# Bài 4: Kiểu dữ liệu hàng đợi

**Giải Chuyên đề Tin học 12 Bài 4: Kiểu dữ liệu hàng đợi**  
**Khởi động trang 16 Chuyên đề Tin học 12**: Từ các bài học trước, em đã biết viết chương trình đơn giản để sử dụng các hàm cơ bản của ngăn xếp được cài đặt bằng danh sách (kiểu list của Python). Em hãy trả lời các câu hỏi sau:  
a) Có thể cài đặt hàng đợi bằng mảng một chiều tương tự như ngăn xếp được không?   
b) Khi cài đặt hàng đợi bằng mảng một chiều, cần có thông tin nào để thực hiện phép toán thêm vảo và lấy ra?  
**Lời giải:**  
**a) Có thể cài đặt hàng đợi bằng mảng một chiều tương tự như ngăn xếp được không?**  
Có, có thể cài đặt hàng đợi (queue) bằng mảng một chiều tương tự như ngăn xếp (stack) bằng cách sử dụng danh sách (list) trong Python. Tuy nhiên, hàng đợi và ngăn xếp có cách thức hoạt động khác nhau. Trong khi ngăn xếp tuân theo nguyên tắc LIFO (Last In, First Out - Vào sau, ra trước), hàng đợi tuân theo nguyên tắc FIFO (First In, First Out - Vào trước, ra trước). Để cài đặt hàng đợi bằng mảng một chiều, ta cần phải quản lý việc thêm phần tử vào cuối hàng đợi (enqueue) và lấy phần tử ra từ đầu hàng đợi (dequeue).  
**b) Khi cài đặt hàng đợi bằng mảng một chiều, cần có thông tin nào để thực hiện phép toán thêm vào và lấy ra?**  
Khi cài đặt hàng đợi bằng mảng một chiều, cần phải quản lý hai chỉ số chính:  
1. **Front (đầu hàng đợi):** Chỉ số này giữ vị trí của phần tử đầu tiên trong hàng đợi. Khi thực hiện phép toán lấy ra (dequeue), phần tử tại vị trí này sẽ được lấy ra và chỉ số front sẽ được cập nhật để trỏ đến phần tử tiếp theo.  
2**. Rear (cuối hàng đợi):** Chỉ số này giữ vị trí của phần tử cuối cùng trong hàng đợi. Khi thực hiện phép toán thêm vào (enqueue), phần tử mới sẽ được thêm vào vị trí kế tiếp của chỉ số rear, và chỉ số rear sẽ được cập nhật để trỏ đến vị trí cuối mới của hàng đợi.  
Ngoài ra, còn cần thêm thông tin về kích thước của mảng để quản lý việc tràn (overflow) khi hàng đợi đầy và để tránh lấy ra từ hàng đợi rỗng (underflow). Một số cách quản lý thông tin bổ sung có thể bao gồm:  
**- Size (kích thước hiện tại):** Số lượng phần tử hiện có trong hàng đợi.  
**- Capacity (dung lượng tối đa):** Dung lượng tối đa của mảng được sử dụng để cài đặt hàng đợi.  
**1. Biểu diễn hàng đợi bằng mảng 1 chiều**  
**Hoạt động 1 trang 16 Chuyên đề Tin học 12**: Quan sát, trao đổi, thảo luận để tìm hiểu cách biểu diễn hàng đợi bằng mảng một chiều. Em hãy trả lời các câu hỏi sau:  
1. Có thể biểu diễn hàng đợi bằng mảng một chiều được không?   
2. Cần có các biến nào để thực hiện các phép toán thêm vào và lấy ra?  
**Lời giải:**  
1. Có thể biểu diễn hàng đợi bằng mảng một chiều được không?  
Có, hoàn toàn có thể biểu diễn hàng đợi (queue) bằng mảng một chiều (array). Trong Python, danh sách (list) có thể được sử dụng để cài đặt hàng đợi. Tuy nhiên, cần quản lý các chỉ số cho phép thêm phần tử vào cuối hàng đợi (enqueue) và lấy phần tử ra từ đầu hàng đợi (dequeue) một cách hiệu quả.  
