Cấu trúc dữ liệu và giải thuật

TS. Phạm Tuấn Minh
Khoa Công nghệ Thông tin, Đại học Phenikaa
minh.phamtuan@phenikaa-uni.edu.vn
https://sites.google.com/site/phamtuanminh/

Hàng đợi ưu tiên

Hàng đợi ưu tiên

- Thao tác trên các cấu trúc dữ liệu:
  - Stack: Lấy ra phần tử mới nhất (Vào sau ra trước - LIFO)
  - Queue: Lấy ra phần tử cũ nhất (Vào trước ra trước - FIFO)
  - Priority queue: Lấy ra phần tử có độ ưu tiên cao nhất
- Độ ưu tiên:
  - Thông tin thêm của mỗi phần tử
  - Phải so sánh được

3

Ví dụ ứng dụng hàng đợi ưu tiên

- Cài đặt hiệu quả giải thuật Dijkstra: Độ ưu tiên là D[u]. Lấy đỉnh x trong hàng đợi mà D[x] nhỏ nhất.
- Cài đặt hiệu quả giải thuật Prim: Độ ưu tiên là trọng số cạnh. Bổ sung các cạnh mỗi khi thêm 1 đỉnh mới vào cây khung và lấy ra cạnh có trọng số nhỏ nhất.
- Cài đặt giải thuật sắp xếp HeapSort.

1-4

Các thao tác trên hàng đợi ưu tiên

- Thêm một phần tử vào hàng đợi ưu tiên: add()
- Lấy một phần tử ra khỏi hàng đợi ưu tiên: poll()
- Xem giá trị của một phần tử ở đầu hàng đợi ưu tiên: peek()

1-5

Cài đặt hàng đợi ưu tiên

LinkedList:

add()	Thêm vào đầu danh sách liên kết – O(1)
poll()	Phải duyệt danh sách liên kết – O(n)
peek()	Phải duyệt danh sách liên kết – O(n)

LinkedList được sắp xếp:

add()	Phải tìm trên danh sách liên kết – O(n)
poll()	Phần tử có độ ưu tiên cao nhất ở đầu danh sách liên kết – O(1)
peek()	Phần tử có độ ưu tiên cao nhất ở đầu danh sách – O(1)

Cây cân đối:

add()	Phải tìm trên cây và tái cân bằng O(log n)
poll()	Phải tìm trên cây và tái cân bằng – O(log n)
peek()	Phải tìm trên cây – O(log n)

Cách tốt hơn?

6

Heap

Đống là cây nhị phân có hai tính chất:

- Hoàn chỉnh (Completeness): Mỗi mức của cây (trừ mức cuối cùng) được điền đầy. Ở mức cuối cùng, các nút điền từ trái sang phải.
- Thứ tự (Heap-order):

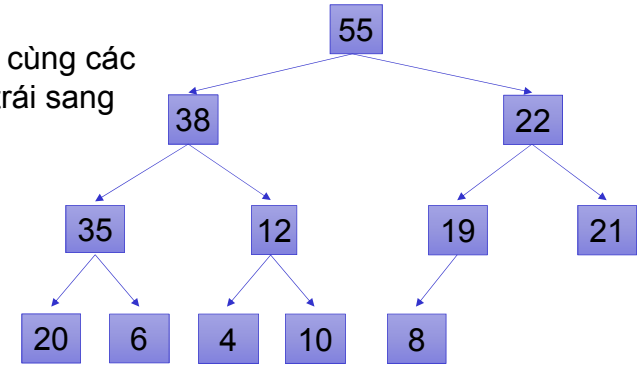
  Max-Heap: Mọi phần tử trên cây <= phần tử cha

  Min-Heap: Mọi phần tử trên cây >= phần tử cha

7

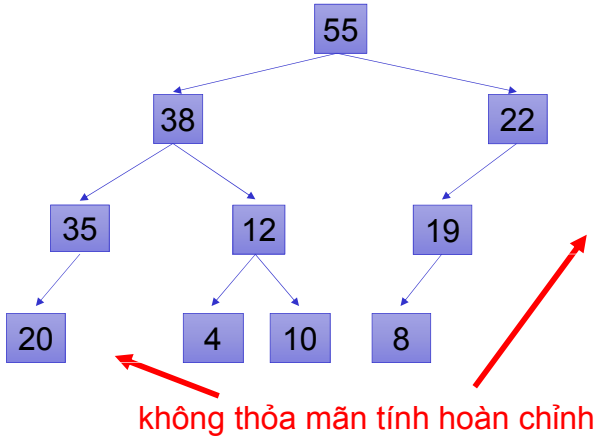
Completeness

- Mỗi mức của cây (trừ mức cuối cùng) được điền đầy.
- Ở mức cuối cùng các nút điền từ trái sang phải.



