == -1:
            break
        indices.append(pos)
```

```
        start = pos + 1

    return indices

# Test
text = "hello hello world hello"
pattern = "hello"
print(find_all_substrings(text, pattern))  # [0, 6, 18]
```

5.6. Ký Tự Chung Giữa Hai Strings (Bài 2 Assessment)

```
def common_characters(s1, s2):
    """
    Tìm ký tự chung giữa 2 strings

    Returns:
        set: Các ký tự chung
    """
    return set(s1) & set(s2)

def count_common_characters(s1, s2):
    """Đếm số ký tự chung (unique)"""
    return len(set(s1) & set(s2))

# Test
s1 = "hello"
s2 = "world"
print(common_characters(s1, s2))       # {'l', 'o'}
print(count_common_characters(s1, s2)) # 2
```

5.7. Most Similar Strings (Bài 2 Assessment - QUAN TRỌNG)

```
def most_similar_strings(strings):
    """
    Tìm 2 chuỗi có nhiều ký tự chung nhất

    Args:
        strings (list): Danh sách strings

    Returns:
        tuple: (str1, str2, common_count)
    """
    max_common = 0
    result = None

    # So sánh từng cặp
    for i in range(len(strings)):
        for j in range(i + 1, len(strings)):
            s1, s2 = strings[i], strings[j]
```

```
            # Đếm ký tự chung (unique)
            common = set(s1) & set(s2)
            num_common = len(common)

            if num_common > max_common:
                max_common = num_common
                result = (s1, s2, num_common)

    return result if result else (None, None, 0)

# Test
strings = ["hello", "world", "help", "hero"]
s1, s2, count = most_similar_strings(strings)
print(f"Most similar: '{s1}' and '{s2}' ({count} common chars)")
# Most similar: 'hello' and 'help' (4 common chars)

# Giải thích: "hello" và "help" có chung 'h', 'e', 'l', 'p' = 4 ký tự unique
```

---

6. Bài Tập Tổng Hợp

Bài 1: Tìm Student Có Môn Điểm Cao Nhất (Bài 3 Assessment)

```
def highest_score_subject(student_id, scores_dict):
    """
    Tìm môn học có điểm cao nhất của sinh viên

    Args:
        student_id (str): Mã sinh viên
        scores_dict (dict): {student_id: {subject: score}}

    Returns:
        str: Tên môn học
    """
    # Kiểm tra student tồn tại
    if student_id not in scores_dict:
        return None

    subjects = scores_dict[student_id]
    if not subjects:
        return None

    # Tìm môn có điểm cao nhất
    max_subject = None
    max_score = -1

    for subject, score in subjects.items():
        if score > max_score:
            max_score = score
            max_subject = subject
```

```
    return max_subject

# Hoặc dùng max()
def highest_score_subject_v2(student_id, scores_dict):
    if student_id not in scores_dict:
        return None

    subjects = scores_dict[student_id]
    if not subjects:
        return None

    return max(subjects, key=subjects.get)

# Test
scores = {
    "S001": {"Math": 85, "Physics": 92, "Chemistry": 78},
    "S002": {"Math": 88, "Physics": 84, "Chemistry": 90}
}

print(highest_score_subject("S001", scores))  # Physics
print(highest_score_subject("S002", scores))  # Chemistry
```

Bài 2: Validate Email

```
def is_valid_email(email):
    """
    Kiểm tra email hợp lệ:
    - Có đúng 1 @
    - Có domain sau @
    - Domain có ít nhất 1 dấu chấm

    Returns:
        bool: True nếu hợp lệ
    """
    # Kiểm tra có @ không
    if email.count('@') != 1:
        return False

    # Tách username và domain
    parts = email.split('@')
    username, domain = parts[0], parts[1]

    # Username không rỗng
    if not username:
        return False

    # Domain phải có ít nhất 1 dấu chấm
    if '.' not in domain:
        return False
```

```
    # Domain không bắt đầu/kết thúc bằng dấu chấm
    if domain.startswith('.') or domain.endswith('.'):
        return False

    return True

# Test
print(is_valid_email("user@example.com"))     # True
print(is_valid_email("user@example"))         # False
print(is_valid_email("user@@example.com"))    # False
print(is_valid_email("@example.com"))         # False
```

Bài 3: Find Duplicates

