Cấu trúc dữ liệu Skew Heap

Đề tài: Skew Heap - Một cấu trúc dữ liệu dạng Heap tự tái cấu trúc

Nhóm thực hiện: Nguyễn Đình Lê Vũ (24120016), Nguyễn Phước Khang (24120189), Phạm Minh Đạt (24120174)
Giảng viên hướng dẫn: Thầy Nguyễn Thanh Phương, Thầy Nguyễn Thanh Tình

Đại Học Quốc Gia TP.HCM
Trường Đại Học Khoa Học Tự Nhiên
Khoa Công Nghệ Thông Tin

Ngày 25 tháng 5 năm 2025

Nội dung chính

1. Giới thiệu tổng quan
2. Các thao tác cơ bản
   - Trộn
   - Chèn một phần tử
   - Trích ra phần tử nhỏ nhất
3. Cài đặt và độ phức tạp
   - Thao tác trộn (`merge`)
   - Thao tác chèn (`insert`)
   - Thao tác trích phần tử nhỏ nhất (`extractMin`)
4. So sánh với các cấu trúc Heap khác
   - Binary Heap
   - Leftist Heap
   - Skew Heap

Khái niệm Skew Heap

Skew Heap [1] là một cấu trúc dữ liệu dạng heap tự điều chỉnh, được triển khai dưới dạng cây nhị phân, tập trung vào việc tối ưu hóa thao tác trộn (merge). Skew Heap không có ràng buộc cấu trúc nào ngoài tính chất heap cơ bản:

- Khóa của mỗi nút nhỏ hơn các khóa của nút con.

(*Qui ước:* Bài báo cáo này sẽ làm việc với min-heap).
Định nghĩa đệ quy của Skew Heap:

i. Cây rỗng là Skew heap.

ii. Một cây nhị phân có nút gốc mang khóa k và hai cây con là L, R, thì cây ấy là Skew heap nếu k bé hơn khóa của mọi nút trong cây L hoặc R (nếu có).

Lịch sử phát triển

Skew Heap là một biến thể của *Leftist Heap*, được thiết kế để đơn giản hóa quá trình trộn. Cấu trúc này được Daniel D. Sleator và Robert E. Tarjan giới thiệu lần đầu trong bài báo năm 1986, *Self-adjusting heaps*, đăng trên Tạp chí SIAM về Tính toán [2].

- Điểm sáng tạo: Thao tác trộn (merge) được sử dụng cho tất cả các thao tác chính (chèn, xóa, trộn).
- Độ phức tạp trung bình: $O(\log n)$ nhờ hoán đổi vô điều kiện các nút con trái/phải trên đường trộn để duy trì tính cân bằng [2].

Ứng dụng

Skew Heap được thiết kế cho *hàng đợi ưu tiên có thể trộn*, nơi trộn hai heap là thao tác thường xuyên. Các ứng dụng chính:

1. *Hàng chờ có độ ưu tiên*: Phù hợp cho hệ thống cần kết hợp hàng chờ, như thuật toán lập lịch hoặc mô phỏng sự kiện.
2. *Quản lý hàng chờ HPC*: Dự án Đại học Maryland (CMSC 341) dùng max-Skew Heap để ưu tiên công việc [3].
3. *Lập trình hàm*: Thư viện `Prioqueue` trong Elixir sử dụng Skew Heap [4].
4. *Thuật toán đồ thị*: Có thể dùng trong Dijkstra hoặc Prim, dù ít phổ biến hơn Binary Heap.

Một số cấu trúc dữ liệu liên quan

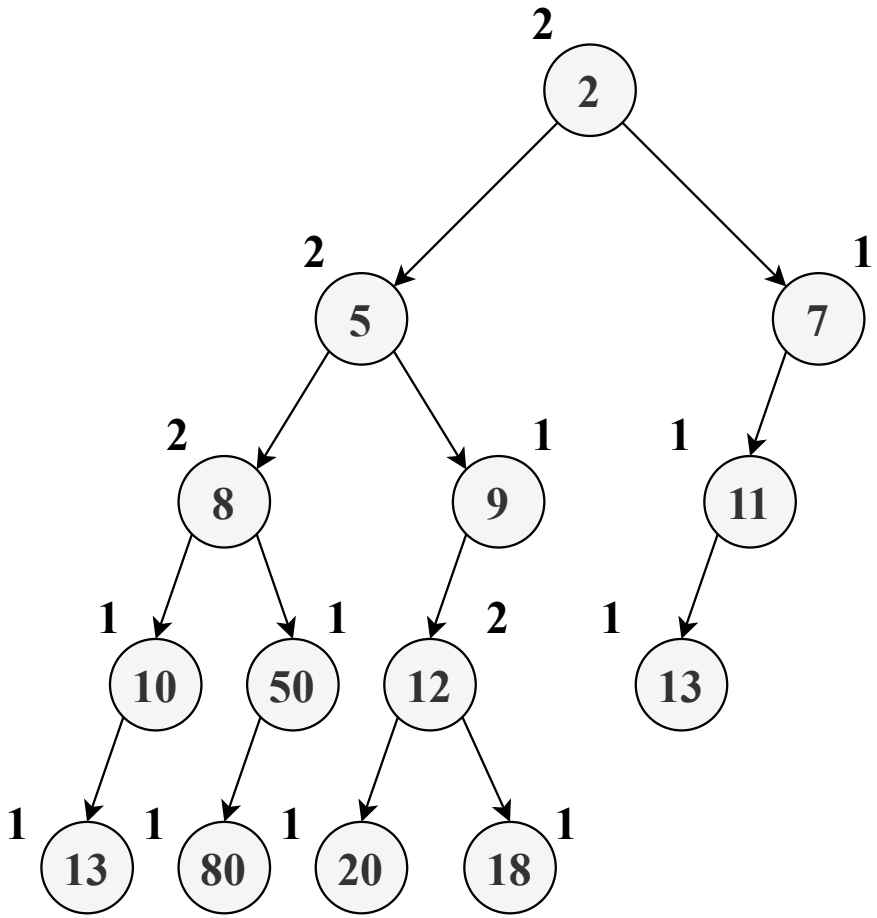
1. **Leftist Heap**: Duy trì tính chất heap và *độ dài đường dẫn null (NPL)*.
2. **Top-Down Skew Heap**: Biến thể của Skew-heap, thao tác trộn thực hiện ngược lại (từ trên xuống).

Leftist Heap

Leftist Heap là cây nhị phân duy trì tính chất heap và ràng buộc cấu trúc là *độ dài đường dẫn null* (NPL).

- NPL là độ dài đường dẫn ngắn nhất đến nút null.
- Đảm bảo NPL nút con trái $\geq$ nút con phải.
- Thời gian $O(\log n)$ cho trộn, chèn, xóa [5].
- Skew Heap là biến thể đơn giản hơn, loại bỏ NPL, dựa vào hoán đổi nút con [2].

Hình minh họa Leftist Heap



Hình: Leftist Heap với các giá trị NPL đặt cạnh các nút khóa

Bottom-Up Skew Heap

Bottom-Up Skew Heap là một biến thể so với Skew Heap thông thường (*Top-down Skew Heap*).

- Thao tác merge sử dụng một cách tiếp cận khác, dựa trên biểu diễn vòng (ring representation) và đi ngược lên các đường dẫn bên phải.
- Giống Skew Heap ở việc không có ràng buộc cấu trúc, khác ở hướng trộn.
- Theo Sleator và Tarjan [2], chi phí amortized của thao tác merge trong bottom-up skew heap là O(1).
- Hiệu suất phụ thuộc vào bài toán do hành vi bộ nhớ đệm hoặc mẫu đệ quy.

Các thao tác cơ bản

1. Trộn (`merge`)
2. Chèn một phần tử (`insert`)
3. Trích gốc (`deleteRoot`, hay `extractMin`)

Thao tác trộn (merge)

- Thao tác *trộn* Kết hợp hai Skew Heap thành một heap duy nhất, giữ nguyên tính chất của min-heap: khóa của mỗi nút nhỏ hơn khóa của các nút con.
- Skew Heap không duy trì ràng buộc cấu trúc như độ dài đường dẫn null (NPL), mà dựa vào việc hoán đổi nút con vô điều kiện để đảm bảo hiệu quả và tính đơn giản của thuật toán [2].

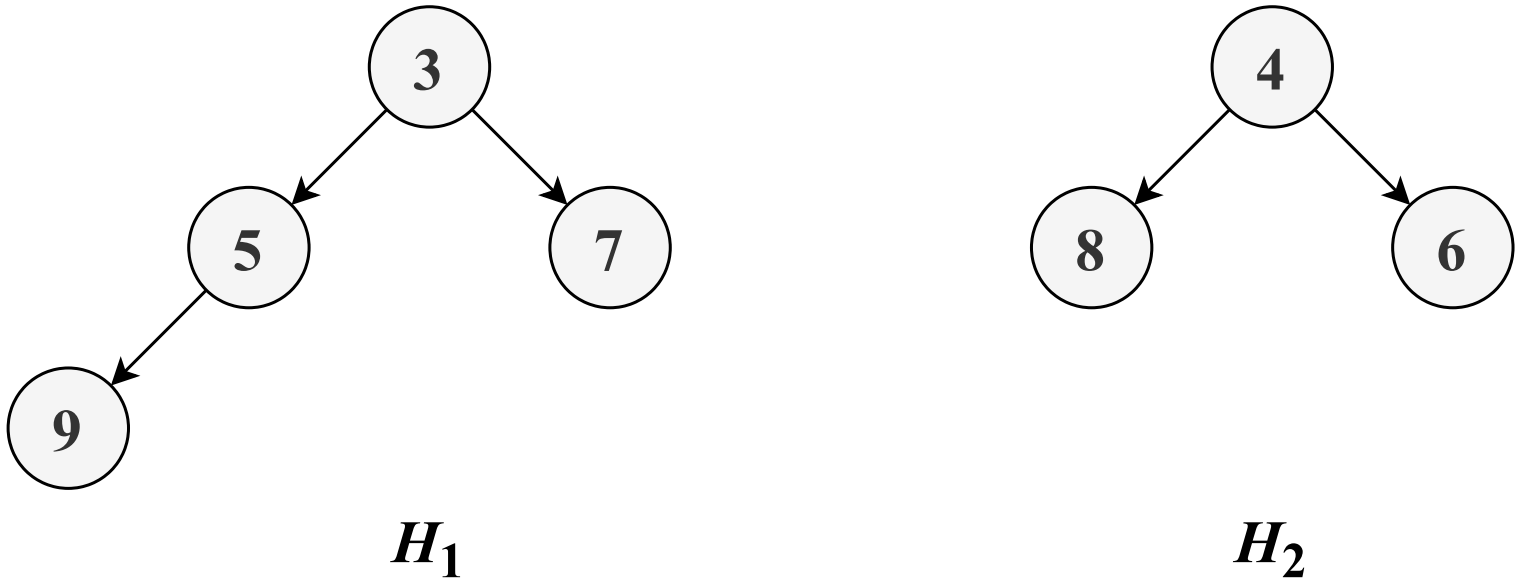
Mô tả thao tác trộn (merge)

Các bước của thao tác trộn (đệ quy đuôi) [6]:

1. *Trường hợp cơ bản*: Nếu một heap rỗng, trả về heap còn lại; nếu cả hai rỗng, trả về rỗng.
2. *So sánh khóa*: Chọn heap có khóa tại gốc nhỏ hơn (Heap A) làm gốc của Heap được trộn.
3. *Hoán đổi nút con*: Hoán đổi cây con trái và phải của gốc A. Sau khi hoán đổi, heap B (có khóa tại gốc lớn hơn) sẽ được trộn với cây con trái của A.
4. *Trộn*: Trộn cây con trái (sau hoán đổi) của A với B; cây con phải của A giữ nguyên.
5. *Trả về*: Heap với gốc tại A, cây con trái là kết quả trộn, cây con phải *giữ nguyên* như ban đầu.