2. Cần có các biến nào để thực hiện các phép toán thêm vào và lấy ra?  
Để thực hiện các phép toán thêm vào và lấy ra trong hàng đợi biểu diễn bằng mảng một chiều, cần có các biến sau:  
**1. Front (đầu hàng đợi)**:  
- Đây là biến giữ vị trí của phần tử đầu tiên trong hàng đợi.  
- Khi thực hiện phép toán lấy ra (dequeue), phần tử tại vị trí front sẽ được lấy ra, và chỉ số front sẽ được cập nhật để trỏ đến phần tử tiếp theo.  
**2. Rear (cuối hàng đợi)**:  
- Đây là biến giữ vị trí của phần tử cuối cùng trong hàng đợi.  
- Khi thực hiện phép toán thêm vào (enqueue), phần tử mới sẽ được thêm vào vị trí kế tiếp của rear, và chỉ số rear sẽ được cập nhật để trỏ đến vị trí cuối mới của hàng đợi.  
**3. Size (kích thước hiện tại)**:  
- Đây là biến theo dõi số lượng phần tử hiện có trong hàng đợi.  
- Biến này giúp kiểm tra hàng đợi có rỗng hay đầy để tránh các lỗi underflow (lấy từ hàng đợi rỗng) và overflow (thêm vào hàng đợi đầy).  
**4. Capacity (dung lượng tối đa)**:  
- Đây là biến xác định dung lượng tối đa của mảng được sử dụng để cài đặt hàng đợi.  
- Giúp kiểm tra và ngăn chặn tình trạng tràn (overflow) khi hàng đợi đầy.  
**Câu hỏi 1 trang 17 Chuyên đề Tin học 12**: Khi hàng đợi được cài đặt bằng danh sách (kiểu list của Python), em hãy cho biết cách tính số phần tử của hàng đợi này.  
**Lời giải:**  
Khi hàng đợi (queue) được cài đặt bằng danh sách (list) trong Python, có thể tính số phần tử của hàng đợi này bằng cách sử dụng thuộc tính size mà ta đã định nghĩa trong lớp Queue. Thuộc tính size sẽ được cập nhật mỗi khi thực hiện các phép toán thêm vào (enqueue) hoặc lấy ra (dequeue).  
- Gợi ý cách tính số phần tử của hàng đợi bằng cách truy cập thuộc tính size:  
class Queue:  
 def \_\_init\_\_(self, capacity):  
 self.capacity = capacity  
 self.queue = [None] \* capacity  
 self.front = 0  
 self.rear = -1  
 self.size = 0  
 def is\_full(self):  
 return self.size == self.capacity  
 def is\_empty(self):  
 return self.size == 0  
 def enqueue(self, item):  
 if self.is\_full():  
 print("Queue is full")  
 return  
 self.rear = (self.rear + 1) % self.capacity  
 self.queue[self.rear] = item  
 self.size += 1  
 print(f"Enqueued {item}")  
 def dequeue(self):  
 if self.is\_empty():  
 print("Queue is empty")  
 return None  
 item = self.queue[self.front]  
 self.front = (self.front + 1) % self.capacity  
 self.size -= 1  
 print(f"Dequeued {item}")  
 return item  
 def get\_front(self):  
 if self.is\_empty():  
 print("Queue is empty")  
 return None  
 return self.queue[self.front]  
 def get\_rear(self):  
 if self.is\_empty():  
 print("Queue is empty")  
 return None  
 return self.queue[self.rear]  
 def get\_size(self):  
 return self.size  
# Test the Queue  
q = Queue(5)  
q.enqueue(10)  
q.enqueue(20)  
q.enqueue(30)  
print(f"Number of elements in the queue: {q.get\_size()}") # Should print 3  
q.dequeue()  
print(f"Number of elements in the queue: {q.get\_size()}") # Should print 2  
\* Trong chương trình trên ta có:  
 - Hàm enqueue thêm phần tử vào cuối hàng đợi và tăng size lên 1.  