```
def find_duplicates(items):
    """
    Tìm các phần tử xuất hiện > 1 lần

    Returns:
        list: Các phần tử trùng lặp
    """
    seen = set()
    duplicates = set()

    for item in items:
        if item in seen:
            duplicates.add(item)
        else:
            seen.add(item)

    return list(duplicates)

# Test
numbers = [1, 2, 3, 2, 4, 1, 5, 3]
print(find_duplicates(numbers))  # [1, 2, 3]
```

Bài 4: Group By First Letter

```
def group_by_first_letter(words):
    """
    Nhóm từ theo chữ cái đầu

    Returns:
        dict: {letter: [words]}
    """
    groups = {}
```

```
    for word in words:
        first_letter = word[0].upper()

        if first_letter not in groups:
            groups[first_letter] = []

        groups[first_letter].append(word)

    return groups

# Hoặc dùng defaultdict
from collections import defaultdict

def group_by_first_letter_v2(words):
    groups = defaultdict(list)
    for word in words:
        groups[word[0].upper()].append(word)
    return dict(groups)

# Test
words = ["apple", "banana", "apricot", "cherry", "avocado"]
result = group_by_first_letter(words)
print(result)
# {
#   'A': ['apple', 'apricot', 'avocado'],
#   'B': ['banana'],
#   'C': ['cherry']
# }
```

Bài 5: Sliding Window - Tìm Subarray Có Tổng Lớn Nhất

```
def max_sum_subarray(arr, k):
    """
    Tìm subarray liên tiếp có k phần tử với tổng lớn nhất

    Args:
        arr (list): Mảng số
        k (int): Kích thước subarray

    Returns:
        int: Tổng lớn nhất
    """
    if len(arr) < k:
        return None

    # Tính tổng k phần tử đầu
    window_sum = sum(arr[:k])
    max_sum = window_sum
```

```
    # Trượt window
    for i in range(k, len(arr)):
        # Thêm phần tử mới, bỏ phần tử cũ
        window_sum = window_sum - arr[i - k] + arr[i]
        max_sum = max(max_sum, window_sum)

    return max_sum

# Test
arr = [1, 4, 2, 10, 23, 3, 1, 0, 20]
k = 4
print(max_sum_subarray(arr, k))  # 39 (10 + 23 + 3 + 1)
```

Bài 6: Two Pointers - Remove Duplicates (In-place)

```
def remove_duplicates_inplace(arr):
    """
    Xóa phần tử trùng lặp trong sorted array (in-place)

    Returns:
        int: Độ dài mảng mới
    """
    if not arr:
        return 0

    # Write pointer
    write = 1

    for read in range(1, len(arr)):
        if arr[read] != arr[read - 1]:
            arr[write] = arr[read]
            write += 1

    return write

# Test
arr = [1, 1, 2, 2, 3, 4, 4, 5]
length = remove_duplicates_inplace(arr)
print(f"Độ dài mới: {length}")  # 5
print(f"Mảng: {arr[:length]}")  # [1, 2, 3, 4, 5]
```

Bài 7: Palindrome Check

```
def is_palindrome(text):
    """
    Kiểm tra chuỗi có phải palindrome không
    (Bỏ qua spaces và case)
```

```
    Returns:
        bool: True nếu là palindrome
    """
    # Chuẩn hóa: lowercase, remove spaces
    cleaned = text.lower().replace(" ", "")

    # So sánh với reversed
    return cleaned == cleaned[::-1]

# Hoặc dùng two pointers
def is_palindrome_v2(text):
    cleaned = text.lower().replace(" ", "")
    left, right = 0, len(cleaned) - 1

    while left < right:
        if cleaned[left] != cleaned[right]:
            return False
        left += 1
        right -= 1

    return True

# Test
print(is_palindrome("A man a plan a canal Panama"))  # True
print(is_palindrome("racecar"))                      # True
print(is_palindrome("hello"))                        # False
```

Bài 8: Frequency Counter Pattern

