# So sánh Array và Linked List + Ví dụ chèn phần tử

## 1. Bảng so sánh

|  |  |  |  |
| --- | --- | --- | --- |
| Tiêu chí | Array (Mảng) | Linked List (Danh sách liên kết) | Ghi chú / Tình huống áp dụng |
| Tốc độ truy cập (random access) | O(1) — truy cập trực tiếp bằng chỉ số (index) | O(n) — phải duyệt từ đầu đến vị trí cần truy cập | Dùng array nếu cần đọc ngẫu nhiên nhanh (ví dụ: bảng, buffer) |
| Thời gian chèn/xóa | Chèn/xóa ở cuối: O(1); chèn/xóa ở giữa: O(n) do cần dịch phần tử | Chèn/xóa ở đầu hoặc khi có con trỏ tới nút trước: O(1); tìm vị trí: O(n) | Dùng linked list khi cần nhiều chèn/xóa ở giữa hoặc đầu (ví dụ: hàng đợi động) |
| Bộ nhớ | Kích thước cố định (hay phải tái cấp phát) và không có overhead cho con trỏ | Sử dụng thêm bộ nhớ cho con trỏ/tham chiếu của mỗi nút (overhead) | Array tiết kiệm bộ nhớ cho dữ liệu thuần; linked list cần overhead nhưng linh hoạt kích thước |
| Tính liên tục trong bộ nhớ | Bộ nhớ liên tục (contiguous) — tốt cho cache, locality | Nút rời rạc, không liên tiếp — locality kém | Array nhanh hơn do cache-friendly; linked list thích hợp khi không biết trước kích thước |

## 2. Ví dụ code minh họa thao tác chèn phần tử

### 2.1. Array (dùng Python list) — chèn phần tử tại vị trí index

arr = [1, 2, 4, 5]  
# Chèn giá trị 3 vào vị trí index 2 (sau phần tử 2)  
index = 2  
value = 3  
arr.insert(index, value) # Python list: dịch các phần tử từ index sang phải và chèn  
print(arr) # Kết quả: [1, 2, 3, 4, 5]

* Giải thích từng dòng (Array):
* arr = [1, 2, 4, 5] — khởi tạo mảng (Python list).
* index = 2; value = 3 — xác định vị trí và giá trị cần chèn.
* arr.insert(index, value) — phương thức của Python list: chèn value vào vị trí index; các phần tử từ index trở đi bị dịch sang phải.
* print(arr) — in mảng sau khi chèn.

### 2.2. Linked List (Singly) — chèn phần tử tại đầu và tại vị trí index

class Node:  
 def \_\_init\_\_(self, val):  
 self.val = val  
 self.next = None  
  
class SinglyLinkedList:  
 def \_\_init\_\_(self):  
 self.head = None  
  
 def insert\_head(self, val):  
 new\_node = Node(val) # Tạo node mới  
 new\_node.next = self.head # Trỏ next của node mới tới head hiện tại  
 self.head = new\_node # Cập nhật head thành node mới  
  
 def insert\_at\_index(self, index, val):  
 if index == 0:  
 return self.insert\_head(val)  
 cur = self.head  
 i = 0  
 # Duyệt đến nút trước vị trí cần chèn  
 while cur is not None and i < index - 1:  
 cur = cur.next  
 i += 1  
 if cur is None:  
 raise IndexError('Index out of bounds')  
 new\_node = Node(val)  
 new\_node.next = cur.next  
 cur.next = new\_node  
  
 def to\_list(self):  
 res = []  
 cur = self.head  
 while cur:  
 res.append(cur.val)  
 cur = cur.next  
 return res  
  
# Sử dụng  
l = SinglyLinkedList()  
l.insert\_head(2)  
l.insert\_head(1) # Danh sách: 1 -> 2  
l.insert\_at\_index(2, 3) # Chèn 3 tại index 2 => 1 -> 2 -> 3  
print(l.to\_list()) # Kết quả: [1, 2, 3]

* Giải thích từng phần (Linked List):
* class Node: — định nghĩa nút chứa val và con trỏ next.
* class SinglyLinkedList: — định nghĩa danh sách với head ban đầu là None.
* insert\_head: tạo node mới, trỏ next của node mới tới head, rồi cập nhật head = node mới (O(1)).
* insert\_at\_index: nếu index = 0 gọi insert\_head; ngược lại duyệt tới nút trước vị trí chèn (O(index)), tạo node mới, nối lại con trỏ để chèn (O(1) cho thao tác nối).
* to\_list: tiện ích để xuất danh sách thành Python list để dễ kiểm tra.

## 3. Prompt gợi ý (nếu AI chỉ trả lời lý thuyết):

“So sánh Array và Linked List theo tốc độ truy cập, bộ nhớ, tình huống áp dụng. Sau đó cung cấp ví dụ code (Python) minh họa thao tác chèn phần tử cho mỗi cấu trúc, kèm giải thích từng dòng.”