

Họ và tên : Lý Gia Huy

BTVN : sesion 5

Lớp : CNTT 5

BÀI TẬP 1 :

1. Prompt 1 (chung chung)

Prompt:

“Giải thích cấu trúc dữ liệu Stack.”

Câu trả lời của AI (tóm tắt):

Stack là một cấu trúc dữ liệu tuyến tính hoạt động theo nguyên tắc LIFO (Last In, First Out). Stack cho phép hai thao tác chính là push (thêm phần tử) và pop (lấy phần tử). Stack thường được sử dụng trong quản lý bộ nhớ, xử lý lời gọi hàm và các thuật toán khác.

2. Prompt 2 (rõ ràng, có ngữ cảnh)

Prompt:

“Tôi chưa biết gì về cấu trúc dữ liệu Stack. Hãy giải thích đơn giản kèm ví dụ đời thực (ví dụ: chồng đĩa).”

Câu trả lời của AI (tóm tắt):

Stack có thể hiểu giống như một chồng đĩa xếp lênh nhau. Khi bạn đặt thêm một chiếc đĩa, bạn đặt lên trên cùng (push). Khi lấy đĩa ra, bạn chỉ có thể lấy chiếc ở trên cùng trước (pop). Vì vậy, chiếc đĩa được đặt vào sau cùng sẽ được lấy ra đầu tiên. Trong lập trình, Stack cũng hoạt động như vậy, dữ liệu vào sau sẽ được lấy ra trước.

3. Bảng so sánh hai câu trả lời

Tiêu chí	Prompt chung chung	Prompt rõ ràng, có ngữ cảnh
Đối tượng người học	Không xác định trình độ	Người mới bắt đầu, chưa biết gì

Họ và tên : Lý Gia Huy

BTVN : sesion 5

Lớp : CNTT 5

Cách diễn đạt	Mang tính học thuật, khái quát	Đơn giản, gần gũi
Ví dụ minh họa	Không có	Có ví dụ đời thực (chồng dĩa)
Mức độ dễ hiểu	Khó với người mới	Dễ hiểu, dễ hình dung
Hiệu quả học tập	Phù hợp người đã có nền tảng	Phù hợp người mới học

4. Nhận xét

Prompt thứ hai giúp AI đưa ra câu trả lời dễ hiểu hơn vì đã **xác định rõ đối tượng người học** và **yêu cầu ví dụ cụ thể**. Nhờ đó, kiến thức trừu tượng như Stack trở nên gần gũi và dễ tiếp cận hơn. Qua bài tập này, có thể thấy rằng việc viết prompt rõ ràng, có ngữ cảnh sẽ giúp AI hỗ trợ học tập hiệu quả hơn rất nhiều so với prompt chung chung.

BÀI TẬP 2 :

1. Prompt ban đầu

Prompt:

“So sánh Bubble Sort, Merge Sort và Quick Sort theo tiêu chí: độ phức tạp thời gian, độ phức tạp bộ nhớ, và tình huống áp dụng.”

Câu trả lời của AI (dạng văn bản – tóm tắt):

Bubble Sort là thuật toán sắp xếp đơn giản, dễ hiểu nhưng có độ phức tạp thời gian cao nên chỉ phù hợp với dữ liệu nhỏ. Merge Sort có hiệu suất ổn định với độ phức tạp thời gian tốt, phù hợp với tập dữ liệu lớn nhưng tốn thêm bộ nhớ. Quick Sort thường rất nhanh trong thực tế, nhưng trong trường hợp xấu nhất có thể chậm hơn.

2. Prompt yêu cầu lại (định dạng bảng)

Prompt:

“Hãy trình bày kết quả so sánh Bubble Sort, Merge Sort và Quick Sort dưới dạng bảng theo các tiêu chí: độ phức tạp thời gian, độ phức tạp bộ nhớ, và tình huống áp dụng.”

