**Sorting Lower Bounds and Binary Search Trees**

( phân loại giới hạn dưới và cây nhị phân tìm kiếm)

1. Sorting Lower Bounds

1. Counting Sort ( đếm sắp xếp)

-  là một [thuật toán](https://en.wikipedia.org/wiki/Algorithm) cho [sắp xếp](https://en.wikipedia.org/wiki/Sorting_algorithm) một bộ sưu tập của các đối tượng theo các phím có dung lượng nhỏ [số nguyên](https://en.wikipedia.org/wiki/Integer) ; đó là, nó là một thuật toán [phân loại nguyên](https://en.wikipedia.org/wiki/Integer_sorting) . Nó hoạt động bằng cách đếm số đối tượng có mỗi giá trị khóa riêng biệt và sử dụng số học trên các số đếm để xác định vị trí của mỗi giá trị khóa trong dãy đầu ra

- Đây là một thuật toán sắp xếp đơn giản có độ phức tạp O(n), thường được dùng để sắp xếp một tập các phần tử có khóa (key: giá trị mà ta dựa vào để xếp thứ tự) là những giá trị nguyên nhỏ. Thuật toán có thời gian thực hiện tuyến tính, các phần tử sắp xếp thuộc về một tập giới hạn, cố định và đã biết trước. Ý tưởng chính của thuật toán là thực hiện việc ánh xạ giá trị key của các phần tử cần sắp xếp thành chỉ mục, thống kê số lần xuất hiện của phần tử sau đó dựa vào thông tin đã có này để chuyển các phần tử từ tập đầu vào sang tập kết quả với vị trí đã sắp xếp

* counting\_sort(A, k)
* chỉ chạy trên tập số nguyên
* Runtime O(n+k)

2. BucketSort ( thuật toán xắp thùng )

* Chia thành nhiều khoảng nhỏ
* là một [thuật toán phân loại](https://en.wikipedia.org/wiki/Sorting_algorithm) hoạt động bằng cách phân phối các phần tử của một [mảng](https://en.wikipedia.org/wiki/Array_data_structure) vào một số [xô](https://en.wikipedia.org/wiki/Bucket_(computing)) . Mỗi nhóm sau đó được sắp xếp riêng lẻ, sử dụng thuật toán phân loại khác hoặc bằng cách đệ quy ứng dụng thuật toán phân loại xô
* độ phức tạp : O(nlogn)

3. Radix Sort ( sắp xếp cơ số )

* là một thuật toán [sắp xếp không so sánh](https://vi.wikipedia.org/w/index.php?title=S%E1%BA%AFp_x%E1%BA%BFp_kh%C3%B4ng_so_s%C3%A1nh&action=edit&redlink=1). Thuật toán này được thực hiện dựa trên ý tưởng nếu một dãy số đã được sắp xếp hoàn chỉnh thì từng chữ số cũng sẽ được sắp xếp hoàn chỉnh dựa trên giá trị của các chữ số đó. Thuật toán này yêu cầu dãy cần được sắp xếp có thể so sánh thứ tự các vị trí vì thế sắp xếp theo cơ số không giới hạn ở tập số nguyên (ta có thể dễ dàng đưa dạng xâu về cơ số nhị phân)
* radix\_sort(A, 3, 10)

31 5 210 14 95 477 555 125

A : mảng gồm các phần tử

d : số các phần tử cùng cấp (vd : 1 chữ số , 2 chữ số …)

k : giá trị các chữ số ( vd : chạy từ 0 -> 9)

+ độ phức tạp : O(d(n+k))

1. Binary Sort (phân loại nhị phân )
2. Định nghĩa

* Cây tìm kiếm ứng với n khóa {\displaystyle k\_{1},k\_{2},...k\_{n}} là [cây nhị phân](https://vi.wikipedia.org/wiki/C%C3%A2y_(c%E1%BA%A5u_tr%C3%BAc_d%E1%BB%AF_li%E1%BB%87u)) mà mỗi nút đều được gán một khóa sao cho với mỗi mỗi nút k:
* Mọi khóa trên cây con trái đều nhỏ hơn khóa trên nút k
* Mọi khóa trên cây con phải đều lớn hơn khóa trên nút k

Cây tìm kiếm nhị phân là một cấu trúc dữ liệu cơ bản được sử dụng để xây dựng các cấu trúc dữ liệu trừu tượng hơn như các tập hợp, đa tập hợp, các dãy kết hợp.

