CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

Bài 04: Hàng đợi ưu tiên

Nguyễn Thị Tâm
nguyenthitam.hus@gmail.com

Ngày 14 tháng 3 năm 2025

Nội dung

1. Hàng đợi ưu tiên - Priority Queues
2. Đống nhị phân - Binary Heaps
3. Sắp xếp vun đống - Heapsort

Nội dung

Collections

Collection

Là một kiểu dữ liệu cho phép lưu trữ và quản lý một nhóm các đối tượng

data type	operations	data structure
stack	push, pop	linked list, resizing array
queue	enqueue, dequeue	linked list, resizing array
priority queue	insert, delete-max	binary heap
symbol table	put, get, delete	BST, hash table
set	add, contains, delete	BST, hash table

Hàng đợi ưu tiên - Priority queue

Collections: Chèn (insert) và xoá (delete). Xoá mục nào?

- Stack: Xoá phần tử được thêm vào cuối cùng
- Queue: Xoá phần tử được thêm vào cuối cùng
- Randomized queue: Xoá ngẫu nhiên
- Priority queue: Xoá phần tử lớn nhất hoặc nhỏ nhất

Priority queue API

Yêu cầu: Các mục dữ liệu kiểu **generic** => cần phải kế thừa từ **Comparable**

```
public interface MaxPriorityQueue<Key extends Comparable <Key>>
{
   public MaxPriorityQueue(); // create an empty priority queue
   public MaxPriorityQueue(Key []a); //create a priority queue with
       given keys
   public void insert(Key v);//insert a key into the priority queue
   public Key deleteMax();//return and remove a largest key
   public boolean isEmpty(); //is the priority queue empty?
   public Key max(); //return a largest key
   public int size(); //number of entries in the priority queue
}
```

Ứng dụng của hàng đợi ưu tiên

- Mô phỏng các sự kiện [khách hàng xếp hàng, các hạt va chạm]
- Tính toán số học [giảm lỗi làm tròn]
- Tối ưu hóa rời rạc. [đóng gói thùng, lập lịch]
- Trí tuệ nhân tạo. [Tìm kiếm A*]
- Mạng máy tính. [web cache]
- Hệ điều hành. [cân bằng tải, xử lý gián đoạn]
- Nén dữ liệu. [Mã Huffman]
- Tìm kiếm đồ thị. [Thuật toán Dijkstra, Thuật toán Prim]
- Lý thuyết số. [Tổng các lũy thừa]
- Lọc thư rác. [Bộ lọc thư rác Bayesian]
- Thống kê. [Trung vị trực tuyến trong luồng dữ liệu]

Hàng đợi ưu tiên - Ví dụ

Operation	Argument	Return value
insert	P	
insert	Q	
insert	E	
remove max		Q
insert	X	
insert	A	
insert	M	
remove max		X
insert	P	
insert	L	
insert	E	
remove max		P

Cài đặt bằng mảng không có thứ tự

Operation	Argument	Return value	Size	Content
insert	P		1	P
insert	Q		2	P Q
insert	E		3	P Q E
remove max		Q	2	P E
insert	X		3	P E X
insert	A		4	P E X A
insert	M		5	P E X A M
remove max		X	4	P E M A
insert	P		5	P E M A P
insert	L		6	P E M A P L
insert	E		7	P E M A P L E
remove max		P	6	E E M A P L

Cài đặt bằng mảng không có thứ tự

```
public class UnorderedMaxPriorityQueue<Key extends
   Comparable<Key>> implements MaxPriorityQueue<Key> {
      private Key[] queue;
      private int n;
      public UnorderedMaxPriorityQueue(int capacity) {
          n = 0;
          queue = (Key[]) new Comparable[capacity];
      }
      public void insert(Key v) {
       queue[n++] = v;
      }
   ...
}
```

Cài đặt bằng mảng không có thứ tự

```
public Key deleteMax() {
    int maxId = 0;
    for (int j = 1; j < n; j++)
        if (queue[j].compareTo(queue[maxId]) > 0) {
            maxId = j;
        }
    swap(maxId, n-1);
    return queue[--n];
}
public int size() {
    return n;
}
```

Cài đặt bằng mảng có thứ tự

Operation	Argument	Return value	Size	Content
insert	P		1	P
insert	Q		2	P Q
insert	E		3	E P Q
remove max		Q	2	E P
insert	X		3	E P X
insert	A		4	A E P X
insert	M		5	A E M P X
remove max		X	4	A E M P
insert	P		5	A E M P P
insert	L		6	A E L M P P
insert	E		7	A E E L M P P
remove max		P	6	A E E L M P

Độ phức tạp

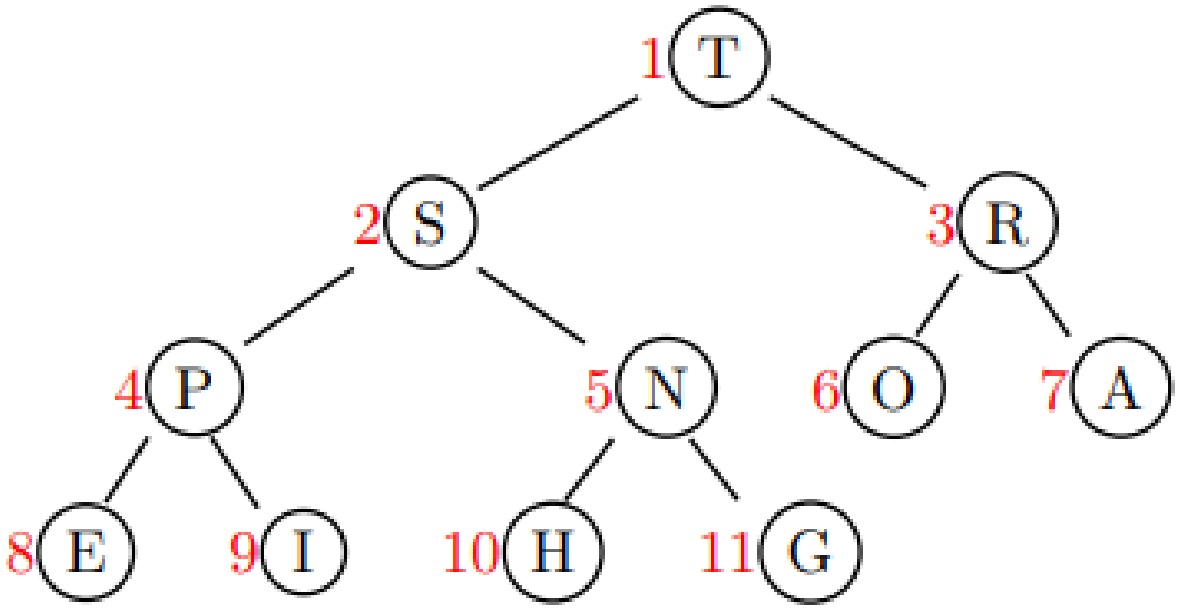
Cài đặt	Insert	Delete max	Max
Mảng không có thứ tự	$O(1)$	$O(n)$	$O(n)$
Mảng có thứ tự	$O(n)$	$O(1)$	$O(1)$
Mục tiêu	$O(logn)$	$O(logn)$	$O(1)$

