HUST

ĐẠI HỌC BÁCH KHOA HÀ NỘI

HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

ONE LOVE. ONE FUTURE.

CẤU TRÚC DỮ LIỆU VÀ THUẬT TOÁN

ĐẠI HỌC
BÁCH KHOA HÀ NỘI
HANOI UNIVERSITY
OF SCIENCE AND TECHNOLOGY

CẤU TRÚC DỮ LIỆU VÀ THUẬT TOÁN

Cây đỏ đen (Red-Black Tree)

ONE LOVE. ONE FUTURE.

MỤC TIÊU

Sau bài học này, người học có thể:

1. Hiểu được khái niệm cây đỏ đen

2. Cài đặt được cấu trúc dữ liệu cây đỏ đen

NỘI DUNG TIẾP THEO

1. Định nghĩa

1.1 Đặt vấn đề

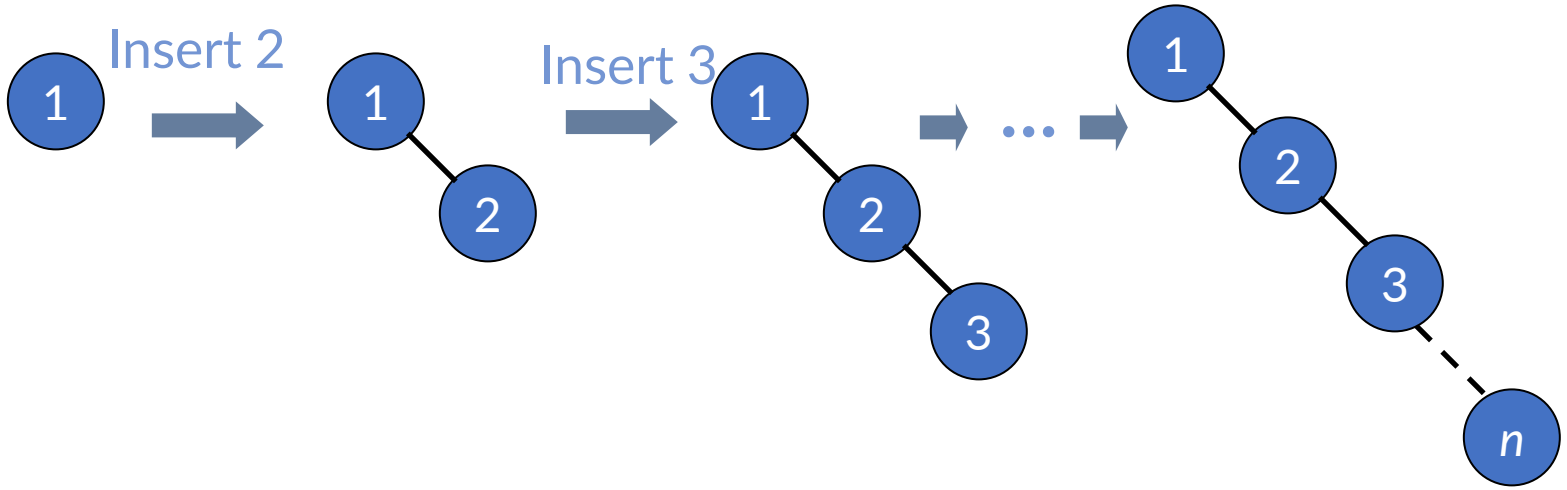
1.2 Tính chất đỏ đen

1.3 Cài đặt

2. Các phép toán

1. ĐỊNH NGHĨA

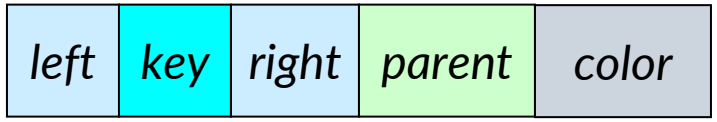
- 1.1. Đặt vấn đề
  - Trên cây nhị phân tìm kiếm với chiều cao *h*, tất cả các phép toán cơ bản đều có thời gian tính là *O*(*h*).
  - Trong tình huống **tồi nhất** $h = n$



Vấn đề đặt ra là: Có cách nào đảm bảo *h* = *O*(log *n*)?

1. ĐỊNH NGHĨA

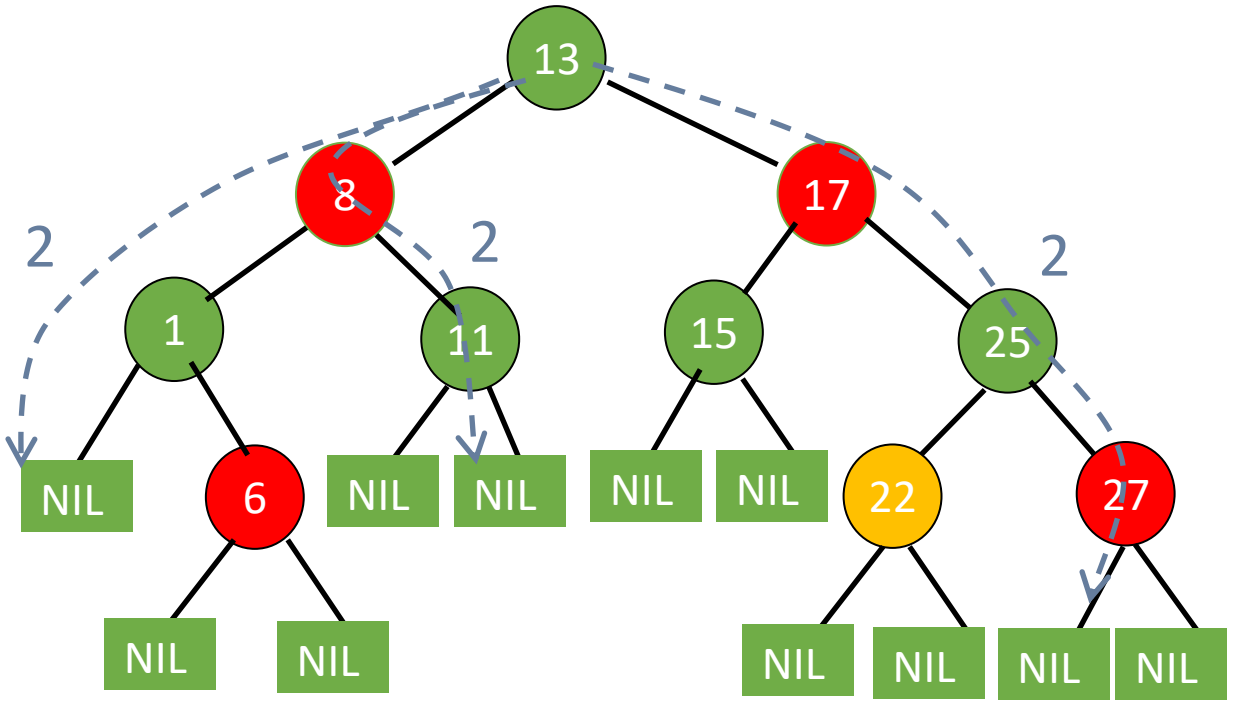
- 1.1. Cây đỏ đen
  - Cây đỏ đen là cây tìm kiếm nhị phân có thêm một trường thông tin cho mỗi nút: **màu sắc** có thể là ĐỎ hoặc ĐEN.
  - Mỗi nút của cây hiện chứa các thuộc tính



