Binary Heap

Binary Heap

Biên soạn: Đỗ Việt Anh (lion_it)

Email: dovietanh.95@gmail.com

Nguồn: wcipeg.com/wiki

0. Kiến thức cần biết trước

Để đọc và hiểu được bài viết các bạn cần có kiến thức về cấu trúc cây (tree) và cây nhị phân đầy đủ (complete binary tree)

1. Giới thiệu chung

1.0. Tính chất

Một cấu trúc Binary Heap thỏa mãn 2 điều kiện sau:

- **Tính chất 1 - Binary** (**TC1**): Là một cây nhị phân đầy đủ (complete binary tree)
- **Tính chất 2 - Heap** (**TC2**) Mỗi nút (node) trên cây đều chứa một nhãn lớn hơn hoặc bằng các con của nó (nếu có) và nhỏ hơn hoặc bằng nút cha (trừ nút gốc là và nó là nút lớn nhất).

Một cấu trúc như trên được gọi là max binary heap vì nhãn ở gốc (root), tương tự ta có thể thay đổi TC 2 để có được min binary heap với nhãn ở gốc là nhỏ nhất trong cây.

1.1. Ứng dụng

Binary Heap được dùng để cài đặt priority queue (trong C++, java...) hay dùng để tăng tốc các thuật toán như Dijkstra , Prim ..

1.2. Cài đặt

- Trong C++, bạn có thể sử dụng CTDL `priority_queue` hoặc `set` , vì vậy việc tự cài đặt lại là không cần thiết.
- Với Pascal, bạn có thể tìm hiểu cách cài đặt trong quyển sách của thầy Lê Minh Hoàng

2. Các thao tác trên Binary Heap

2.0. Các thao tác thường dùng với Binary Heap là

- Tìm nhãn lớn nhất trên Binary Heap (nút gốc)
- Thêm một nút vào Binary Heap
- Xóa bỏ nút gốc (root) hay nhãn lớn nhất trên Binary Heap
- Xây dựng một Binary Heap từ một tập N phần tử

*(**Các bạn có thể vào visualgo để có thể hình dung cụ thể về các thao tác trên Heap**)*

Đặt h là độ cao của cây. Nút gốc ở độ sâu 0, 2 nút con của gốc ở độ sâu 1, và nút sâu nhất có độ sâu là h. Ở độ sâu k, cây có tối đa 2^k nút, do đó tổng số nút trên cây $N \leq 2^{h+1}$.

2.1. Tìm phần tử lớn nhất

- Rõ ràng gốc (root) luôn chứa nhãn lớn nhất theo **TC2** (các nút con luôn không nhỏ hơn nút cha)
- **Độ phức tạp thuật toán:** $O(1)$.
  - Thông thường thao tác này sẽ đi kèm với thao tác xóa nút gốc để tìm nhãn tiếp theo (như khi sort). Thao tác xóa sẽ được miêu tả ở mục **2.3**

2.2. Thêm một nút (node) trên cây

- Chọn vị trí để thêm nút:
  - Nếu Binary Heap là rỗng ta chỉ cần cho nút cần thêm làm gốc.
  - Nếu Binary Heap không rỗng thêm nó vào vị trí phải nhất ở của lớp đáy, nếu lớp đáy đã đủ (số phần tử đúng bằng 2^h) thêm nút này vào lớp mới.
- Vun đống từ dưới lên (bottom-up heapify):
  - Việc thêm như trên sẽ đảm bảo được tính chất **TC1** cây nhị phân đầy đủ (complete binary tree) nhưng **TC2** có thể không được thỏa mãn.
  - Nếu nút mới này nhỏ hơn nút cha của nó ta không cần phải làm gì thêm nữa.
  - Ngược lại nếu phần tử mới lớn hơn nút cha ta đổi chỗ 2 nút cho nhau, so sánh với nút cha mới đổi chỗ nếu nó lớn hơn nút cha cứ thế đến khi nó nhỏ hoặc bằng nút cha hoặc nó là gốc (không còn nút cha nào nữa).
- **Độ phức tạp:**
  - Quá trình chọn vị trí cho nút chỉ mất $O(1)$
  - Quá trình vun đống từ dưới lên (bottom-up heapify): ta có thể thấy số lần so sánh và đổi chỗ 2 phần tử không quá độ sâu h của cây nhị phân hay ($\leq log(N+1)$ với N là số nút hiện có trên cây).
  - Độ phức tạp của cả quá trình này là $O(logN)$

2.3. Xóa nút gốc khỏi cây

- Ta chỉ có thể xóa phần tử lớn nhất hay góc của Binary Heap ra khỏi cây.