Nội dung

Binary heap

- Là cây nhị phân hoàn chỉnh (tức là tất cả các tầng đều được điền đủ, ngoại trừ tầng cuối, và tầng cuối cần có càng nhiều phần tử bên trái càng tốt)
- Tính chất đống: Có hai loại đống nhị phân chính dựa trên cách sắp xếp giá trị:
  - Min-Heap: Giá trị của mỗi nút cha nhỏ hơn hoặc bằng giá trị của các nút con của nó. Nút gốc (root) là phần tử nhỏ nhất.
  - Max-Heap: Giá trị của mỗi nút cha lớn hơn hoặc bằng giá trị của các nút con của nó. Nút gốc là phần tử lớn nhất

Biểu diễn Binary Heap



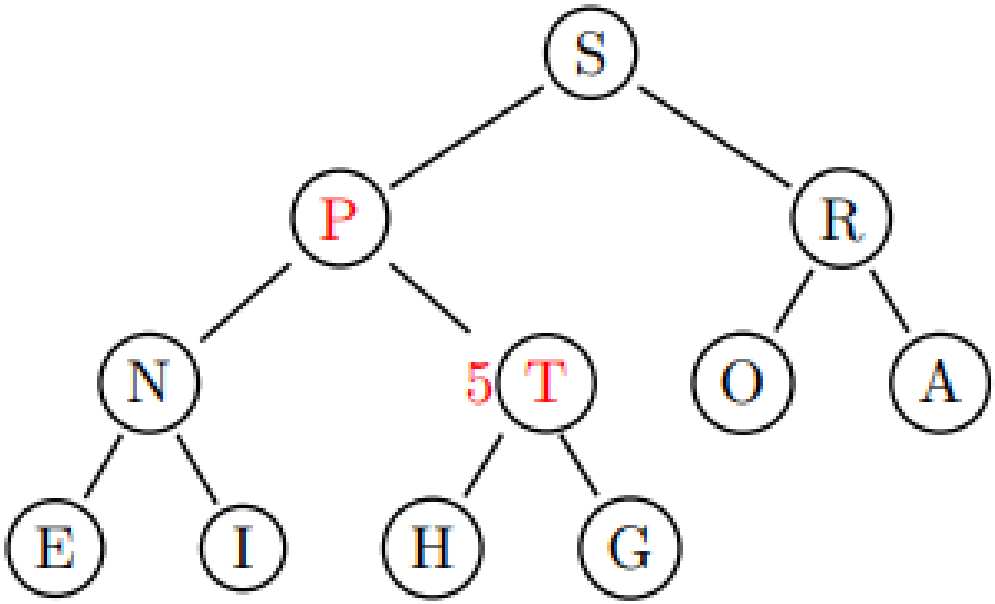
i	0	1	2	3	4	5	6	7	8	9	10	11
heap[i]	-	T	S	R	P	N	O	A	E	I	H	G

- Khoá lớn nhất là `heap[1]` là gốc của cây
- Nút cha của nút ở vị trí k là nút ở vị trí $k/2$
- Nút con của nút ở vị trí k là nút ở vị trí $2k$ và $2k+1$

Upheap

Khi giá trị khoá của nút con lớn hơn giá trị khoá của nút cha, cần thực hiện thao tác:

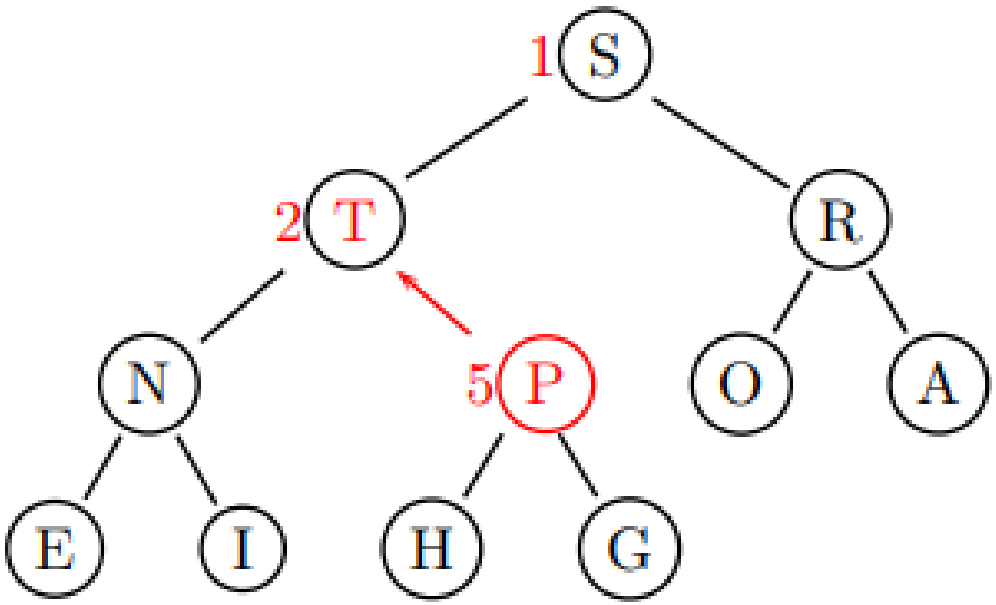
- Đổi khoá của nút con với nút cha
- Lặp lại quá trình đến khi heap đúng thứ tự



Upheap

Khi giá trị khoá của nút con lớn hơn giá trị khoá của nút cha, cần thực hiện thao tác:

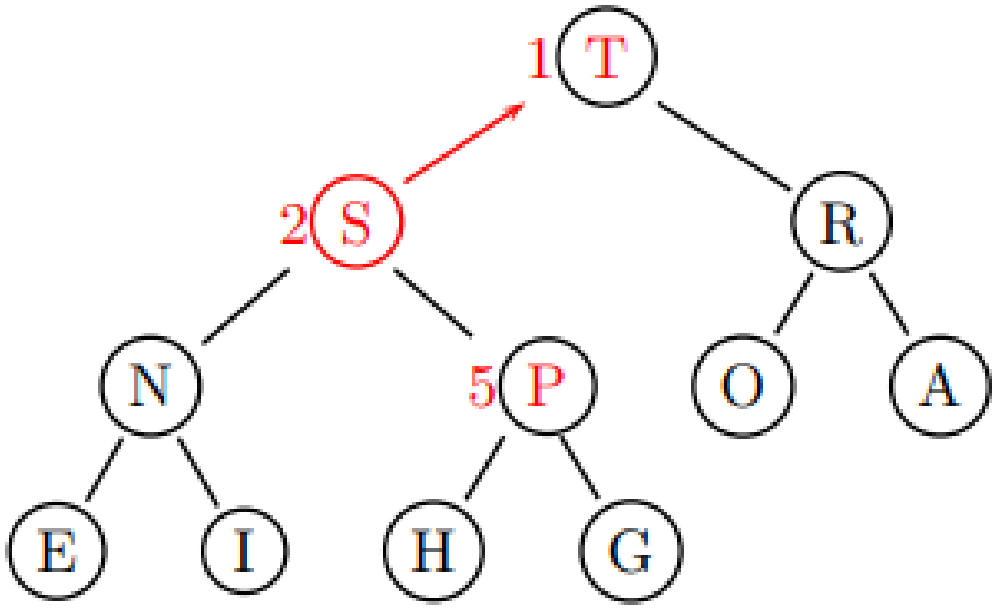
- Đổi khoá của nút con với nút cha
- Lặp lại quá trình đến khi heap đúng thứ tự