Nếu một BST có chứa các giá trị giống nhau thì nó biểu diễn một đa tập hợp. Cây loại này sử dụng các bất đẳng thức không nghiêm ngặt. Mọi nút trong cây con trái có khóa nhỏ hơn khóa của nút cha, mọi nút trên cây con phải có nút lớn hơn hoặc bằng khóa của nút cha.

Nếu một BST không chứa các giá trị giống nhau thì nó biểu diễn một tập hợp đơn trị như trong lý thuyết tập hợp. Cây loại này sử dụng các bất đẳng thức nghiêm ngặt. Mọi nút trong cây con trái có khóa nhỏ hơn khóa của nút cha, mọi nút trên cây con phải có nút lớn hơn khóa của nút cha.

1. Các thao tác

* Tìm kiếm( Searching)

+ Việc tìm một khóa trên BST có thể thực hiện nhờ đệ quy. Chúng ta bắt đầu từ gốc. Nếu khóa cần tìm bằng khóa của gốc thì khóa đó trên cây, nếu khóa cần tìm nhỏ hơn khoa ở gốc, ta phải tìm nó trên cây con trái, nếu khóa cần tìm lớn hơn khóa ở gốc, ta phải tìm nó trên cây con phải. Nếu cây con (trái hoặc phải) là rỗng thì khóa cần tìm không có trên cây.

* Chèn (Insertion)

+ Phép chèn bắt đầu giống như phép tìm kiếm; Nếu khóa của gốc khác khóa cần chèn ta tìm nó trong cây con trái hoặc phải. Nếu cây con trái hoặc phải tương ứng là rỗng (không tìm thấy) thì thêm một nút và gán cho nút ấy khóa cần chèn.

* Xóa ((deletion)
* Xảy ra các trường hợp sau :

+ Xóa 1 lá : Vì lá không có con nên chỉ cần giải phóng nó khỏi cây

+ Xóa nút có 1 con : xóa và thay thế nó bằng con duy nhất của nó

+ Xóa 1 nút có 2 con : xóa nút đó và thay thế bằng nút có khóa lớn nhất

* Phép duyệt cây
* Phép sắp xếp

**2 . Cây đỏ đen**

* **Cây đỏ đen** ([tiếng Anh](https://vi.wikipedia.org/wiki/Ti%E1%BA%BFng_Anh): *red-black tree*) là một dạng [cây tìm kiếm nhị phân](https://vi.wikipedia.org/wiki/C%C3%A2y_t%C3%ACm_ki%E1%BA%BFm_nh%E1%BB%8B_ph%C3%A2n) tự cân bằng, một [cấu trúc dữ liệu](https://vi.wikipedia.org/wiki/C%E1%BA%A5u_tr%C3%BAc_d%E1%BB%AF_li%E1%BB%87u) được sử dụng trong [khoa học máy tính](https://vi.wikipedia.org/wiki/Khoa_h%E1%BB%8Dc_m%C3%A1y_t%C3%ADnh). Cấu trúc ban đầu của nó được đưa ra vào năm [1972](https://vi.wikipedia.org/wiki/1972) bởi [Rudolf Bayer](https://vi.wikipedia.org/w/index.php?title=Rudolf_Bayer&action=edit&redlink=1). Ông gọi chúng là các "[B-cây](https://vi.wikipedia.org/wiki/B-c%C3%A2y) cân bằng" còn tên hiện nay được đưa ra từ [1978](https://vi.wikipedia.org/wiki/1978) bởi Leo J. Guibas và [Robert Sedgewick](https://vi.wikipedia.org/w/index.php?title=Robert_Sedgewick&action=edit&redlink=1). Nó là cấu trúc phức tạp nhưng cho kết quả tốt về thời gian trong trường hợp xấu nhất. Các phép toán trên chúng như tìm kiếm (search), chèn (insert), và xóa (delete) trong thời gian [O](https://vi.wikipedia.org/w/index.php?title=K%C3%BD_hi%E1%BB%87u_O_l%E1%BB%9Bn&action=edit&redlink=1) (log *n*), trong đó *n* là số các phần tử của cây.
* **Tính chất :**

+ Một nút hoặc là đỏ hoặc đen.

+ Gốc là đen.

