BÁO CÁO VỀ VIỆC SỬ DỤNG PROMPT HIỆU QUẢ CHO CÁC TÁC VỤ HỌC TẬP

Nguyễn Hoàng Vũ

Table of contents

1 Giới thiệu

Báo cáo này được thực hiện nhằm phát triển kỹ năng viết prompt (câu lệnh) hiệu quả, từ đó tận dụng tối đa khả năng của các mô hình ngôn ngữ lớn (LLMs) để phục vụ cho việc học tập các môn chuyên ngành ở bậc đại học. Qua quá trình thử nghiệm với chủ đề **Độ phức tạp thời gian của thuật toán**, em mong muốn tìm ra công thức giao tiếp với AI sao cho kết quả trả về chính xác, giúp em hiểu sâu cách phân tích và đánh giá hiệu năng của các thuật toán.

2 Lựa chọn các tác vụ học tập để thử nghiệm prompt

Trong phần này, em đã chọn 3 tác vụ học tập phổ biến của sinh viên khoa CNTT liên quan đến chủ đề độ phức tạp thời gian và phân tích yêu cầu cho từng tác vụ đó.

Tác vụ	Mục tiêu	Đối tượng
Tác vụ 1: Giải thích khái niệm ký hiệu Big-O và các lớp độ phức tạp thời gian phổ biến.	Phải làm rõ được ý nghĩa toán học của Big-O và phân biệt được các lớp O(1), O(log n), O(n), O(n log n), $O(n^2)$ một cách trực quan.	Là sinh viên năm hai mới bắt đầu học môn Cấu trúc Dữ liệu và Giải thuật, cần giải thích dễ hiểu, có ví dụ minh họa cụ thể bằng ngôn ngữ lập trình Java.
Tác vụ 2: Phân tích độ phức tạp thời gian của một đoạn mã nguồn cho sẵn.	Phải xác định chính xác độ phức tạp của từng vòng lặp, hàm đệ quy hoặc khối lệnh lồng nhau và đưa ra kết luận Big-O tổng thể.	Yêu cầu AI đóng vai trò trợ giảng, giải thích từng bước phân tích để sinh viên có thể học được phương pháp, không chỉ nhận kết quả.
Tác vụ 3: Tạo một bộ câu hỏi ôn tập về phân tích độ phức tạp thời gian cho môn Cấu trúc Dữ liệu và Giải thuật.	Cần một bộ câu hỏi bao quát từ lý thuyết (định nghĩa, so sánh) đến thực hành (phân tích code, tối ưu hóa), có độ phân hóa rõ ràng theo mức độ khó.	Yêu cầu bộ câu hỏi phải rèn luyện tư duy phân tích độ phức tạp từ đoạn code thực tế, không chỉ kiểm tra thuộc lòng định nghĩa.

3 Xây dựng các phiên bản prompt cho từng tác vụ

Với các tác vụ ở trong phần I, em đã tiến hành xây dựng các prompt để thử nghiệm trên AI.

- **Tác vụ 1:**
  - Prompt cơ bản: “Big-O là gì?”
  - Prompt cải tiến: “Hãy giải thích ký hiệu Big-O và liệt kê các lớp độ phức tạp thời gian phổ biến từ nhanh đến chậm, kèm theo ví dụ cho mỗi loại.”
  - Prompt nâng cao: “Đóng vai một Giảng viên môn Cấu trúc Dữ liệu và Giải thuật với nhiều năm kinh nghiệm giảng dạy. Hãy giải thích ký hiệu Big-O cho một sinh viên năm hai chưa từng học về phân tích thuật toán. Yêu cầu cấu trúc trình bày: 1 đoạn giải thích ý nghĩa trực quan của Big-O bằng một phép ẩn dụ đời thường, 1 bảng so sánh các lớp độ phức tạp O(1), O(log n), O(n), O(n log n), $O(n^2)$, $O(2^n)$ với ví dụ mã Java tương ứng cho mỗi lớp, và 1 đoạn lưu ý những sai lầm phổ biến khi sinh viên mới học về Big-O.”
- **Tác vụ 2:**
  - Prompt cơ bản: “Đoạn code này có độ phức tạp bao nhiêu?”
  - Prompt cải tiến: “Hãy phân tích độ phức tạp thời gian của đoạn code Java dưới đây và giải thích tại sao.”
  - Prompt nâng cao: “Đóng vai một Trợ giảng môn Giải thuật kiên nhẫn và cẩn thận. Hãy phân tích độ phức tạp thời gian của đoạn code Java dưới đây theo từng bước:

1. Xác định và giải thích độ phức tạp của từng khối lệnh / vòng lặp riêng lẻ (có đánh số dòng tương ứng).
2. Áp dụng các quy tắc cộng và nhân để tính độ phức tạp tổng thể.
3. Kết luận Big-O cuối cùng và giải thích tại sao các số hạng bậc thấp hơn được bỏ qua.
4. Gợi ý (nếu có) cách tối ưu hóa thuật toán để cải thiện độ phức tạp."

- **Tác vụ 3:**
  - Prompt cơ bản: "Cho tôi một số câu hỏi ôn tập về Big-O."
  - Prompt cải tiến: "Hãy tạo 10 câu hỏi trắc nghiệm ôn tập về độ phức tạp thời gian của thuật toán. Mỗi câu cần có 4 đáp án A, B, C, D và in đậm đáp án đúng."
  - Prompt nâng cao: "Đóng vai người ra đề thi môn Cấu trúc Dữ liệu và Giải thuật. Tôi cần một bộ câu hỏi kiểm tra giữa kỳ về chủ đề Độ phức tạp Thời gian. Hãy tạo một bộ đề mô phỏng cấu trúc sau:
    - 3 câu trắc nghiệm lý thuyết (Mức độ Nhớ - định nghĩa Big-O, so sánh các lớp độ phức tạp).
    - 3 câu hỏi phân tích đoạn code Java ngắn (Mức độ Hiểu - xác định Big-O của đoạn code cho sẵn).
    - 1 bài tập tự luận (Mức độ Vận dụng - so sánh hai thuật toán sắp xếp về độ phức tạp và đề xuất lựa chọn theo từng trường hợp sử dụng cụ thể).

    Yêu cầu: Cung cấp đáp án chi tiết và lời giải từng bước ngay bên dưới mỗi câu hỏi."

4 Thử nghiệm và so sánh kết quả

Dựa vào các prompt đã được xây dựng ở trên, em tiến hành thử nghiệm các phiên bản prompt trên Google Gemini.

Cho tôi một số câu hỏi ôn tập về Big-O

Dưới đây là các câu hỏi ôn tập về độ phức tạp thuật toán (Big-O) giúp bạn củng cố kiến thức.

Câu hỏi lý thuyết

- Big-O là gì?
- Nó đo thời gian chạy chính xác hay xu hướng tăng trưởng?
- Vì sao ta bỏ qua hằng số trong Big-O?
- Độ phức tạp tốt nhất và xấu nhất khác nhau thế nào?
- Ký hiệu Big-O khác gì với Big-Omega và Big-Theta?

Sắp xếp độ phức tạp (Từ nhỏ đến lớn)

Hãy sắp xếp các độ phức tạp sau theo chiều tăng dần:

- $O(N \log N)$
- $O(1)$
- $O(N^2)$
- $O(\log N)$
- $O(N)$
- $O(2^N)$

Xác định Big-O của mã nguồn

Hãy tìm Big-O cho các đoạn mã dưới đây:

Đoạn mã 1:

python

```python
def print_items(n):
    for i in range(n):
        print(i)
```

Use code with caution.

Đoạn mã 2:

python

```python
def print_pairs(n):
    for i in range(n):
        for j in range(n):
            print(i, j)
```

Use code with caution.

AI Mode All Images Videos Shopping More

python

```python
def binary_search_style(n):
    i = n
    while i > 1:
        i = i // 2
        print(i)
```

Use code with caution.