  - Nút lá (leaf):
    - Không chứa key hay dữ liệu
    - Nút con có giá trị NULL của một nút có chứa key.
  - Chiều cao của cây Đỏ-Đen luôn là O(log n) trong đó n là số nút trên cây.

1. ĐỊNH NGHĨA

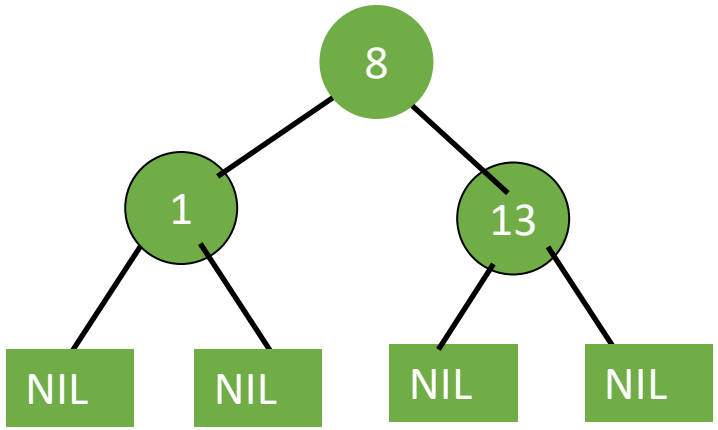
- 1.2. Tính chất đỏ đen
  - Cây đỏ đen là cây nhị phân thỏa mãn các tính chất đỏ đen sau:
    - Mọi nút đều có màu đỏ hoặc đen.
    - Nút gốc có màu đen.
    - Mỗi lá (NIL) đều có màu đen.
    - Nếu một nút có màu đỏ thì cả 2 nút con của nó đều có màu đen.
    - Đối với mỗi nút, tất cả các đường đi từ nút đến các nút lá đều chứa cùng số nút đen



1. ĐỊNH NGHĨA

- 1.2. Tính chất đỏ đen
  - Cây đỏ đen là cây nhị phân thỏa mãn các tính chất đỏ đen sau:
    - Mọi nút đều có màu đỏ hoặc đen.
    - Nút gốc có màu đen.
    - Mỗi lá (NIL) đều có màu đen.
    - Nếu một nút có màu đỏ thì cả 2 nút con của nó đều có màu đen.
    - Đối với mỗi nút, tất cả các đường đi từ nút đến các nút lá đều chứa cùng số nút đen

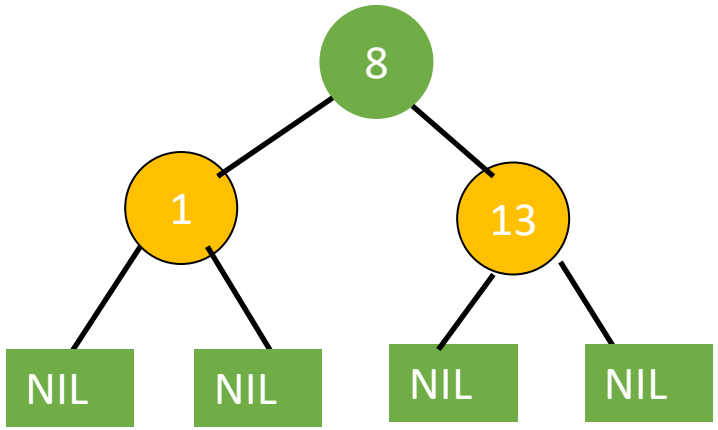
Cây đỏ đen



1. ĐỊNH NGHĨA

- 1.2. Tính chất đỏ đen
  - Cây đỏ đen là cây nhị phân thỏa mãn các tính chất đỏ đen sau:
    - Mọi nút đều có màu đỏ hoặc đen.
    - Nút gốc có màu đen.
    - Mỗi lá (NIL) đều có màu đen.
    - Nếu một nút có màu đỏ thì cả 2 nút con của nó đều có màu đen.
    - Đối với mỗi nút, tất cả các đường đi từ nút đến các nút lá đều chứa cùng số nút đen

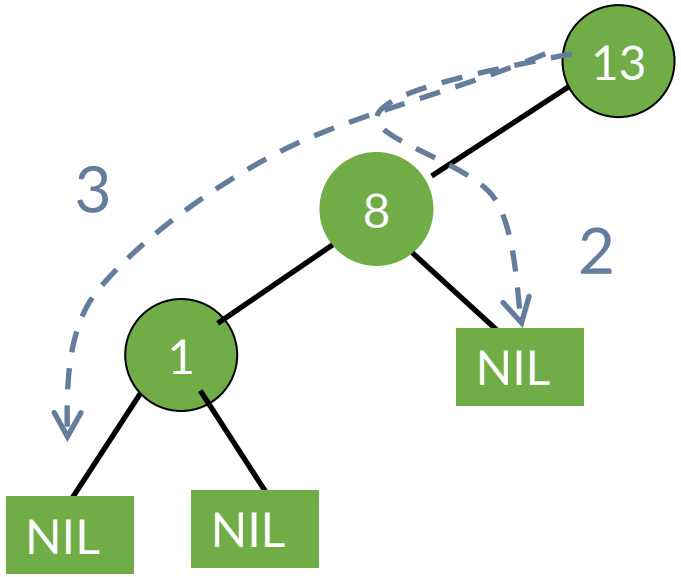
Cây đỏ đen



1. ĐỊNH NGHĨA

- 1.2. Tính chất đỏ đen
  - Cây đỏ đen là cây nhị phân thỏa mãn các tính chất đỏ đen sau:
    - Mọi nút đều có màu đỏ hoặc đen.
    - Nút gốc có màu đen.
    - Mỗi lá (NIL) đều có màu đen.
    - Nếu một nút có màu đỏ thì cả 2 nút con của nó đều có màu đen.
    - Đối với mỗi nút, tất cả các đường đi từ nút đến các nút lá đều chứa cùng số nút đen