- Nếu cây chỉ có nút gốc ta xóa nó khỏi cây, nếu không
- Gán nhãn nút góc bằng nhãn của nút phải nhất ở lớp sâu nhất
- Xóa nút phải nhất ở lớp sâu nhất đi (nút này vừa được chuyển nhãn lên gốc)
- Khi này tính chất **TC1** vẫn được đảm bảo nhưng **TC2** thì có thể không, do đó ta cần vun đống từ trên xuống (top-down heapify):
  - So sánh nhãn nút gốc với nút lớn hơn trong 2 nút con của nó (nếu chỉ có 1 nút con, thì nút con đó được coi là nút lớn hơn)
  - Nếu nó nhỏ hơn nút con lớn hơn của nó: đổi chỗ 2 nút này, tiếp tục xét tiếp với nút con mới cho đến khi nó lớn hơn hoặc bằng nút con lớn hơn của nó hoặc nó không có con hay nút lá (leaf)

- **Độ phức tạp:**
  - Nếu cây chỉ có duy nhất nút gốc độ phức tạp là $O(1)$.
  - Nếu cây có N nút:
    - Xóa nút phải nhất ở lớp sâu nhất khỏi cây có độ phức tạp $O(1)$
    - Vun đống từ trên xuống (top-down heapify) cũng như bottom-up heapify không vượt quá độ sâu h của cây nên có độ phức tạp là $O(logN)$

2.4. Tăng, giảm nhãn của một nút

- Trước tiên cần xác định vị trí của nút ta cần thay đổi nhãn
- Thay đổi nhãn
- Vun đống heap
  - Nếu nhãn tăng so với nhãn trước đó cần thực hiện bottom-up heapify như khi thêm nút
  - Nếu nhãn giảm đi so với nhãn trước đó cần thực hiện top-down heapify như khi xóa nút.
- **Độ phức tạp:** độ thức tạp của thao tác này bằng độ phức tạp của top-down heapify hoặc bottom-up heapify hay bằng $O(logN)$

2.5. Xây đựng Binary Heap từ tập N phần tử

- Một cách đơn giản ta có thể thực hiện N phép thêm nút. Nhưng có một kĩ thuật hiệu quả hơn để xây dựng binary heap được gọi là **bottom-up construction**.
- **Bottom-up construction:** Kỹ thuật này yêu cầu xây dựng một cây nhị phân đầy đủ trước và thực hiện top-down heapify các nút trên cây theo tứ tự giảm dần độ cao của cây (từ các nút lá lên các nút cha và tiếp tục cho đến gốc). Chứng minh kết quả của cách xây dựng là một Binary Heap không phải là khó.
- **Độ phức tạp:**
  - Khi thực hiện N bước thêm nút ta có thể thấy độ phức tạp là $O(log1 + log2 + \ldots + logN) = O(NlogN)$.
  - Ở cách xây dựng thứ 2 ta thấy nếu một nút ở độ xâu k nó sẽ mất không quá $h - k$ lần so sánh với nút con (nhắc lại h là độ sâu của cây và số nút của cây $N < 2^{h+1}$) và một nửa số nút trên cây là lá và sẽ không phải so sánh với nút con nào cả, ta suy ra được:
    - Ở độ sâu $k = h - 1$ có 2^{h-1} nút số phép so sánh là $1 * 2^{h-1} = 1/4 * N$
    - Ở độ sâu $k = h - 2$ có 2^{h-2} nút số phép so sánh là $2 * 2^{h-2} = 2/8 * N$

- Ở độ sâu $k = h - 3$ có 2^{h-3} nút số phép so sánh là $3 * 2^{h-3} = 3/16 * N$

...

- Ở độ sâu $k = 0$ (gốc) có 1 nút số phép so sánh là $h * 1 = h/N * N$
- Tổng hợp lại ta có: $1/4 * N + 2/8 * N + 3/16 * N + \ldots + h/N * N = N$ phép so sánh

3. Câu hỏi thêm cho bạn đọc

- Tại sao Binary Heap nên là một cây nhị phân đầy đủ **TC1** ?
- Nếu Heap không phải là một cây nhị phân mà là một cây tam phân, tứ phân, k-phân thì độ phức tạp của các thao tác sẽ thay đổi thế nào ?
- **TC1** cần thêm điều kiện tập nhãn phải là một totally ordered set (2 giá trị bất kì trong tập đều có thể so sánh được và có tính chất bắc cầu trong các phép so sánh, ví dụ như tập số thực $\mathbb{R}$)