Câu hỏi tình huống và cấu trúc dữ liệu

- Độ phức tạp khi tìm kiếm trên mảng đã sắp xếp là gì?
- Thêm một phần tử vào cuối mảng (Array) tốn bao lâu?
- Tìm một nút trong cây tìm kiếm nhị phân mất bao lâu?
- Độ phức tạp của thuật toán sắp xếp nhanh (Quick Sort) là gì?

Để tôi có thể hỗ trợ bạn tốt nhất, bạn muốn tiếp tục theo hướng nào? Nếu bạn muốn, tôi có thể:

- Cung cấp **đáp án và lời giải chi tiết** cho các câu hỏi trên.
- Đổi các câu hỏi này thành dạng **trắc nghiệm có đáp án (A, B, C, D)**.
- Tạo thêm các bài tập nâng cao về **độ phức tạp không gian (Space Complexity)**.

Hãy cho tôi biết **lựa chọn của bạn** nhé!

Hãy tạo 10 câu hỏi trắc nghiệm ôn tập về độ phức tạp thời gian của thuật toán. Mỗi câu cần có 4 đáp án A, B, C, D và in...

Chào bạn! Dưới đây là 10 câu hỏi trắc nghiệm giúp bạn ôn tập về độ phức tạp thời gian của thuật toán. Các câu hỏi được thiết kế từ mức độ cơ bản đến nâng cao để bạn dễ dàng hệ thống lại kiến thức.

Câu hỏi trắc nghiệm về độ phức tạp thời gian

Câu 1: Thuật toán tìm kiếm nhị phân (Binary Search) trên một mảng đã sắp xếp có độ phức tạp thời gian trong trường hợp xấu nhất là bao nhiêu?
A. $O(1)$
B. $\boldsymbol{O(\log n)}$
C. $O(n)$
D. $O(n \log n)$

Câu 2: Độ phức tạp thời gian của thuật toán sắp xếp nổi bọt (Bubble Sort) trong trường hợp xấu nhất là gì?
A. $O(n)$
B. $O(n \log n)$
C. $\boldsymbol{O(n^2)}$
D. $O(2^n)$

Câu 3: Nếu một thuật toán có hai vòng lặp lồng nhau, vòng lặp ngoài chạy n lần và vòng lặp trong cũng chạy n lần, thì độ phức tạp thời gian tổng thể là bao nhiêu?
A. $O(2n)$
B. $O(n \log n)$
C. $\boldsymbol{O(n^2)}$
D. $O(n^3)$

Câu 4: Ký hiệu Big-O (O) dùng để biểu diễn điều gì trong phân tích thuật toán?
A. Thời gian chạy trung bình của thuật toán.
B. **Giới hạn trên (trường hợp xấu nhất) của thời gian chạy thuật toán.**
C. Giới hạn dưới (trường hợp tốt nhất) của thời gian chạy thuật toán.
D. Bộ nhớ tối đa mà thuật toán cần dùng.

Câu 5: Thuật toán nào sau đây có độ phức tạp thời gian là $O(1)$?
A. Tìm phần tử lớn nhất trong một mảng chưa sắp xếp.
B. **Truy cập một phần tử trong mảng thông qua chỉ số (index).**
C. Duyệt qua tất cả các phần tử của một danh sách liên kết.
D. Sắp xếp một dãy số bằng thuật toán Quick Sort.

Câu 6: Thuật toán sắp xếp hòa trộn (Merge Sort) có độ phức tạp thời gian trong trường hợp xấu nhất là bao nhiêu?
A. $O(n)$
B. $\boldsymbol{O(n \log n)}$
C. $O(n^2)$
D. $O(\log n)$

Câu 7: Độ phức tạp thời gian nào sau đây tăng trưởng nhanh nhất (tệ nhất) khi kích

AI Mode All Images Videos Shopping More

C. $O(n^3)$
D. $\boldsymbol{O(2^n)}$

Câu 8: Một thuật toán thực hiện một công việc cố định, không phụ thuộc vào kích thước dữ liệu đầu vào n. Độ phức tạp của thuật toán này là gì?
A. $\boldsymbol{O(1)}$
B. $O(n)$
C. $O(\log n)$
D. Không xác định được.