+ Tất cả các lá là đen.

+ Cả hai con của mọi nút đỏ là đen. (và suy ra mọi nút đỏ có nút cha là đen.)

+ Tất cả các đường đi từ một nút đã cho tới các lá chứa một số như nhau các nút đen.

1. Các phép toán

a.1. Phép chèn

Phép chèn bắt đầu bằng việc bổ sung một nút [như trong cây tìm kiếm nhị phân bình thường](https://vi.wikipedia.org/wiki/C%C3%A2y_t%C3%ACm_ki%E1%BA%BFm_nh%E1%BB%8B_ph%C3%A2n#ph.C3.A9p_ch.C3.A8n) và gán cho nó màu đỏ. Ta xem xét để bảo toàn tính chất đỏ đen từ các nút lân cận với nút mới bổ sung. Thuật ngữ *nút chú bác* sẽ dùng để chỉ nút anh (hoặc em) với nút cha của nút đó như trong cây phả hệ. Chú ý rằng:

* Tính chất 3 (Tất cả các lá -là các nút *null* là đen) giữ nguyên.
* Tính chất 4 (Cả hai con của nút đỏ là đen) nếu bị thay đổi chỉ bởi việc thêm một nút đỏ có thể sửa bằng cách gán màu đen cho một nút đỏ hoặc một phép quay.
* Tính chất 5 (Tất cả các đường đi từ gôc tới các lá có cùng một số nút đen) nếu bị thay đổi chỉ bởi việc thêm một nút đỏ có thể sửa bằng cách gán màu đen cho một nút đỏ hoặc một phép quay.

*Chú ý*: Nhãn **N** sẽ dùng để chỉ nút đang chèn vào, **P** chỉ nút cha của **N'**, **G** chỉ ông của **N'**, và **U** chỉ chú bác của **N'**. Nhớ rằng,giữa các trường hợp, vai trò và nhãn của các nút có thể thay đổi còn trong cùng một trường hợp thì không.

+ TH1 : Nút mới thêm **N** ở tại gốc. Trong trường hợp này, gán lại màu đen cho **N**, để bảo toàn tính chất 2 (Gốc là đen). Vì mới chỉ bổ sung một nút, Tính chất 5 được bảo đảm vì mọi đường đi chỉ có một nút.

+ TH2 : Nút cha **P** của nút mới thêm là đen, khi đó Tính chất 4 (Cả hai nút con của nút đỏ là đen) không bị vi phạm vì nút mới thêm có hai con là "null' là đen. Tính chất 5 cũng không vi phạm vì nút mới thêm là đỏ không ẩnh hưởng tới số nút đen trên tất cả đường đi.

+ TH3 : Cả cha **P** và bác **U** là đỏ, thì thể đổi cả hai thành đen còn **G** thành đỏ (để bảo toàn tính chất 5.Khi đó nút mới **N**có cha đen. Vì đường đi bất kỳ đi qua cha và bác của "N" phải đi qua ông của *N* nên số các nút đen trên đường đi này không thay đổi. Tuy thế nút ông **G** có thể vi phạm tính chất 2 (Gốc là đen) hoặc 4 (Cả hai con của nút đỏ là nút đen) (tính chất 4 bị vi phạm khi cha của **G** là đỏ). Để sửa chữa trường hợp này gọi một thủ tục đệ quy trên **G** từ trường hợp 1. Note that this is the only recursive call, and it occurs prior to any rotations, which proves that a constant number of rotations occur.



////// Note : *Chú ‎ý:* Trong các trường hợp tiếp theo, giả sử rằng nút cha **P** là con trái của cha của nó. Nếu nó là con phải, *left* và *right* đổi chỗ cho nhau trong cases 4 and 5.