8

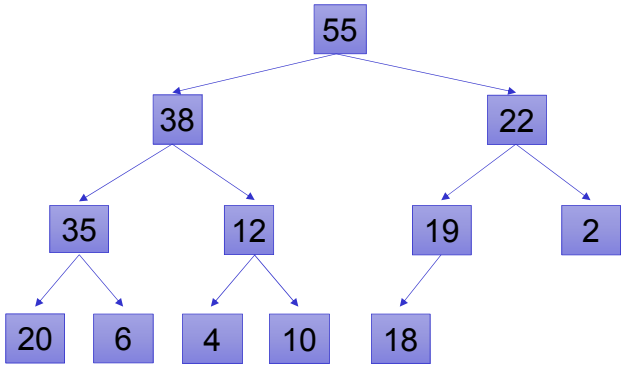
Completeness



9

Max-heap

Mọi phần tử trên cây <= phần tử cha

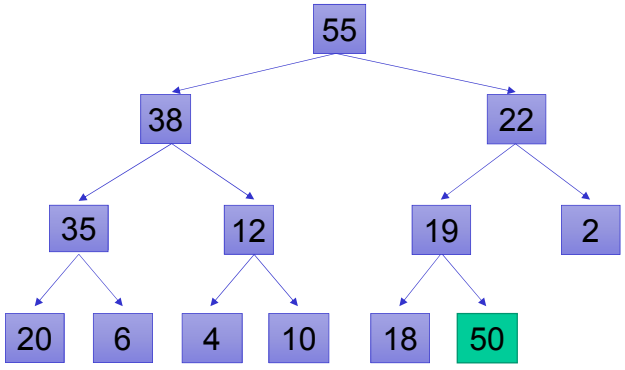


Cài đặt hàng đợi ưu tiên dùng heap

- Mỗi nút trong heap chứa độ ưu tiên của phần tử trong hàng đợi
- Độ phức tạp của các thao tác:
  - add(): O(log n)
  - poll(): O(log n)
  - peek(): O(1)
- Không có thao tác nào O(n): Tốt hơn cài đặt hàng đợi ưu tiên dùng danh sách liên kết
- Tốt hơn cây cân đối: peek() có độ phức tạp O(1)

11

Heap: add()

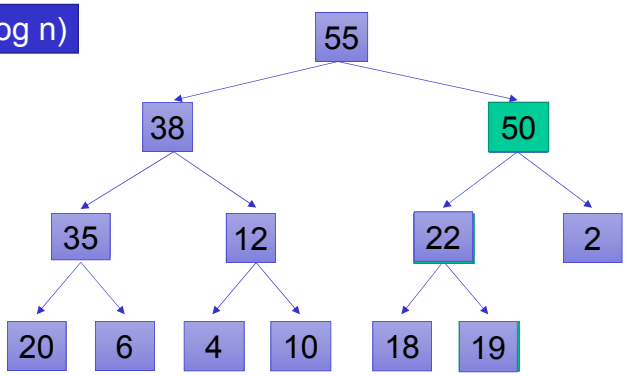


1. Thêm phần tử mới vào nút lá trống trái nhất

12

Heap: add()

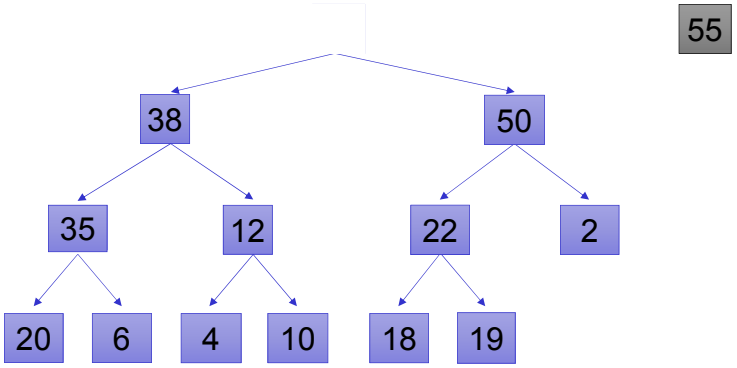
O(log n)