Upheap

Khi giá trị khoá của nút con lớn hơn giá trị khoá của nút cha, cần thực hiện thao tác:

- Đổi khoá của nút con với nút cha
- Lặp lại quá trình đến khi heap đúng thứ tự



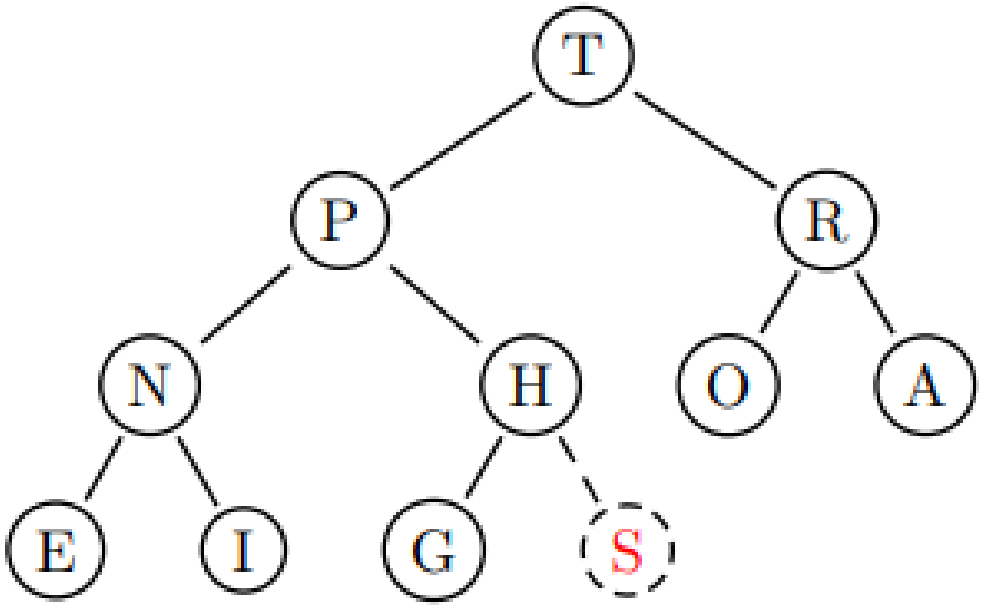
Upheap

```
private void upheap(int k) {
    while (k > 1 && heap[k/2].compareTo(heap[k]) < 0) {
        swap(k, k/2);
        k = k/2;
    }
}
```