 - Hàm dequeue lấy phần tử ra từ đầu hàng đợi và giảm size xuống 1.  
 - Hàm get\_size trả về giá trị của thuộc tính size, tức là số phần tử hiện có trong hàng đợi.  
Do đó, số phần tử trong hàng đợi luôn được theo dõi và cập nhật thông qua biến size, và ta có thể biết được số phần tử hiện có trong hàng đợi bất cứ lúc nào bằng cách gọi phương thức get\_size.  
**Câu hỏi 2 trang 17 Chuyên đề Tin học 12**: Ban đầu, hàng đợi là rỗng. Em hãy cho biết giá trị của phần tử ở đầu (front) và đuôi (rear) sau khi thực hiện tuần tự các phép toán enqueue(Q,2); enqueue(Q,19); dequeue(Q), enaqueue (Q, 6); dequeue(Q); enqueue(Q, 9); enqueue(Q,1).  
**Lời giải:**  
Để xác định giá trị của phần tử ở đầu (front) và đuôi (rear) sau khi thực hiện tuần tự các phép toán trên hàng đợi (queue), ta sẽ theo dõi từng phép toán một cách chi tiết. Giả sử hàng đợi ban đầu có dung lượng đủ lớn để chứa tất cả các phần tử được thêm vào mà không cần phải xử lý trường hợp tràn (overflow).  
Ban đầu, hàng đợi là rỗng:  
front = 0  
rear = -1  
size = 0  
Các bước thực hiện tuần tự các phép toán như sau:  
**1. enqueue(Q, 2)**  
Thêm 2 vào cuối hàng đợi.  
rear tăng lên 0.  
size tăng lên 1.  
queue = [2]  
front = 0  
rear = 0  
**2. enqueue(Q, 19)**  
Thêm 19 vào cuối hàng đợi.  
rear tăng lên 1.  
size tăng lên 2.  
queue = [2, 19]  
front = 0  
rear = 1  
**3. dequeue(Q)**  
Lấy 2 ra từ đầu hàng đợi.  
front tăng lên 1.  
size giảm xuống 1.  
queue = [2, 19] (phần tử tại front không còn được sử dụng)  
front = 1  
rear = 1  
**4. enqueue(Q, 6)**  
Thêm 6 vào cuối hàng đợi.  
rear tăng lên 2.  
size tăng lên 2.  
queue = [2, 19, 6]  
front = 1  
rear = 2  
**5. dequeue(Q)**  
Lấy 19 ra từ đầu hàng đợi.  
front tăng lên 2.  
size giảm xuống 1.  
queue = [2, 19, 6] (phần tử tại front không còn được sử dụng)  
front = 2  
rear = 2  
**6. enqueue(Q, 9)**  
Thêm 9 vào cuối hàng đợi.  
rear tăng lên 3.  
size tăng lên 2.  
queue = [2, 19, 6, 9]  
front = 2  
rear = 3  
**7. enqueue(Q, 1)**  
Thêm 1 vào cuối hàng đợi.  
rear tăng lên 4.  
size tăng lên 3.  
queue = [2, 19, 6, 9, 1]  
front = 2  
rear = 4  
- Sau khi thực hiện tuần tự các phép toán, trạng thái của hàng đợi sẽ như sau:  
queue = [2, 19, 6, 9, 1]  
front = 2 (phần tử ở đầu hàng đợi là 6)  
rear = 4 (phần tử ở cuối hàng đợi là 1)  
**2. Các phép toán của kiểu dữ liệu hàng đợi**  
**Hoạt động 2 trang 17 Chuyên đề Tin học 12**: Đọc, trao đổi để biết các hàm cơ bản của hàng đợi được cài đặt bằng danh sách (kiểu list của Python).  
