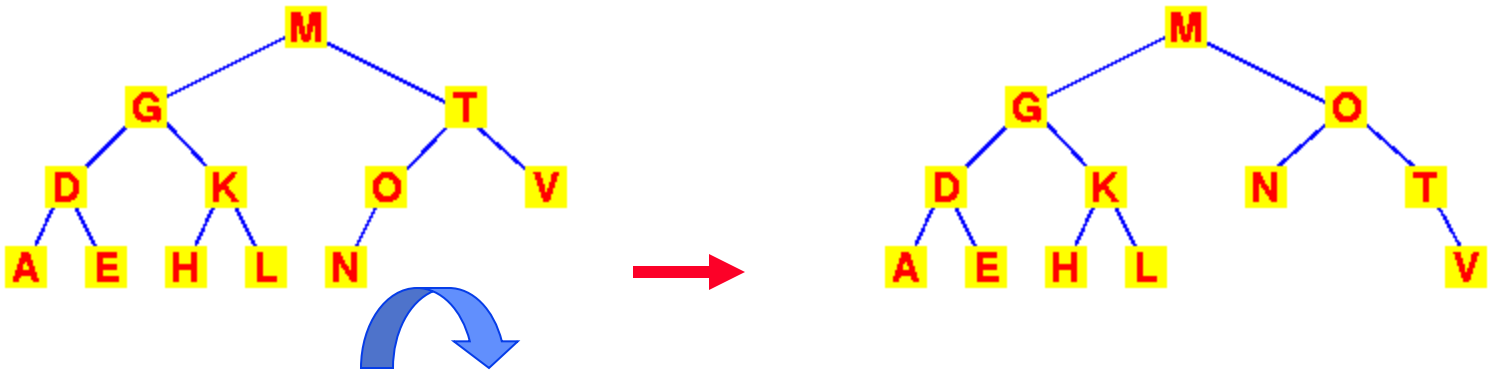
Cây đỏ đen - *Đặt vấn đề*

- Cây BST có một số vấn đề cần phải xem xét. Thao tác thêm vào một node trên cây sẽ thực hiện tốt nếu dữ liệu được chưa sắp xếp.
- Ngược lại, nếu dữ liệu được chèn vào đã được sắp xếp thì trở nên chậm hơn nhiều. Lý do là khi dữ liệu cần chèn đã được sắp xếp thì cây nhị phân trở nên mất cân bằng dẫn đến mất đi khả năng tìm kiếm nhanh (trong thao tác chèn hoặc xóa) một phần tử đã cho.
- Một cách tiếp cận cách giải quyết vấn đề của cây không cân bằng: đó là *cây đỏ đen* - là cây tìm kiếm nhị phân có thêm một số đặc điểm khác.

Phép quay trên cây BST

Nhắc lại về phép quay trong cây BST

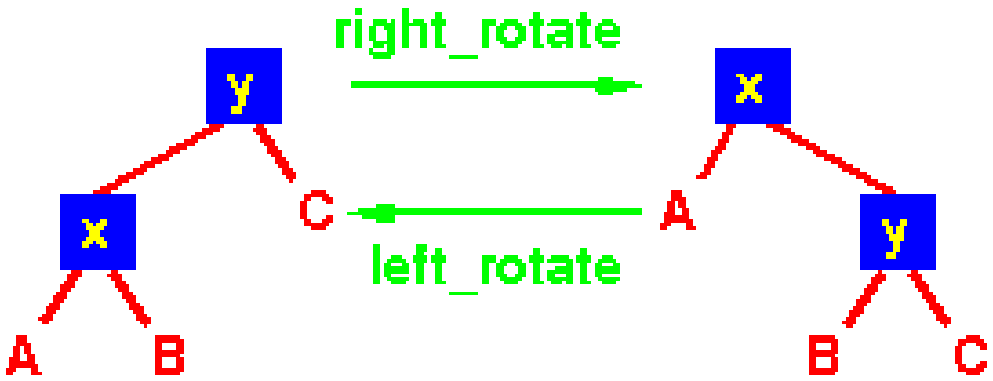
Để bảo đảm thứ tự cây BST chúng ta quan sát phép quay biểu diễn như sau:



Thực hiên phép quay cây con quanh T và O .