```
def can_form_pairs(arr):
    """
    Kiểm tra có thể ghép tất cả phần tử thành cặp không
    (Mỗi số phải xuất hiện số lần chẵn)

    Returns:
        bool: True nếu có thể ghép cặp
    """
    freq = {}

    # Đếm tần suất
    for num in arr:
        freq[num] = freq.get(num, 0) + 1

    # Kiểm tra tất cả số chẵn lần
    for count in freq.values():
        if count % 2 != 0:
            return False

    return True
```

```
# Test
print(can_form_pairs([1, 2, 1, 2]))     # True
print(can_form_pairs([1, 2, 1, 2, 3]))  # False
```

---

7. Patterns Quan Trọng

7.1. Pattern: Track Min/Max While Iterating

```
def find_min_max(numbers):
    """Tìm min và max trong 1 lần duyệt"""
    if not numbers:
        return None, None

    min_num = max_num = numbers[0]

    for num in numbers[1:]:
        if num < min_num:
            min_num = num
        if num > max_num:
            max_num = num

    return min_num, max_num

# Test
numbers = [3, 7, 2, 9, 1, 5]
print(find_min_max(numbers))  # (1, 9)
```

7.2. Pattern: Early Return

```
def has_duplicate(items):
    """Kiểm tra có phần tử trùng lặp không"""
    seen = set()

    for item in items:
        if item in seen:
            return True  # Early return khi tìm thấy
        seen.add(item)

    return False  # Không tìm thấy

# Test
print(has_duplicate([1, 2, 3, 2]))  # True
print(has_duplicate([1, 2, 3, 4]))  # False
```

7.3. Pattern: Accumulator

```python
def running_sum(numbers):
    """Tính tổng tích lũy"""
    result = []
    total = 0

    for num in numbers:
        total += num
        result.append(total)

    return result

# Test
numbers = [1, 2, 3, 4]
print(running_sum(numbers))  # [1, 3, 6, 10]
```

7.4. Pattern: Flag Variable

```python
def contains_consecutive_duplicates(items):
    """Kiểm tra có 2 phần tử liên tiếp giống nhau không"""
    found = False

    for i in range(len(items) - 1):
        if items[i] == items[i + 1]:
            found = True
            break

    return found

# Test
print(contains_consecutive_duplicates([1, 2, 2, 3]))  # True
print(contains_consecutive_duplicates([1, 2, 3, 4]))  # False
```

7.5. Pattern: Multiple Passes

```python
def normalize_then_filter(numbers, threshold):
    """
    Bước 1: Normalize về 0-100
    Bước 2: Lọc >= threshold
    """
    # Pass 1: Normalize
    min_num = min(numbers)
    max_num = max(numbers)
    range_num = max_num - min_num

    normalized = []
    for num in numbers:
        norm = ((num - min_num) / range_num) * 100
        normalized.append(norm)
```

```
    # Pass 2: Filter
    filtered = []
    for norm in normalized:
        if norm >= threshold:
            filtered.append(norm)

    return filtered

# Test
numbers = [10, 20, 30, 40, 50]
result = normalize_then_filter(numbers, 50)
print(result)  # [50.0, 75.0, 100.0]
```

---

8. Tổng Kết và Checklist

☑ Kỹ năng cần nắm vững

Tìm kiếm:

- ☐ Linear search - duyệt từng phần tử
- ☐ Tìm với điều kiện
- ☐ Tìm tất cả vị trí
- ☐ Kiểm tra tồn tại

Min/Max:

- ☐ Tìm min/max đơn giản
- ☐ Tìm element có thuộc tính min/max
- ☐ Tìm min/max với điều kiện
- ☐ Top K elements

Lọc và so sánh:

- ☐ Lọc với 1 điều kiện
- ☐ Lọc với nhiều điều kiện
- ☐ Lọc và transform
- ☐ Phân loại (categorize)
- ☐ So sánh 2 tập dữ liệu

String:

- ☐ Substring check
- ☐ Prefix/suffix matching
- ☐ Tìm common characters
- ☐ Most similar strings
- ☐ Longest common prefix

🎯 Patterns quan trọng

1. **Track min/max while iterating** - Theo dõi trong 1 lần duyệt
2. **Early return** - Return ngay khi tìm thấy
3. **Accumulator** - Tích lũy kết quả
4. **Flag variable** - Biến đánh dấu
5. **Multiple passes** - Nhiều lần duyệt

Chiến lược giải bài

1. **Đọc kỹ đề** - Hiểu input, output, constraints
2. **Tìm pattern** - Bài này giống pattern nào?
3. **Viết pseudocode** - Các bước cần làm
4. **Implement từng bước** - Không cố viết một lần
5. **Test với ví dụ nhỏ** - Edge cases

---

9. Câu Hỏi Thường Gặp (FAQ)

Q1: Khi nào dùng built-in vs tự viết?

A:

- **Dùng built-in** khi có sẵn (max, min, sorted)
- **Tự viết** khi cần logic phức tạp hơn