Không phải cây đỏ đen

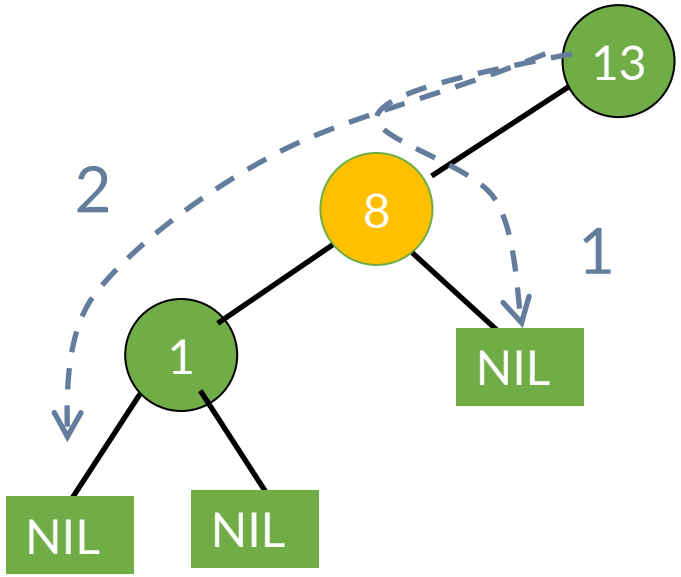


Vi phạm tính chất 5

1. ĐỊNH NGHĨA

- 1.2. Tính chất đỏ đen
  - Cây đỏ đen là cây nhị phân thỏa mãn các tính chất đỏ đen sau:
    - Mọi nút đều có màu đỏ hoặc đen.
    - Nút gốc có màu đen.
    - Mỗi lá (NIL) đều có màu đen.
    - Nếu một nút có màu đỏ thì cả 2 nút con của nó đều có màu đen.
    - Đối với mỗi nút, tất cả các đường đi từ nút đến các nút lá đều chứa cùng số nút đen

Không phải cây đỏ đen

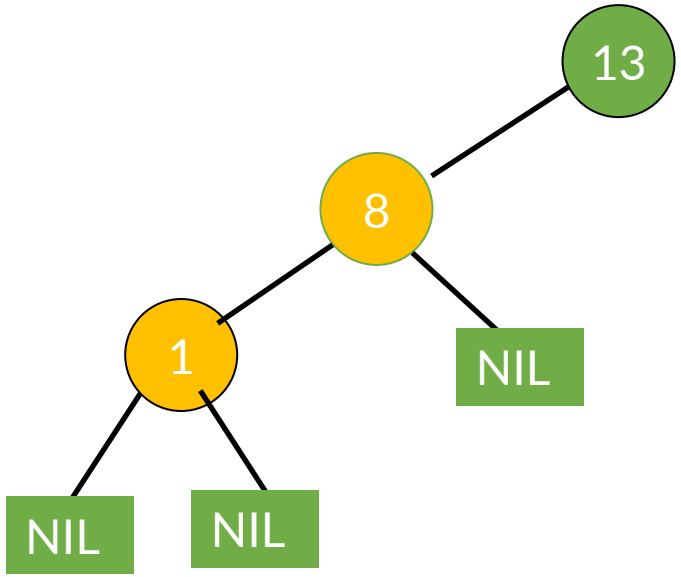


Vi phạm tính chất 5

1. ĐỊNH NGHĨA

- 1.2. Tính chất đỏ đen
  - Cây đỏ đen là cây nhị phân thỏa mãn các tính chất đỏ đen sau:
    - Mọi nút đều có màu đỏ hoặc đen.
    - Nút gốc có màu đen.
    - Mỗi lá (NIL) đều có màu đen.
    - Nếu một nút có màu đỏ thì cả 2 nút con của nó đều có màu đen.
    - Đối với mỗi nút, tất cả các đường đi từ nút đến các nút lá đều chứa cùng số nút đen

Không phải cây đỏ đen

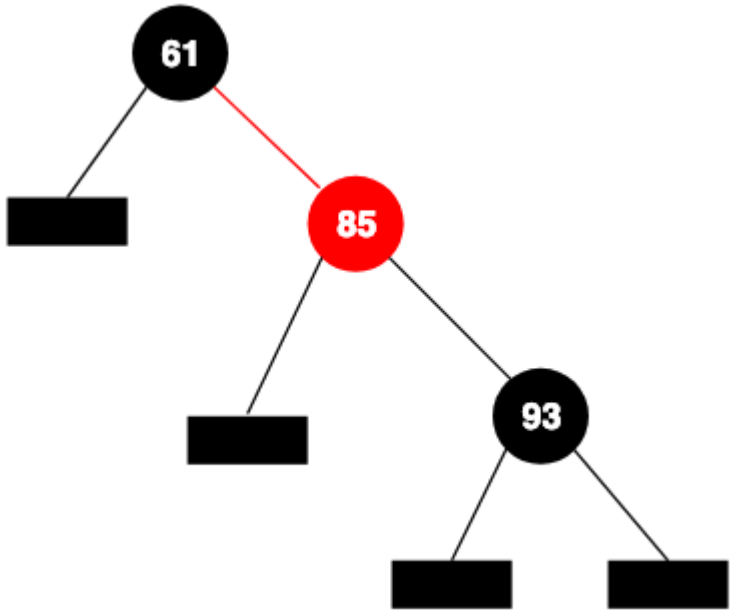


Vi phạm tính chất 4

1. ĐỊNH NGHĨA

- 1.2. Tính chất đỏ đen
  - Cây đỏ đen là cây nhị phân thỏa mãn các tính chất đỏ đen sau:
    - Mọi nút đều có màu đỏ hoặc đen.
    - Nút gốc có màu đen.
    - Mỗi lá (NIL) đều có màu đen.
    - Nếu một nút có màu đỏ thì cả 2 nút con của nó đều có màu đen.
    - Đối với mỗi nút, tất cả các đường đi từ nút đến các nút lá đều chứa cùng số nút đen