Phép quay trên cây BST

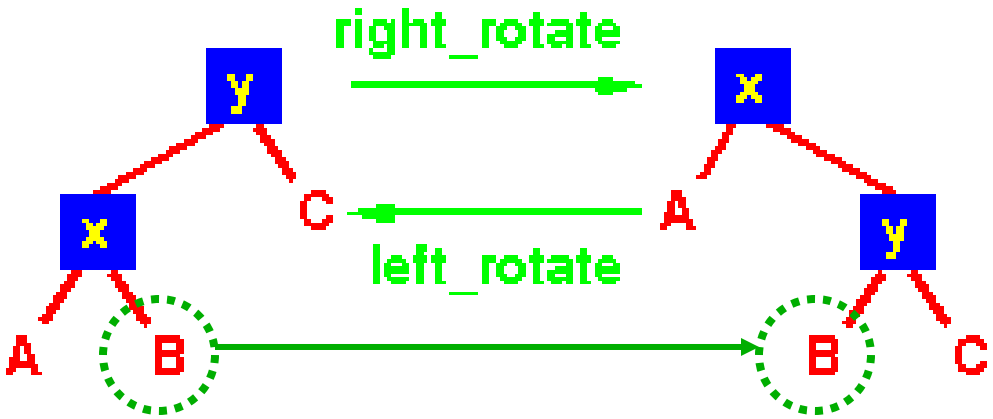
Minh hoạ phép quay Trái và quay phải (left-rotations or right-rotations)



Kết qủa của 2 phép quay trên thứ tự trong phép duyệt cây là: A x B y C

Phép quay trên cây BST

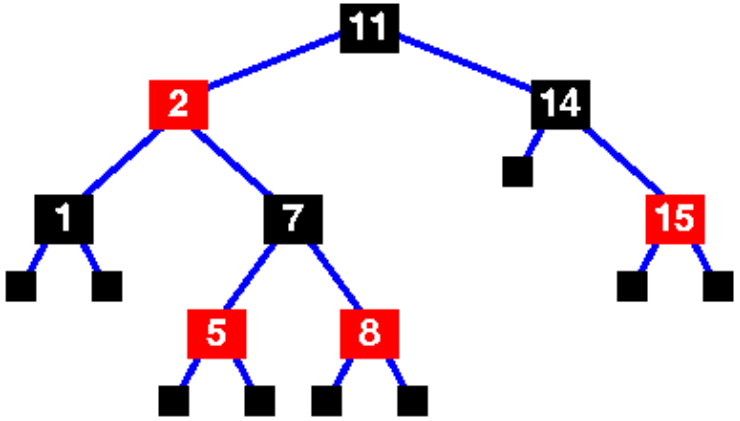
Các phép quay có thể là quay trái hoặc quay phải



Để ý rằng trong phép quay này chúng ta có thể B từ con phải của X thành con trái của Y

Giới thiệu Cây đỏ đen

- **Một cây BST là cây đỏ đen nếu:**
- **Mỗi một nút của cây là đỏ hoặc đen.**
- **Tất cả các nút lá là đen.**
- **Nếu một nút là đỏ thì cả hai nút con là đen.**
- **Đường đi từ một nút đến nút lá có cùng số nút đen.**



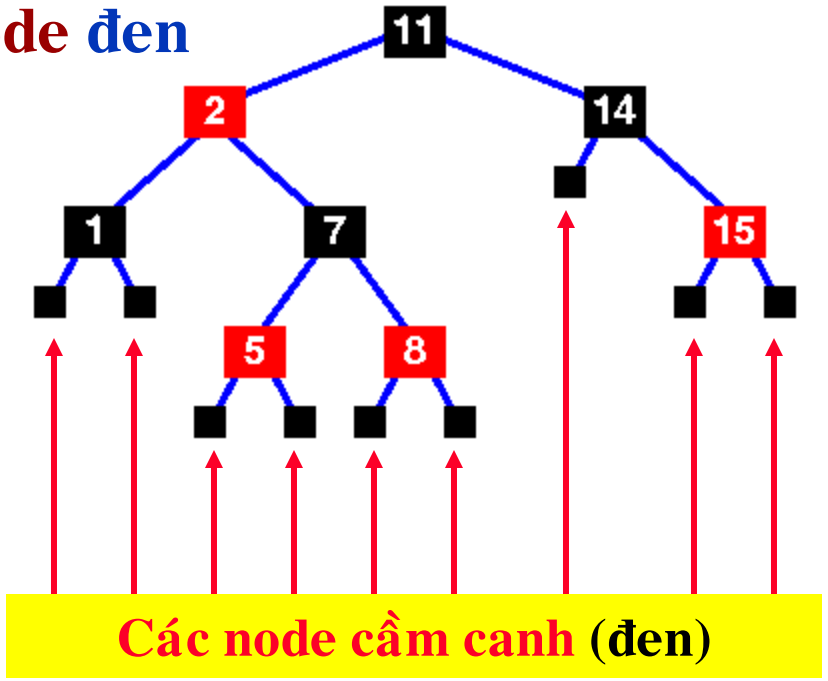
Giới thiệu Cây đỏ đen (tt)