Insert

```
public void insert(Key v) {
    heap[++n] = v;
    upheap(n);
}
```

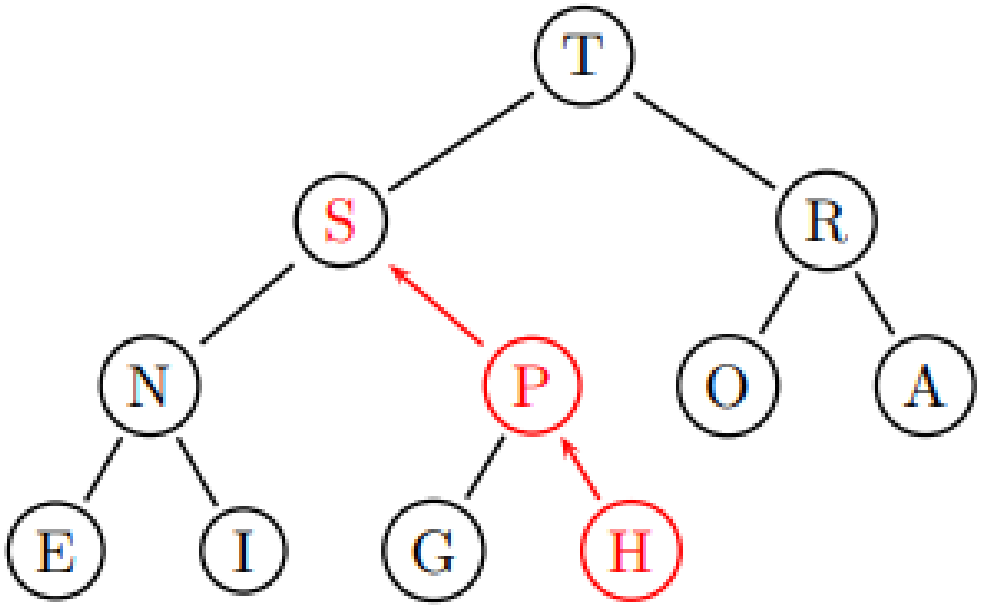
Cần $1 + log_2(n)$ phép so sánh



Insert

```
public void insert(Key v) {
    heap[++n] = v;
    upheap(n);
}
```

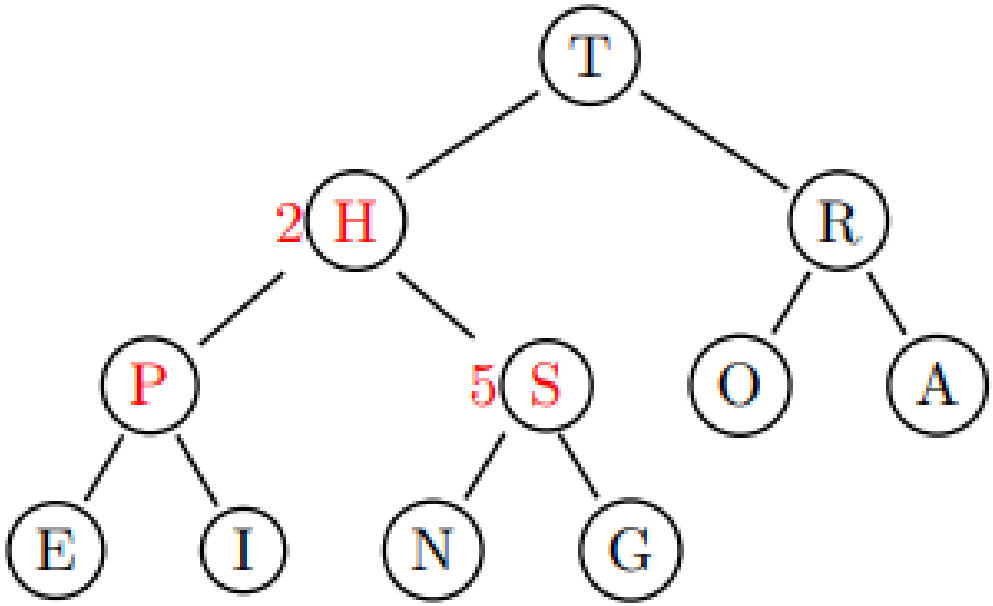
Cần $1 + log_2(n)$ phép so sánh



Downheap

Khi khoá của nút cha nhỏ hơn một (hoặc cả hai) khoá của nút con

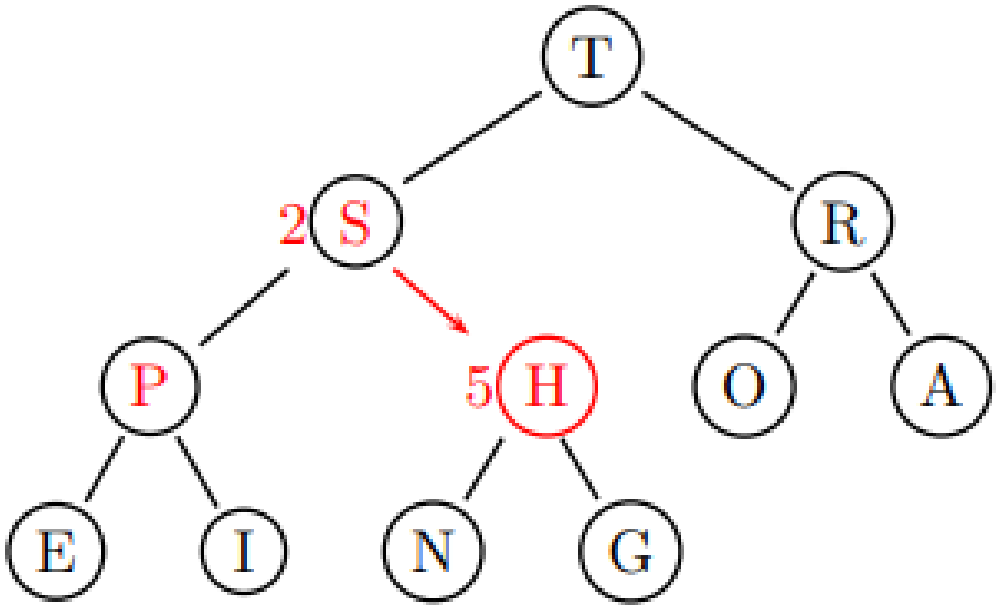
- Đổi giá trị khoá của cha với giá trị khoá của con
- Lặp lại quá trình đến khi heap đúng thứ tự



Downheap

Khi khoá của nút cha nhỏ hơn một (hoặc cả hai) khoá của nút con

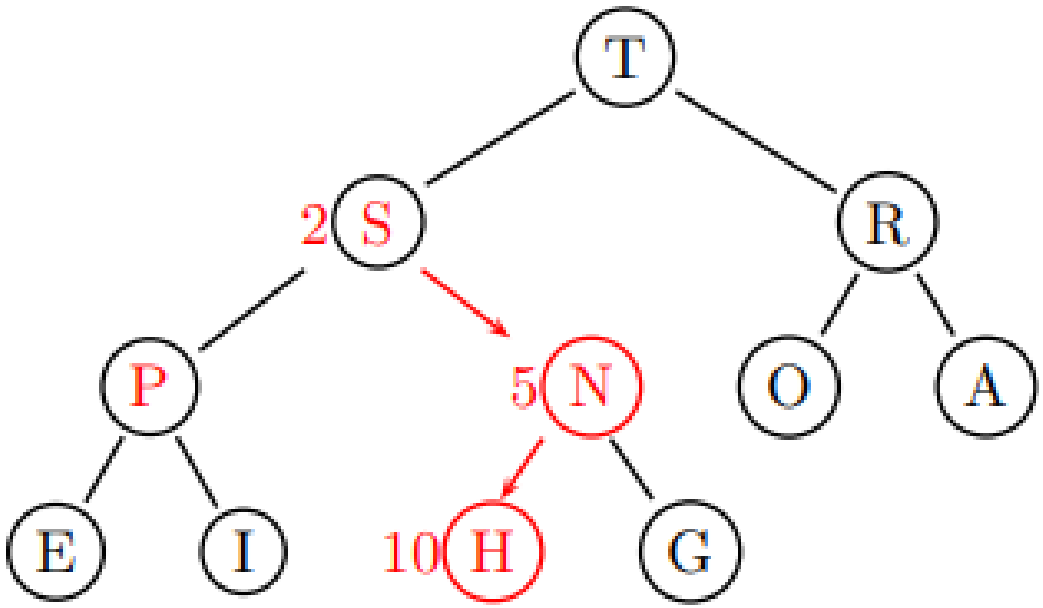
- Đổi giá trị khoá của cha với giá trị khoá của con
- Lặp lại quá trình đến khi heap đúng thứ tự



Downheap

Khi khoá của nút cha nhỏ hơn một (hoặc cả hai) khoá của nút con

- Đổi giá trị khoá của cha với giá trị khoá của con
- Lặp lại quá trình đến khi heap đúng thứ tự

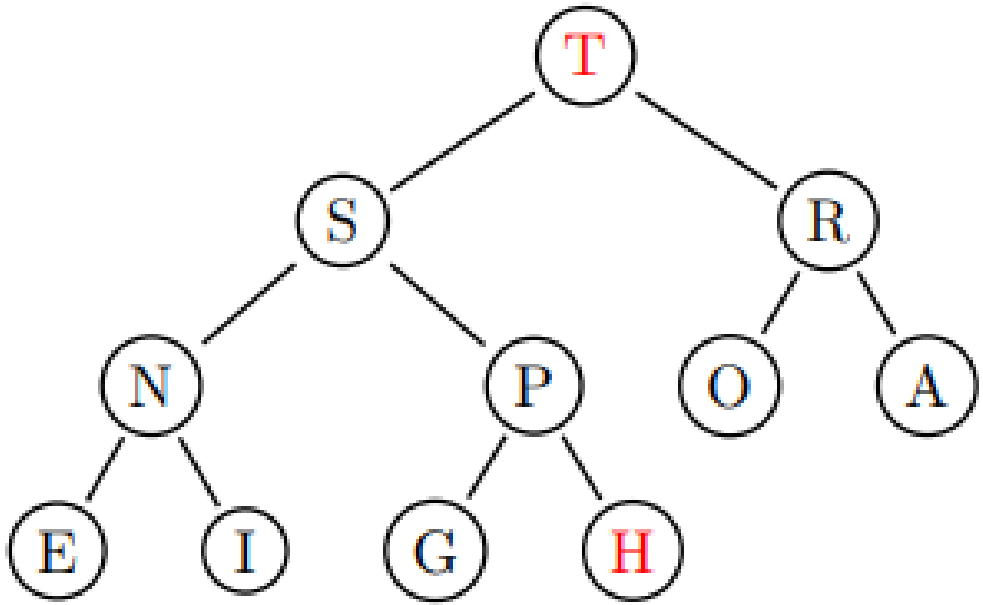


Downheap

```
private void downheap(int k) {
    while (2 * k <= n) {
        int j = 2*k;
        if (j < n && heap[j].compareTo(heap[j+1]) < 0)
            j++;
        if (heap[k].compareTo(heap[j]) > 0)
            break;
        swap(k, j);
        k = j;
    }
}
```