Ví dụ thao tác trộn

Xét hai Skew Heap (H_1, H_2):



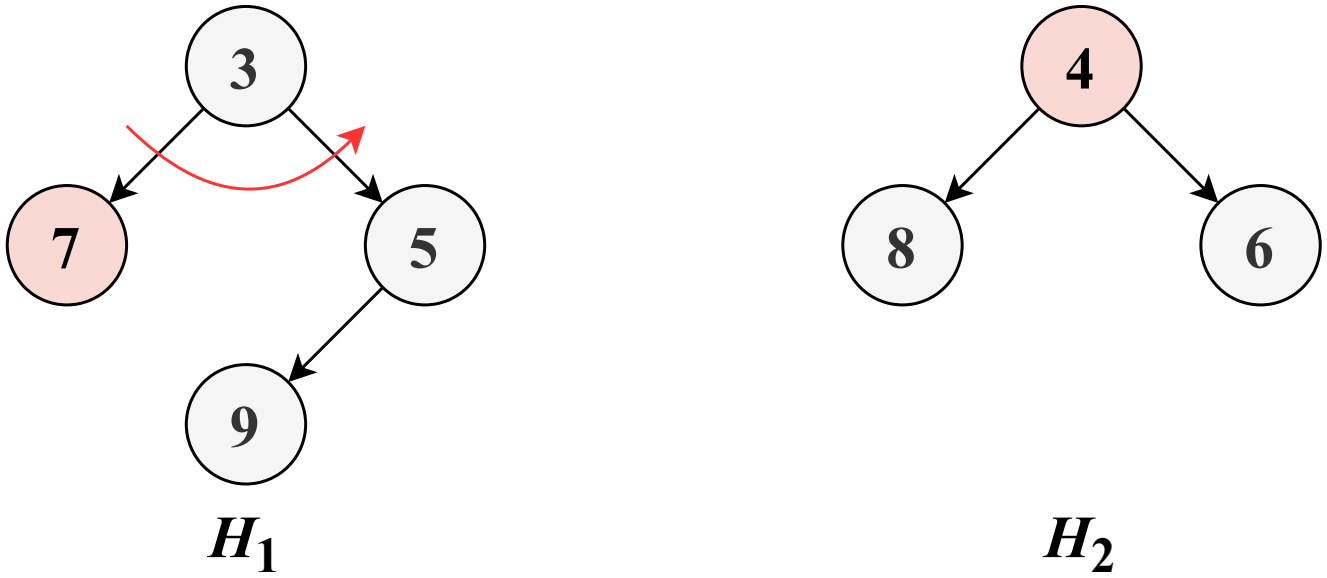
Hình: Hai Skew Heap cần trộn

- H_1 là cây bên trái, H_2 là cây bên phải.
- Mỗi mức đệ quy giữ nguyên nút cũ, tô màu nút gốc cần trộn.

Mức đệ quy thứ 1

Trộn hai cây có gốc 3 và 4:

- Vì $3 < 4$, chọn 3 làm gốc A, 4 làm gốc B.
- *Hoán đổi*: Hoán đổi nút con trái (5) và phải (7) của A (3). Ghi nhớ (7) (con trái của A) để trộn với B.

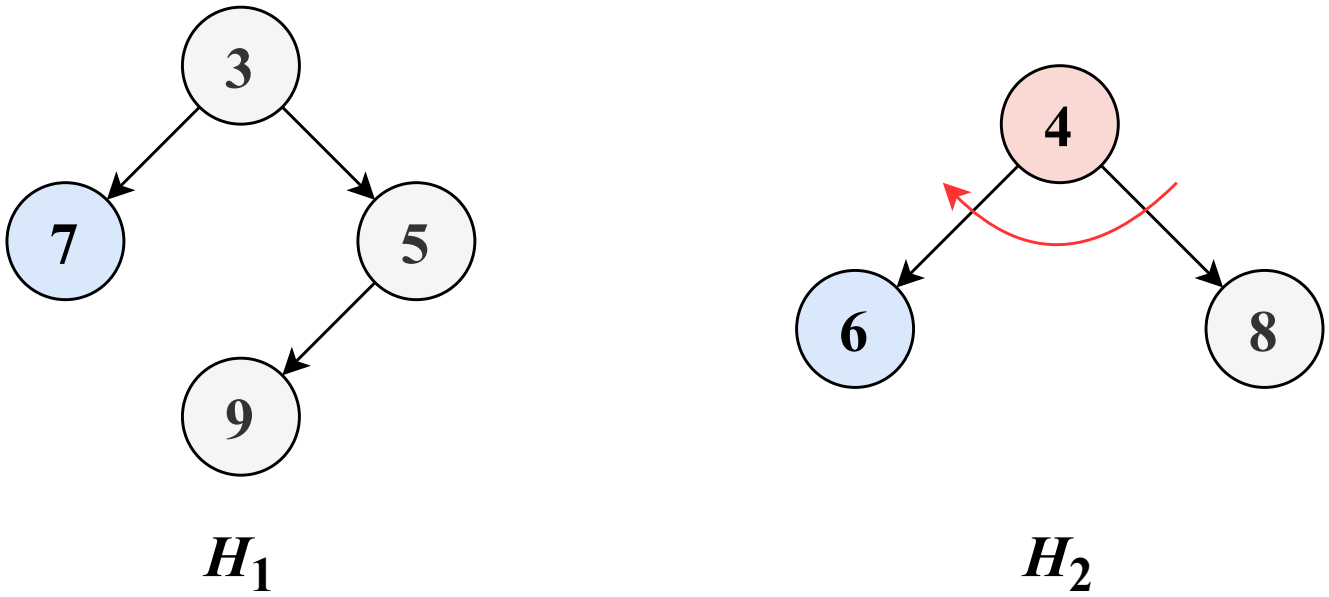


Hình: Mức đệ quy đầu tiên (nút 7, 4 màu đỏ)

Mức đệ quy thứ 2

Trộn hai cây có gốc 7 và 4:

- Vì $4 < 7$, chọn 4 làm gốc A, 7 làm gốc B.
- *Hoán đổi*: Hoán đổi nút con trái (8) và phải (6) của A (4). Ghi nhớ (6) (con trái của A) để trộn với B.



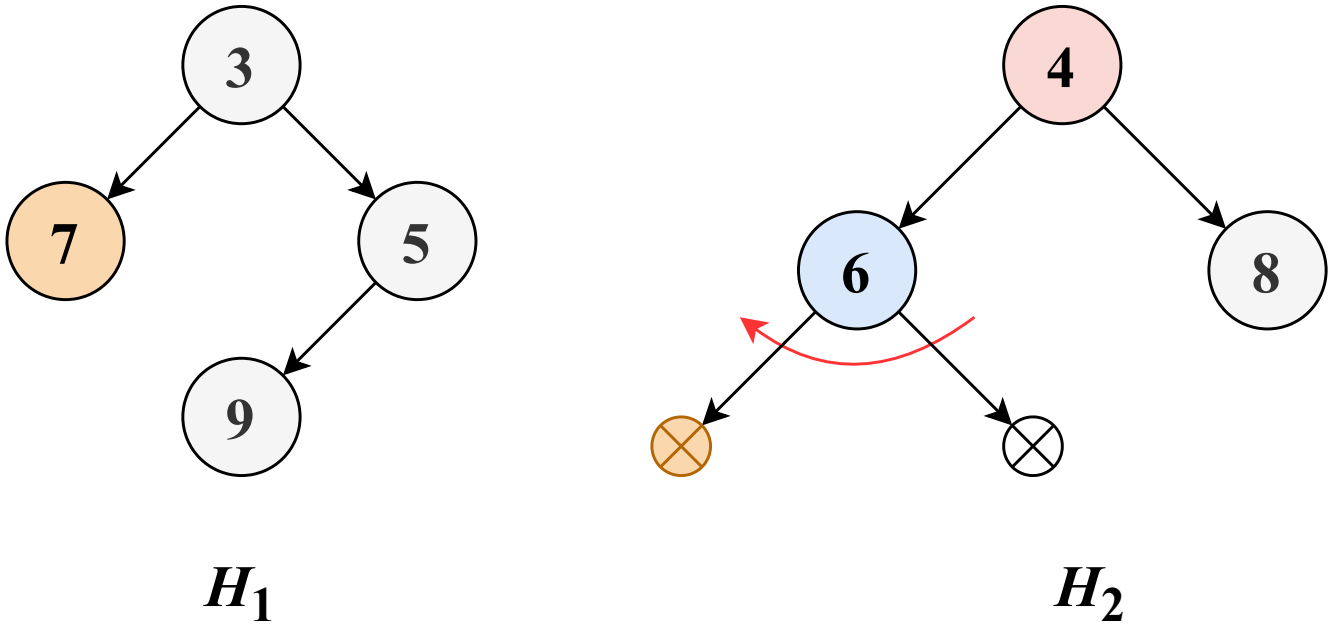
H_1 H_2

Hình: Mức đệ quy thứ 2 (nút 7, 6 màu xanh)

Mức đệ quy thứ 3

Trộn hai cây có gốc 7 và 6:

- Vì $6 < 7$, chọn 6 làm gốc A, 7 làm gốc B.
- *Hoán đổi*: Hoán đổi nút con trái (`NULL`) và phải (`NULL`) của A (6). Ghi nhớ `NULL` (con trái của A) để trộn với B.



Hình: Mức đệ quy thứ 3 (nút 7, NULL màu vàng)

Mức đệ quy thứ 4

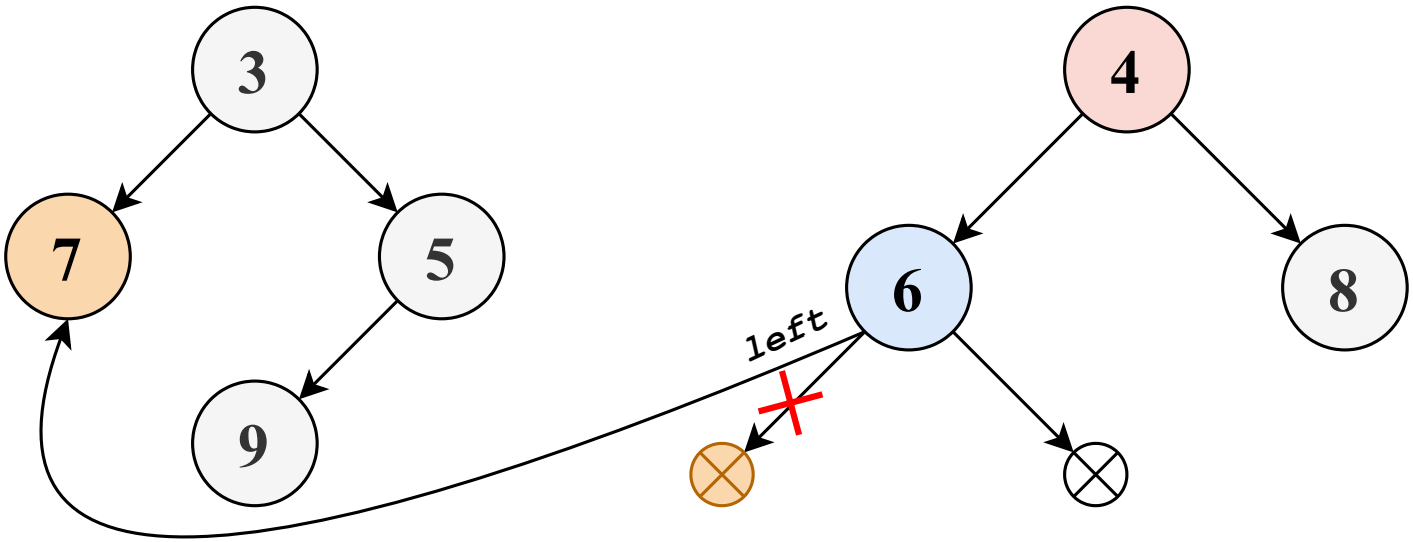
Trộn hai cây có gốc 7 và `NULL`:

- Một heap rỗng, trả về 7.

Quá trình trả đệ quy - Mức 3

Trả đệ quy về mức 3:

- Cây con trái của 6 trỏ đến 7 (kết quả trộn).