Câu 9: Khi tính toán độ phức tạp thời gian Big-O cho biểu thức $T(n) = 5n^2 + 10n + 100$, kết quả chính xác là bao nhiêu?
A. $O(100)$
B. $O(10n)$
C. $\boldsymbol{O(n^2)}$
D. $O(5n^2)$

Câu 10: Hàm đệ quy tính số Fibonacci bằng cách gọi lại chính nó hai lần ở mỗi bước (không dùng nhớ/quy hoạch động) có độ phức tạp thời gian là bao nhiêu?
A. $O(n)$
B. $O(n^2)$
C. $O(\log n)$
D. $\boldsymbol{O(2^n)}$

Để việc ôn tập hiệu quả hơn, bạn có muốn tôi giải thích chi tiết **cách xác định** độ phức tạp cho một câu hỏi cụ thể nào ở trên không? Hoặc tôi có thể cung cấp thêm các câu hỏi về **độ phức tạp không gian (Space Complexity)** nếu bạn cần!

Prompt nâng cao: "Đóng vai người ra đề thi môn Cấu trúc Dữ liệu và Giải thuật. Tôi cần một đề cương bài kiểm tra giữa kỳ về ...

Chào bạn! Dưới đây là bộ đề thi giữa kỳ môn Cấu trúc Dữ liệu và Giải thuật về chủ đề **Độ phức tạp Thời gian (Time Complexity)**, được thiết kế đúng theo cấu trúc và mức độ bạn yêu cầu.

ĐỀ THI GIỮA KỲ: ĐỘ PHỨC TẠP THỜI GIAN

Phần 1: Câu hỏi trắc nghiệm lý thuyết (Mức độ Nhớ)

Câu 1: Định nghĩa nào sau đây mô tả đúng nhất về ký hiệu Big-O (O)?

- A. Đường giới hạn dưới của thời gian chạy thuật toán.
- B. Đường giới hạn trên của thời gian chạy thuật toán trong trường hợp xấu nhất.
- C. Thời gian chạy chính xác tính bằng giây của thuật toán.
- D. Số dòng code của một chương trình máy tính.

Đáp án đúng: B

- **Lời giải chi tiết:** Ký hiệu Big-O (O) được dùng để chỉ ra một đường giới hạn trên (upper bound) của một hàm số. Trong khoa học máy tính, nó dùng để biểu diễn tốc độ tăng tối đa của thời gian chạy thuật toán khi kích thước đầu vào n tiến ra vô cùng, tương ứng với trường hợp xấu nhất.

Câu 2: Sắp xếp các lớp độ phức tạp sau theo thứ tự tăng dần về tốc độ thực thi (từ nhanh nhất đến chậm nhất):

- A. $O(1) < O(\log n) < O(n) < O(n \log n) < O(n^2) < O(2^n)$
- B. $O(1) < O(n) < O(\log n) < O(n \log n) < O(n^2) < O(2^n)$
- C. $O(2^n) < O(n^2) < O(n \log n) < O(n) < O(\log n) < O(1)$
- D. $O(1) < O(\log n) < O(n \log n) < O(n) < O(n^2) < O(2^n)$

Đáp án đúng: A

- **Lời giải chi tiết:** Khi kích thước dữ liệu n lớn lên, hàm hằng số $O(1)$ không đổi nên nhanh nhất. Hàm lôga tăng rất chậm. Sắp theo là hàm tuyến tính ($O(n)$, hàm $O(n \log n)$, hàm bậc hai $O(n^2)$, và chậm nhất trong nhóm này là hàm mũ $O(2^n)$.

Câu 3: Khi phân tích độ phức tạp thời gian Big-O của một hàm $f(n) = 5n^2 + 10n + 100$, kết quả chính xác là gì?

- A. $O(100)$
- B. $O(10n)$
- C. $O(5n^2)$
- D. $O(n^2)$

Đáp án đúng: D

- **Lời giải chi tiết:** Theo quy tắc tính Big-O: (1) Chúng ta chỉ giữ lại số hạng có bậc cao nhất vì nó quyết định tốc độ tăng của hàm khi n rất lớn (ở đây là $5n^2$); (2) Loại bỏ các hằng số nhân phía trước số hạng đó (loại bỏ số 5). Vì vậy, kết quả rút gọn cuối cùng là $O(n^2)$.