1. Thêm phần tử mới vào nút lá trống trái nhất
2. Đẩy phần tử đó lên trong khi còn lớn hơn nút cha

13

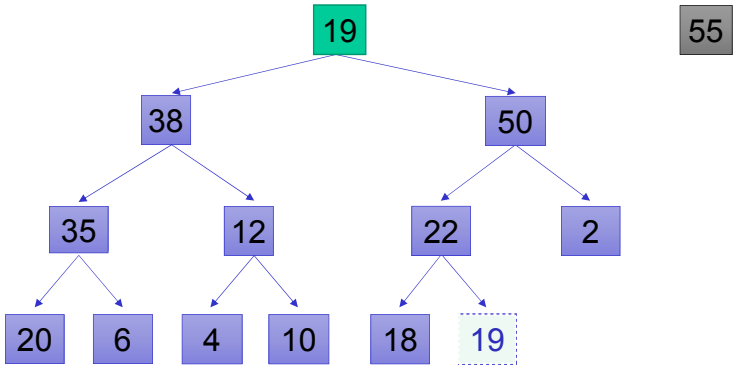
Heap: poll()



1. Ghi lại giá trị nút gốc vào một biến

14

Heap: poll()

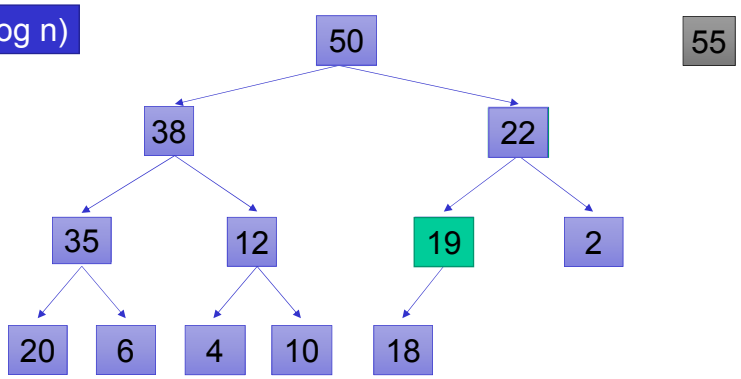


1. Ghi lại giá trị nút gốc vào một biến
2. Gán giá trị nút ở đáy cho nút gốc, xóa nút ở đáy

15

Heap: poll()

O(log n)

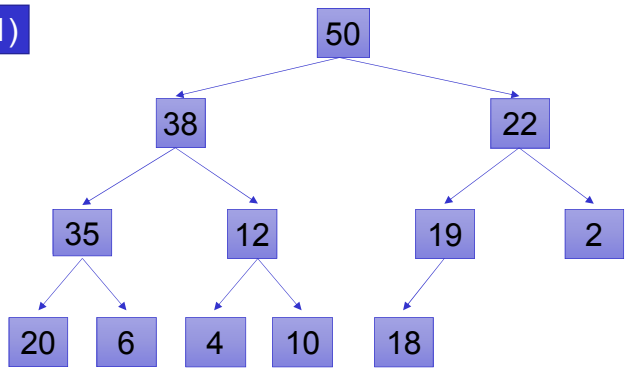


1. Ghi lại giá trị nút gốc vào một biến
2. Gán giá trị nút ở đáy cho nút gốc, xóa nút ở đáy
3. Đổi chỗ với nút con lớn trong khi còn nhỏ hơn giá trị nút con

16

Heap: peek()

O(1)