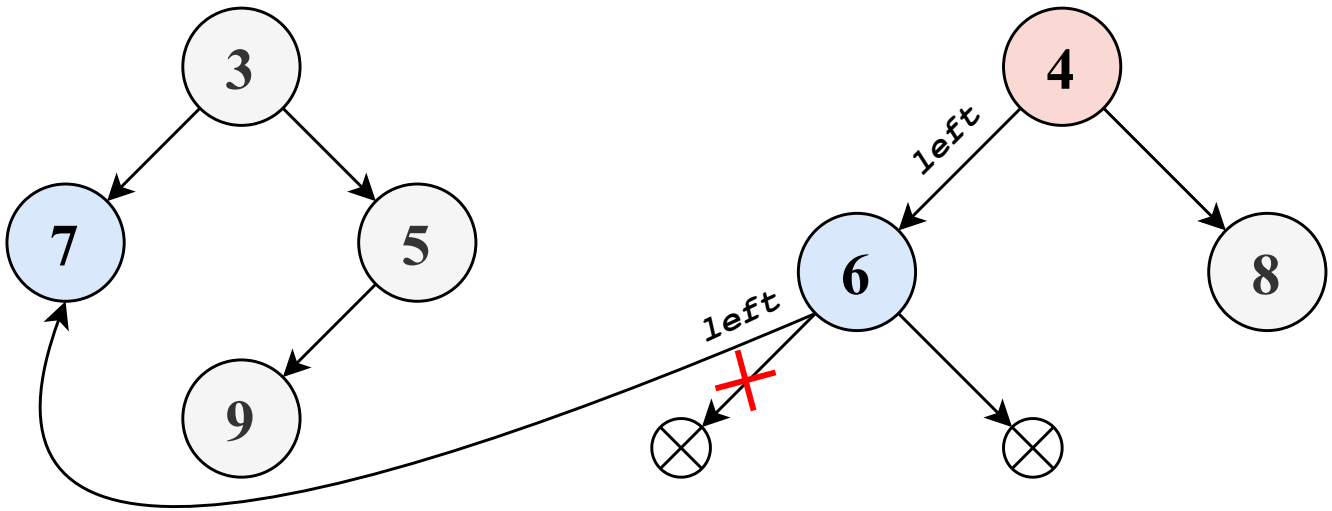


Hình: Trả đệ quy về mức 3

Quá trình trả đệ quy - Mức 2

Trả đệ quy về mức 2:

- Cây con trái của 4 trỏ đến 6 (kết quả trộn).
- Thực ra kết quả không thay đổi.

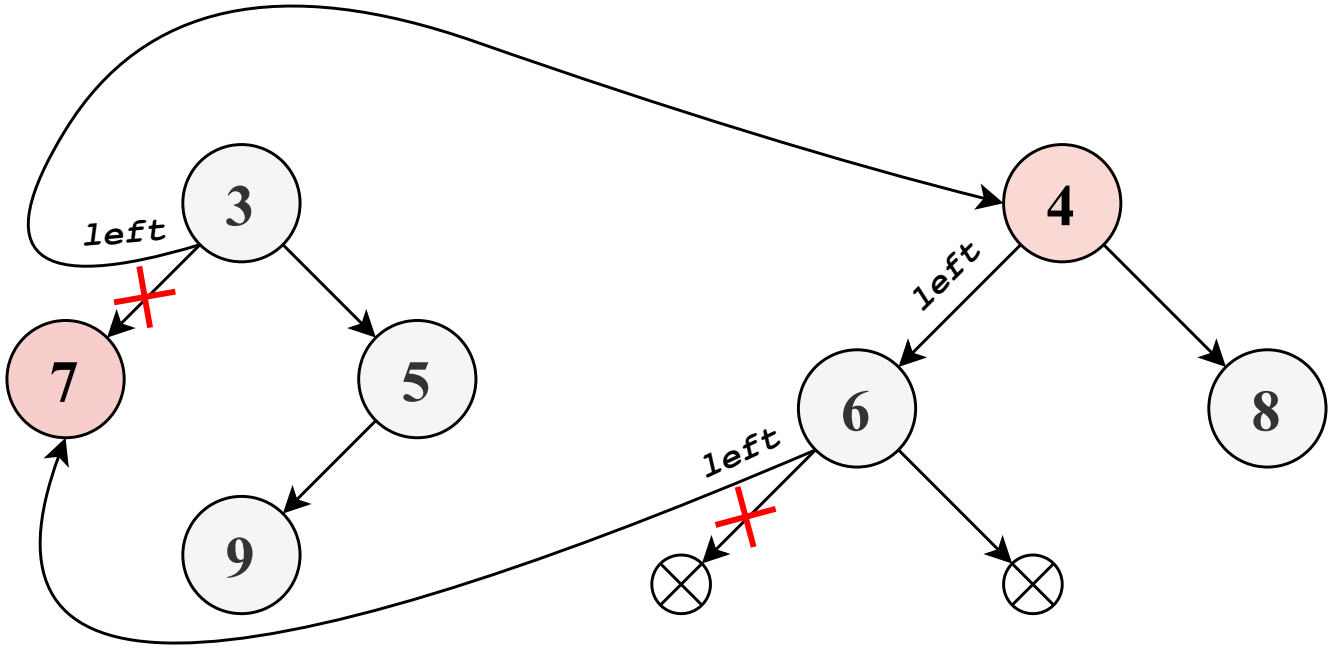


Hình: Trả đệ quy về mức 2

Quá trình trả đệ quy - Mức 1

Trả đệ quy về mức 1:

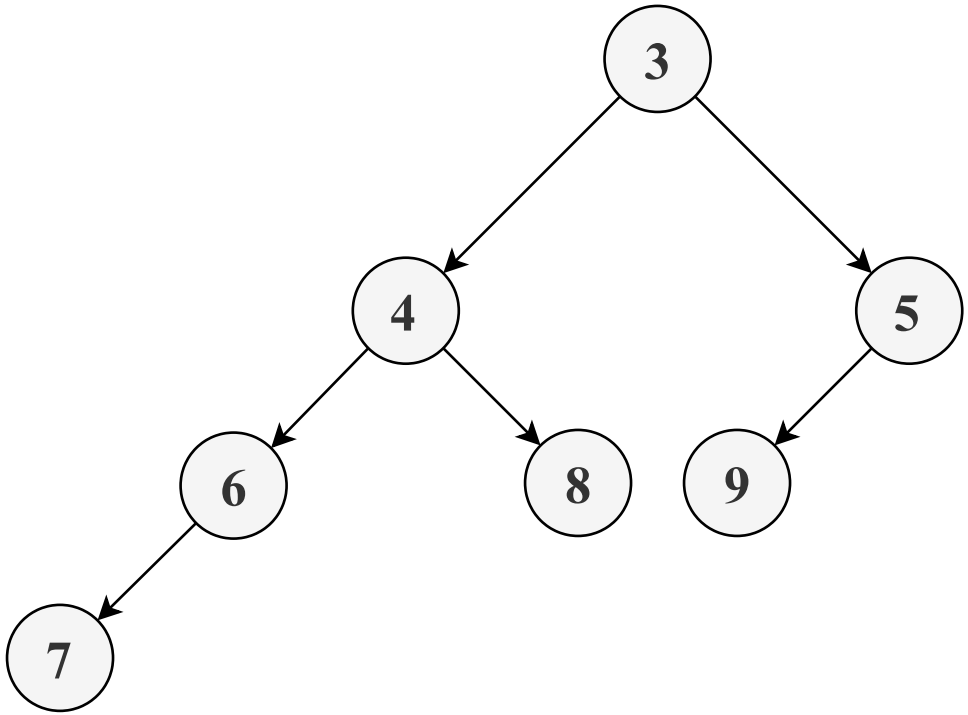
- Cây con trái của 3 trỏ đến 4.



Hình: Trả đệ quy về mức 1

Kết quả trộn

Sau khi trộn H_1 và H_2, kết quả là một skew heap (rõ ràng thỏa tính chất heap - khóa của một nút nhỏ hơn khóa của các nút con nếu có):



Hình: Kết quả của phép trộn Skew Heap H_1, H_2

Mở rộng: Trộn không đệ quy với Ngăn xếp

Ngoài cách cài đặt đệ quy, thao tác trộn có thể được thực hiện không đệ quy bằng cách sử dụng cấu trúc dữ liệu Ngăn xếp (Stack).
Các bước cơ bản:

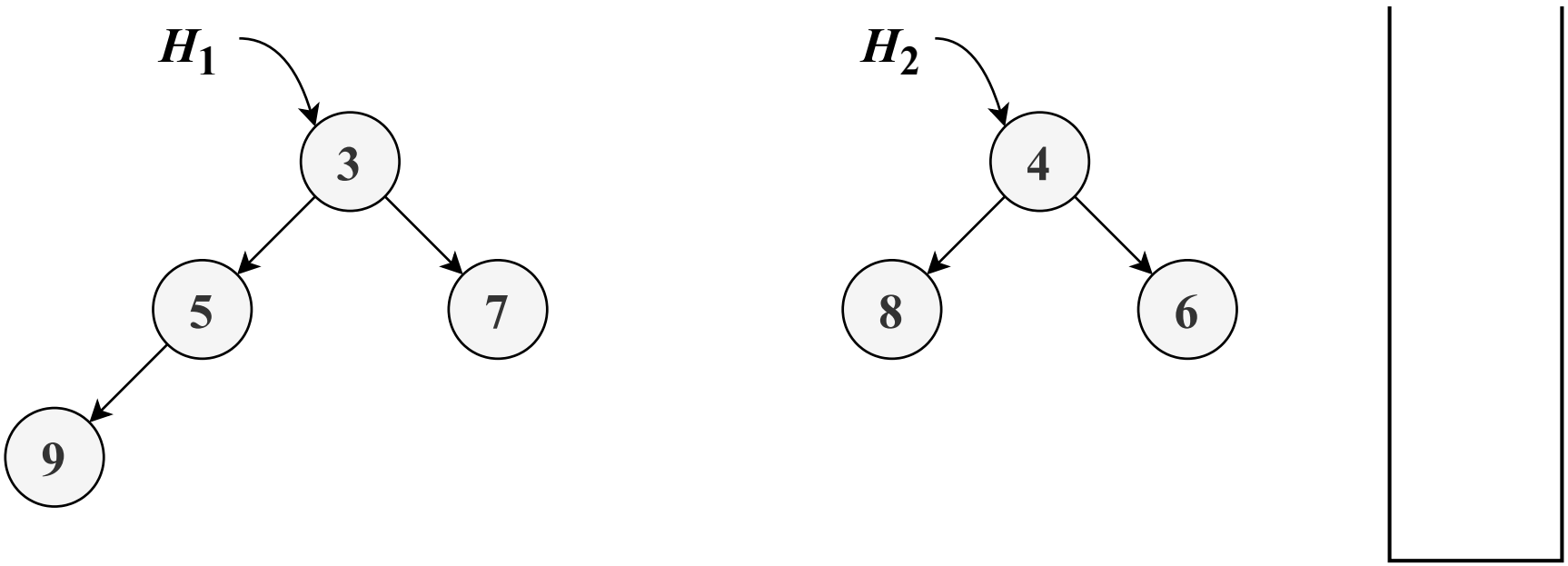
1. *Trường hợp cơ bản*: Nếu một heap rỗng, trả về heap còn lại; nếu cả hai rỗng, trả về rỗng.
2. *Khởi tạo ngăn xếp*: Tạo một ngăn xếp rỗng để lưu trữ các cặp nút.
3. *Vòng lặp chính*: Trong khi H_1 và H_2 khác rỗng:
   - *So sánh*: Nếu khóa gốc H_1 lớn hơn H_2, hoán đổi H_1 và H_2.
   - *Đẩy*: Đẩy nút gốc của H_1 vào ngăn xếp.
   - *Hoán đổi*: Hoán đổi cây con trái và phải của gốc H_1.
   - *Cập nhật*: Gán H_1 bằng cây con trái của nó.

Mở rộng: Trộn không đệ quy với Ngăn xếp

4. *Vòng lặp liên kết*: Gọi nút gốc của heap không rỗng là *nút gốc của heap đã trộn xong hiện tại*. Trong khi ngăn xếp khác rỗng:
   - *Trích*: Trích phần tử đầu tiên; cây con trái của nút trích trỏ đến *nút gốc của heap trộn xong hiện tại*.
   - *Gán*: Gán *nút gốc của heap trộn xong hiện tại* bằng nút vừa trích.
5. Trả về *nút gốc của heap trộn xong hiện tại*.