Phần 2: Câu hỏi phân tích đoạn code Java (Mức độ Hiểu)

Câu 4: Hãy xác định độ phức tạp Big-O của đoạn code Java sau:

```java
void printItems(int n) {
    for (int i = 0; i < n; i++) {
        System.out.println(i);
    }
    for (int j = 0; j < n; j++) {
        System.out.println(j);
    }
}
```

Use code with caution.

- **Đáp án:** $O(n)$
- **Lời giải chi tiết:**
  - Vòng lặp `for` đầu tiên chạy từ 0 đến $n-1$, thực hiện n lần bước in.
  - Vòng lặp `for` thứ hai nằm độc lập (phía sau vòng lặp một), cũng chạy từ 0 đến $n-1$, thực hiện n lần bước in.
  - Tổng số lần thực hiện lệnh là $n + n = 2n$.
  - Áp dụng quy tắc bỏ hằng số trong Big-O, ta bỏ số 2. Kết quả cuối cùng là $O(n)$.

Câu 5: Hãy xác định độ phức tạp Big-O của đoạn code Java sau:

```java
void printPairs(int n) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            System.out.println(i + " " + j);
        }
    }
}
```

Use code with caution.

- **Đáp án:** $O(n^2)$
- **Lời giải chi tiết:**
  - Đây là dạng hai vòng lặp lồng nhau (nested loops).
  - Vòng lặp bên ngoài chạy n lần (từ i đến $n-1$).
  - Với mỗi lần vòng lặp ngoài chạy, vòng lặp bên trong lại chạy đủ n lần.
  - Tổng số lần lệnh in được thực thi là $n \times n = n^2$ lần. Do đó, độ phức tạp là $O(n^2)$.

Câu 6: Hãy xác định độ phức tạp Big-O của đoạn code Java sau:

```java
void divideAndConquer(int n) {
    while (n > 1) {
        n = n / 2;
        System.out.println(n);
    }
}
```

Use code with caution.

- **Đáp án:** $O(\log n)$
- **Lời giải chi tiết:**
  - Mỗi lần vòng lặp `while` chạy, giá trị của biến n lại bị giảm đi một nửa ($n = n / 2$).
  - Số lần vòng lặp chạy chính là số lần chúng ta có thể chia n cho 2 trước khi n giảm xuống bằng hoặc nhỏ hơn 1.
  - Phép toán ngược của việc chia đôi liên tiếp chính là logarit cơ số 2. Vì vậy, số vòng lặp sẽ tỷ lệ thuận với $\log_2(n)$. Trong Big-O, ta bỏ cơ số và viết gọn là $O(\log n)$.

Phần 3: Bài tập tự luận (Mức độ Vận dụng)

Câu 7 (Bài báo):

Hãy so sánh độ phức tạp thời gian trong trường hợp tốt nhất (Best case) và trường hợp xấu nhất (Worst case) của hai thuật toán: **Sắp xếp chèn (Insertion Sort)** và **Sắp xếp trộn (Merge Sort)**.

Dựa vào bảng so sánh đó, hãy đề xuất thuật toán tối ưu hơn cho hai trường hợp sử dụng cụ thể sau và giải thích lý do:

1. Trường hợp 1: Dữ liệu đầu vào là một mảng gồm các số gần như đã được sắp xếp sẵn (chỉ có 1 hoặc 2 phần tử sai vị trí).
2. Trường hợp 2: Dữ liệu đầu vào là một mảng cực kỳ lớn gồm hàng triệu phần tử lộn xộn ngẫu nhiên, và hệ thống yêu cầu thời gian xử lý ổn định, không được quá chậm trong mọi tình huống.