Họ và tên : Lý Gia Huy

BTVN : sesion 5

Lớp : CNTT 5

3. Kết quả trả lời của AI (dạng bảng)

Thuật toán	Độ phức tạp thời gian	Độ phức tạp bộ nhớ	Tình huống áp dụng
Bubble Sort	$O(n^2)$ trong mọi trường hợp	$O(1)$	Phù hợp để học thuật toán cơ bản, dữ liệu rất nhỏ
Merge Sort	$O(n \log n)$ trong mọi trường hợp	$O(n)$	Dữ liệu lớn, cần hiệu suất ổn định
Quick Sort	Trung bình $O(n \log n)$, xấu nhất $O(n^2)$	$O(\log n)$	Thực tế chạy nhanh, dùng phổ biến trong thư viện chuẩn

4. Nhận xét

Khi chỉ yêu cầu so sánh chung, AI thường trả lời dưới dạng văn bản dài, khó quan sát và so sánh trực tiếp. Việc bổ sung yêu cầu “**trình bày dưới dạng bảng**” giúp câu trả lời rõ ràng, dễ đọc và phù hợp cho việc học tập hoặc ghi chép. Qua bài tập này, có thể thấy rằng prompt càng cụ thể về **tiêu chí và định dạng**, thì kết quả AI trả về càng hữu ích và dễ sử dụng.

BÀI TẬP 3 :

Prompt đã sử dụng

Prompt:

“Viết ví dụ Python minh họa nguyên lý kế thừa (Inheritance) trong OOP với lớp Animal và Dog. Giải thích từng dòng code.”

Đoạn code Python minh họa

```
# Định nghĩa lớp cha Animal  
class Animal:
```

Họ và tên : Lý Gia Huy

BTVN : sesion 5

Lớp : CNTT 5

```
def __init__(self, name):
    self.name = name

def speak(self):
    print("Animal makes a sound")

# Định nghĩa lớp con Dog kế thừa từ Animal
class Dog(Animal):
    def __init__(self, name):
        # Gọi constructor của lớp cha
        super().__init__(name)

    def speak(self):
        print("Dog barks")

# Tạo đối tượng từ lớp Animal
animal = Animal("Generic Animal")
animal.speak()

# Tạo đối tượng từ lớp Dog
dog = Dog("Buddy")
dog.speak()
```

Giải thích từng phần code

- **class Animal:**
Khai báo một lớp tên là Animal, đóng vai trò là **lớp cha**.
- **def __init__(self, name):**
Hàm khởi tạo (constructor), được gọi khi tạo một đối tượng mới từ lớp Animal.
- **self.name = name**
Lưu tên của con vật vào thuộc tính name của đối tượng.
- **def speak(self):**
Phương thức mô tả hành động chung của các loài động vật.

Họ và tên : Lý Gia Huy

BTVN : sesion 5

Lớp : CNTT 5

- `class Dog(Animal):`
Khai báo lớp Dog **kế thừa** từ lớp Animal, nghĩa là Dog có thể sử dụng các thuộc tính và phương thức của Animal.
- `super().__init__(name)`
Gọi lại constructor của lớp cha để khởi tạo thuộc tính name.
- `def speak(self):`
Ghi đè (override) phương thức speak của lớp Animal để thể hiện hành vi riêng của chó.
- `animal = Animal("Generic Animal")`
Tạo một đối tượng từ lớp Animal.
- `dog = Dog("Buddy")`
Tạo một đối tượng từ lớp Dog.
- `animal.speak() và dog.speak()`
Gọi phương thức speak để thấy sự khác nhau giữa lớp cha và lớp con.

Nhận xét

AI đã tạo đoạn code minh họa đúng nguyên lý kế thừa trong OOP và giải thích rõ từng phần. Việc yêu cầu “**giải thích từng dòng code**” giúp người học không chỉ sao chép mà còn hiểu được mối quan hệ giữa lớp cha và lớp con, cũng như cách ghi đè phương thức trong Python. Điều này cho thấy prompt càng cụ thể thì AI hỗ trợ học tập càng hiệu quả.