Ví dụ: Khởi tạo

Trộn hai heap từ hình 2:



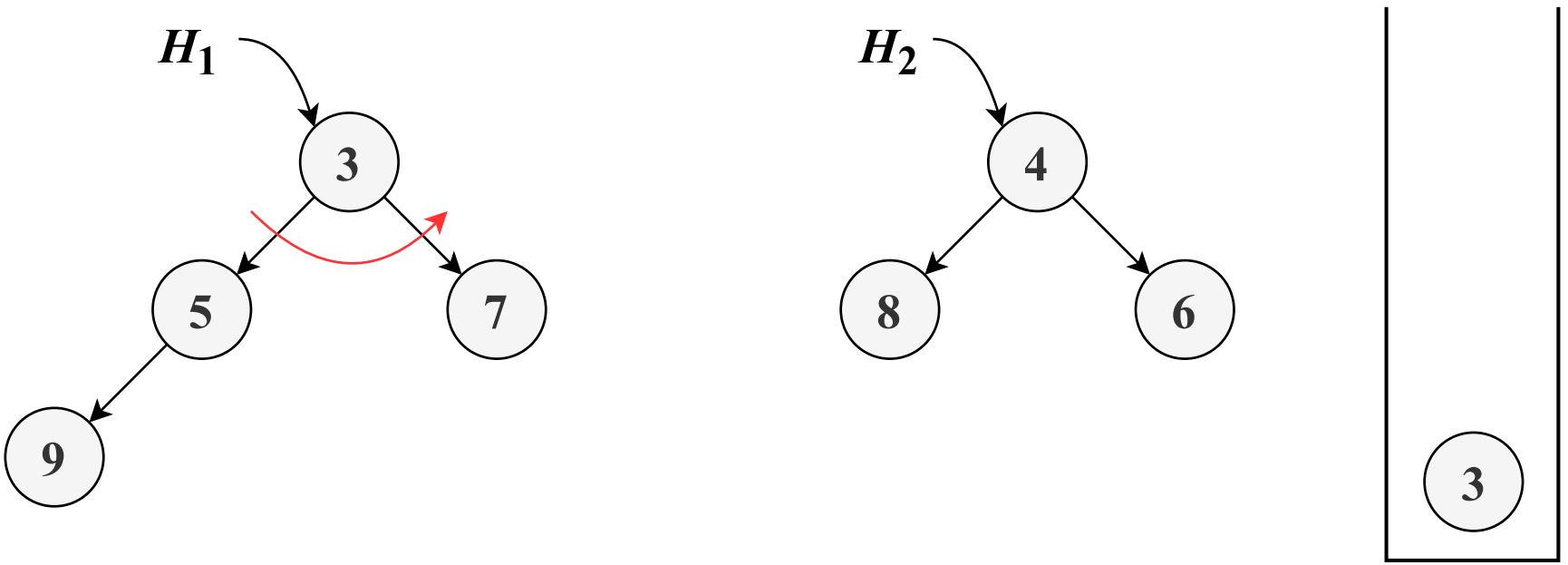
Hình: Khởi tạo ngăn xếp và gán con trỏ H_1, H_2

Vòng lặp chính 1: Bước 1.1 đến 1.3

1.1 *So sánh*: Khóa H_1 (3), H_2 (4). Vì $3 < 4$, giữ nguyên H_1, H_2.

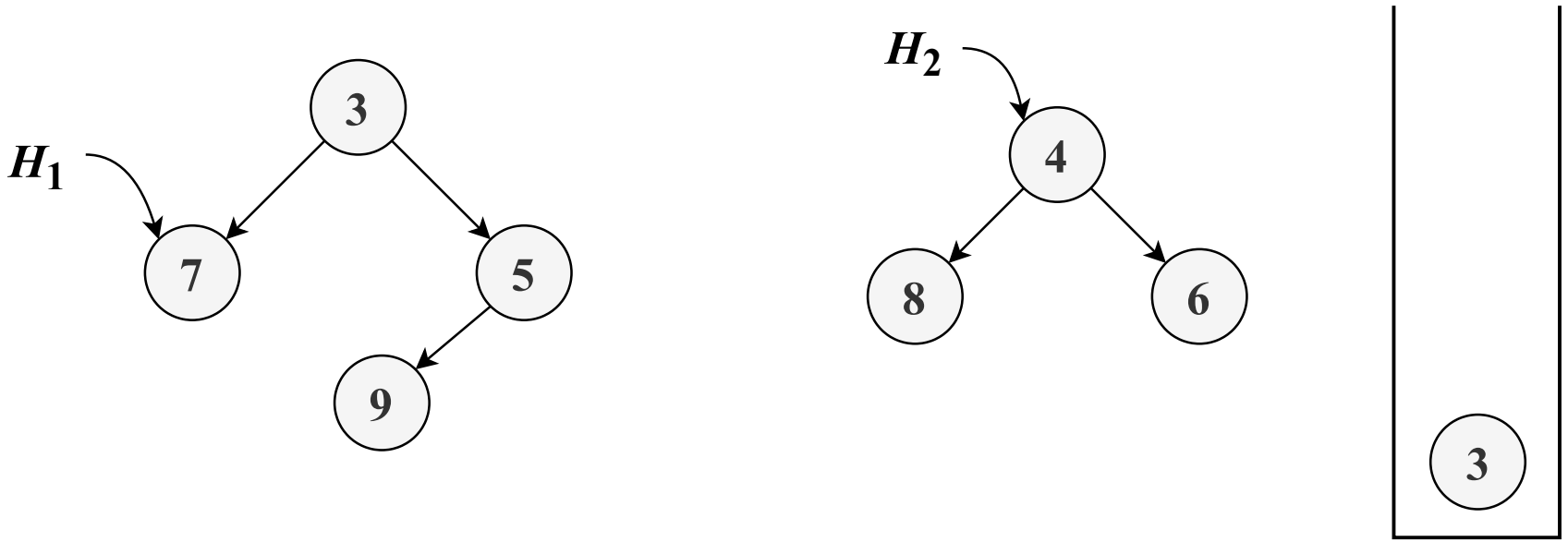
1.2 *Đẩy*: Đẩy nút 3 vào ngăn xếp.

1.3 *Hoán đổi*: Hoán đổi cây con trái (5) và phải (7) của H_1 (3).



Hình: Các bước 1.1, 1.2, 1.3

1.4 *Cập nhật*: Gán H_1 bằng cây con trái, H_1 trỏ đến 7.



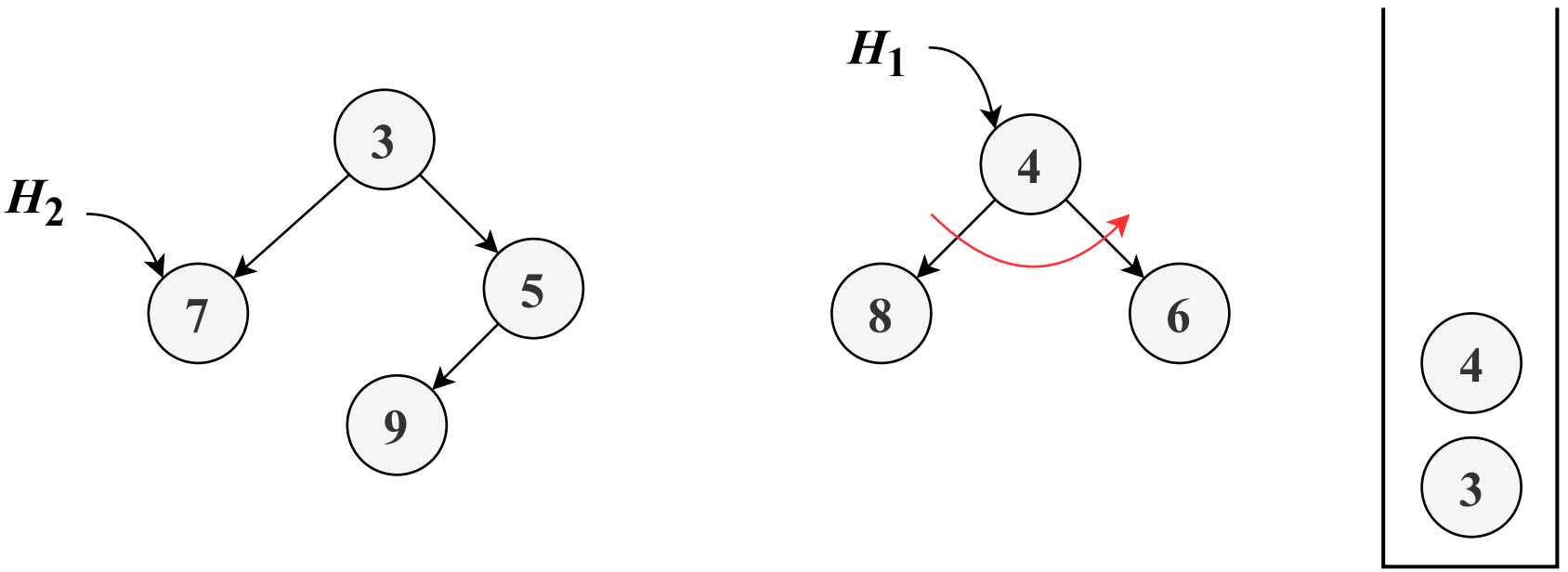
Hình: Kết quả sau bước 1.3 và 1.4

Vòng lặp chính 2: Bước 2.1 đến 2.3

2.1 *So sánh*: Khóa H_1 (7), H_2 (4). Vì $7 > 4$, hoán đổi H_1, H_2.

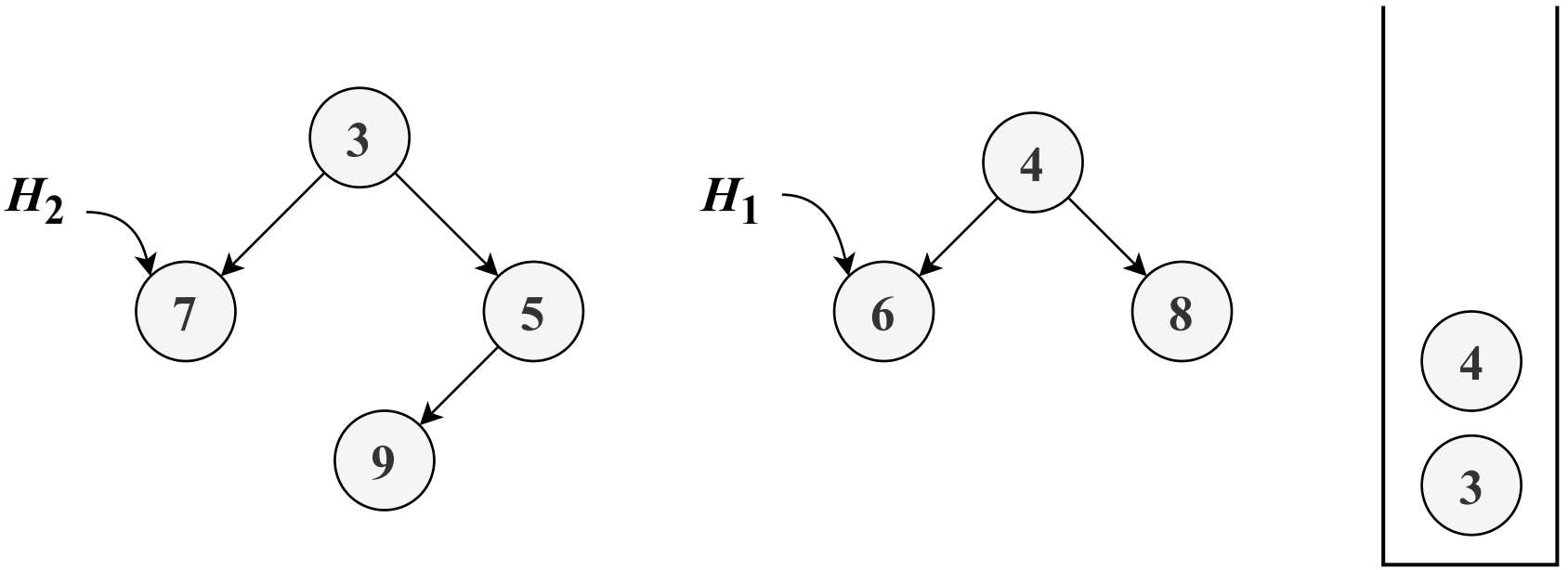
2.2 *Đẩy*: Đẩy nút 4 vào ngăn xếp.

2.3 *Hoán đổi*: Hoán đổi cây con trái (8) và phải (6) của H_1 (4).



Hình: Các bước 2.1, 2.2, 2.3

2.4 *Cập nhật*: Gán H_1 bằng cây con trái, H_1 trỏ đến 6.