+ TH4 : Nút cha **P** là đỏ nhưng nút chú bác **U** là đen, nút mới **N** là con phải của nút **P**, và **P** là con trái của nút **G**. Trong trường hợp này, thực hiện [quay trái](https://vi.wikipedia.org/w/index.php?title=Ph%C3%A9p_quay_(c%C3%A2y)&action=edit&redlink=1) chuyển đổi vai trò của nút mới **N** và nút cha **P** do đó định dạng lại nút **P** bằng Trường hợp 5 (đổi vai trò **N** và **P**) vì tính chất 4 bị vi phạm (Cả hai con của nút đỏ là đen). Phép quay cũng làm thay đổi một vài đường đi (các đường đi qua cây con nhãn "1") phải đi qua thêm nút mới **N**, nhưng vì **N** là đỏ nên không làm chúng vi pham tính chất 5



+ TH5 : Nút cha **P** là đỏ nhưng nút bác **U** là đen, nút mới **N** là con trái của nút **P**, và **P** là con trái của nút ông **G**. Trong trường hợp này, một phép quay phải trên nút ông **G** được thực hiện; kết quả của phep quay là trong cây mới nút **P** trở thành cha của cả hai nít **N** và nút **G**. Đã biết **G** là đen, vì bây giờ nó là con của **P**. Đổi màu của **P** và **G** thì cây thỏa mãn tính chất 4. Tính chất 5 không bị vi phạm vì các đường đi qua G trước đây bây giờ đi qua **P**.



a.2. Phép xóa

- Trong cây tìm kiếm nhị phân bình thường khi xóa một nút có cả hai con (không là *lá null*), ta tìm phần tử lớn nhất trong cây con trái hoặc phần tử nhỏ nhất trong cây con phải, chuyển giá trị của nó vào nút đang muốn xóa (xem [Cây tìm kiếm nhị phân](https://vi.wikipedia.org/wiki/C%C3%A2y_t%C3%ACm_ki%E1%BA%BFm_nh%E1%BB%8B_ph%C3%A2n#ph.C3.A9p_x.C3.B3a)). Khi đó chúng ta xóa đi nút đã được copy giá trị, nút này có ít hơn hai con (không là *lá null*). Vì việc copy giá trị không làm mất tính chất đỏ đen nên không cần phải sửa chữa gì cho thao tác này. Việc này chỉ đặt ra khi xóa các nút có nhiều nhất một con (không là *lá null*).

Chúng ta sẽ thảo luận về việc xóa một nút có nhiều nhất một con (không là lá null).

Nếu ta xóa một nút đỏ, ta có thể chắc chắn rằng con của nó là nút đen. Tất cả các đường đi đi qua nút bị xóa chỉ đơn giản bớt đi một nút đỏ do đó tính chất 5 không thay đổi. Ngoài ra, cả nút cha và nút con của nút bị xóa đều là nút đen, do đó tính chất 3 và 4 vẫn giữa nguyên.. Một trường hợp đơn giản khác là khi xóa một nút đen chỉ có một con là nút đỏ. Khi xóa nút đó các tính chất 4 và 5 bị phá vỡ, nhưng nếu gán lại màu cho nút con là đen thì chúng lại được khôi phục.

Trường hợp phức tạp xảy ra khi cả nút bị xóa và nút con của nó đều là đen. Chúng ta sẽ bắt đầu bằng việc thay nút bị xóa bằng nút con của nó. Chúng ta sẽ gọi nút con này (trong vị trí mới của nó là **N**, và anh em với nó (con khác của nút cha mới) là **S**. Tiếp theo ta vẫn dùng **P** chỉ cha mới của **N**, **SL** chỉ con trái của **S**, và **SR** chỉ con phải của **S** (chúng tồn tại vì **S** không thể là lá).

*Chú ‎ý*: Giữa các trường hợp khác nhau, vai trò và nhãn của các nút có thể thay đổi, nhưng trong một trường hợp mọi nhãn giữ vai trò không thay đổi. Trong hình vẽ các màu đỏ đen được thể hiện khi màu của nút đã rõ ràng, màu trắng biểu thị một màu chưa rõ (hoặc đỏ hoặc đen).

+ Nếu cả **N** và gốc ban đầu của nó là đen thì sau khi xóa các đường qua "N" giảm bớt một nút đen. Do đó vi phạm Tính chất 5, cây cần phải cân bằng lại. Có các trường hợp sau :

* **Trường hợp 1:** **N** là gốc mới. Trong trường hợp này chúng ta dừng lại. Ta đã giải phóng một nút đen khỏi mọi đường đi và gôc mới lại là đen. Không tính chất nào bị vi phạm.
* **Trường hợp 2:** **S** là đỏ. Trong trường hợp này tráo đổi màu của **P** và **S**, và sau đó [quay trái](https://vi.wikipedia.org/w/index.php?title=Ph%C3%A9p_quay&action=edit&redlink=1) tai **P**, nó sẽ làm cho **S** trở thành nút ông của **N**. Chú ý rằng **P** có màu đen và có một con màu đỏ. Tất cả các đường đi có số các nút đen giống nhau, bây giờ **N** có một anh em màu đen và cha màu đỏ, chúng ta có thể tiếp tục với các trường hợp 4, 5, hoặc 6. (anh em mới của nó là đen ví chỉ có một con của nút đỏ **S**.) Trong các trường hợp sau la sẽ gọi anh em mới của **N'** là **S**.