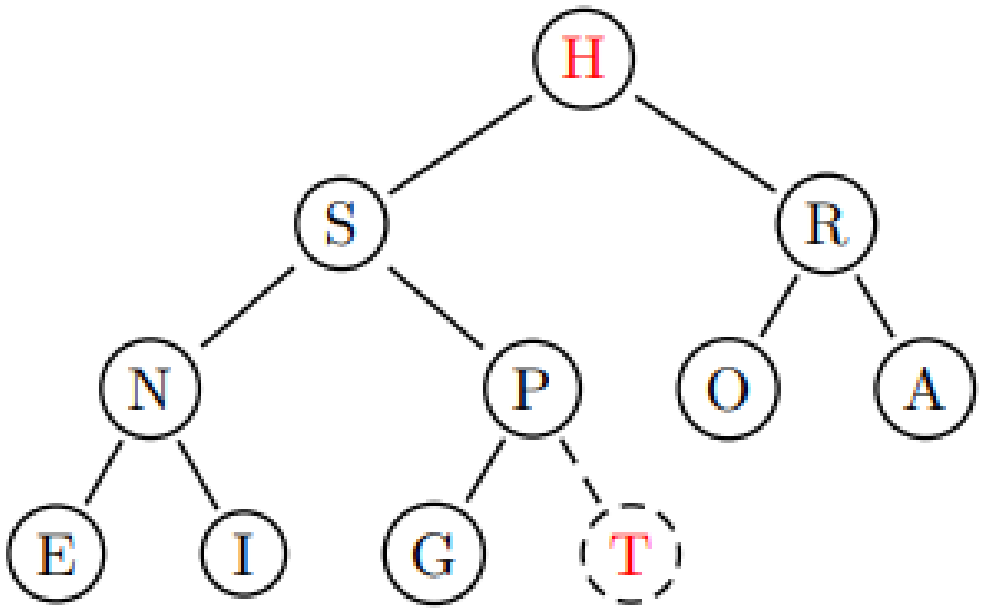
Delete Maximum

```
public Key deleteMax() {
     Key max = heap[1];
     swap(1, n--);
     downheap(1);
     heap[n+1] = null;
     return max;
 }
```



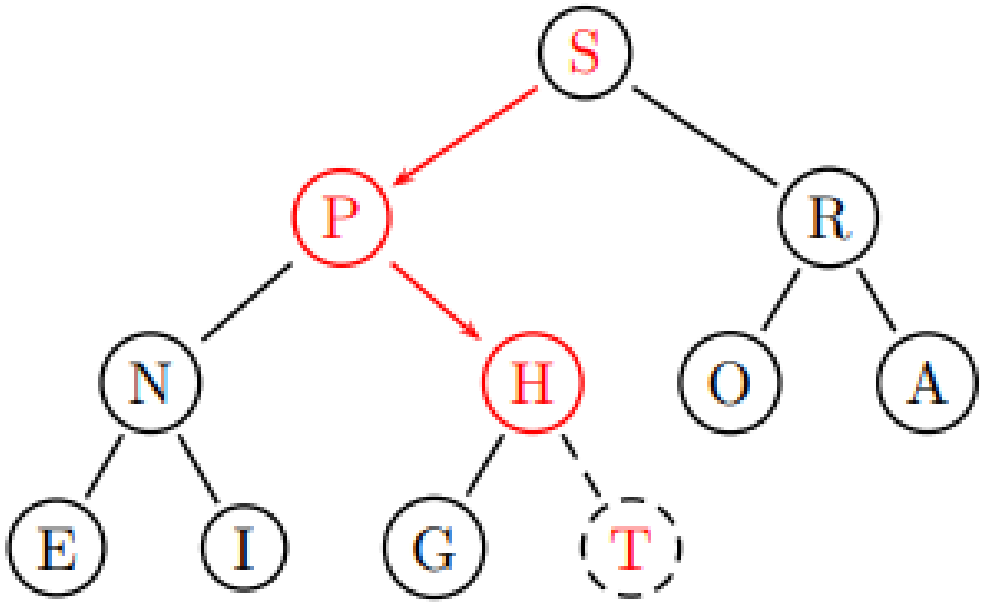
Delete Maximum

```
public Key deleteMax() {
     Key max = heap[1];
     swap(1, n--);
     downheap(1);
     heap[n+1] = null;
     return max;
 }
```



Delete Maximum

```
public Key deleteMax() {
     Key max = heap[1];
     swap(1, n--);
     downheap(1);
     heap[n+1] = null;
     return max;
 }
```



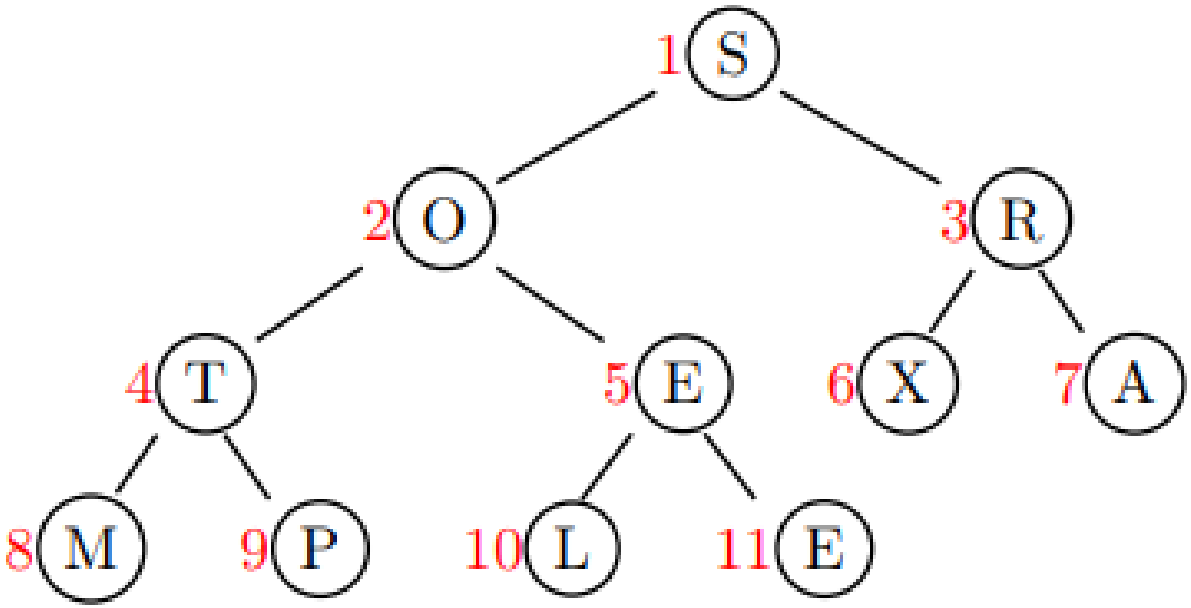
Độ phức tạp

Cài đặt	Insert	Delete Max	Max
unorder array	O(1)	O(n)	O(n)
order array	O(n)	O(1)	O(1)
binary heap	O(logn)	O(logn)	O(1)