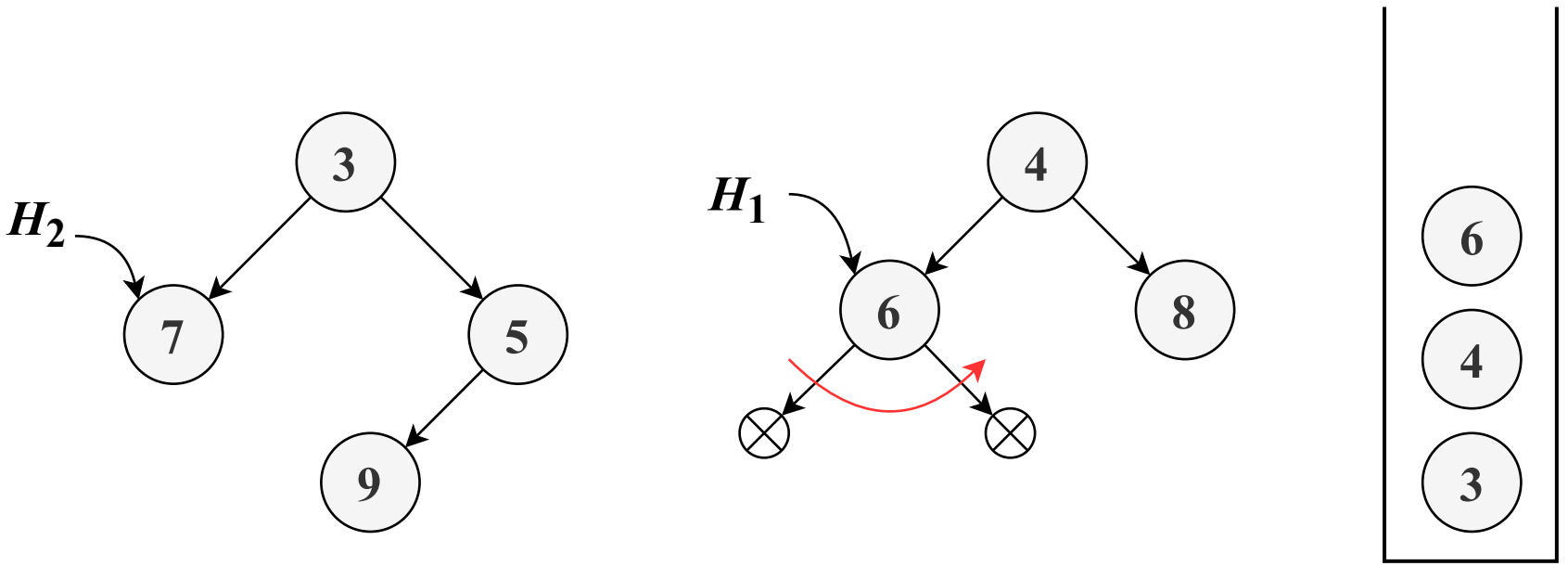
Hình: Kết quả sau bước 2.3 và 2.4

Vòng lặp chính 3: Bước 3.1 đến 3.3

3.1 *So sánh*: Khóa H_1 (6), H_2 (7). Vì $6 < 7$, giữ nguyên H_1, H_2.

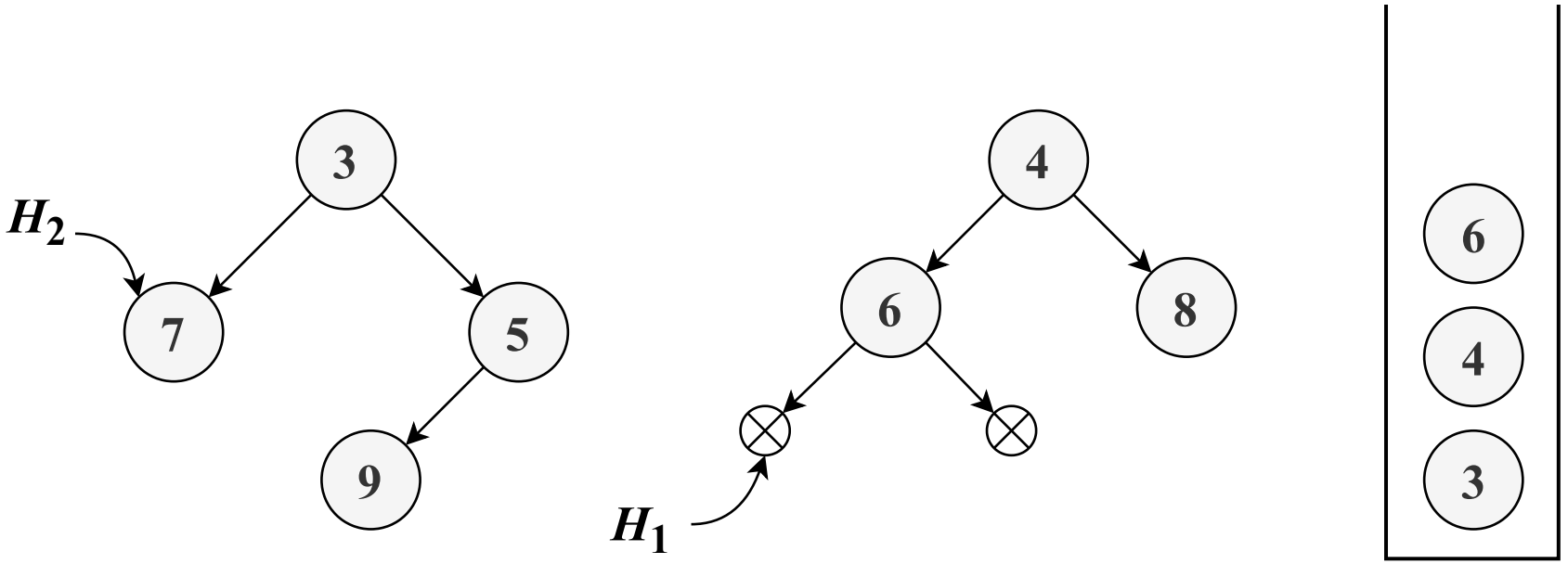
3.2 *Đẩy*: Đẩy nút 6 vào ngăn xếp.

3.3 *Hoán đổi*: Hoán đổi cây con trái (`NULL`) và phải (`NULL`) của H_1 (6).



Hình: Các bước 3.1, 3.2, 3.3

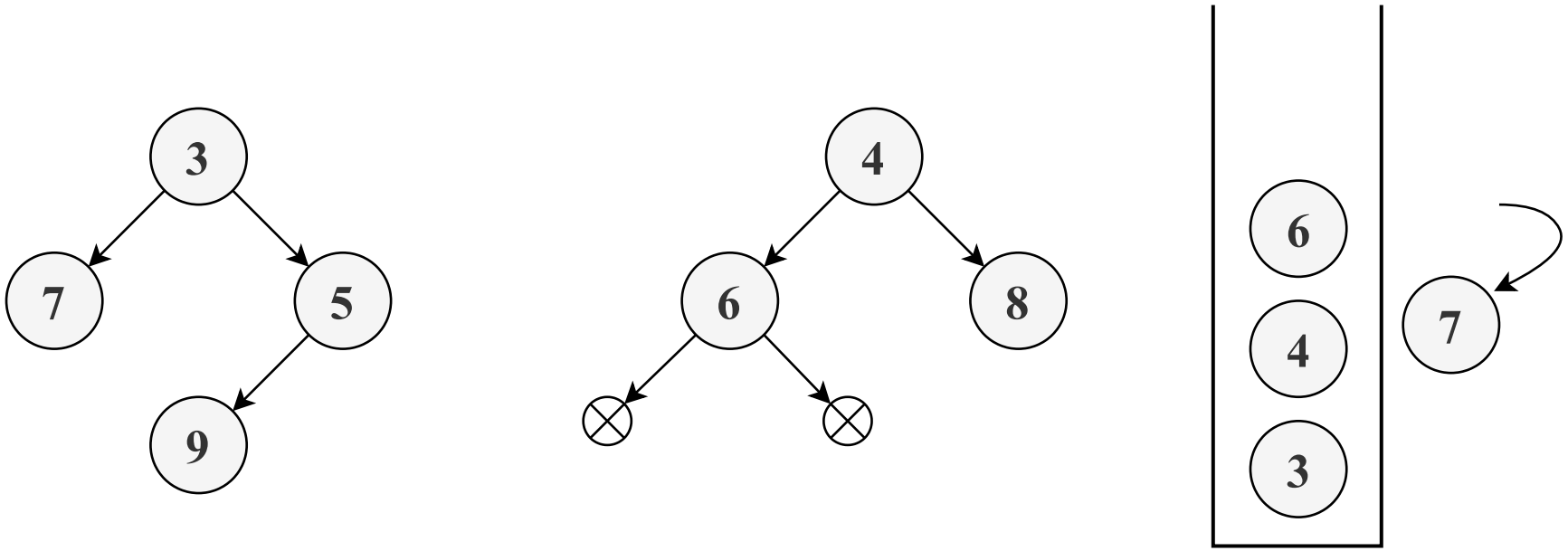
3.4 *Cập nhật*: Gán H_1 bằng cây con trái, H_1 trỏ đến `NULL`.



Hình: Kết quả sau bước 3.3 và 3.4

Chuẩn bị vòng lặp liên kết

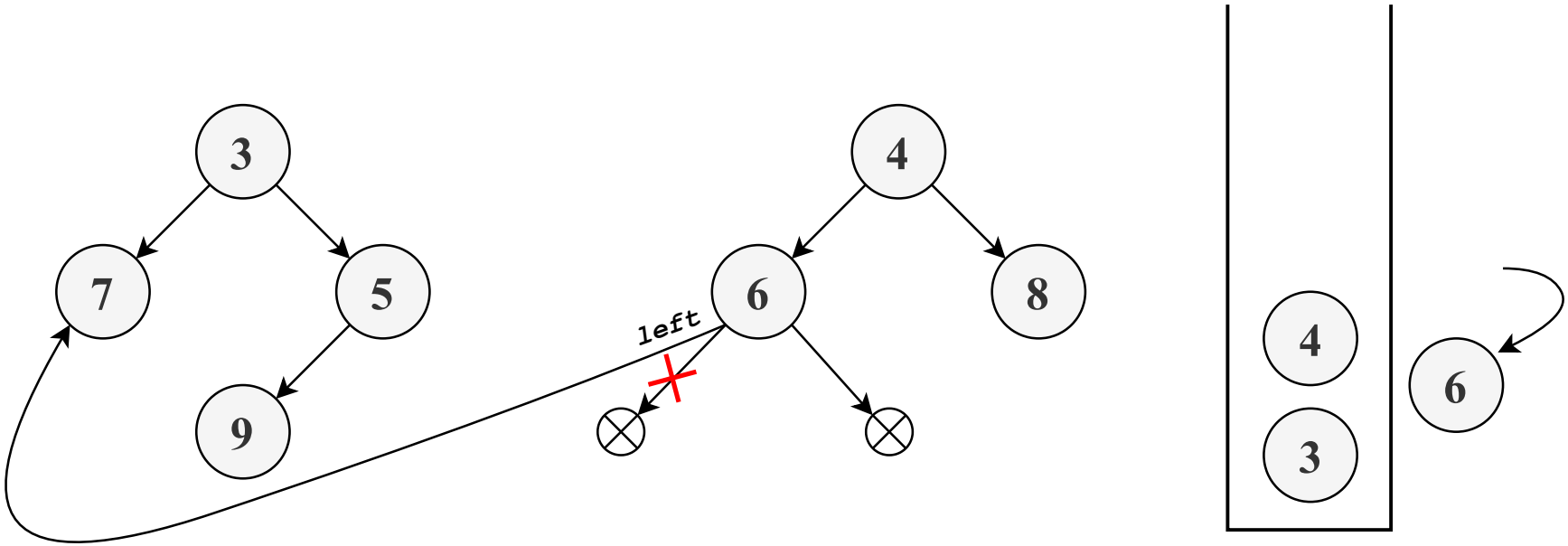
- Kết thúc vòng lặp chính vì đúng một trong hai nút H_1, H_2 trỏ đến `NULL` (lúc này là H_1).
- Gán H_2 (7) (nút khác `NULL`) là *nút gốc của heap đã trộn xong hiện tại*