Cây đỏ đen là:

Mỗi một node hoặc là node đỏ hoặc node đen

Tất cả các node lá là node đen

Các node lá không chứa dữ liệu-goi là node cầm canh (đen)

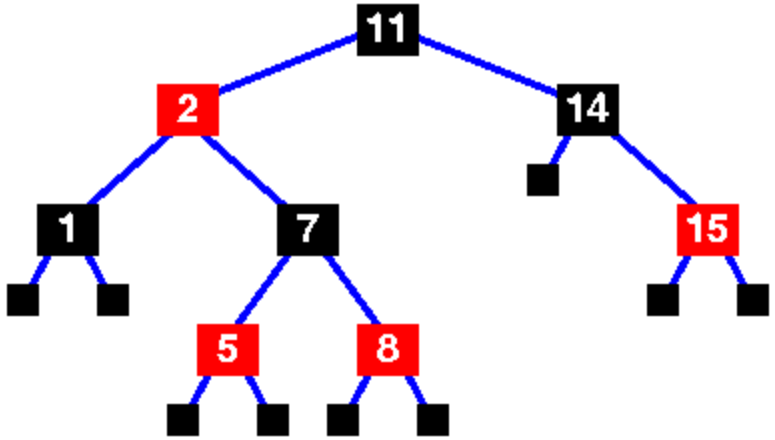


Giới thiệu Cây đỏ đen(tt)

Mỗi một node hoặc là node đỏ hoặc node đen

Tất cả các node lá là node đen

- **Nếu là node đỏ thì 2 node con là đen**



Không có 2 node đỏ kề nhau trên một đường đi
(Các node đen có thể kề nhau trên cùng một đường đi)

Giới thiệu Cây đỏ đen(tt)

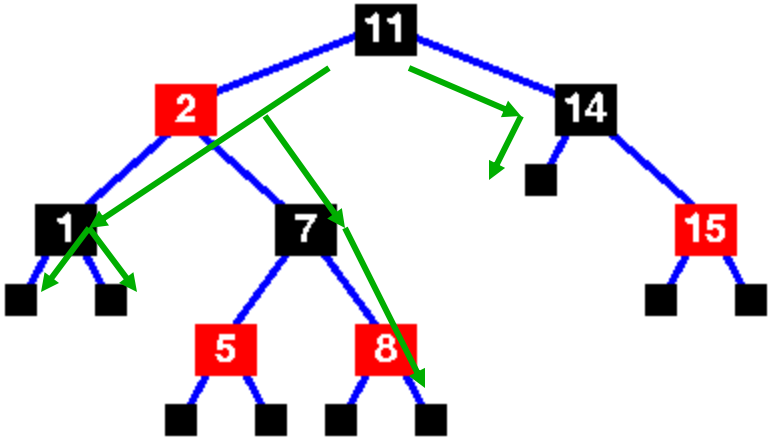
Cây đỏ đen là cây:

Mỗi một node hoặc là node đỏ hoặc node đen

Tất cả các node lá là node đen

- **Nếu là node đỏ thì 2 node con là đen**

Mỗi đường đi từ một nút đến node lá có cùng số node đen



Từ node gốc có 3 nodes Đen trên mỗi đường đi

Chiều dài đường đi này được gọi là chiều cao đen của cây

Chiều cao đen của nút x, bh(x) là số nút đen từ x đến nút lá (không tính x)

Giới thiệu Cây đỏ đen(tt)

Bổ đề

Một cây đỏ đen n- node có

$height <= 2 \log(n+1)$

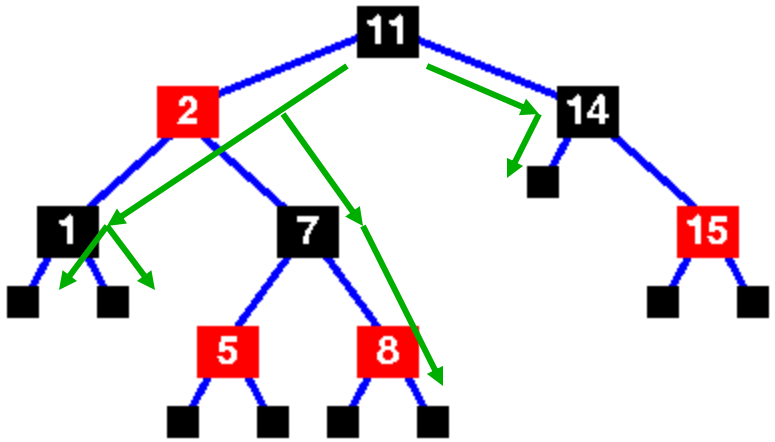
***height*- Chiều cao của cây**

Tính chất:,

$height <= 2 * bh(x)$

Thời gian tìm kiếm:

$O(\log n)$



Giới thiệu Cây đỏ đen-Cài đặt

Cấu trúc dữ liệu

Cây đỏ đen có cấu trúc của cây BST và thêm vào thuộc tính màu của node và liên kết đến node cha

```
struct t_red_black_node {
    enum { red, black } colour;
    void *item;
    struct t_red_black_node *left,
                            *right,
                            *parent;
    }
```

Tương tự cây BST thêm vào 2 thuộc tính

Cây đỏ đen – Phép toán thêm vào(1)

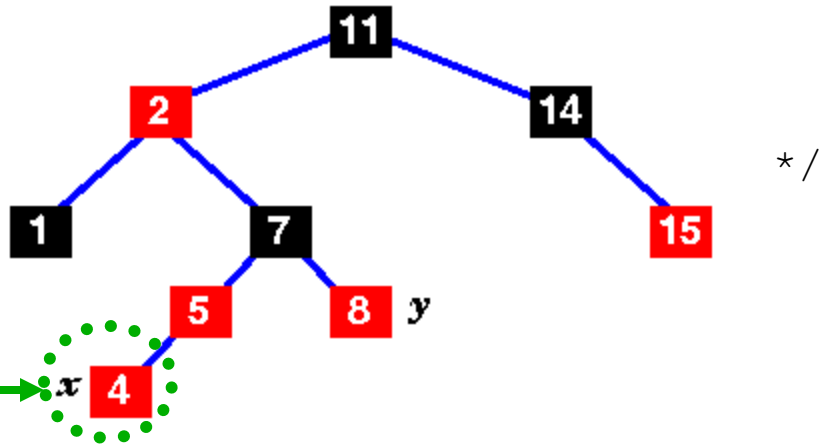
Thêm một node vào cây

Yêu cầu cân bằng lại cây

```
rb_insert( Tree T, node x ) {
    /* Phép toán thêm vào giống cây BST  */
    tree_insert( T, x );
       /* Phục hồi lại thuộc tính đỏ đen  */
    x->colour = red;
                root) &&
                == x->par
                arent is                                    */
            y = x->parent->paren
            if ( y->colour == re
                /* case 1 - chan
                colou
                y->colour = blac
                x->parent->parent->colour = red;
                /* Move x up the tree */
                x = x->parent->parent;
```

Node thêm vào là 4
Đánh dấu màu đỏ

Node thêm vào: X



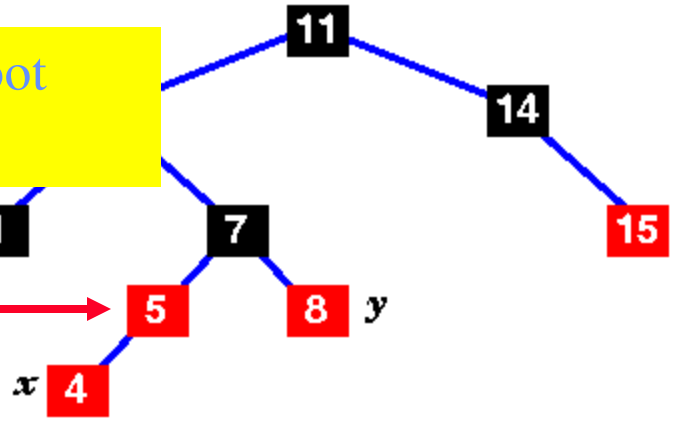
Cây đỏ đen – Phép toán thêm vào(2)

```
rb_insert( Tree T, node x ) {
    /* Phép toán thêm vào giống cây BST */
    tree_insert( T, x );
     /* Phục hồi lại thuộc tính đỏ đen */
     x->colour = red;
     while ( (x != T->root) && (x->parent->colour == red) )
       if ( x->parent == x->parent
```

