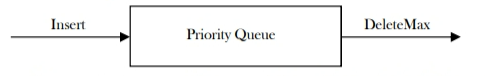
# What is a Priority Queue?

Hàng đợi ưu tiên ADT là một cấu trúc dữ liệu giúp tìm phần tử tối thiểu/tối đa trong một tập hợp các phần tử. Nó hỗ trợ các hoạt động Insert, DeleteMin (trả về và xóa phần tử tối thiểu) và DeleteMax (trả về và xóa phần tử tối đa). Thứ tự các phần tử vào hàng đợi có thể không giống với thứ tự chúng được xử lý. Một ứng dụng ví dụ của hàng đợi ưu tiên là lập kế hoạch công việc, được ưu tiên thay vì phục vụ theo kiểu ai đến trước được phục vụ trước.



Hàng đợi ưu tiên có thể được chia thành hai loại: tăng dần và giảm dần. Hàng đợi ưu tiên tăng dần là khi mục có khóa nhỏ nhất có mức ưu tiên cao nhất, tức là phần tử nhỏ nhất sẽ luôn được xóa. Ngược lại, hàng đợi ưu tiên giảm dần là khi mục có khóa lớn nhất có mức ưu tiên cao nhất, tức là phần tử lớn nhất sẽ luôn được xóa. Hai loại này đối xứng với nhau.

Priority Queue ADT

Một hàng đợi ưu tiên là một cấu trúc dữ liệu trừu tượng (ADT) được xây dựng từ hàng đợi thông thường, nhưng có thêm tính chất ưu tiên dựa trên khóa của các phần tử. Dưới đây là tóm tắt các thao tác chính để biến hàng đợi thành hàng đợi ưu tiên:

### Chèn (Insert):

Thêm một phần tử mới có khóa và dữ liệu tương ứng vào hàng đợi ưu tiên. Phần tử này được chèn sao cho các phần tử trong hàng đợi vẫn được sắp xếp theo thứ tự tăng hoặc giảm dần dựa trên khóa.

### Delete Min/Delete Max:

Xóa và trả về phần tử có khóa nhỏ nhất (đối với Delete Min) hoặc khóa lớn nhất (đối với Delete Max) trong hàng đợi ưu tiên. Sau khi thực hiện thao tác này, hàng đợi vẫn duy trì tính chất ưu tiên.

### Get Minimum/Get Maximum:

Trả về phần tử có khóa nhỏ nhất (đối với Get Minimum) hoặc khóa lớn nhất (đối với Get Maximum) trong hàng đợi ưu tiên mà không xóa nó. Không gây ảnh hưởng đến cấu trúc của hàng đợi.

### Triển khai mảng không có thứ tự

Các phần tử được chèn vào mảng mà không cần quan tâm đến thứ tự. Việc xóa (deleteMax) được thực hiện bằng cách tìm kiếm khóa

rồi xóa.

Độ phức tạp của phần chèn: O(1). độ phức tạp của deleteMin: O(n).

### Thực hiện danh sách không có thứ tự

Nó rất giống với việc triển khai mảng, nhưng thay vì sử dụng mảng, người ta sử dụng danh sách liên kết.

Độ phức tạp của phần chèn: O(1). độ phức tạp của deleteMin: O(n).

### Triển khai mảng theo thứ tự

Các phần tử được chèn vào mảng theo thứ tự sắp xếp dựa trên trường khóa. Việc xóa chỉ được thực hiện ở một đầu.

Độ phức tạp của phần chèn: O(n). độ phức tạp của deleteMin: O(1).

### Thực hiện danh sách có thứ tự

Các phần tử được chèn vào danh sách theo thứ tự sắp xếp dựa trên trường khóa. Việc xóa chỉ được thực hiện ở một đầu, do đó duy trì trạng thái của hàng đợi ưu tiên. Tất cả các chức năng khác liên quan đến danh sách liên kết ADT đều được thực hiện mà không cần sửa đổi.

Độ phức tạp của phần chèn: O(n). độ phức tạp của deleteMin: O(1).

### Triển khai cây tìm kiếm nhị phân

Cả hai lần chèn và xóa đều mất trung bình O(logn) nếu các lần chèn là ngẫu nhiên (tham khảo chương Cây).

### Triển khai cây tìm kiếm nhị phân cân bằng

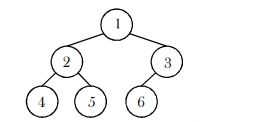
Cả việc chèn và xóa đều lấy O(logn) trong trường hợp xấu nhất (tham khảo chương Cây).

### Triển khai heap nhị phân

Trong các phần tiếp theo chúng ta sẽ thảo luận chi tiết về điều này. Hiện tại, giả sử rằng việc triển khai vùng heap nhị phân mang lại độ phức tạp O(logn) cho việc tìm kiếm, chèn và xóa và O(1) để tìm phần tử tối đa hoặc tối thiểu.

Heaps and Binary Heaps

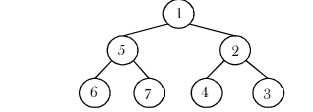
Heap là một dạng cây nhị phân đặc biệt, nơi mỗi nút có một giá trị (khóa) lớn hơn hoặc bằng (hoặc nhỏ hơn hoặc bằng) giá trị của nút con của nó. Điều này được gọi là thuộc tính heap. Một đống còn có thuộc tính bổ sung là tất cả các lá phải ở mức h hoặc h-1, nghĩa là đống phải tạo thành một cây nhị phân hoàn chỉnh. H là chiều cao của cây.



## Types of Heaps?

Dựa vào tính chất của heap, chúng ta có thể phân heap thành 2 loại:

Min heap: Giá trị của một nút phải nhỏ hơn hoặc bằng giá trị của các nút con của nó



Max heap: Giá trị của một nút phải lớn hơn hoặc bằng giá trị của các nút con của nó

## 

# Binary Heaps

Trong Binary Heap, mỗi nút có thể có tối đa hai nút con. Chúng ta tập trung vào Heap Min. Heap có thể được biểu diễn bằng cách sử dụng mảng, với giả định rằng các phần tử được lưu trữ tạo thành một cây nhị phân hoàn chỉnh trong mảng, bắt đầu từ chỉ mục 0. Điều này giúp tránh lãng phí không gian do có nhiều cây. Heap Max trước đó có thể được biểu diễn dưới dạng một mảng.



Declaration of Heap

class Heap:

    def \_\_init\_\_(self, capacity, heap\_type):

        self.array = []

        self.count = 0  # Number of elements in Heap

        self.capacity = capacity  # Size of the heap

        self.heap\_type = heap\_type  # Min Heap or Max Heap

    def parent(self, i):

        pass  # Refer Below sections

    def left\_child(self, i):

        pass  # Refer Below sections

    def right\_child(self, i):

        pass  # Refer Below sections

    def get\_maximum(self, i):

        pass  # Refer Below sections

Creating Heap

class Heap:

    def \_\_init\_\_(self, capacity, heap\_type):

        self.heap\_type = heap\_type

        self.count = 0

        self.capacity = capacity

        self.array = [0]\*capacity

Độ phức tạp thời gian: O(1)