

- Chọn 1 khái niệm nâng cao (Heap, Trie, Graph...).
 - Thiết kế ít nhất 4 Prompt liên tiếp, mỗi Prompt phải mở rộng hoặc đào sâu thêm:
 - Prompt 1: Giải thích khái niệm cơ bản, phù hợp cho người mới.
 - Prompt 2: So sánh với một khái niệm liên quan, trình bày bằng bảng.
 - Prompt 3: Viết ví dụ minh họa (code hoặc tình huống thực tế).
 - Prompt 4: Giải thích lại cho 2 đối tượng khác nhau (người mới và sinh viên năm 3).
 - chạy cả 4 Prompt, ghi lại kết quả AI. Ghi nhận Prompt và kết quả AI trả về.
-

Các Prompt về Graph :

- Prompt 1 : “Tôi chưa biết gì về cấu trúc dữ liệu. Hãy giải thích khái niệm Graph (Đồ thị) một cách đơn giản nhất, bao gồm định nghĩa các thành phần chính là Đỉnh (Vertex) và Cạnh (Edge), và cho một ví dụ đời thường.”
- Prompt 2 : “Hãy so sánh chi tiết giữa Graph và Tree (Cây), bao gồm cấu trúc, nguyên tắc, và các giới hạn về cạnh. Trình bày kết quả dưới dạng bảng.”
- Prompt 3 : “Viết một ví dụ code Python để tạo và biểu diễn một Graph vô hướng đơn giản sử dụng Adjacency List (Danh sách Kề). Minh họa cách thêm Đỉnh và thêm Cạnh.”
- Prompt 4 : “Giải thích lại khái niệm Graph theo hai cấp độ khác nhau: (1) cho người mới học lập trình (dùng ẩn dụ đời thường), và (2) cho sinh viên năm 3 ngành Khoa học máy tính (tập trung vào thuật toán và độ phức tạp).”

Kết quả AI trả về từ các prompt :

- Prompt 1 :

I. Định nghĩa Đơn giản

Graph là một cấu trúc dùng để mô tả mối quan hệ giữa các đối tượng.

- **Đối tượng** là các thành phố, con người, hoặc các trang web.
- **Mối quan hệ** là đường đi, tình bạn, hoặc liên kết siêu văn bản.

II. Các Thành phần Chính

1. **Đỉnh (Vertex hoặc Node):** Là các đối tượng riêng lẻ.
 - **Ví dụ:** Mỗi thành phố trên bản đồ là một Đỉnh.
2. **Cạnh (Edge):** Là các mối liên hệ hoặc kết nối giữa các Đỉnh.
 - **Ví dụ:** Con đường nối hai thành phố với nhau là một Cạnh.

III. Ví dụ Đời thường: Mạng Xã hội

Hãy xem xét Facebook hoặc LinkedIn:

- **Đỉnh (Vertex):** Mỗi tài khoản người dùng là một Đỉnh.
- **Cạnh (Edge):** Mỗi quan hệ "Bạn bè" hoặc "Kết nối" giữa hai người là một Cạnh.

Nếu bạn có 100 người bạn, bạn có một Đỉnh (là bạn) và 100 Cạnh nối bạn với 100 Đỉnh khác.

- **Prompt 2 :**

Tiêu chí	Graph (Đồ thị)	Tree (Cây)
Cấu trúc Cơ bản	Bộ sưu tập các Đỉnh và Cạnh ($G = (V, E)$).	Bộ sưu tập các Đỉnh và Cạnh ($T = (V, E)$).
Mối quan hệ Cạnh	Không có giới hạn. Có thể có nhiều cạnh, tạo thành chu trình (cycles) .	Có giới hạn nghiêm ngặt. Không được phép có chu trình. Luôn là đồ thị liên thông (connected graph) .
Định hướng	Có thể là Có hướng (Directed) hoặc Vô hướng (Undirected) .	Thường là Có hướng (từ Root đi xuống), nhưng cũng có thể xét vô hướng.

Số lượng Cạnh	Có thể có tối đa $V \times (V - 1)$ cạnh (đồ thị có hướng hoàn chỉnh).	Luôn có chính xác $V - 1$ cạnh, với V là số đỉnh.
Phần tử Đặc biệt	Không có đỉnh đặc biệt.	Luôn có một đỉnh đặc biệt được gọi là Root (Gốc) .
Ví dụ Ứng dụng	Mạng xã hội, Bản đồ đường đi, Mạng máy tính.	Hệ thống tập tin (File System), Cây phả hệ, Cây quyết định (Decision Tree).

- **Prompt 3 :**

```
python Copy code

# Biểu diễn graph bằng danh sách kế
graph = {
    'A': ['B', 'C'],
    'B': ['A', 'D'],
    'C': ['A'],
    'D': ['B']
}

# In graph
for vertex in graph:
    print(vertex, "->", graph[vertex])
```

Giải thích:

- Mỗi key là một đỉnh
- Danh sách giá trị là các đỉnh kề
- Đây là đồ thị vô hướng
- Dễ mở rộng và tiết kiệm bộ nhớ

- **prompt 4 :**

(1) Giải thích cho Người mới học Lập trình (Ân dụ Đời thường)

Hãy coi Graph như Hệ thống tàu điện ngầm.

- **Đỉnh (Vertex/Node):** Là mỗi ga tàu điện (điểm dừng).
- **Cạnh (Edge):** Là đường ray nối trực tiếp hai ga tàu lại với nhau.

Mục tiêu của Graph là giúp ta trả lời các câu hỏi:

- Có đường đi không? (Tôi có thể đi từ Ga A đến Ga Z không?)
- Đường đi ngắn nhất? (Tuyến đường nào có ít điểm dừng nhất?)

Tất cả các thuật toán trên Graph (như Breadth-First Search hay Dijkstra) chỉ là cách chúng ta tìm kiếm thông minh trên mạng lưới đường ray và nhà ga này.

(2) Giải thích cho Sinh viên năm 3 Khoa học Máy tính (Chuyên sâu)

Graph là một cấu trúc dữ liệu không tuyến tính, vô cùng linh hoạt, thường được triển khai bằng Adjacency Matrix (Ma trận Kề) hoặc Adjacency List (Danh sách Kề).

- **Danh sách Kề (List):** Hiệu quả về Bộ nhớ ($O(V+E)$), lý tưởng cho các đồ thị thưa (sparse graphs).
- **Ma trận Kề (Matrix):** Hiệu quả về Truy cập ($O(1)$) để kiểm tra cạnh tồn tại, lý tưởng cho các đồ thị dày (dense graphs).

Các Thuật toán trọng tâm:

1. **Duyệt:** DFS (Depth-First Search) dùng Stack, BFS (Breadth-First Search) dùng Queue.
2. **Đường đi ngắn nhất:** Dijkstra (cho cạnh không âm) và Bellman-Ford (cho cạnh có âm, nhưng không có chu trình âm).
3. **Cây bao trùm tối thiểu (MST):** Prim và Kruskal, dùng để tìm tập hợp cạnh nhỏ nhất để kết nối tất cả các đỉnh.

Việc chọn thuật toán và biểu diễn Graph phụ thuộc trực tiếp vào độ phức tạp thời gian bạn cần đạt được cho bài toán cụ thể.