Hình: Chuẩn bị cho vòng lặp liên kết

Vòng lặp liên kết 1

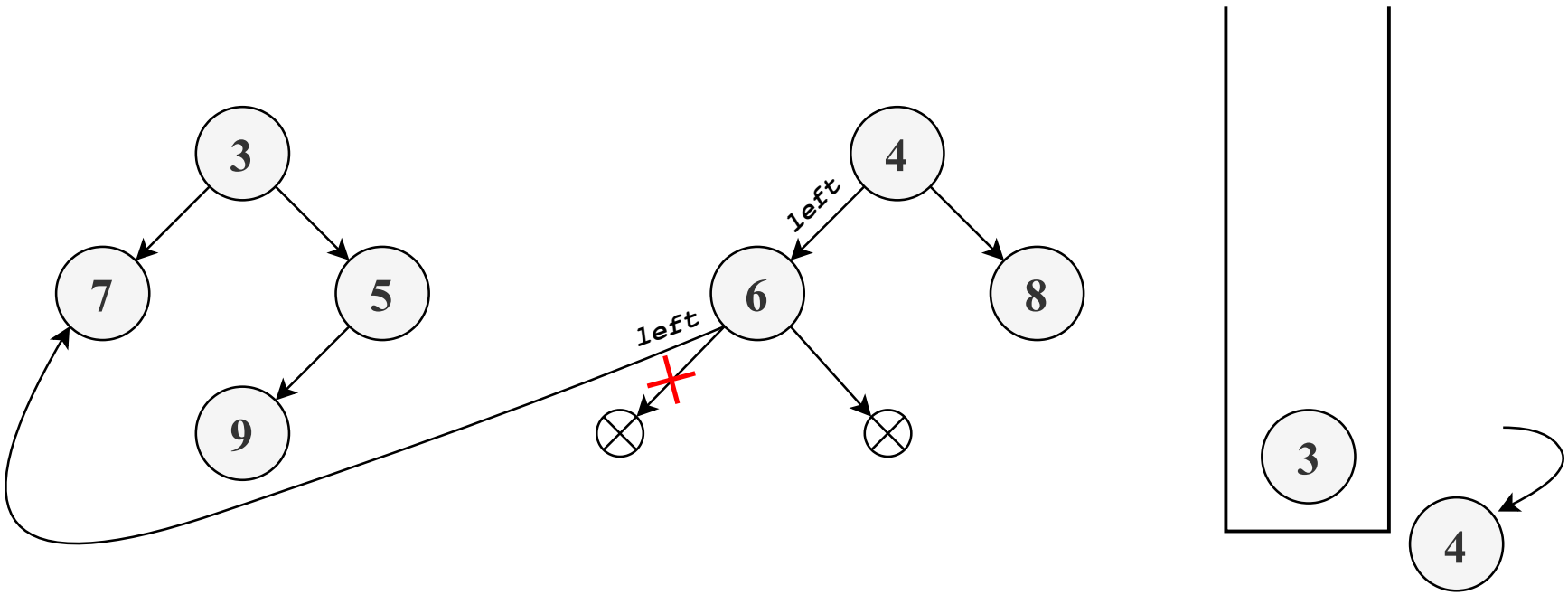
- Trích nút 6 ra khỏi ngăn xếp, cây con trái của 6 trỏ đến 7.
- Gán *nút gốc của heap trộn xong hiện tại* thành 6.



Hình: Kết quả của vòng lặp liên kết 1

Vòng lặp liên kết 2

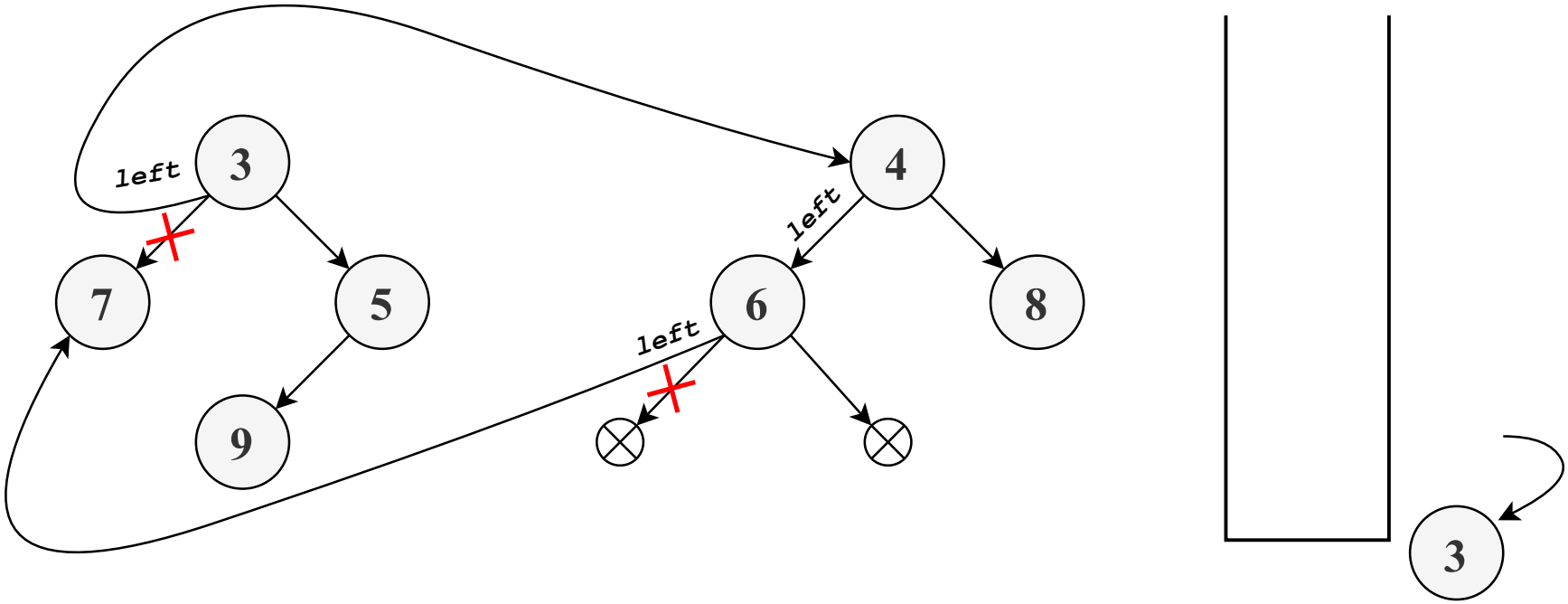
- Trích nút 4, cây con trái của 4 trỏ đến 6 (không thay đổi).
- Gán *nút gốc của heap trộn xong hiện tại* thành 4.



Hình: Kết quả của vòng lặp liên kết 2

Vòng lặp liên kết 3

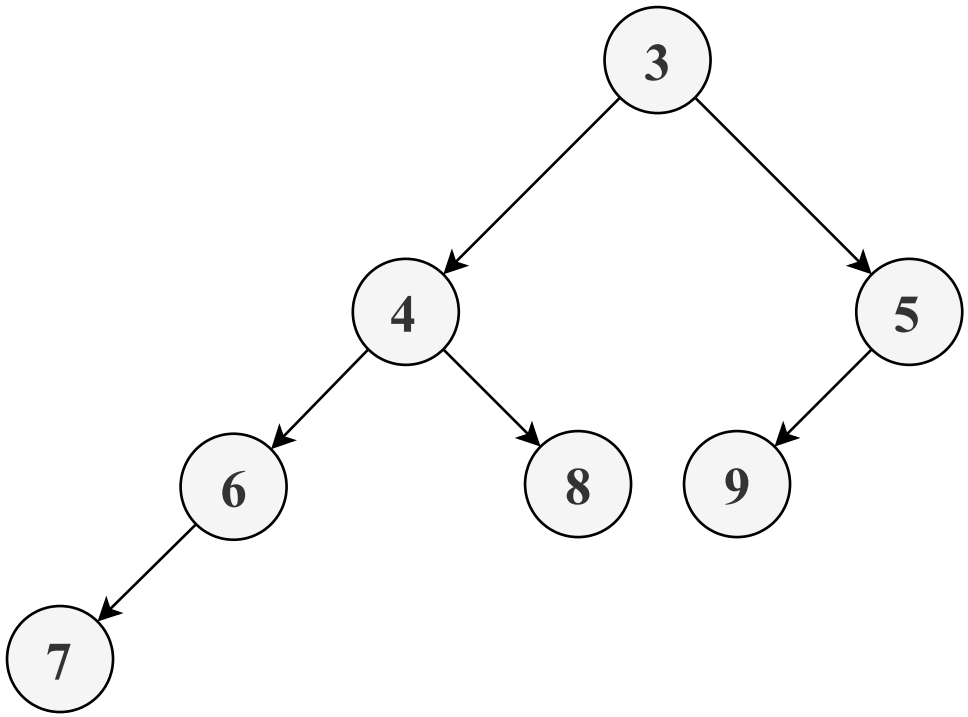
- Trích nút 3, cây con trái của 3 trỏ đến 4.
- Gán *nút gốc của heap trộn xong hiện tại* thành 3.



Hình: Kết quả của vòng lặp liên kết 3

Kết quả cuối cùng

Kết quả giống hình 21 của phép trộn đệ quy:



Hình: Kết quả của phép trộn không đệ quy

Chèn một phần tử

Thao tác *chèn một phần tử* vào Skew Heap được thực hiện bằng cách tạo một Skew Heap mới chứa chỉ phần tử đó, sau đó hợp nhất (merge) heap này với Skew Heap hiện có. Cách tiếp cận này tận dụng thao tác trộn – thao tác cốt lõi của Skew Heap – để đảm bảo tính chất min-heap.

Mô tả thao tác chèn

Các bước chèn (cho min-heap):

1. *Tạo heap mới:* Tạo một Skew Heap đơn gồm một nút chứa phần tử cần chèn (không có nút con).
2. *Trộn*: Gọi thao tác trộn giữa heap hiện có và heap đơn vừa tạo, sử dụng thuật toán trộn của Skew Heap.
3. *Trả về kết quả:* Heap sau hợp nhất là kết quả, với phần tử mới được tích hợp, duy trì tính chất heap.

Thực hành thao tác chèn

Vì tính đơn giản, ta lược qua ví dụ trực quan. Thực hành chèn một phần tử vào Skew Heap qua trang web:

- Truy cập [7] để trực quan hóa.

Trích ra phần tử nhỏ nhất

Thao tác *trích xuất phần tử nhỏ nhất* (extract-min) loại bỏ và trả về nút có khóa nhỏ nhất (gốc của min-heap), sau đó tái cấu trúc heap bằng cách trộn hai cây con của gốc để duy trì tính chất min-heap.

Mô tả thao tác trích xuất

Các bước trích xuất:

1. *Kiểm tra rỗng:* Nếu heap rỗng, trả về lỗi hoặc giá trị đặc biệt.
2. *Lấy gốc:* Lấy nút gốc (chứa khóa nhỏ nhất) và lưu giá trị để trả về.
3. *Trộn hai cây con:* Gọi thao tác hợp nhất giữa cây con trái và cây con phải của gốc để tạo heap mới.
4. *Trả về:* Trả về giá trị của gốc và heap mới sau khi trộn.

Thực hành thao tác trích xuất

Vì tính đơn giản, ta lược qua ví dụ trực quan. Thực hành trích xuất phần tử nhỏ nhất từ Skew Heap qua trang web:

- Truy cập [7] để trực quan hóa.

Cài đặt và độ phức tạp

- Phần này giới thiệu mã giả và chi phí thực tế của các thao tác: trộn, chèn, trích xuất phần tử nhỏ nhất trong Skew Heap. Khuôn khổ của phần trình bày này có thể bỏ qua phần mã giả của các của thuật toán dài, bạn đọc có thể tìm thấy trong bài báo cáo chính.

Mã giả: Trộn đệ quy

Thuật toán trộn phiên bản đệ quy, phù hợp với miêu tả và ví dụ trước:

Algorithm 1 Thuật toán trộn hai Skew Heap sử dụng đệ quy

```
1:  function MERGE(H1, H2)
2:      if H1 = null then
3:          return H2
4:      else if H2 = null then
5:          return H1
6:      else if H1.key > H2.key then
7:          H1 ⇌ H2
8:      end if
9:      H1.left ⇌ H1.right
10:     H1.left ← MERGE(H1.left, H2)
11:     return H1
12: end function
```

Mã giả: Trộn khử đệ quy

Thuật toán trộn phiên bản khử đệ quy, phù hợp với miêu tả và ví dụ trước:

Algorithm 2 Thuật toán trộn hai Skew Heap sử dụng kỹ thuật khử đệ quy

```
function nonRecursionMerge(a, b)
    if a = NULL then
        a ← b
        b ← NULL
        return
    end if
    if b = NULL then
        b ← a
        a ← NULL
        return
    end if
    ...
```

Algorithm 2 Thuật toán trộn hai Skew Heap sử dụng kỹ thuật khử đệ quy

function NONRECURSIONMERGE(a, b)

...

stack $S \leftarrow$ rỗng

$p \leftarrow a$, $q \leftarrow b$

$prev, Cur, temp \leftarrow$ NULL

while $p \neq$ NULL **and** $q \neq$ NULL **do**

  if p.key $>$ q.key **then**

    $p \rightleftharpoons q$

  end if

  S.push(p)

  p.left $\rightleftharpoons$ p.right

  $p \leftarrow p$.left

end while

...