Không phải cây đỏ đen



Vi phạm tính chất 5

1. ĐỊNH NGHĨA

- 1.3. Cài đặt

```
typedef enum {BLACK,RED} NodeColor;
typedef int DataType; //Kiểu dữ liệu
typedef struct Node {
      DataType key; //Dữ liệu
      NodeColor color; //Màu của node
      struct Node *pLeft;
      struct Node *pRight;
      struct Node *pParent; //Để dễ cài đặt
} RBNode;
typedef struct RBNode* RBTREE;
```

NỘI DUNG TIẾP THEO

1. Định nghĩa

2. Các phép toán

2.1 Chèn một nút mới vào cây

2.2 Xóa một nút khỏi cây

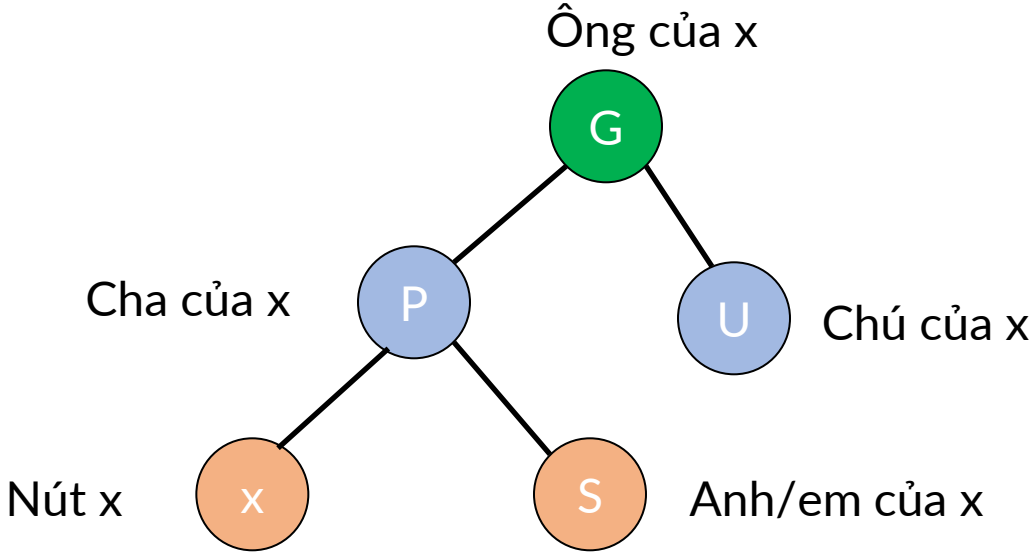
2. CÁC PHÉP TOÁN

- **2.1. Chèn một nút mới vào cây**
  - Thực hiện giống như cây BST
  - Node mới thêm luôn luôn có màu đỏ
  - Nếu xảy ra vi phạm qui tắc ⇒ điều chỉnh cây
  - Những qui tắc có thể bị vi phạm khi thêm nút mới
    - Mọi node phải là đỏ hoặc đen ⇒ OK
    - Node gốc là đen ⇒ not OK ! Nếu node mới là root
    - Các node ngoài (NULL) phải luôn luôn đen ⇒ OK
    - Nếu một node là đỏ, những node con của nó phải là đen ⇒ not OK ! vì có thể parent[z] = RED ⇒ 2 node liên tiếp màu đỏ
    - Mọi đường dẫn từ gốc đến nút lá phải có cùng số lượng node đen ⇒ OK vì không làm thay đổi số node đen

2. CÁC PHÉP TOÁN

- **2.1. Chèn một nút mới vào cây**
  - Nếu vi phạm phải cân bằng lại cây
  - Sử dụng 2 thao tác:
    - Đổi màu
      - Nếu nút có màu đỏ thì đổi thành màu đen và ngược lại.
      - Lưu ý: màu của nút lá (NULL) luôn là màu đen.
      - Luôn thử tô màu lại trước, nếu đổi màu không hiệu quả thì sẽ tiến hành xoay.
    - Xoay cây

- **2.1. Chèn một nút mới vào cây**
  - Quan hệ các nút trên cây



2. CÁC PHÉP TOÁN

- **2.1. Chèn một nút mới vào cây**

1. Chèn 1 nút vào cây nhị phân và gán màu đỏ cho nó.
2. Nếu nút mới là nút gốc → đổi màu của nó thành màu đen
3. Nếu không phải → kiểm tra màu của nút cha.

    3.1. Nếu nút cha màu đen → không đổi màu

    3.2. Nếu là màu đỏ → kiểm tra màu của nút chú.

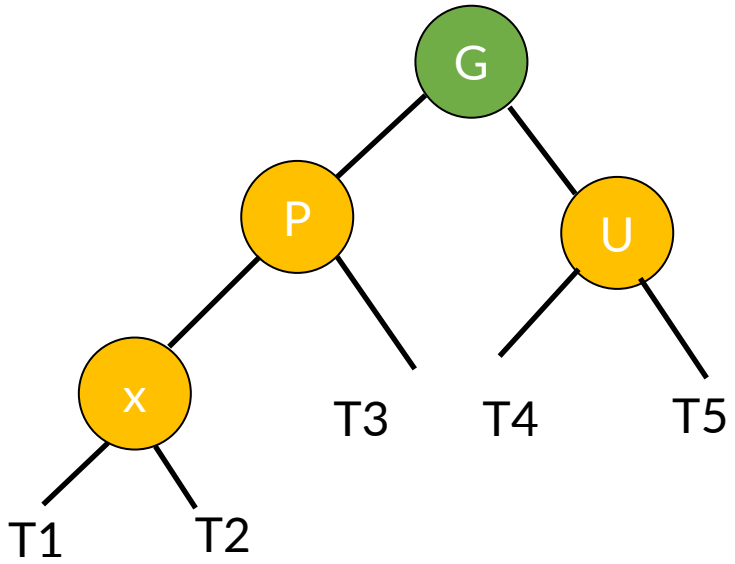
    3.2.1. Nếu nút chú màu đỏ → đổi màu của nút cha và chú thành màu đen và của nút ông nội thành màu đỏ và lặp lại quy trình tương tự cho nút đó (tức là ông nội).
    Nếu nút ông là nút gốc thì không đổi ông thành màu đỏ.