**Lời giải:**  
Hàng đợi được cài đặt với các phép toán chính như thêm phần tử vào cuối hàng đợi (enqueue), lấy phần tử ra từ đầu hàng đợi (dequeue), kiểm tra xem hàng đợi có rỗng không (is\_empty), và trả về số phần tử hiện tại trong hàng đợi (size).  
**Câu hỏi 1 trang 19 Chuyên đề Tin học 12**: Khi hàng đợi Q được cài đặt bằng danh sách (kiểu list của Python), em hãy cho biết chỉ số của các phân tử tại đầu (front) và đuôi (rear). So sánh các chỉ số này với chỉ số của các phần tử tại đáy (bottom) và đỉnh (top) của ngăn xếp (cũng được cài đặt bằng danh sách).  
**Lời giải:**  
Khi hàng đợi (queue) và ngăn xếp (stack) được cài đặt bằng danh sách (list) trong Python, cách quản lý chỉ số của các phần tử có một số điểm khác nhau. Dưới đây là sự phân tích chi tiết về chỉ số của các phần tử tại đầu (front) và đuôi (rear) của hàng đợi so với chỉ số của các phần tử tại đáy (bottom) và đỉnh (top) của ngăn xếp.  
**Hàng đợi (Queue)**  
Trong hàng đợi (FIFO - First In, First Out):  
- **Front (đầu hàng đợi)**: Chỉ số của phần tử đầu tiên (phần tử được thêm vào đầu tiên và sẽ được lấy ra đầu tiên).  
- **Rear (đuôi hàng đợi)**: Chỉ số của phần tử cuối cùng (phần tử được thêm vào gần đây nhất).  
**Ngăn xếp (Stack)**  
Trong ngăn xếp (LIFO - Last In, First Out):  
- **Bottom (đáy ngăn xếp)**: Chỉ số của phần tử đầu tiên được thêm vào (phần tử nằm ở đáy ngăn xếp, ít được truy cập nhất).  
- **Top (đỉnh ngăn xếp)**: Chỉ số của phần tử cuối cùng được thêm vào (phần tử n**So sánh các chỉ số**  
- **Hàng đợi**:  
+ **Front**: Chỉ số của phần tử đầu tiên được thêm vào và sẽ được lấy ra đầu tiên. Trong ví dụ, **Front = 0**.  
+ **Rear**: Chỉ số của phần tử cuối cùng được thêm vào. Trong ví dụ, **rear = 4**  
- **Ngăn xếp**:  
+ **Bottom**: Chỉ số của phần tử đầu tiên được thêm vào, nằm ở đáy ngăn xếp. Trong ví dụ, **Bottom = 0** .  
+ **Top**: Chỉ số của phần tử cuối cùng được thêm vào, nằm ở đỉnh ngăn xếp. Trong ví dụ, **Top = 4**  
Cả hàng đợi và ngăn xếp đều sử dụng danh sách Python, và cả hai đều sử dụng chỉ số 0 cho phần tử đầu tiên được thêm vào. Sự khác biệt chính nằm ở cách mà các phần tử được thêm vào và lấy ra:  
- Trong **hàng đợi**, phần tử ở **Front** được lấy ra trước và phần tử ở **Rear** được thêm vào cuối.  
- Trong **ngăn xếp**, phần tử ở **Top** được lấy ra trước và phần tử ở **Bottom** nằm ở đáy ngăn xếp.  
- Nằm ở đỉnh ngăn xếp, phần tử sẽ được lấy ra đầu tiên).  
**Câu hỏi 2 trang 19 Chuyên đề Tin học 12**: Em hãy nêu sự giống nhau và khác nhau giữa các hàm của ngăn xếp và hàng đợi được cài đặt bằng danh sách (kiểu list của Python).  
**Lời giải:**  
So sánh giữa các hàm của ngăn xếp (stack) và hàng đợi (queue) được cài đặt bằng danh sách (list) của Python:  
\* Sự giống nhau  
1. Dữ liệu cơ bản:  
- Cả hai đều sử dụng danh sách (list) của Python để lưu trữ các phần tử.  