Mã giả: Trộn khử đệ quy

Algorithm 2 Thuật toán trộn hai Skew Heap sử dụng kỹ thuật khử đệ quy

```
function nonRecursionMerge(a, b)
    ...
    if p ≠ NULL then
        prev ← p
    else
        prev ← q
    end if
    while not S.empty() do
        Cur ← S.top()
        S.pop()
        Cur.left ← prev
        prev ← Cur
    end while
    return prev
end function
```

Chi phí thực tế

- Chi phí thời gian phụ thuộc vào số nút trên đường đi phải (*right path*) của hai heap đầu vào. Ký hiệu r_1, r_2 là độ dài đường đi phải của H_1, H_2 (tính theo cạnh, từ nút đến nút tận cùng bên phải).
- Tiến trình đệ quy diễn ra dọc đường đi phải của hai nút gốc.
- Để dễ tưởng tượng, bỏ qua thao tác hoán đổi. Các nút được gọi đệ quy lần lượt trên đường đi phải của hai cây, đến khi một trong hai nút là rỗng.
- Suy ra số lần thực thi thao tác cơ sở không vượt quá $r_1 + r_2$.
- Chi phí thực tế: $T_{\text{actual}} = O(r_1 + r_2)$.

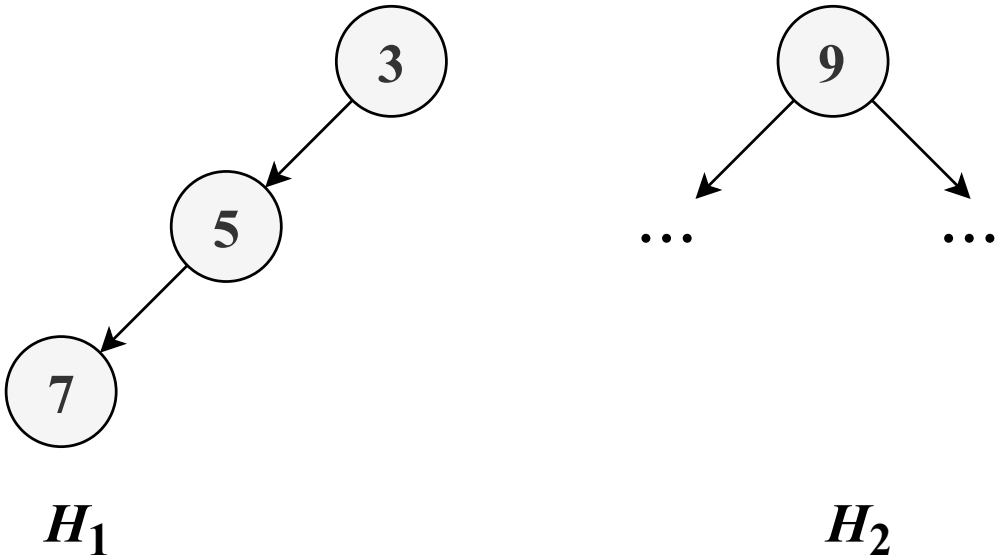
Độ phức tạp trong trường hợp tốt nhất, xấu nhất

Phân tích thời gian thực hiện trong các trường hợp tốt nhất và xấu nhất của thao tác trộn.

Trường hợp tốt nhất

Thời gian thực hiện trong trường hợp tốt nhất là $\Theta(1)$.

- Khi một heap rỗng hoặc heap có nút gốc nhỏ hơn có đường đi phải độ dài 1, phép `merge` chỉ cần vài phép gán con trỏ. Ví dụ:



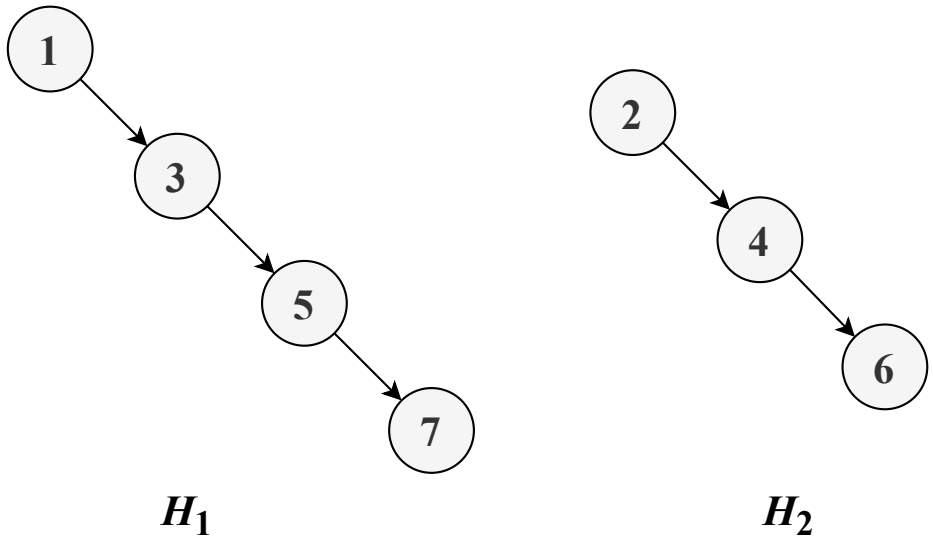
Hình: Trộn H_1, H_2. Để ý cây con của (9) không bị ảnh hưởng

- Chi phí: $\Theta(1)$, hằng số.

Trường hợp xấu nhất

Thời gian thực hiện trong trường hợp xấu nhất là $\Theta(n)$.

- Khi cả hai heap là cây lệch phải hoàn toàn, với giá trị nút tăng dần xen kẽ. Tất cả nút trên đường đi phải của cả hai cây đều được trộn. Ví dụ:



Hình: Trộn hai cây lệch phải với giá trị xen kẽ

- Chi phí $\Theta(r_1 + r_2) \equiv \Theta(n)$.

Chi Phí Amortized

- Độ phức tạp trung bình cho thao tác *merge*, *insert*, *extract-min* của Skew Heap vẫn chưa rõ ràng.
- Xác định độ dài đường dẫn bên phải kỳ vọng là vấn đề mở, khó hơn Leftist Heap.
- Phân tích trường hợp trung bình (thời gian kỳ vọng) chưa được giải quyết đầy đủ.
- Bài báo cáo này tập trung vào phân tích chi phí amortized.
- Phân tích amortized xem xét thời gian trung bình qua chuỗi thao tác.
- Hữu ích khi chi phí cao được bù đắp bởi các thao tác rẻ hơn.

Phương Pháp Thế Năng

- Sử dụng phương pháp thế năng (tiềm năng) từ [2].
- Hàm thế năng Φ ánh xạ trạng thái thành số không âm.
- Chi phí amortized:

$$a_i = c_i + (\Phi_i - \Phi_{i-1}),$$

trong đó c_i là chi phí thực tế, Φ_i và Φ_{i-1} là tiềm năng sau và trước thao tác.

Các Định Nghĩa

- Một nút p là:
  - *Nút nặng*: $\text{size}(p.\text{right}) > \text{size}(p.\text{left})$.
  - *Nút nhẹ*: $\text{size}(p.\text{right}) \leq \text{size}(p.\text{left})$.
  - *Nút bên phải*: Là con phải của nút cha.
- Hàm thế năng $\Phi(H)$ là số *nút nặng bên phải* của heap H.
- Ý nghĩa: Đo lường "khó khăn" tiềm tàng khi trộn, đặc biệt nếu nhiều nút nặng trên đường phải.

Bổ Đề

- *Trên đường đi phải của heap có n nút, số nút nhẹ không vượt quá* $\log_2 n$ [2].
- Chứng minh: Với k nút nhẹ, mỗi nút nhẹ p có $\text{size}(\text{right}(p)) \leq \text{size}(\text{left}(p))$, nên $\text{size}(p) \geq 2 \cdot \text{size}(\text{right}(p))$.
- Gọi n_i là kích thước cây con tại nút nhẹ thứ i: $n_0 \geq 2n_1 \geq 4n_2 \geq \cdots \geq 2^k n_k$.
- Suy ra: $n \geq 2^k$, nên $k \leq \log_2 n$.

Đánh Giá Chi Phí Amortized

- Tổng chi phí amortized:

$$T_{\text{amortized}} = \sum a_i = \sum [c_i + (\Phi_i - \Phi_{i-1})] = T_{\text{actual}} + \Delta\Phi,$$

  với T_{actual} là chi phí thực tế, $\Delta\Phi = \Phi_{\text{end}} - \Phi_0$.
- Thao tác trộn hai heap H_1 và H_2:
  - l_1, l_2: Số nút nhẹ trên đường trộn của H_1, H_2.
  - h_1, h_2: Số nút nặng trên đường trộn của H_1, H_2.
- Thời gian amortized = số nút trên đường trộn + thay đổi thế năng.

Số Nút Thực Tế trên Đường Trộn

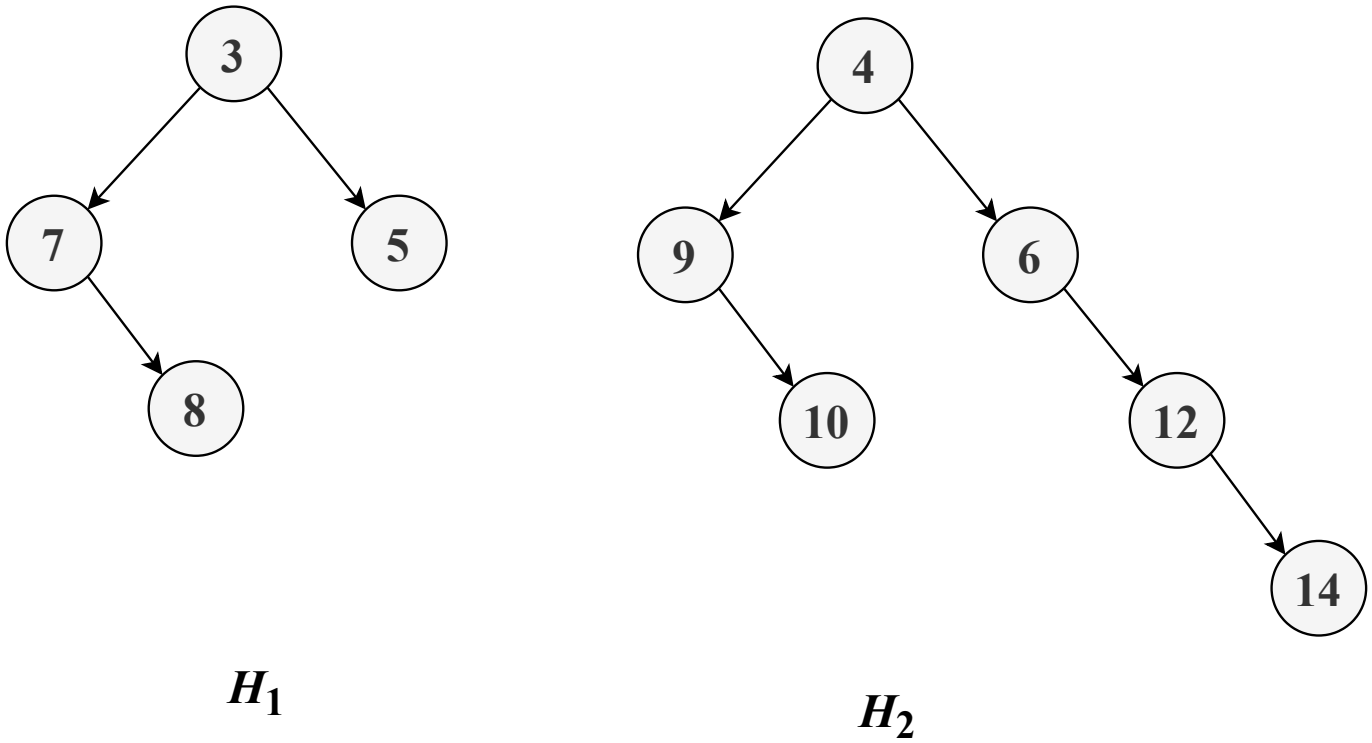
- $T_{\text{actual}} = (l_1 + h_1) + (l_2 + h_2)$.
- Theo bổ đề: $l_1 \leq \log_2 n_1$, $l_2 \leq \log_2 n_2$.
- Vậy: $T_{\text{actual}} \leq (\log_2 n_1 + h_1) + (\log_2 n_2 + h_2) \leq 2 \log_2 n + h_1 + h_2$.

Phân Tích Sự Thay Đổi Thế Năng

- n_1, n_2: Tổng số nút của H_1, H_2.
- k: Số nút trở thành con phải, làm cha thành nút nặng, $\Delta\Phi = k - h_1 - h_2$. (?)