* Trường hợp 3 : **P**, **S**, và các con của **S** là đen. Trong trường hợp này, chúng ta gán lại cho **S** màu đỏ. Kết quả là mọi đường đi qua **S**, (tất nhiên chúng không qua **N**,có ít hơn một nút đen. Vì việc xóa đi cha trước đây của **N'** làm tất cả các đương đi qua **N**bớt đi một nút đen, nên chúng bằng nhau. Tuy nhiên tất cả các đường đi qua **P** bây giờ có ít hơn một nút đen so với các đường không qua **P**, do đó Tính chất 5 (Tất cả các đường đi từ gốc tới các nút lá có cùng số nút đen) sẽ bị vi phạm. Để sửa chữa nó chúng ta lại tái cân bằng tại **P**, bắt đầu từ trường hợp 1.



* **Trường hợp 4 : S** và các con của **S** là đen nhưng **P** là đỏ. Trong trường hợp này, chúng ta đổi ngược màu của **S** và **P**. Điều này không ảnh hưởng tới số nút đen trên các đường đi không qua **N**, nhưng thêm một nút đen trên các đường đi qua **N**, thay cho nút đen đã bị xóa trên các đường này.
* **Trường hợp 5:** **S** là đen, con trái của **S** là đỏ, con phải của **S** là đen, còn **N** là con trái của cha nó. Trong trường hợp này chúng ta quay phải tại **S**, khi đó con trái của **S** trở thành cha của **S** và **N** là anh em mới của nó. Sau đó ta tráo đổi màu của **S** và cha mới của nó. Tất cả các đường đi sẽ có số nút đen như nhau, nhưng bây giờ **N** có một người anh em đen mà con phải của nó lại là đỏ, chúng ta chuyển sang Trường hợp 6. Hoặc **N** hoặc cha của nó bị tác động bởi việc dịch chuyên này.

(Lưu ý trong trường hợp 6, ta đặt lại nút anh em mới của **N** là **S**.)



**Trường hợp 6:** **S** là đen, con phải của **S** là đỏ và **N** là con trái của nút cha **P**. Trong trường hợp này chúng ta quay trái tại **P**, khi đó **S** trở thành cha của **P** và con phải của **S**. Chúng ta hoán đổi màu của **P** và **S**, và gán cho con phải của **S** màu đen. Cây con giữ nguyên màu của gốc do đó Tính chất 4 (Cả hai con của nút đỏ là đen) và Tính chất 5 không bị vi phạm trong cây con này. Tuy nhiên, **N** bây giờ có thêm một nút đen tiền nhiệm: hoặc **P** mới bị tô đen, nó đã là đen và **S** là nút ông của nó trở thành đen. Như cậy các đương đi qua **N** có thêm một nút đen.

Trong lúc đó, với một đường đi không đi qua **N**, có hai khả năng:

* đi qua nút anh em của **N**. Khi đó cả trước và sau khi quay nó phải đi qua **S** và **P**, khi thay đổi màu sắc hai nút này đã tráo đổi màu cho nhau. Như vây đường đi này không bị thay đổi số nút đen.
* đi qua nút bác của **N'**, là con phải của **S**. Khi đó trước khi quay nó đi qua **S**, cha của **S**, và con phải của **S**, nhưng sau khi quay nó chỉ đi qua nút **S** và con phải của **S**, khi này **S** đã nhận màu cũ của cha **P** còn con phải của **S'**s đã đổi màu từ đỏ thành đen. Kết quả là số các nút đen trên đường đi này không thay đổi.

Như vậy, số các nút đen trên các đường đi là không thay đổi. Do đó các tính chất 4 và 5 đã được khôi phục. Nút trắng trong hình vẽ có thể là đỏ hoặc đen, nhưng phải ghi lại trước và sau khi thay đổi.