BÀI TẬP 4:

1. Giải thích Breadth First Search (BFS) theo từng đối tượng

(1) Dành cho học sinh lớp 10

Breadth First Search là cách tìm kiếm theo từng lớp. Có thể tưởng tượng bạn đang đứng ở một điểm và muốn thăm tất cả các điểm xung quanh trước, rồi mới đi xa hơn. Ví dụ, khi tìm bạn bè trên mạng xã hội, bạn sẽ xem bạn trực tiếp của mình trước, sau đó mới xem bạn của bạn. BFS luôn đi từng vòng gần nhất rồi mới mở rộng ra các vòng xa hơn.

Họ và tên : Lý Gia Huy

BTVN : sesion 5

Lớp : CNTT 5

(2) Dành cho sinh viên năm 1 CNTT

Breadth First Search (BFS) là một thuật toán duyệt đồ thị hoặc cây theo chiều rộng. Thuật toán bắt đầu từ một đỉnh gốc, sau đó duyệt tất cả các đỉnh kề với nó trước khi chuyển sang các đỉnh ở mức sâu hơn. BFS thường sử dụng cấu trúc dữ liệu hàng đợi (queue) để đảm bảo thứ tự duyệt. Thuật toán này được ứng dụng trong việc tìm đường đi ngắn nhất trong đồ thị không có trọng số.

(3) Dành cho lập trình viên có 5 năm kinh nghiệm

Breadth First Search là thuật toán duyệt đồ thị theo lớp (level-order traversal) với độ phức tạp thời gian $O(V + E)$. BFS sử dụng queue để quản lý frontier và đảm bảo duyệt theo thứ tự tăng dần khoảng cách từ node nguồn. Thuật toán này đặc biệt hữu ích trong các bài toán shortest path trên unweighted graph, crawl dữ liệu, hoặc phân tích dependency. BFS đảm bảo tính tối ưu về số cạnh đi qua nhưng đánh đổi bằng bộ nhớ do phải lưu toàn bộ frontier.

2. So sánh cách giải thích theo cấp độ

Tiêu chí	Học sinh lớp 10	Sinh viên năm 1	Lập trình viên 5 năm
Ngôn ngữ	Đơn giản, đời thường	Có thuật ngữ cơ bản	Thuật ngữ chuyên sâu
Mức độ chi tiết	Ít, tập trung ý chính	Vừa phải	Chi tiết, kỹ thuật
Ví dụ	Gần gũi, trực quan	Ví dụ học thuật	Ví dụ ứng dụng thực tế
Yêu cầu kiến thức nền	Không cần	Có kiến thức lập trình	Có kinh nghiệm

Họ và tên : Lý Gia Huy

BTVN : sesion 5

Lớp : CNTT 5

3. Nhận xét cá nhân

Cách giải thích dành cho **sinh viên năm 1 CNTT** là phù hợp nhất với tôi vì vừa có thuật ngữ chuyên môn, vừa giải thích rõ cách hoạt động của BFS mà không quá phức tạp. Cách giải thích cho học sinh lớp 10 thì dễ hiểu nhưng thiếu chiều sâu, còn cách dành cho lập trình viên có kinh nghiệm lại hơi nặng về kỹ thuật. Điều này cho thấy việc yêu cầu AI giải thích đúng cấp độ người học giúp việc tiếp thu kiến thức hiệu quả hơn rất nhiều.

BÀI TẬP 5:

1. Prompt đã sử dụng

Prompt:

“So sánh Array và Linked List theo tốc độ truy cập, bộ nhớ, tình huống áp dụng. Sau đó đưa code minh họa thao tác chèn phần tử trong mỗi cấu trúc.”

2. Bảng so sánh Array và Linked List

Tiêu chí	Array	Linked List
Tốc độ truy cập	O(1) theo chỉ số	O(n), phải duyệt từng node
Bộ nhớ	Ít overhead, dùng liên tục	Tốn thêm bộ nhớ cho con trỏ
Chèn phần tử	Chậm nếu chèn giữa (O(n))	Nhanh nếu đã có vị trí (O(1))
Tình huống áp dụng	Khi cần truy cập ngẫu nhiên nhanh	Khi cần chèn/xóa thường xuyên