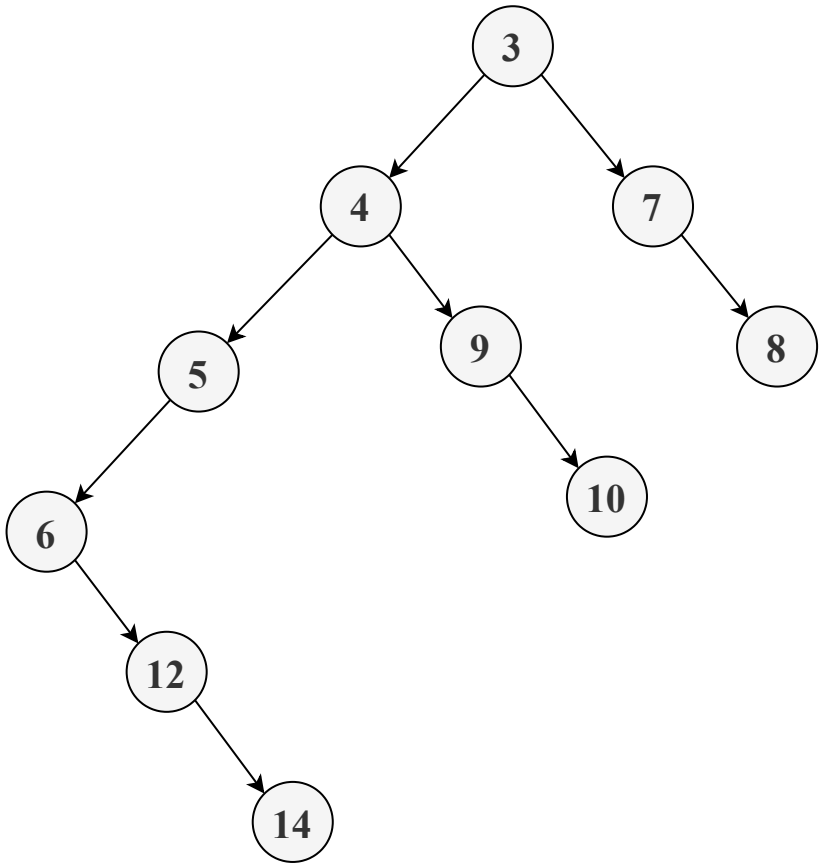
Phân Tích Sự Thay Đổi Thế Năng

Xét ví dụ sau:



Hình: 2 cây trước khi trộn, các nút nặng bên phải trên đường trộn gồm 4, 6.

Phân Tích Sự Thay Đổi Thế Năng



Hình: Cây sau khi trộn, nút nặng 7, 9 trở thành nút bên phải.

Kết Luận Chi phí Amortized

- $k \leq \log_2 n$ (theo bổ đề).
- $T_{\text{actual}} \leq 2\log_2 n + h_1 + h_2$ (chứng minh trên).
- $T_{\text{amortized}} = T_{\text{actual}} + \Delta\Phi \leq (2\log_2 n + h_1 + h_2) + (k - h_1 - h_2) = 2\log_2 n + k \leq 3\log_2 n$.
- Chi phí amortized là $3\log_2 n \in O(\log n)$.

Độ Phức Tạp của Phép Chèn và Trích Phần Tử Nhỏ Nhất

- Phân tích độ phức tạp của `insert` và `extractMin` trong Skew Heap.
- Độ phức tạp phụ thuộc vào thao tác `merge`.

Thao tác chèn (`insert`)

- Phép chèn: Tạo heap mới với phần tử, rồi trộn với heap ban đầu bằng `merge`.

Algorithm 5 `insert`(H, key)

```
1: function INSERT(H, key)
2:     H_new ← một nút mới với khóa key
3:     return MERGE(H, H_new)
4: end function
```

- Tạo heap mới: $O(1)$.
- Độ phức tạp: $O(1)$: trường hợp tốt nhất, $O(\log n)$: amortized, $O(n)$: trường hợp xấu nhất

Thao tác trích phần tử nhỏ nhất (extractMin)

- Phép trích: Lấy nút gốc, trộn hai cây con trái và phải của nút gốc.

Algorithm 6 extractMin(H, k)

```
1: function EXTRACTMIN((H, k))
2:     k = H.key
3:     return MERGE(H.left, H.right)
4: end function
```

- Độ phức tạp: $O(1)$: trường hợp tốt nhất, $O(\log n)$: amortized, $O(n)$: trường hợp xấu nhất

Xây Dựng Skew Heap từ Dãy Ban Đầu

- Trình bày hai cách xây dựng Skew Heap từ dãy số ban đầu.
- Phân tích độ phức tạp chi tiết.
- Hai phương pháp:
  - *Hướng 1:* Chèn tuần tự bằng `insert`.
  - *Hướng 2:* Ghép đôi từng cặp theo phương pháp chia để trị.

Cách 1: Chèn Tuần Tự bằng `insert`

Algorithm 7 `buildSkewHeapInsert(items)`

```
1: function BUILDSKEWHEAPINSERT(items)
2:     H ← null
3:     for all x trong items do
4:         node ← nút mới chứa x
5:         H ← merge(H, node)
6:     end for
7:     return H
8: end function
```

- *Phân tích chi phí amortized:*
- Với n phần tử, chi phí amortized của trộn phần tử thứ i là $O(\log i)$.
- Tổng chi phí:

$$T_{\text{amortized}}(n) = \sum_{i=1}^{n} O(\log i) = O\left(\sum_{i=1}^{n} \log i\right) \leq n \log n \in O(n \log n)$$

Cách 2: Ghép Đôi Từng Cặp

Algorithm 8 `buildSkewHeapPairwise(items)`

```
1:  function BUILDSKEWHEAPPAIRWISE(items)
2:      Q ← hàng đợi rỗng
3:      for i ← 0 đến items.size() − 1 bước nhảy 2 do
4:          p ← nút mới chứa items[i]
5:          if i + 1 < items.size() then
6:              q ← nút mới chứa items[i + 1]
7:              p ← merge(p, q)
8:          end if
9:          Q.enqueue(p)
10:     end for
11:     ...
```

Cách 2: Ghép Đôi Từng Cặp

Algorithm 9 `buildSkewHeapPairwise(items)`

```
1: function BUILDSKEWHEAPPAIRWISE(items)
2:     ...
3:     while Q.size() > 1 do
4:         sz ← Q.size()
5:         for j ← 1 đến ⌊sz/2⌋ do
6:             p ← Q.dequeue()
7:             q ← Q.dequeue()
8:             Q.enqueue(merge(p, q))
9:         end for
10:        if sz lẻ then
11:            Q.enqueue(Q.dequeue())
12:        end if
13:    end while
14:    return Q.dequeue()
15: end function
```

Phân Tích Chi Phí Amortized (Cách 2)

- Thuật toán ghép đôi giống merge sort, có $\lfloor n/2 \rfloor$ tầng trộn.
- Mỗi tầng k có $\lfloor n/2^{k+1} \rfloor$ cặp heap, chi phí amortized trộn là $O(k)$.
- Chi phí mỗi tầng: $O\left(\frac{nk}{2^{k+1}}\right)$.
- Tổng chi phí:

$$T_{\text{amortized}}(n) = \sum_{k=0}^{\lceil \log_2 n \rceil - 1} O\left(\frac{nk}{2^k}\right)$$

Kết Luận Phân Tích (Cách 2)

- Tổng $S = \sum_{k=0}^{\infty} \frac{k}{2^k} = 2$ (dựa trên khai triển Taylor).
- $T_{\text{amortized}}(n) \leq n \sum_{k=0}^{\lceil \log_2 n \rceil - 1} \frac{k}{2^k} \leq 2n$.
- Vậy $T_{\text{amortized}}(n) \in O(n)$.

So sánh với các cấu trúc Heap khác

- Heap là cấu trúc dữ liệu dạng cây, quản lý phần tử theo thứ tự ưu tiên.
- Min-Heap: Gốc là phần tử nhỏ nhất; Max-Heap: Gốc là phần tử lớn nhất.
- Ứng dụng: Thuật toán Dijkstra, HeapSort, hàng đợi ưu tiên.
- Sự đa dạng: Tối ưu hóa chèn, xóa, hoặc trộn tùy kịch bản.
- So sánh: **Binary Heap**, **Leftist Heap**, **Skew Heap** (tập trung vào Skew Heap).

Mô Tả Từng Loại Heap

- **Binary Heap**: Cây nhị phân hoàn chỉnh, lưu trữ bằng mảng.
- **Leftist Heap**: Cây không hoàn chỉnh, tối ưu trộn với NPL.
- **Skew Heap**: Cây tự điều chỉnh, đơn giản, linh hoạt.

Binary Heap

- Cấu trúc: Cây nhị phân hoàn chỉnh, lưu bằng mảng.
- *Chèn (Insert)*: Sift up, $O(\log n)$.
- *Trích xuất Min*: Sift down, $O(\log n)$.
- *trộn (Merge)*: Xây dựng lại, $O(n)$.
- Ưu điểm: Truy cập nhanh, quản lý bộ nhớ hiệu quả.

Leftist Heap

- Cấu trúc: Cây không hoàn chỉnh, dùng NPL để cân bằng.
- *Chèn (Insert)*: Tạo heap đơn và merge, $O(\log n)$.
- *Trích xuất Min*: trộn cây con, $O(\log n)$.
- *Trộn (Merge)*: So sánh gốc, cập nhật NPL, $O(\log n)$.
- Ưu điểm: Hiệu suất ổn định, phù hợp cho merge thường xuyên.

Skew Heap

- Cấu trúc: Cây tự điều chỉnh, không cần thuộc tính phụ.
- *Chèn (Insert)*: Tạo heap và merge, amortized $O(\log n)$, xấu nhất $O(n)$.
- *Trích xuất Min*: Trộn cây con, amortized $O(\log n)$, xấu nhất $O(n)$.
- *Trộn (Merge)*: Hoán đổi cây con, amortized $O(\log n)$, xấu nhất $O(n)$.
- Ưu điểm: Đơn giản, dễ cài đặt.

So Sánh Thao Tác

Bảng: So sánh độ phức tạp thời gian

Thao tác	**Binary Heap**	**Leftist Heap**	**Skew Heap**
Chèn (Insert)	$O(\log n)$	$O(\log n)$	$O(\log n)^*$
Xóa Min (Delete)	$O(\log n)$	$O(\log n)$	$O(\log n)^*$
trộn (Merge)	$O(n)$	$O(\log n)$	$O(\log n)^*$
Tìm Min (Find Min)	$O(1)$	$O(1)$	$O(1)$

*Trường hợp xấu nhất của Skew Heap là $O(n)$, nhưng amortized là $O(\log n)$.

Các Đặc Điểm Khác Biệt

- *Binary Heap*: Tối ưu cho chèn/xóa, phù hợp Dijkstra, HeapSort.
- *Leftist Heap*: Tốt cho merge thường xuyên (quản lý tiến trình, Huffman).
- *Skew Heap*: Đơn giản, lý tưởng cho thử nghiệm, nhưng không ổn định.

Ứng Dụng Thực Tế

- *Binary Heap*: Phù hợp với ứng dụng thời gian thực, thi đấu lập trình.
- *Leftist Heap*: Ổn định, dùng trong lập lịch ưu tiên.
- *Skew Heap*: Thử nghiệm học thuật, chấp nhận mất cân bằng.
- Lựa chọn tùy nhu cầu: Binary cho đơn lẻ, Leftist cho merge ổn định, Skew cho đơn giản.

Tài liệu tham khảo I

- Skew heap. Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Skew_heap
- D. D. Sleator and R. E. Tarjan, "Self-adjusting heaps," *SIAM Journal on Computing*, vol. 15, no. 1, 1986. [Online]. Available: https://www.cs.cmu.edu/~sleator/papers/adjusting-heaps.pdf
- University of Maryland, "CMSC 341: Data Structures - Project on Skew Heap," https://www.cs.umd.edu/class/fall2023/cmsc341, 2023, accessed: May 20, 2025.
- Elixir Community, "Prioqueue: Priority Queue Implementation in Elixir," https://hexdocs.pm/prioqueue, 2022, accessed: May 20, 2025.
- Leftist tree. Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Leftist_tree
- (2023) Skew heap. GeeksforGeeks. [Online]. Available: https://www.geeksforgeeks.org/skew-heap/

Tài liệu tham khảo II


D. Galles. (2023) Skew heap visualization. University of San Francisco. [Online]. Available: https://www.cs.usfca.edu/~galles/visualization/SkewHeap.html