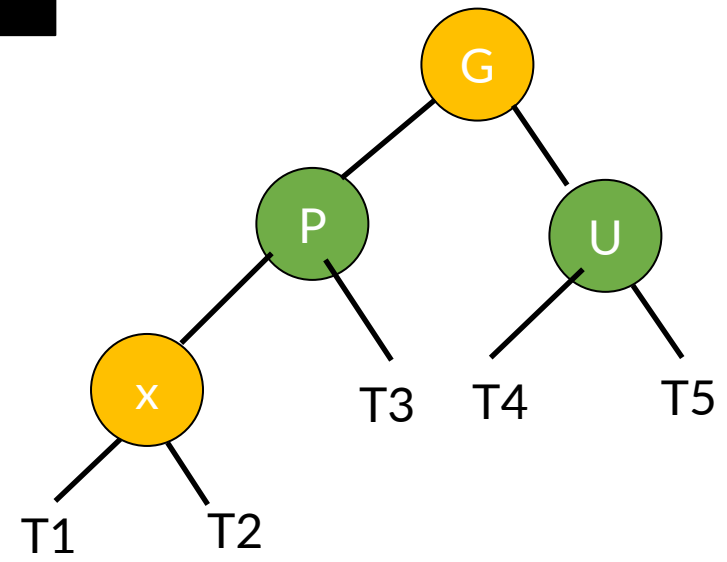
2. CÁC PHÉP TOÁN

- **2.1. Chèn một nút mới vào cây**

2.1.1 Nếu nút chú màu đỏ

1. Đổi màu của nút cha và chú thành màu đen
2. Nút ông nội thành màu đỏ



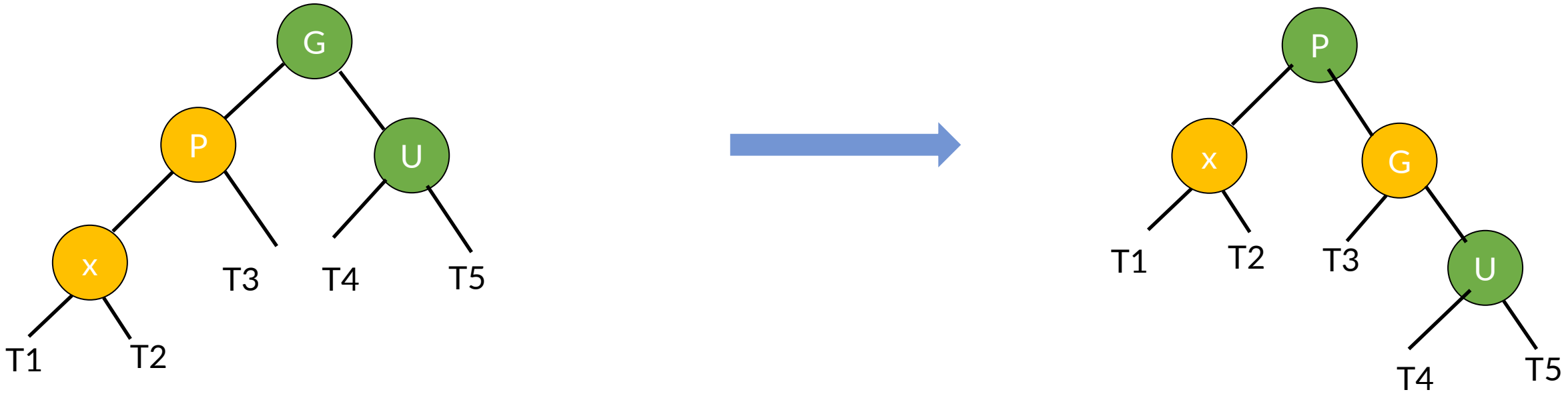


3. Lặp lại quy trình tương tự với nút ông là nút x

- **2.1. Chèn một nút mới vào cây**

2.1.2 Nếu nút chú màu đen: Trường hợp 1 - LL

1. Xoay phải tại nút ông
2. Đổi màu nút ông và nút cha

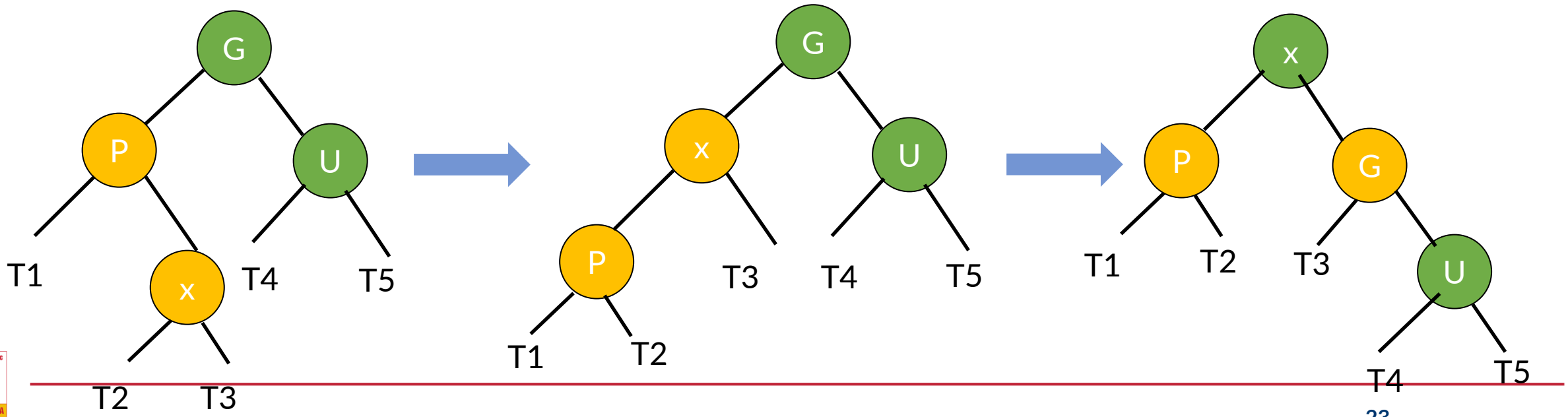


2. CÁC PHÉP TOÁN

- **2.1. Chèn một nút mới vào cây**

  2.1.2 Nếu nút chú màu đen: Trường hợp 2 - LR

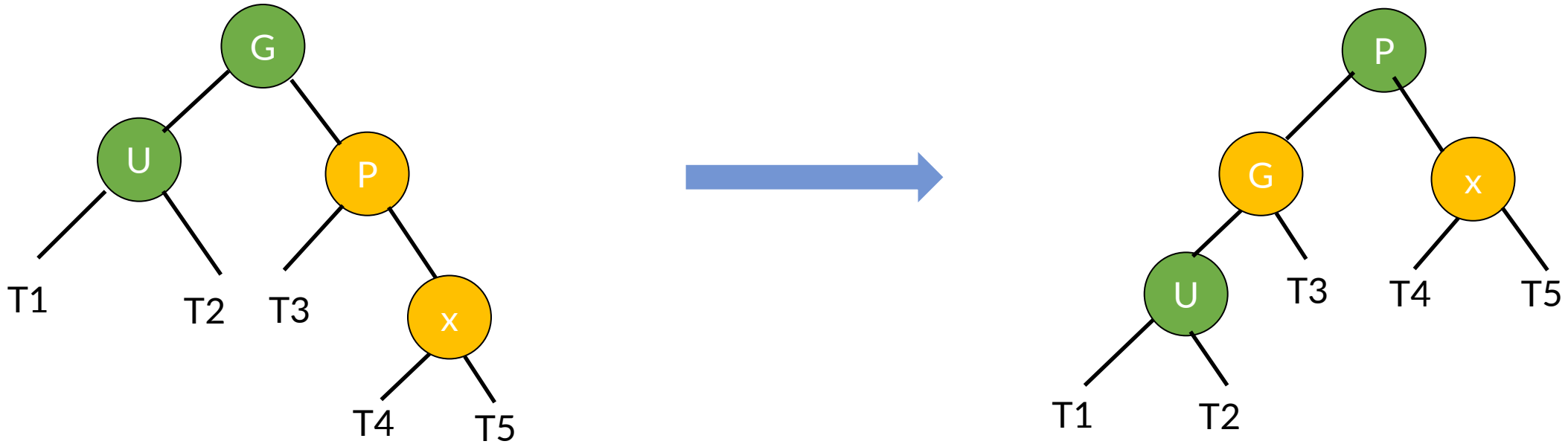
1. Xoay trái tại nút cha
2. Xoay LL



- **2.1. Chèn một nút mới vào cây**

  2.1.2 Nếu nút chú màu đen: Trường hợp 3 - RR

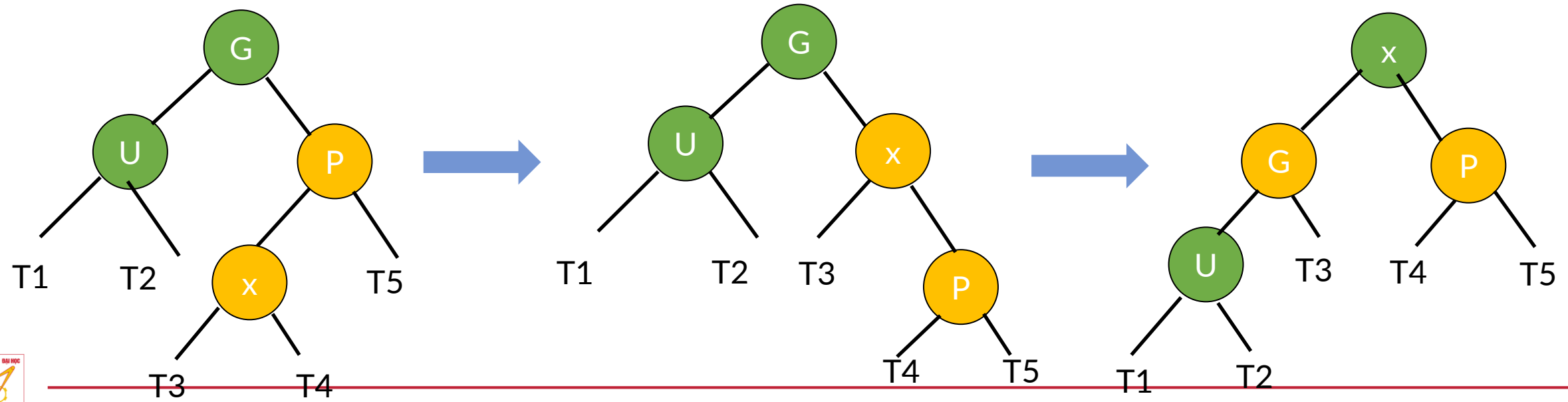
1. Xoay trái tại nút ông
2. Đổi màu nút ông và nút cha



- **2.1. Chèn một nút mới vào cây**

