Prompt ban đầu: “So sánh Mảng và Danh sách Liên kết theo tốc độ truy cập, bộ nhớ, ứng dụng vấn đề. Sau đó đưa mã minh họa thao tác chèn phần tử trong mỗi cấu trúc.”

## Phản hồi AI (lần 1 – chỉ lý thuyết):

AI giải thích: Mảng có tốc độ truy cập nhanh (O(1)), nhưng chèn/xóa chậm do phải dời phần tử. Danh sách liên kết chèn/xóa nhanh hơn (O(1) nếu biết vị trí), nhưng truy cập chậm (O(n)) và tốn thêm bộ nhớ cho con trỏ. Ứng dụng: Mảng phù hợp khi cần truy cập ngẫu nhiên nhanh, danh sách phù hợp khi cần chèn/xóa thường xuyên.

## Prompt chỉnh sửa (bổ sung yêu cầu mã minh họa):

“So sánh Mảng và Danh sách Liên kết theo tốc độ truy cập, bộ nhớ và ứng dụng. Viết thêm mã Python minh họa thao tác \*\*chèn phần tử\*\* trong mỗi cấu trúc để dễ hiểu hơn.”

## Bảng so sánh Mảng và Danh sách Liên kết

|  |  |  |  |
| --- | --- | --- | --- |
| Tiêu chí | Mảng (Array) | Danh sách Liên kết (Linked List) | Ghi chú |
| Tốc độ truy cập | O(1) – truy cập trực tiếp bằng chỉ số. | O(n) – phải duyệt lần lượt từng nút. | Mảng nhanh hơn khi truy cập. |
| Tốc độ chèn/xóa | O(n) – phải dời phần tử. | O(1) nếu biết vị trí chèn. | Danh sách nhanh hơn khi chèn/xóa. |
| Bộ nhớ sử dụng | Liên tục, ít overhead. | Tốn thêm bộ nhớ cho con trỏ (next). | Danh sách tốn nhiều RAM hơn. |
| Ứng dụng | Khi cần truy cập ngẫu nhiên nhanh. | Khi cần chèn/xóa thường xuyên. | Tùy tình huống sử dụng. |

## Mã minh họa thao tác chèn phần tử

\*\*1. Mảng (Array)\*\* – chèn phần tử vào vị trí bất kỳ:

* arr = [10, 20, 30, 40]  
  arr.insert(2, 25) # Chèn 25 vào vị trí thứ 2  
  print(arr) # Kết quả: [10, 20, 25, 30, 40]

\*\*Giải thích:\*\* Dùng phương thức insert() của list trong Python. Thao tác này phải dời các phần tử phía sau nên tốn O(n) thời gian.

\*\*2. Danh sách Liên kết (Linked List)\*\* – chèn phần tử mới sau một nút cụ thể:

* class Node:  
   def \_\_init\_\_(self, data):  
   self.data = data  
   self.next = None  
    
  # Tạo danh sách: 10 -> 20 -> 30  
  head = Node(10)  
  second = Node(20)  
  third = Node(30)  
  head.next = second  
  second.next = third  
    
  # Chèn 25 sau nút có giá trị 20  
  new\_node = Node(25)  
  new\_node.next = second.next  
  second.next = new\_node  
    
  # In danh sách sau khi chèn  
  current = head  
  while current:  
   print(current.data, end=" -> ")  
   current = current.next  
  # Kết quả: 10 -> 20 -> 25 -> 30 ->

\*\*Giải thích:\*\* Tạo nút mới, điều chỉnh con trỏ next để chèn phần tử. Thao tác chèn là O(1) nếu biết vị trí cần chèn.

## Tổng kết:

Phiên bản prompt sau khi bổ sung mã minh họa giúp người học dễ hình dung sự khác biệt giữa hai cấu trúc dữ liệu. Thay vì chỉ hiểu lý thuyết, việc có ví dụ Python giúp trực quan hóa thao tác chèn và thấy rõ sự khác nhau về hiệu suất.