3. Code minh họa chèn phần tử

3.1 Chèn phần tử vào Array (Python)

```
arr = [1, 2, 3, 4]
```

```
# Chèn số 99 vào vị trí index = 2
arr.insert(2, 99)
```

Họ và tên : Lý Gia Huy

BTVN : sesion 5

Lớp : CNTT 5

```
print(arr)
```

Giải thích:

- `insert(2, 99)` chèn giá trị 99 vào vị trí thứ 2
- Các phần tử phía sau phải dời vị trí, nên tốn thời gian

3.2 Chèn phần tử vào Linked List (Python)

```
class Node:  
    def __init__(self, data):  
        self.data = data  
        self.next = None  
  
# Tạo danh sách liên kết: 1 -> 2 -> 4  
head = Node(1)  
head.next = Node(2)  
head.next.next = Node(4)  
  
# Chèn số 99 sau node có giá trị 2  
new_node = Node(99)  
new_node.next = head.next.next  
head.next.next = new_node
```

Giải thích:

- Tạo một node mới chứa giá trị 99
- Trỏ `next` của node mới tới node tiếp theo
- Cập nhật liên kết để chèn node vào danh sách
- Không cần dời các phần tử khác

Họ và tên : Lý Gia Huy

BTVN : sesion 5

Lớp : CNTT 5

4. Nhận xét

Việc kết hợp bảng so sánh với đoạn code minh họa giúp sự khác biệt giữa Array và Linked List trở nên rõ ràng hơn. Array dễ dùng và truy cập nhanh, nhưng chèn phần tử tốn chi phí. Ngược lại, Linked List phức tạp hơn về cấu trúc nhưng lại hiệu quả trong các bài toán cần chèn/xóa nhiều. Qua bài tập này, có thể thấy rằng việc yêu cầu AI vừa **so sánh lý thuyết** vừa **viết code minh họa** giúp người học hiểu sâu hơn thay vì chỉ ghi nhớ khái niệm.

BÀI TẬP 6 :

Prompt đã sử dụng

Prompt:

“Giải thích cách hoạt động của Queue bằng sơ đồ minh họa (ASCII art hoặc mô tả trực quan).”

Giải thích cách hoạt động của Queue

Queue (hàng đợi) là một cấu trúc dữ liệu hoạt động theo nguyên tắc **FIFO – First In, First Out**, nghĩa là **vào trước thì ra trước**. Có thể tưởng tượng Queue giống như một hàng người xếp hàng mua vé. Người đến trước sẽ được phục vụ trước, người đến sau phải đứng phía sau chờ tới lượt.

Queue có hai thao tác cơ bản:

- **Enqueue:** thêm phần tử vào cuối hàng đợi
- **Dequeue:** lấy phần tử ra ở đầu hàng đợi

Họ và tên : Lý Gia Huy

BTVN : sesion 5

Lớp : CNTT 5

Sơ đồ minh họa (ASCII art)

Trạng thái ban đầu (Queue rỗng)

FRONT -> | | | <- REAR

Thao tác Enqueue (thêm phần tử)

Thêm các phần tử A, B, C vào Queue:

FRONT -> | A | B | C | <- REAR

Giải thích:

- A được thêm đầu tiên nên đứng ở đầu hàng
- B và C được thêm sau, xếp phía sau A

Thao tác Dequeue (lấy phần tử ra)

Lấy phần tử đầu tiên (A) ra khỏi Queue:

FRONT -> | B | C | <- REAR

Giải thích:

- A bị lấy ra trước vì vào Queue sớm nhất
- B trở thành phần tử đứng đầu Queue

Họ và tên : Lý Gia Huy

BTVN : sesion 5

Lớp : CNTT 5

Nhận xét

Việc kết hợp sơ đồ trực quan giúp tôi dễ hình dung cách Queue hoạt động hơn so với chỉ đọc phần giải thích bằng chữ. Nhờ sơ đồ, tôi có thể thấy rõ vị trí **FRONT** và **REAR**, cũng như sự thay đổi của Queue sau mỗi thao tác enqueue và dequeue. Qua bài tập này, có thể thấy rằng yêu cầu AI tạo sơ đồ minh họa là một cách hiệu quả để hiểu các cấu trúc dữ liệu trừu tượng trong lập trình.