2.1.2 Nếu nút chú màu đen: Trường hợp 4 - RL

1. Xoay phải tại nút cha
2. Xoay RR



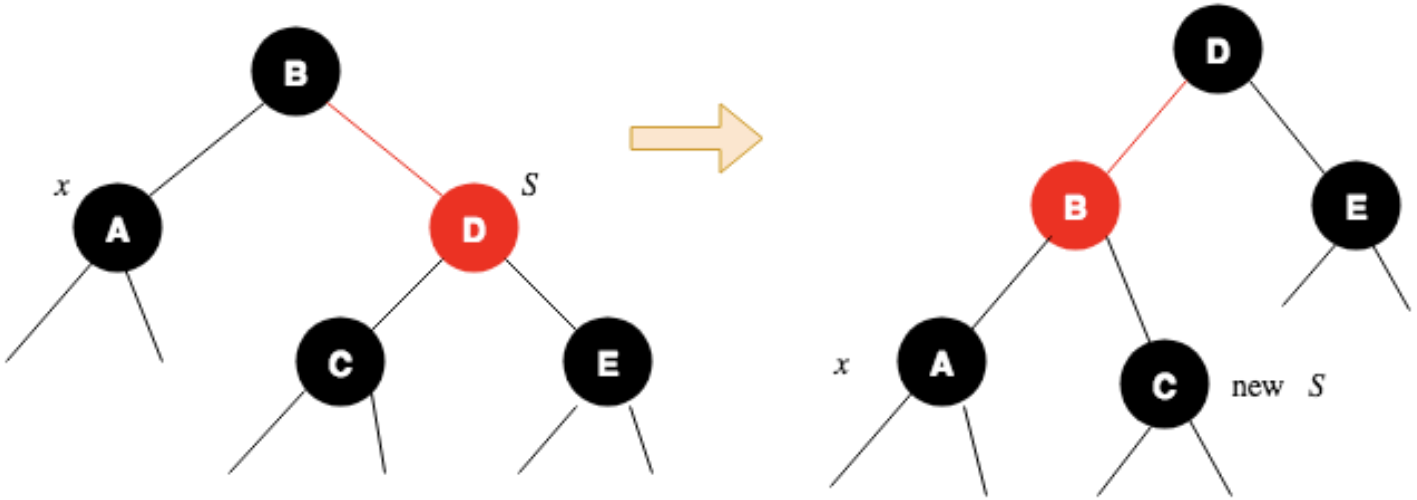
- **2.1. Chèn một nút mới vào cây**

```
RB-INSERT(T, k)
     BST-INSERT(T, k) //normal BST insertion
     while k.parent.color == RED
         if k.parent == k.parent.parent.right
            u = k.parent.parent.left //uncle
             if u.color == RED // case 3.1
                u.color = BLACK
                k.parent.color = BLACK
                k.parent.parent.color = RED
                k = k.parent.parent
             else if k == k.parent.left // case 3.3.1 and 3.3.2
                      k = k.parent
                      LEFT-ROTATE(T, k)
                k.parent.color = BLACK
                k.parent.parent.color = RED
                RIGHT-ROTATE(T, k.parent.parent)
        else (same as then clause with “left” and “right” exchanged)
     T.root.color = BLACK
```