Đáp án chi tiết và lời giải từng bước:

Bước 1: Bảng so sánh độ phức tạp thời gian

Thuật toán	Trường hợp tốt nhất (Best Case)	Trường hợp xấu nhất (Worst Case)
Insertion Sort	$O(n)$	$O(n^2)$

AI Mode All Images Videos Shopping More

Giải quyết Trường hợp 1: Mảng gần như đã sắp xếp sẵn

- **Đề xuất lựa chọn:** Chọn thuật toán **Insertion Sort**.
- **Giải thích lý do:**
  - Khi mảng đã gần như sắp xếp xong, Insertion Sort chỉ cần thực hiện một lần duyệt qua mảng. Tại mỗi phần tử, lệnh so sánh sẽ nhận thấy phần tử đó đã đứng đúng vị trí và không cần dịch chuyển nhiều.
  - Độ phức tạp thực tế lúc này sẽ tiệm cận mức tốt nhất là $O(n)$.
  - Trong khi đó, Merge Sort luôn chia đôi mảng và trộn lại dù mảng có sẵn thứ tự hay chưa, nên nó vẫn tốn thời gian là $O(n \log n)$. Vì $n <$ nên Insertion Sort chạy nhanh hơn và hiệu quả hơn hẳn trong kịch bản này.

Giải quyết Trường hợp 2: Mảng cực lớn, lộn xộn và cần sự ổn định

- **Đề xuất lựa chọn:** Chọn thuật toán **Merge Sort**.
- **Giải thích lý do:**
  - Với mảng lớn xộn có kích thước hàng triệu phần tử (ví dụ tăng), nếu dùng Insertion Sort trong trường hợp xấu nhất, thời gian chạy sẽ tỷ lệ với n^2 (hàng nghìn tỷ phép tính). Hệ thống chắc chắn sẽ bị treo hoặc chạy rất chậm.
  - Merge Sort có một ưu điểm lớn là tính **ổn định và thời gian**. Dù dữ liệu đầu vào có bị lộn mức nào, độ phức tạp của nó luôn được đảm bảo ở mức $O(n \log n)$.
  - Với n bằng 1 triệu, $n \log n$ chỉ tương đương khoảng 20 triệu phép tính, nhỏ hơn rất nhiều so với n^2. Do đó, Merge Sort là lựa chọn an toàn và tối ưu cho dữ liệu lớn.

Nếu bạn muốn **thay đổi ngôn ngữ lập trình** (ví dụ sang C++ hoặc Python) hay muốn **bổ sung thêm câu hỏi** về chủ đề nào khác, hãy cho tôi biết nhé!

4.1 So sánh kết quả

Tác vụ	Cơ bản	Cải tiến	Nâng cao
Tác vụ 1	**Kết quả:** AI trả về định nghĩa toán học khô khan, chứa nhiều ký hiệu và thuật ngữ kỹ thuật như "giới hạn trên tiệm cận" khiến người mới học khó tiếp thu.	**Kết quả:** AI đưa ra giải thích dễ hiểu hơn và có kèm ví dụ, nhưng các ví dụ còn rời rạc, chưa liên kết được các lớp độ phức tạp với nhau một cách có hệ thống.	**Kĩ thuật:** Role prompting, constraints, structured output **Kết quả:** AI xây dựng phép ẩn dụ trực quan (ví dụ: tìm tên trong danh bạ điện thoại được sắp xếp vs. chưa sắp xếp) để minh họa sự khác biệt giữa O(1), O(log n) và O(n). Bảng so sánh đầy đủ giúp người học hình thành bức tranh tổng thể về các lớp độ phức tạp.
Tác vụ 2	**Kết quả:** AI chỉ đưa ra kết luận Big-O mà không giải thích quá trình suy luận, khiến sinh viên không học được phương pháp phân tích.	**Kết quả:** AI có giải thích, nhưng cách trình bày thiếu cấu trúc, đôi khi bỏ qua một số vòng lặp hoặc không làm rõ lý do bỏ qua các số hạng bậc thấp.	**Kĩ thuật:** Role prompting, chain-of-thought, constraints **Kết quả:** AI phân tích từng dòng code một cách có hệ thống, áp dụng rõ ràng quy tắc cộng và nhân để hợp nhất độ phức tạp từng khối, và chủ động gợi ý phương án tối ưu hóa. Sinh viên học được quy trình tư duy chứ không chỉ nhận đáp số.
Tác vụ 3	**Kết quả:** AI đưa ra các câu hỏi định nghĩa lặp đi lặp lại, ít câu liên quan đến phân tích code thực tế và không có sự phân hóa độ khó rõ ràng.	**Kết quả:** AI tạo được 10 câu trắc nghiệm đa dạng hơn, không bị lặp, nhưng phần lớn vẫn là câu hỏi thuần lý thuyết, thiếu bài tập phân tích code.	**Kĩ thuật:** Role prompting, chain-of-thought, structured output **Kết quả:** Bộ đề có cấu trúc phân tầng rõ ràng theo thang đo Bloom, từ ghi nhớ định nghĩa đến phân tích code thực tế và vận dụng so sánh thuật toán. Đáp án kèm lời giải từng bước giúp bộ đề có giá trị ôn tập cao.