BÀI TẬP 7 :

[Bài Tập] GIẢI THÍCH THEO BỐI CẢNH KHÁC NHAU

1. Mục Tiêu:

- Hiểu được tầm quan trọng của “bối cảnh người học” khi đặt Prompt.
- Thực hành cùng một khái niệm nhưng với 2 bối cảnh khác nhau.
- Rèn kỹ năng so sánh, phân tích kết quả AI.

2. Mô Tả:

- Prompt 1: “Tôi là học sinh cấp 3, chưa biết gì về thuật toán sắp xếp. Hãy giải thích Quick Sort.”
- Prompt 2: “Tôi là lập trình viên junior, đã biết Bubble Sort. Hãy giải thích Quick Sort và cho ví dụ code Python.”
- So sánh sự khác biệt giữa 2 câu trả lời.
- Viết phân tích: AI thay đổi cách giải thích như thế nào khi bối cảnh khác nhau.

BÀI TẬP 8 :

Họ và tên : Lý Gia Huy

BTVN : sesion 5

Lớp : CNTT 5

Prompt 1 – Giải thích cơ bản cho người mới

Prompt 1:

“Hãy giải thích khái niệm Heap trong cấu trúc dữ liệu một cách đơn giản, phù hợp cho người mới bắt đầu học lập trình.”

Kết quả AI trả về (tóm tắt):

Heap là một cấu trúc dữ liệu dạng cây nhị phân đặc biệt, trong đó giá trị của nút cha luôn **lớn hơn hoặc nhỏ hơn** giá trị của các nút con, tùy theo loại Heap. Có hai loại Heap phổ biến là Max Heap (cha lớn hơn con) và Min Heap (cha nhỏ hơn con). Heap thường được dùng để tìm phần tử lớn nhất hoặc nhỏ nhất một cách nhanh chóng.

Prompt 2 – So sánh với khái niệm liên quan (bảng)

Prompt 2:

“So sánh Heap và Binary Search Tree (BST) theo các tiêu chí: cấu trúc, tốc độ tìm phần tử lớn nhất/nhỏ nhất, và tình huống áp dụng. Trình bày dưới dạng bảng.”

Kết quả AI trả về:

Tiêu chí	Heap	Binary Search Tree
Cấu trúc	Cây nhị phân gần đầy	Cây nhị phân có thứ tự
Phần tử lớn nhất/nhỏ nhất	Truy cập O(1)	O(log n) (trung bình)
Tìm kiếm phần tử bất kỳ	O(n)	O(log n)
Tình huống áp dụng	Priority Queue, Heap Sort	Tìm kiếm, lưu trữ có thứ tự

Prompt 3 – Ví dụ minh họa (code)

Prompt 3:

“Hãy viết ví dụ Python minh họa Min Heap và thao tác thêm phần tử vào Heap.”

Kết quả AI trả về:

Họ và tên : Lý Gia Huy

BTVN : sesion 5

Lớp : CNTT 5

```
import heapq

# Tạo một Min Heap rỗng
heap = []

# Thêm các phần tử vào heap
heapq.heappush(heap, 5)
heapq.heappush(heap, 2)
heapq.heappush(heap, 8)

print(heap)          # Heap sau khi thêm
print(heap[0])       # Phần tử nhỏ nhất
```

AI giải thích rằng heapq trong Python luôn tạo Min Heap, phần tử nhỏ nhất luôn nằm ở đầu danh sách.

Prompt 4 – Giải thích cho 2 đối tượng khác nhau

Prompt 4:

“Hãy giải thích Heap cho (1) người mới học lập trình và (2) sinh viên năm 3 CNTT.”

Kết quả AI trả về (tóm tắt):

- **Cho người mới:**

Heap giống như một cái tháp số, trong đó số nhỏ nhất (hoặc lớn nhất) luôn nằm trên cùng. Khi thêm số mới, hệ thống tự sắp xếp lại để đảm bảo thứ tự đó.