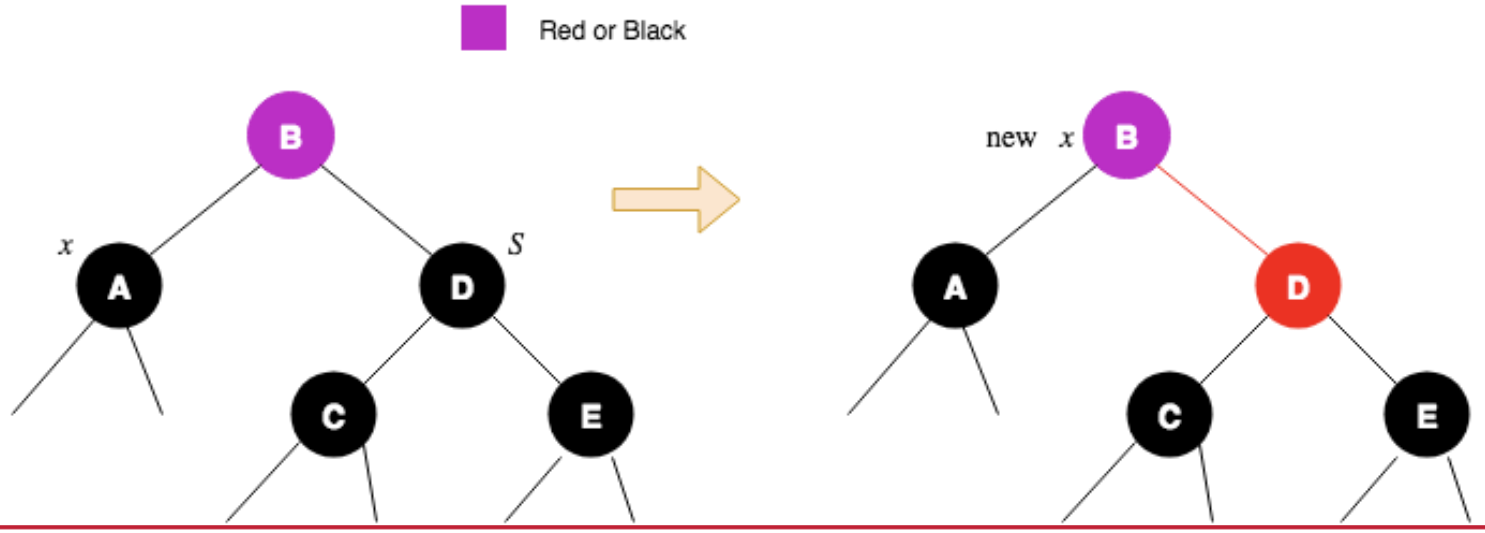
- **2.2. Xóa một nút khỏi cây**

  - Thực hiện theo quy trình xóa BST thông thường để đảm bảo rằng x là một nút lá hoặc có một nút con.
  - Có một số trường hợp của hoạt động xóa.
  - Trường hợp 1: x là một nút màu đỏ → chỉ cần xóa x
  - Trường hợp 2: x có một nút con màu đỏ → thay thế x bởi nút con màu đỏ của nó và đổi màu của nút con thành màu đỏ.
  - Trường hợp 3: x là một nút đen →thêm một nút đen bổ sung vào nút đã xóa và gọi nó là nút 'đen kép' → chuyển đổi nút đen kép thành 1 nút đen duy nhất.

- **2.2. Xóa một nút khỏi cây**
  - 2.2.1: Trường hợp nút x có nút anh chị em S màu đỏ
    - Đổi màu của S và nút cha P rồi thực hiện thao tác xoay trái tại nút cha
    - Chuyển các mục 2.2.1 thành các mục 2.2.2, 2.2.3 hoặc 2.2.4 .