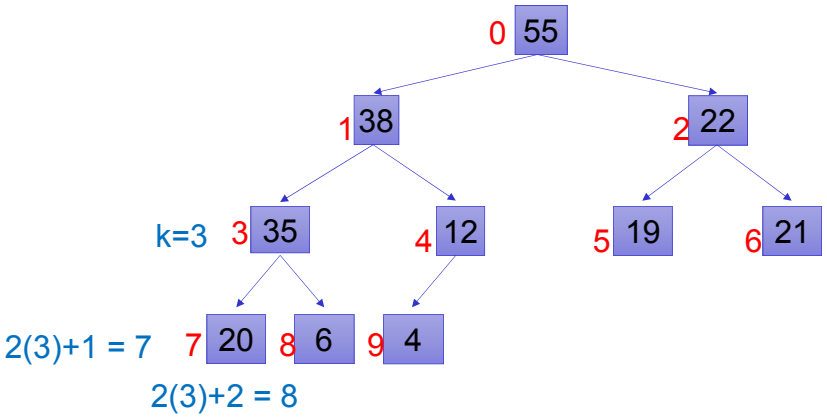
50

1. Trả về giá trị nút gốc

Cài đặt hàng đợi ưu tiên dùng heap

18

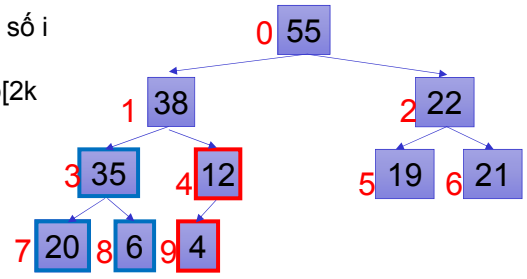
Biểu diễn cây bằng mảng: Đánh số các nút trên cây

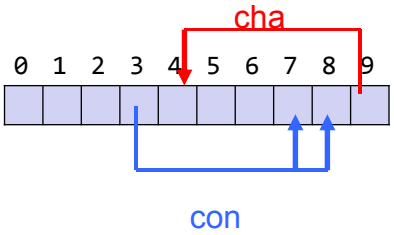


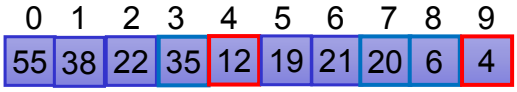
Con của **k** là nút **2k+1** và **2k+2**
Cha của nút **k** là nút **(k-1)/2**

Biểu diễn cây bằng mảng

- Nút thứ i chứa ở phần tử chỉ số i của mảng
- Con của b[k] là b[2k +1] và b[2k +2]
- Cha của b[k] là b[(k-1)/2]







Biểu diễn heap bằng mảng

```
int b[MAX];
int n; // hàng đợi của n phần tử b[0..n-1]
```

21

add() (Giả sử mảng còn trống)

```
// Thêm phần tử có độ ưu tiên e
void add(int e) {
    b[n]= e;
    n= n + 1;
    bubbleUp(n - 1);
}
```

22

add()

```
// Đẩy phần tử vị trí k vào đúng vị trí
void bubbleUp(int k) {
    // p là cha của k
   // và mọi phần tử trừ k là nhỏ hơn cha của nó
    int p= (k-1)/2;
    while (k > 0  && b[k] > b[p]) {
        swap(b[k], b[p]);
        k = p;
        p = (k-1)/2;
   }
```

23

peek()

```
// Trả về phần tử có độ ưu tiên lớn nhất
// trả về NULL nếu rỗng
int peek() {
     if (n == 0) return NULL;
     return b[0];   // Giá trị lớn nhất tại gốc
}
```

24

poll()

```
// Lấy phần tử có độ ưu tiên cao nhất khỏi hàng đợi
// Trả về NULL nếu rỗng
int poll() {
   if (n == 0) return NULL;
   int v= b[0]; // Phần tử ưu tiên cao nhất tại gốc
   n = n – 1;  // Chuyển phần tử đáy lên gốc
   b[0]= b[n];
   bubbleDown(); // Chuyển phần tử gốc về đúng vị trí
   return v;
}
```

25

poll()

```
// Chuyển phần tử gốc về đúng vị trí
void bubbleDown() {
   // b[0..n-1] là đogns trừ b[k]
   // và b[c] là nút con có giá trị lớn của b[k]
   int k= 0;
   int c= biggerChild(k);
   while (c < n &&  b[k] < b[c]) {
        swap(b[k], b[c]);
        k= c;
        c= biggerChild(k);
   }
}
```

26

poll()

```
// Trả về chỉ số của nút con có giá trị lớn của nút k
int biggerChild(int k) {
    int c= 2*k + 2;     // con phải của nút k
    if (c >= n || b[c-1] > b[c])
       c= c-1;
    return c;
}
```

27

Cấu trúc dữ liệu và giải thuật

- Nội dung bài giảng được biên soạn bởi TS. Phạm Tuấn Minh.

1-28