Nội dung

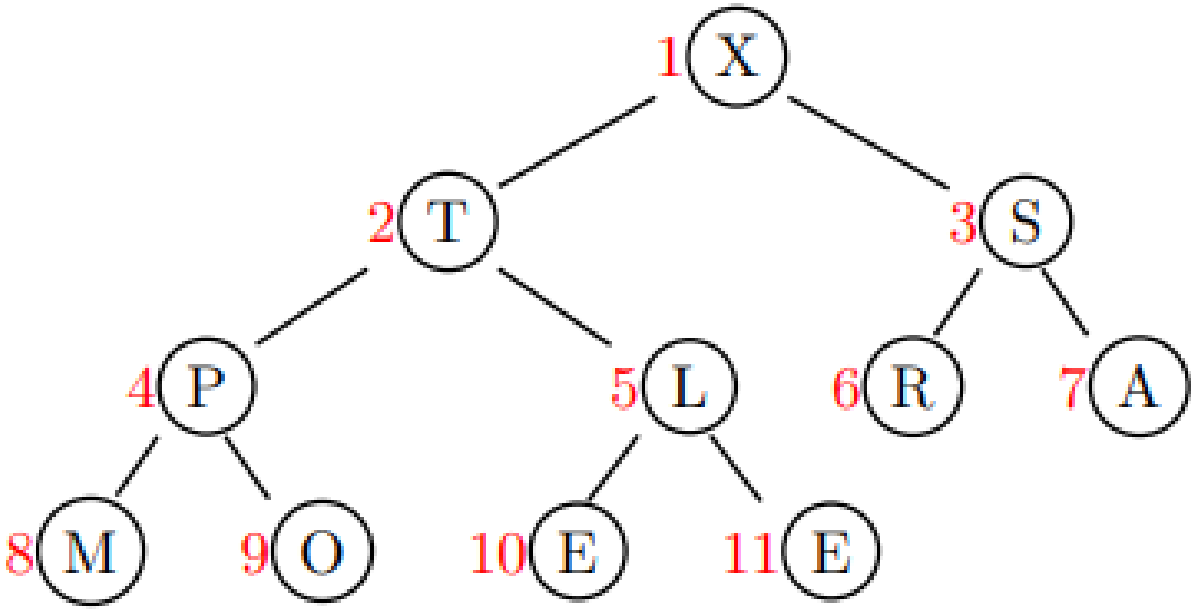
Heapsort

- Tạo đống của phần tử gốc có giá trị lớn nhất từ mảng đã cho
- Đổi chỗ gốc (phần tử lớn nhất) với phần tử cuối cùng trong mảng
- Loại bỏ nút cuối cùng bằng cách giảm kích thước của đống đi một
- Thực hiện vun lại đống với gốc mới
- Lặp lại quá trình đến khi đống chỉ còn một nút