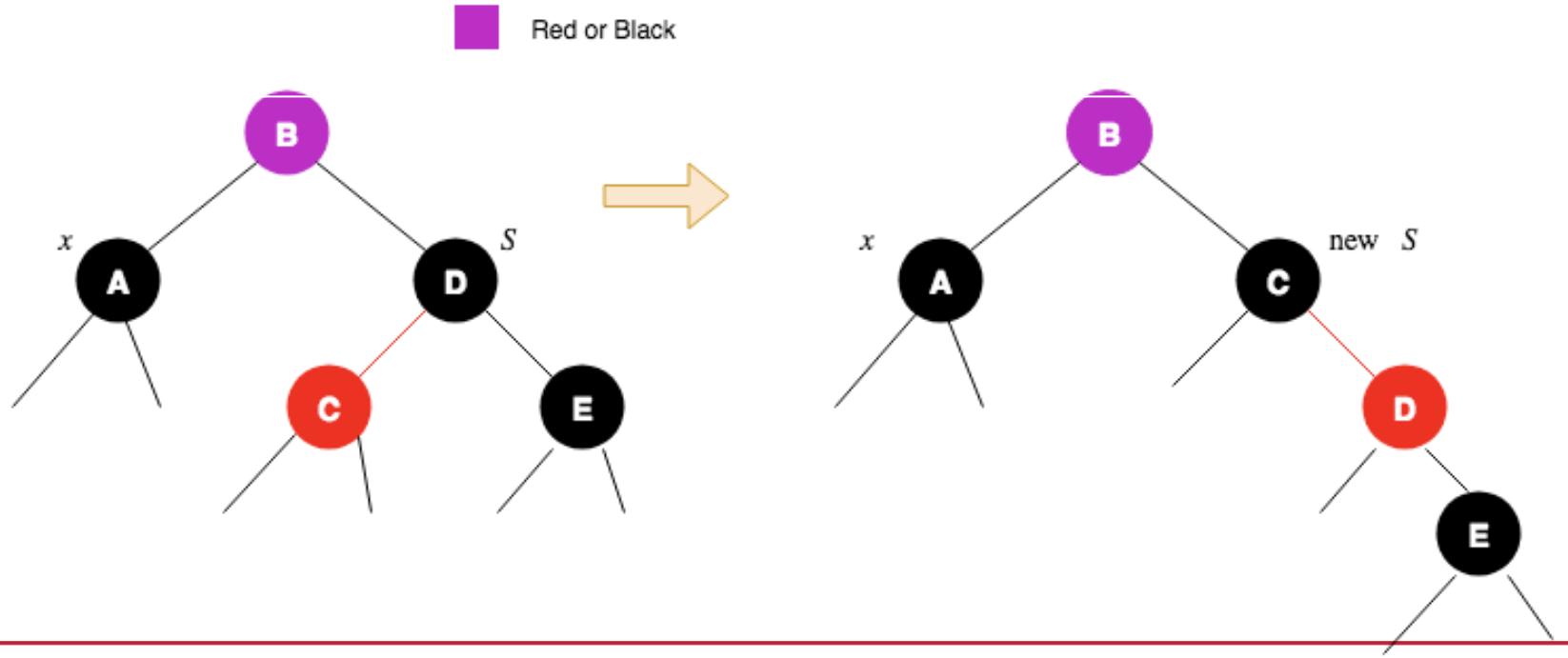


- **2.2. Xóa một nút khỏi cây**
  - 2.2.2: Trường hợp nút anh chị em S màu đen và cả 2 nút con đều màu đen
    - Màu của nút cha có thể đỏ hoặc đen. đổi màu của S thành màu đỏ
    - Nếu màu của nút cha là màu đỏ, đổi thành màu đen → trở thành mục 2.2.4
    - Ngược lại, nút cha thành một x mới và lặp lại quá trình từ trường hợp 2.2.1

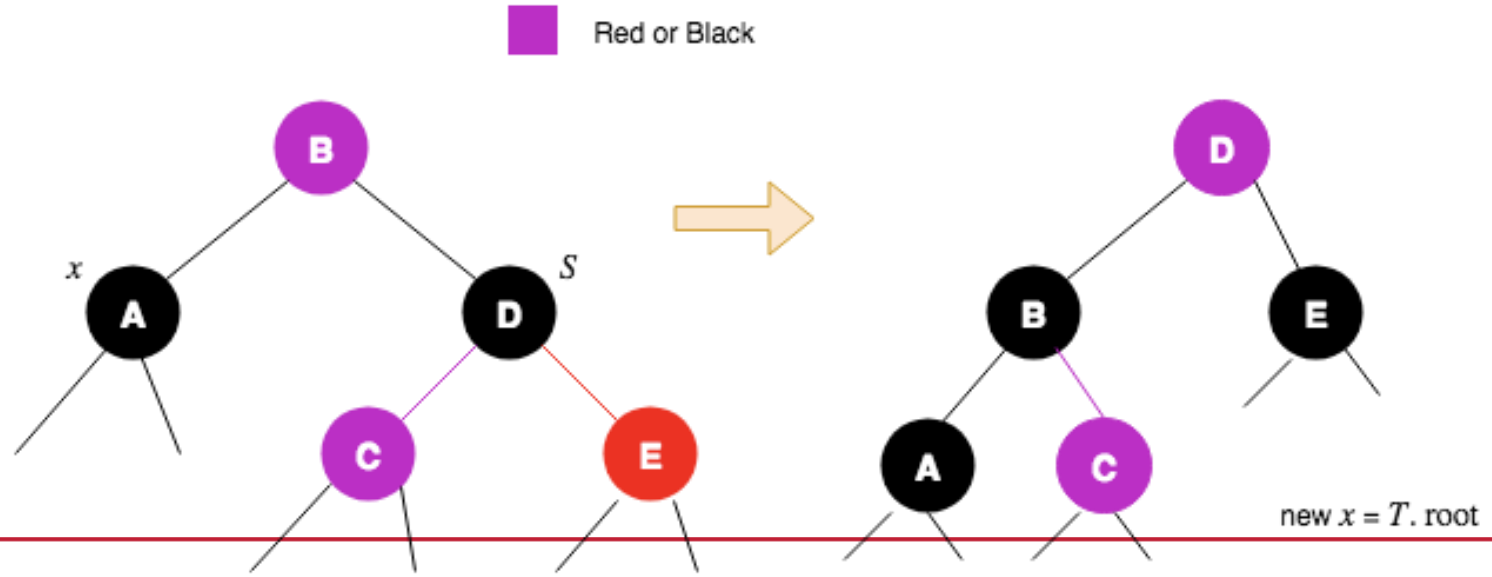


- **2.2. Xóa một nút khỏi cây**
  - 2.2.3: Trường hợp nút anh em S màu đen, S có con trái đỏ và con phải đen
    - Đổi màu S và con trái của nó, sau đó thực hiện xoay phải tại S

→ cây thành trường hợp 2.2.4



- **2.2. Xóa một nút khỏi cây**
  - 2.2.4: Trường hợp nút anh em S màu đen, S có con phải đỏ
    - Đổi màu S và con bên phải của nó thành đen
    - Đổi nút cha P thành màu đen
    - Thực hiện phép quay trái tại nút cha. (loại bỏ nút đen thừa trên x)



- **2.2. Xóa một nút khỏi cây**

```
RB-DELETE(T, x)
    BST-DELETE(T, x)
    while x ≠
 T.root and x.color == BLACK
        if x == x.parent.left
            s = x.parent.right
            if s.color == RED
                s.color = BLACK // case 3.1
                x.parent.color = RED // case 3.1
                LEFT-ROTATE(T, x.parent) // case 3.1
                s = x.parent.right // case 3.1
            if s.left.color == BLACK and s.right.color == BLACK
                s.color = RED // case 3.2
                x = x.parent //case 3.2
```

```
else if s.right.color == BLACK
                    s.left.color = BLACK // case 3.3
                    s.color = RED //case 3.3
                    RIGHT-ROTATE(T, s) // case 3.3
                    s = x.parent.right // case 3.3
                s.color = x.parent.right // case 3.4
                x.parent.color = BLACK // case 3.4
                s.right.color = BLACK // case 3.4
                LEFT-ROTATE(T, x.parent) // case 3.4
                x = T.root
        else (same as then close with “right” and “left”
exchanged)
    x.color = BLACK
```

HUST

hust.edu.vn fb.com/dhbkhn

THANK YOU !