- **Cho sinh viên năm 3:**

Heap là một cấu trúc dữ liệu cây nhị phân hoàn chỉnh, thỏa mãn heap property. Heap hỗ trợ các thao tác insert và extract với độ phức tạp $O(\log n)$, thường được dùng trong Priority Queue và thuật toán Heap Sort.

Họ và tên : Lý Gia Huy

BTVN : sesion 5

Lớp : CNTT 5

Nhận xét chung

Qua 4 Prompt liên tiếp, có thể thấy rằng mỗi Prompt đào sâu khái niệm Heap theo một góc nhìn khác nhau: từ giải thích cơ bản, so sánh lý thuyết, minh họa bằng code, đến giải thích theo cấp độ người học. Việc thiết kế Prompt hỗn hợp giúp khai thác AI hiệu quả hơn và hỗ trợ học tập có định hướng, thay vì chỉ nhận câu trả lời đơn lẻ.

BÀI TẬP 9 :

Prompt đã đặt cho AI

Prompt 1:

“Hãy giải thích quá trình quang hợp của cây xanh theo từng bước chi tiết, giống như một chuỗi gọi hàm (step-by-step). Minh họa bằng sơ đồ ASCII thể hiện ánh sáng → diệp lục → phản ứng tạo năng lượng → tạo đường (glucose).”

Kết quả AI trả về – Giải thích theo từng bước (step-by-step)

Quá trình quang hợp là quá trình cây xanh sử dụng năng lượng ánh sáng để tạo ra chất dinh dưỡng cho mình. Nếu mô tả giống như truy vết lời gọi hàm, ta có thể chia thành các bước sau:

Bước 1: Nhận ánh sáng mặt trời

Lá cây hấp thụ ánh sáng mặt trời thông qua sắc tố diệp lục có trong lục lạp.

Bước 2: Kích hoạt diệp lục

Diệp lục sử dụng năng lượng ánh sáng để kích thích các phản ứng hóa học ban đầu.

Bước 3: Phản ứng tạo năng lượng

Nước (H_2O) bị phân tách, tạo ra năng lượng hóa học (ATP, NADPH) và giải phóng khí oxy (O_2).

Họ và tên : Lý Gia Huy

BTVN : sesion 5

Lớp : CNTT 5

Bước 4: Tổng hợp chất hữu cơ

Khí CO₂ kết hợp với năng lượng vừa tạo ra để tổng hợp thành glucose (C₆H₁₂O₆), là nguồn dinh dưỡng cho cây.

Sơ đồ minh họa quá trình (ASCII)

Ánh sáng mặt trời

|

v

Diệp lục

|

v

Phản ứng quang hóa
(tạo năng lượng)

|

v

CO₂ + H₂O

|

v

Glucose (đường)
+ O₂ (thải ra)

Nhận xét về cách trình bày từng bước

Việc yêu cầu AI giải thích theo kiểu “truy vết từng bước” giúp quá trình quang hợp trở nên rõ ràng và có logic hơn, giống như khi theo dõi từng dòng lệnh trong một chương trình. Sơ đồ ASCII giúp tôi dễ hình dung luồng diễn ra của quá trình thay vì chỉ đọc mô tả bằng chữ.

Họ và tên : Lý Gia Huy

BTVN : sesion 5

Lớp : CNTT 5

Prompt mở rộng do tự đặt thêm (ngoài lập trình)

Prompt mở rộng:

“Hãy giải thích quy trình kỹ thuật gấp máy bay giấy theo từng bước, giống như trace một thuật toán, kèm sơ đồ ASCII minh họa.”

Nhận xét chung

Qua bài tập này, tôi nhận ra kỹ thuật “truy vết từng bước” không chỉ áp dụng hiệu quả trong lập trình (debug, trace hàm), mà còn rất hữu ích khi học các **quy trình** trong nhiều lĩnh vực khác như sinh học, kỹ thuật hay đời sống hằng ngày. Khi biết cách đặt Prompt đúng, AI có thể trở thành công cụ giúp phân tích và hiểu sâu một quá trình phức tạp một cách có hệ thống.