```
# Built-in
max_num = max(numbers)

# Tự viết - cần điều kiện
max_even = None
for num in numbers:
    if num % 2 == 0:
        if max_even is None or num > max_even:
            max_even = num
```

Q2: List comprehension vs vòng lặp?

A:

- **Comprehension** cho filter/transform đơn giản
- **Vòng lặp** cho logic phức tạp

```
# ☑ Comprehension - đơn giản
evens = [x for x in numbers if x % 2 == 0]

# ☑ Loop - phức tạp
result = []
for student in students:
    if student.score >= 8 and student.attendance > 0.8:
```

```
        result.append({
            "name": student.name,
            "grade": "A" if student.score >= 9 else "B"
        })
```

Q3: Khởi tạo None vs 0 cho min/max?

A:

- **None** khi có thể không tìm thấy
- **First element** khi chắc chắn có kết quả

```
# None - có thể không tìm thấy
max_even = None
for num in numbers:
    if num % 2 == 0:
        if max_even is None or num > max_even:
            max_even = num

# First element - chắc chắn có
max_num = numbers[0]
for num in numbers[1:]:
    if num > max_num:
        max_num = num
```

Q4: Set vs List để track seen?

A:

- **Set** cho membership check (O(1))
- **List** khi cần thứ tự

```
# ☑ Set - nhanh
seen = set()
for item in items:
    if item in seen:  # O(1)
        return True
    seen.add(item)

# ❌ List - chậm
seen = []
for item in items:
    if item in seen:  # O(n)
        return True
    seen.append(item)
```

Q5: Khi nào dùng enumerate vs range?

A:

- **enumerate** khi cần cả index và value
- **range** khi chỉ cần index

```
# ☑ enumerate - cần cả 2
for i, item in enumerate(items):
    print(f"{i}: {item}")

# ☑ range - chỉ cần index
for i in range(len(items) - 1):
    if items[i] == items[i + 1]:
        print("Found duplicate")
```

---

10. Thử Thách Cuối Khóa

Challenge 1: Longest Substring Without Repeating

Tìm substring dài nhất không có ký tự lặp lại.

```
"""
Input: "abcabcbb"
Output: 3 ("abc")

Input: "bbbbb"
Output: 1 ("b")

Gợi ý: Sliding window + set
"""
```

Challenge 2: First Missing Positive

Tìm số nguyên dương nhỏ nhất không có trong mảng.

```
"""
Input: [3, 4, -1, 1]
Output: 2

Input: [1, 2, 0]
Output: 3

Gợi ý: Dùng set cho O(1) lookup
"""
```

Challenge 3: Subarray Sum Equals K

Đếm số subarray có tổng = k.

```
"""
Input: arr = [1, 1, 1], k = 2
Output: 2 (arr[0:2], arr[1:3])

Gợi ý: Prefix sum + hashmap
"""
```

Challenge 4: Valid Parentheses

Kiểm tra dấu ngoặc hợp lệ.

```
"""
Input: "()[]{}
Output: True

Input: "([)]"
Output: False

Gợi ý: Dùng stack (list)
"""
```