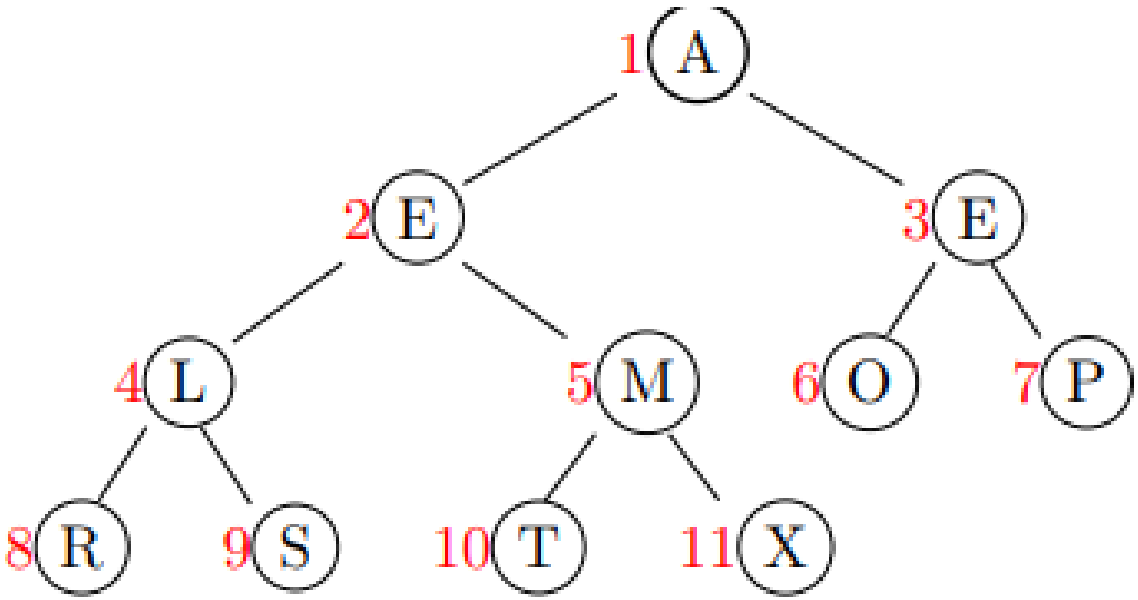
Heapsort

Tạo max-heap



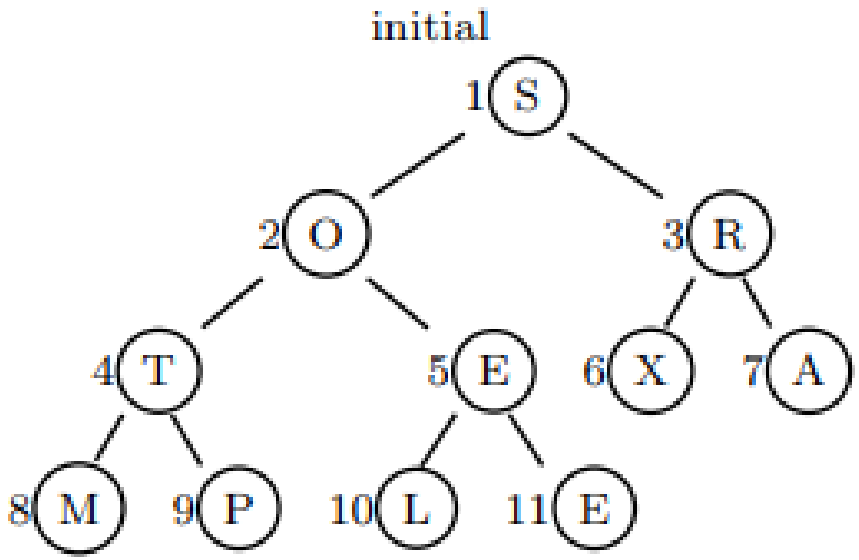
Heapsort

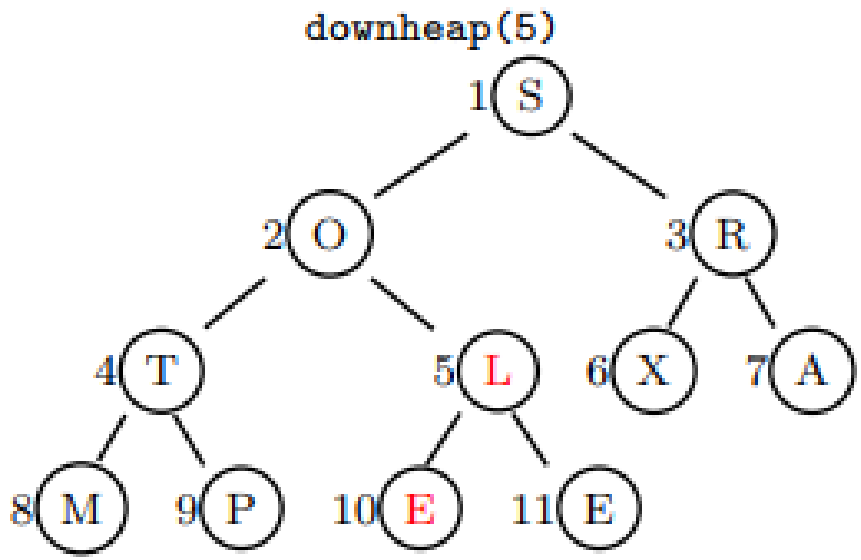
Kết quả sắp xếp



Sắp xếp tại chỗ

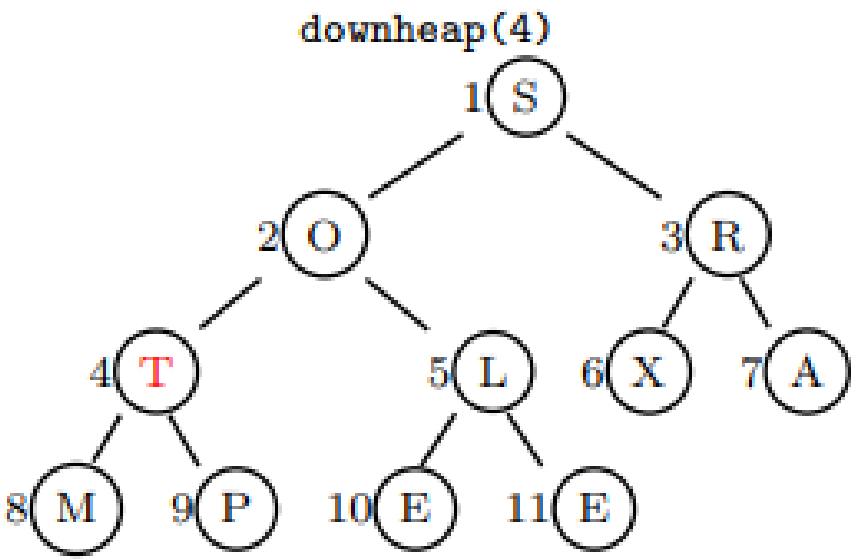
```
for (int k = n/2; k >= 1; k--) {
    downheap(k);
}
```

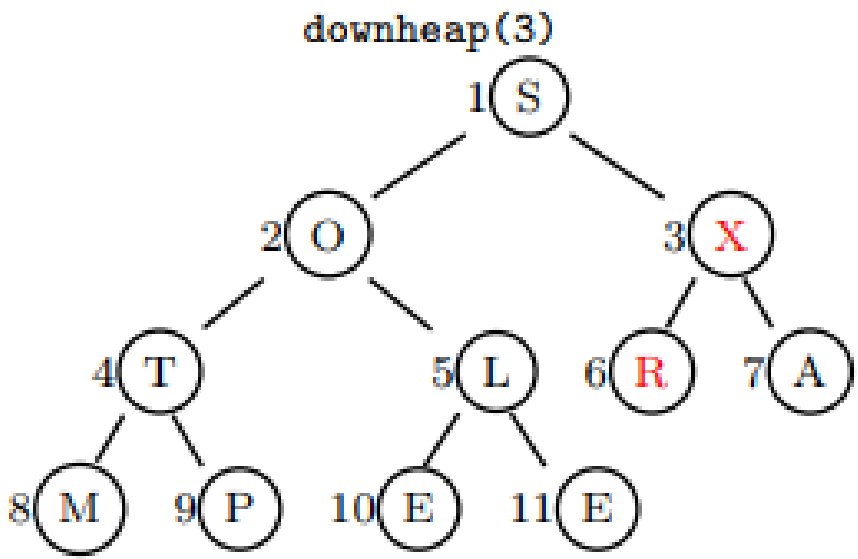




Sắp xếp tại chỗ

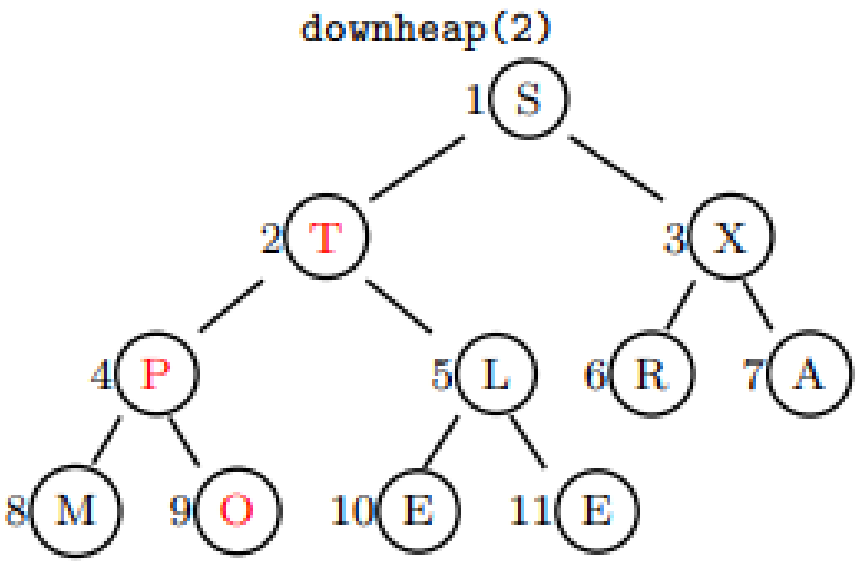
```
for (int k = n/2; k >= 1; k--) {
    downheap(k);
}
```

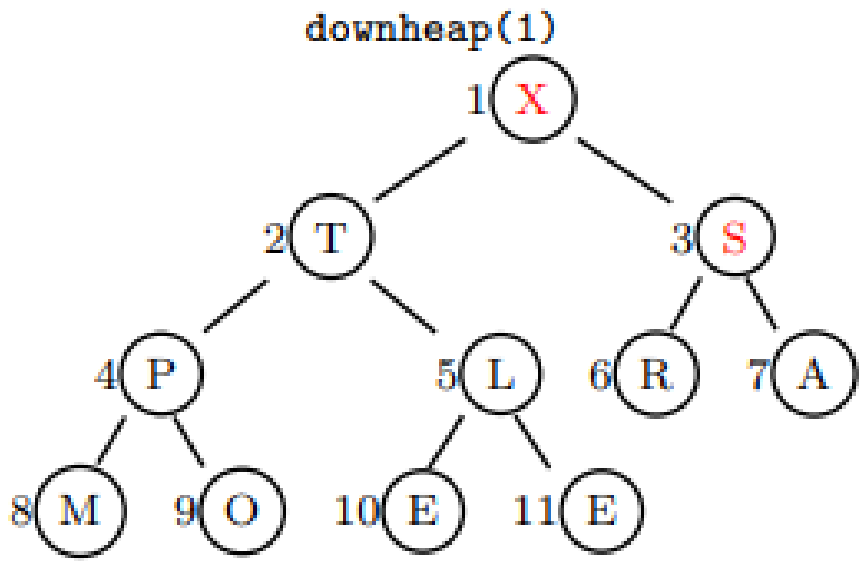




Sắp xếp tại chỗ

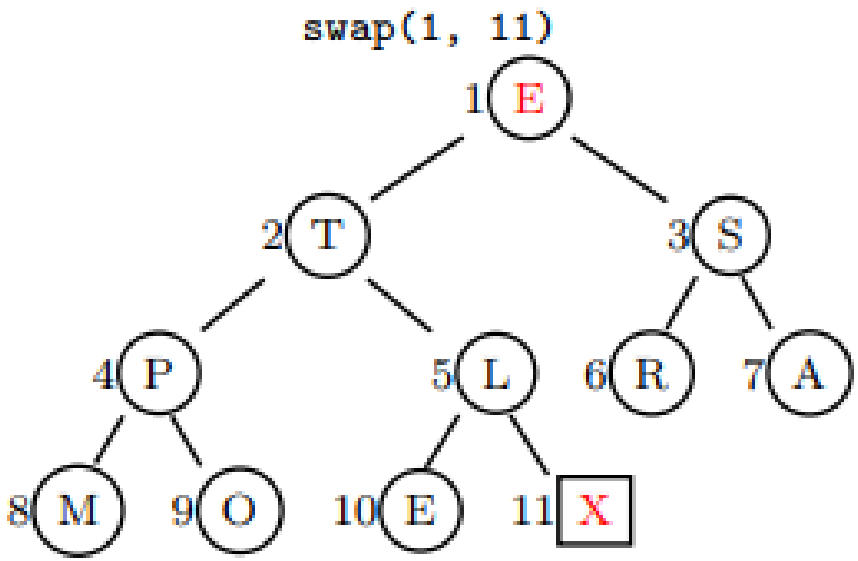
```
for (int k = n/2; k >= 1; k--) {
    downheap(k);
}
```