2. Khởi tạo:  
- Cả hai đều cần một phương thức khởi tạo (\_\_init\_\_) để tạo một danh sách trống.  
3. Kiểm tra rỗng:  
- Cả hai đều có một phương thức để kiểm tra xem cấu trúc dữ liệu có rỗng hay không (is\_empty).  
\* Sự khác nhau:  
1. Thêm phần tử:  
- Ngăn xếp (stack):  
+ Hàm push thêm một phần tử vào cuối danh sách (đỉnh ngăn xếp).  
+ self.stack.append(item)  
- Hàng đợi (queue):  
+ Hàm enqueue thêm một phần tử vào cuối danh sách (đuôi hàng đợi).  
+ self.queue.append(item)  
- Lấy phần tử:  
+ Ngăn xếp (stack):  
+ Hàm pop lấy phần tử cuối cùng của danh sách (đỉnh ngăn xếp) và trả về phần tử đó.  
+ return self.stack.pop()  
+ Hàng đợi (queue):  
+ Hàm dequeue lấy phần tử đầu tiên của danh sách (đầu hàng đợi) và trả về phần tử đó.  
+ return self.queue.pop(0)  
- Lấy phần tử nhưng không xóa:  
+ Ngăn xếp (stack):  
Hàm top trả về phần tử cuối cùng của danh sách (đỉnh ngăn xếp) mà không xóa nó.  
return self.stack[-1]  
Hàm bottom trả về phần tử đầu tiên của danh sách (đáy ngăn xếp) mà không xóa nó.  
return self.stack[0]  
+ Hàng đợi (queue):  
Hàm front trả về phần tử đầu tiên của danh sách (đầu hàng đợi) mà không xóa nó.  
return self.queue[0]  
Hàm rear trả về phần tử cuối cùng của danh sách (đuôi hàng đợi) mà không xóa nó.  
return self.queue[-1]  
**Luyện tập 1 trang 19 Chuyên đề Tin học 12**: Sửa lại hàm dequeue(Q) và front(Q) trong chương trình trên như sau: Nếu hàng đợi rỗng thì thông báo: "Hàng đợi rỗng không thể thực hiện được lệnh".  
**Lời giải:**  
**Chương trình sửa lại như sau:**  
def front(Q):  
 if isEmptyQueue(Q):  
 print("Hàng đợi rỗng không thể hiện thực được lệnh")  
 return None  
 else:  
 return Q[0]  
def dequeue(Q):  
 if isEmptyQueue(Q):  
 print("Hàng đợi rỗng không thể hiện thực được lệnh")  
 return None  
 else:  
 return Q.pop(0)  
- Chương trình thay thế **raise ValueError** bằng **print** để thông báo lỗi và trả về **None** thay vì ném ra một ngoại lệ. Điều này giúp chương trình tiếp tục chạy mà không bị gián đoạn bởi lỗi.  
**Luyện tập 2 trang 19 Chuyên đề Tin học 12**: Viết hàm length(Q) trả về số phần tử của hàng đợi  
**Lời giải:**  
Trong ngôn ngữ lập trình Python nói riêng, có thể sử dụng một danh sách (list) để biểu diễn một hàng đợi (queue). Để viết một hàm length(Q) trả về số phần tử của hàng đợi, bạn có thể sử dụng phương thức len() của Python để đếm số phần tử trong danh sách. Dưới đây là chương trình mẫu thực hiện điều đó:  
def length(Q):  
 return len(Q)  
# Ví dụ sử dụng  
queue = [1, 2, 3, 4, 5] # Giả sử hàng đợi được biểu diễn bằng một danh sách print(length(queue)) # Output: 5  
**Vận dụng 1 trang 19 Chuyên đề Tin học 12**: Hãy giải thích vì sao lệnh dequeue(Q) lại có độ phức tạp thời gian là O(n), với n là độ dài của hàng đợi hiện thời  
**Lời giải:**  
Nếu biểu diễn hàng đợi (queue) bằng một danh sách (list), lệnh dequeue(Q) thường được thực hiện bằng cách sử dụng phương thức pop(0), tức là lấy và loại bỏ phần tử đầu tiên của danh sách. Độ phức tạp thời gian của lệnh này là O(n)O(n)O(n), với nnn là số phần tử trong hàng đợi hiện thời. Để hiểu rõ tại sao lại như vậy, ta cần xem xét cách thức hoạt động của danh sách trong Python.  