5 Phân tích hiệu quả của prompt nâng cao

Qua những thử nghiệm prompt trên, em thấy prompt sử dụng các kỹ thuật prompt engineering có độ hiệu quả tốt hơn so với prompt cơ bản vì những lí do sau:

- **Sự rõ ràng về bối cảnh:** Prompt cơ bản như "Big-O là gì?" không cung cấp thông tin gì về người hỏi, dẫn đến câu trả lời mang tính học thuật cao và thiếu tính ứng dụng. Prompt nâng cao xác định rõ đối tượng giúp AI điều chỉnh ngôn ngữ và mức độ trừu tượng phù hợp.

- **Tạo kết quả tối ưu ngay từ đầu:** Khi yêu cầu cấu trúc cụ thể, AI không trả về một khối văn bản dài mà tổ chức thông tin thành các phần có chủ đích, giúp người đọc tiếp thu từng lớp kiến thức một cách tuần tự.
- **Định hướng tư duy phân tích:** Đặc biệt với Tác vụ 2, yêu cầu AI giải thích "từng bước" (chain-of-thought) thay vì chỉ đưa đáp án đã giúp em học được phương pháp tư duy đằng sau kết quả.
- **Kiểm soát chất lượng đầu ra:** Việc yêu cầu cụ thể về cấu trúc đề thi giúp loại bỏ các câu hỏi mơ hồ hoặc lặp lại, tạo ra bộ tài liệu ôn tập có giá trị thực tiễn cao hơn.

6 Các nguyên tắc viết prompt hiệu quả

Từ bài tập này, em rút ra được những nguyên tắc quan trọng trong việc viết prompt hiệu quả cho các tác vụ học tập:

- **Bối cảnh:** Luôn cho AI biết bạn là ai, trình độ của bạn đến đâu và bạn đang học trong hoàn cảnh nào (VD: "Hãy giải thích Big-O cho tôi như thể tôi là sinh viên năm hai chưa học về phân tích thuật toán").
- **Hướng dẫn rõ ràng:** Chia nhỏ các yêu cầu phức tạp thành các bước 1, 2, 3,… có thứ tự. Tránh dùng từ ngữ chung chung như "giải thích chi tiết hơn" mà hãy nói cụ thể "giải thích theo từng bước, có kèm ví dụ mã Java".
- **Xác định vai trò:** Yêu cầu AI đóng vai một chuyên gia cụ thể (Giảng viên, Trợ giảng, Người ra đề) sẽ định hướng văn phong, mức độ chuyên sâu và cách tổ chức thông tin của câu trả lời.
- **Chuỗi tư duy (Chain-of-thought):** Với các tác vụ cần phân tích (như phân tích độ phức tạp code), hãy yêu cầu AI trình bày suy luận từng bước thay vì chỉ đưa kết luận.
- **Tinh chỉnh:** Đừng chấp nhận câu trả lời đầu tiên của AI. Hãy luôn tương tác lại dựa vào câu trả lời của nó (Ví dụ: *"Giải thích lại bước 2 nhưng dùng một ví dụ code phức tạp hơn"*) để tinh chỉnh kết quả cho đến khi thực sự ưng ý.