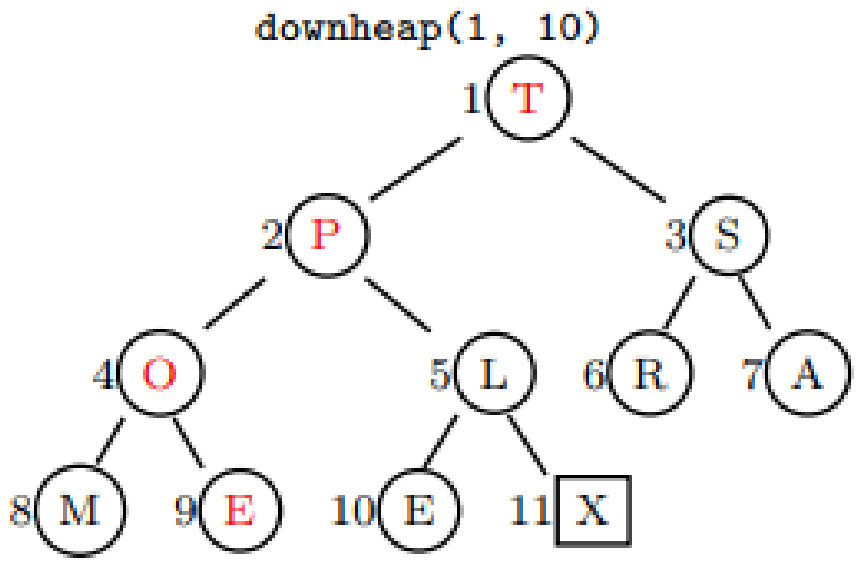




Sắp xếp tại chỗ

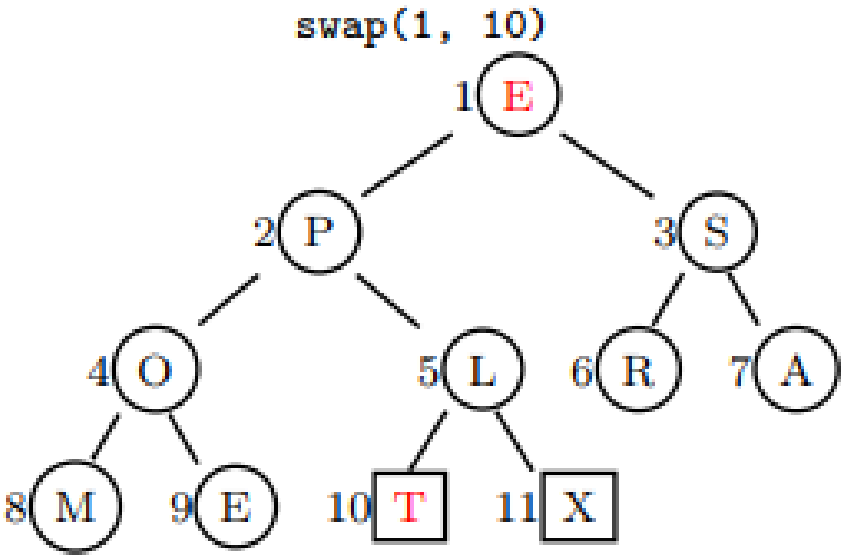
```
while (n > 1) {
    swap(1, n);
    n--;
    downheap(1, n);
}
```

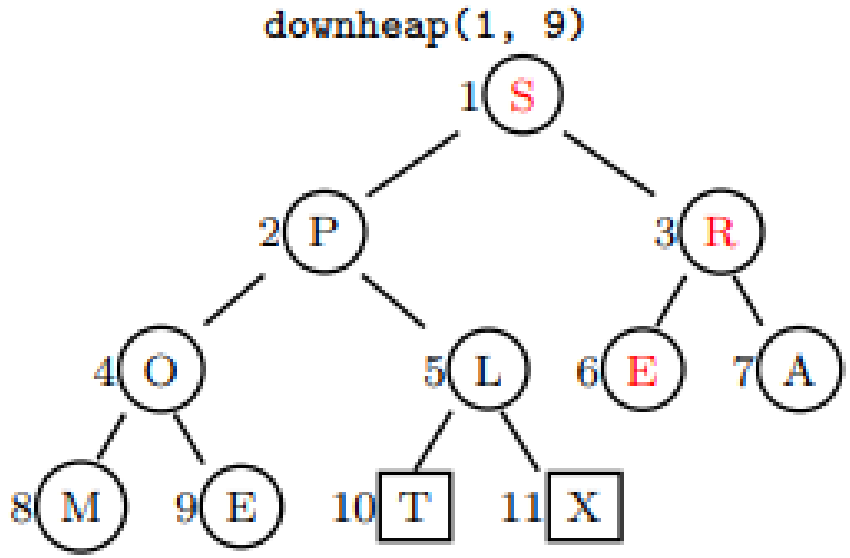




Sắp xếp tại chỗ

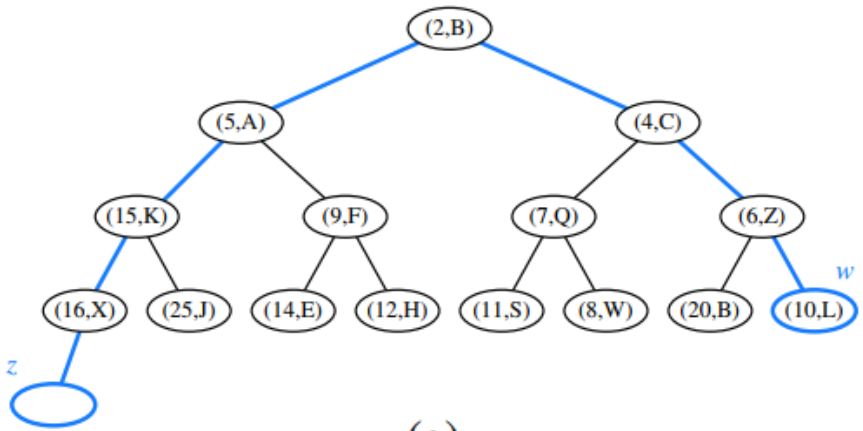
```
while (n > 1) {
    swap(1, n);
    n--;
    downheap(1, n);
}
```





Bài tập

1. Minh hoạ thuật toán heap sort với dãy đầu vào: (2, 5, 16, 4, 10, 23, 39, 18, 26, 15)
2. Trình bày cách cài đặt ngăn xếp sử dụng 1 hàng đợi ưu tiên và 1 biến nguyên
3. Cho một heap H và một khóa k, hãy thiết kế một thuật toán để tìm tất cả các phần tử trong heap có giá trị nhỏ hơn hoặc bằng k. Ví dụ: Giả sử heap được biểu diễn như trong hình sau, với truy vấn k = 7, thuật toán cần trả về các phần tử 2, 4, 5, 6, 7 (thứ tự không quan trọng). Lưu ý: không thay đổi thứ tự của đống



(a)