Hoạt động của lệnh pop(0)  
1. Truy cập phần tử đầu tiên:  
Truy cập phần tử đầu tiên của danh sách (phần tử có chỉ số 0) là O(1)O(1)O(1), vì Python có thể truy cập bất kỳ phần tử nào của danh sách theo chỉ số trong thời gian hằng số.  
2. Loại bỏ phần tử đầu tiên:  
- Khi phần tử đầu tiên bị loại bỏ, tất cả các phần tử còn lại trong danh sách phải được di chuyển về phía trước một vị trí để lấp chỗ trống. Cụ thể, phần tử thứ hai trở thành phần tử thứ nhất, phần tử thứ ba trở thành phần tử thứ hai, và cứ thế tiếp tục.  
- Quá trình di chuyển tất cả các phần tử này có độ phức tạp là O(n)O(n)O(n), vì cần phải di chuyển n−1n-1n−1 phần tử trong danh sách (với nnn là tổng số phần tử ban đầu).  
Độ phức tạp thời gian  
- Việc di chuyển tất cả các phần tử về phía trước một vị trí có độ phức tạp thời gian là O(n−1)O(n-1)O(n−1), nhưng về mặt phân tích độ phức tạp, chúng ta bỏ qua hằng số và bậc thấp, do đó kết quả là O(n)O(n)O(n).  
Vì lý do này, lệnh dequeue(Q) thực hiện bằng Q.pop(0) có độ phức tạp thời gian là O(n)O(n)O(n).  
Lựa chọn thay thế  
Để cải thiện hiệu suất có thể sử dụng collections.deque trong Python, một cấu trúc dữ liệu được thiết kế cho việc thêm và loại bỏ phần tử ở cả hai đầu với độ phức tạp thời gian là O(1)O(1)O(1):  
from collections import deque  
Q = deque([1, 2, 3, 4, 5])  
Q.popleft() # Loại bỏ phần tử đầu tiên, độ phức tạp là O(1)  
Với deque , lệnh dequeue(Q) sẽ có độ phức tạp thời gian là O(1)O(1)O(1), hiệu quả hơn rất nhiều so với việc sử dụng danh sách.  
**Vận dụng 2 trang 19 Chuyên đề Tin học 12**: Cho trước mảng T gồm N phần tử T[0], T[1], ..., T[N-1]. Hãy viết hàm thiết lập hàng đợi và các thao tác cơ bản với hàng đợi từ mảng T.  
Gợi ý:  
Để thiết lập dữ liệu hàng đợi từ mảng T cho trước cần có thêm biến backldx mô tả chỉ số của phần tử đuôi của hàng đợi. Ban đầu thiết lập backIdx = -1 tương ứng với hàng đợi rỗng.  
– Cần viết thêm hàm is FullQueue (Q) kiểm tra xem hàng đợi đã đầy chưa. Hàm trả về True nếu hàng đợi Q đã đầy (backIdx = N-1), ngược lại trả về False.  
**Lời giải:**  
Thiết lập hàng đợi và các thao tác cơ bản với hàng đợi từ mảng T như sâu:  
- Để thiết lập dữ liệu hàng đợi từ mảng T cho trước cần có thêm biến backldx mô tả chỉ số của phần tử đuôi của hàng đợi. Ban đầu thiết lập backldx = -1 tương ứng với hàng đợi rỗng.  
- Cần viết thêm hàm isFullQueue(Q) kiểm tra xem hàng đợi đã đầy chưa. Hàm trả về True nếu hàng đợi Q đã đầy (backIdx = N-1), ngược lại trả về False.  
Để thiết lập hàng đợi từ mảng TTT và thực hiện các thao tác cơ bản với hàng đợi, ta có thể làm như sau:  
- Khởi tạo hàng đợi từ mảng TTT.  
- Thiết lập biến backIdx để theo dõi chỉ số của phần tử đuôi của hàng đợi.  
- Viết các hàm kiểm tra hàng đợi có đầy không, thêm phần tử vào hàng đợi, và loại bỏ phần tử khỏi hàng đợi.  
Hướng dẫn chương trình ghi mã nguồn chi tiết trong Python:  
class Queue:  
 def \_\_init\_\_(self, N):  
 self.queue = [None] \* N # Khởi tạo mảng với kích thước N  
 self.N = N # Lưu trữ kích thước của hàng đợi  
 self.backIdx = -1 # Khởi tạo chỉ số phần tử đuôi của hàng đợi  
 def isFullQueue(self):  
 return self.backIdx == self.N - 1  
 def isEmptyQueue(self):  
 return self.backIdx == -1  
 def enqueue(self, value):  
 if self.isFullQueue():  
 print("Queue is full. Cannot enqueue.")  
 else:  
 self.backIdx += 1  
 self.queue[self.backIdx] = value  
 def dequeue(self):  
 if self.isEmptyQueue():  
 print("Queue is empty. Cannot dequeue.")  
 return None  
 else:  
 value = self.queue[0]  
 for i in range(1, self.backIdx + 1):  
 self.queue[i - 1] = self.queue[i]  
 self.queue[self.backIdx] = None # Xóa phần tử cuối cùng  
 self.backIdx -= 1  
 return value  
 def front(self):  
 if self.isEmptyQueue():  
 print("Queue is empty. No front element.")  
 return None  
 else:  
 return self.queue[0]  
 def rear(self):  
 if self.isEmptyQueue():  
 print("Queue is empty. No rear element.")  
 return None  
 else:  
 return self.queue[self.backIdx]  
 def length(self):  
 return self.backIdx + 1  
 def display(self):  
 if self.isEmptyQueue():  
 print("Queue is empty.")  
 else:  
 print("Queue contents:", end=" ")  
 for i in range(self.backIdx + 1):  
 print(self.queue[i], end=" ")  
 print()  
# Khởi tạo mảng T  
T = [1, 2, 3, 4, 5]  
N = len(T)  
# Tạo hàng đợi từ mảng T  
queue = Queue(N)  
for item in T:  
 queue.enqueue(item)  
# Hiển thị hàng đợi  
queue.display()  
# Ví dụ về các thao tác cơ bản  
print("Dequeue:", queue.dequeue())  
queue.display()  
print("Enqueue 6")  
queue.enqueue(6)  
queue.display()  
print("Front element:", queue.front())  
print("Rear element:", queue.rear())  
print("Queue length:", queue.length())  
Ghi chú giải thích chi tiết:  
Lớp Queue:  
- Phương thức init: Khởi tạo hàng đợi với kích thước N và biến **backIdx** ban đầu là -1 để biểu diễn hàng đợi rỗng.  
- Phương thức isFullQueue: Kiểm tra xem hàng đợi đã đầy chưa bằng cách so sánh backIdx với N-1.  
- Phương thức isEmptyQueue: Kiểm tra xem hàng đợi có rỗng không bằng cách kiểm tra backIdx có bằng -1 hay không.  
- Phương thức enqueue: Thêm một phần tử vào cuối hàng đợi nếu hàng đợi chưa đầy.  
- Phương thức dequeue: Loại bỏ phần tử đầu tiên của hàng đợi nếu hàng đợi không rỗng. Các phần tử còn lại sẽ được dịch chuyển lên một vị trí.  
- Phương thức front: Trả về phần tử đầu tiên của hàng đợi.  
- Phương thức rear: Trả về phần tử cuối cùng của hàng đợi.  
- Phương thức length: Trả về số lượng phần tử hiện có trong hàng đợi.  
- Phương thức display: Hiển thị các phần tử trong hàng đợi.