Trong khi X không phải là Root và cha của X la màu đỏ

```
            /* case 1 - change
            x->parent->colour =
            y->colour = black;
            x->parent->parent->
            /* Move x up the tr
            x = x->parent->parent;
```

x->parent



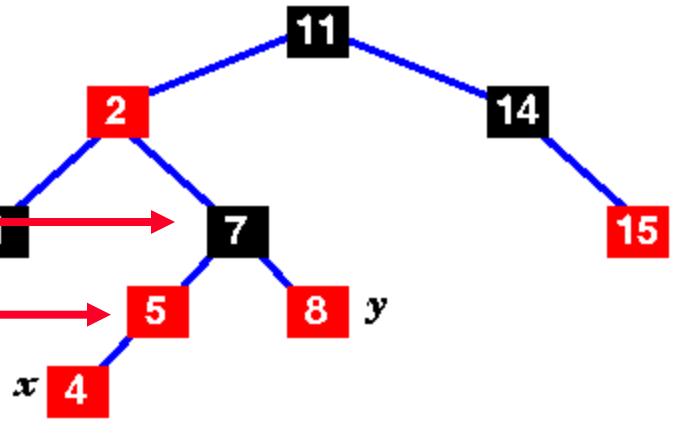
Cây đỏ đen – Phép toán thêm vào(3)

```
rb_insert( Tree T, node x ) {
    /* Phép toán thêm vào giống cây BST */
   tree_insert( T, x );
   /* Phục hồi lại thuộc tính đỏ đen */
     x->colour = red;
     while ( (x != T->root) && (x->parent->colour == red) )
       if ( x->parent == x->parent->parent->left ) {
```

Nếu cha của X là node bên trái

```
            y = x->parent->parent->right;
            if ( y->colour == red ) {
                /* case 1 - change the col
                                        ack;
                x->parent->parent
                x->par                  >colour
                /* Mov                  ree */
                x = x->parent->parent;
```

x->parent



Cây đỏ đen – Phép toán thêm vào(4)

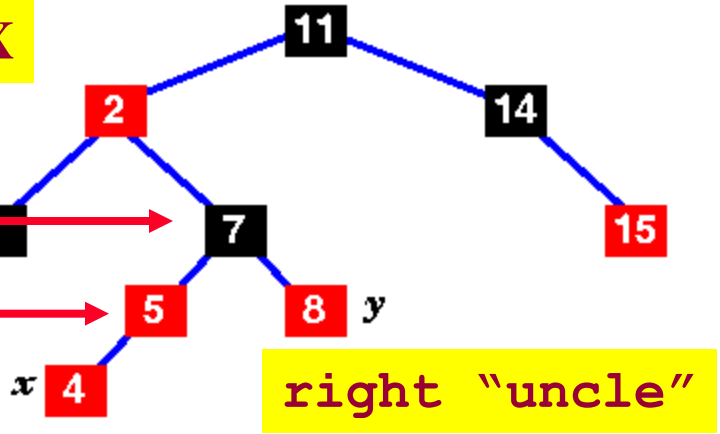
```
/* Now restore the red-black property */
x->colour = red;
while ( (x != T->root) && (x->parent->colour == red) )
  if ( x->parent == x->parent->parent->left ) {
     /* If x's parent is a left, y is x's right 'uncle'
     y = x->parent->parent->right;
```

Y là node chú bác "uncle" bên phải của X

```
     x->parent->colour = b
     y->colour = black;
     /* Move x up the tree
     x = x->parent;
```

x->parent->parent

x->parent

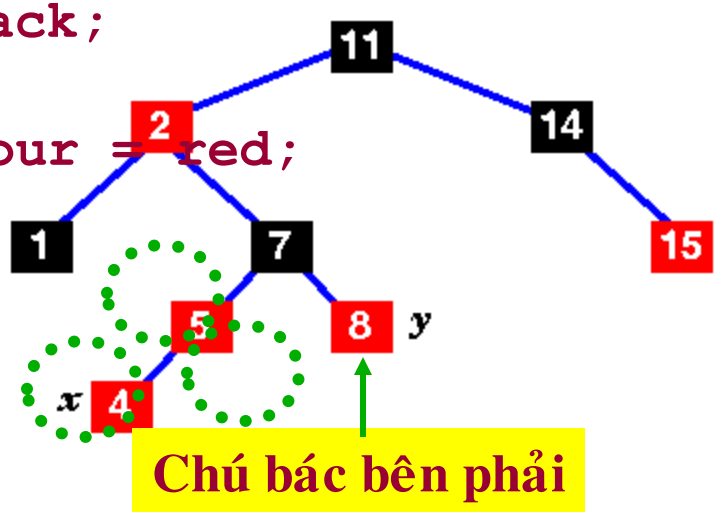


right "uncle"

Cây đỏ đen – Phép toán thêm vào(5)

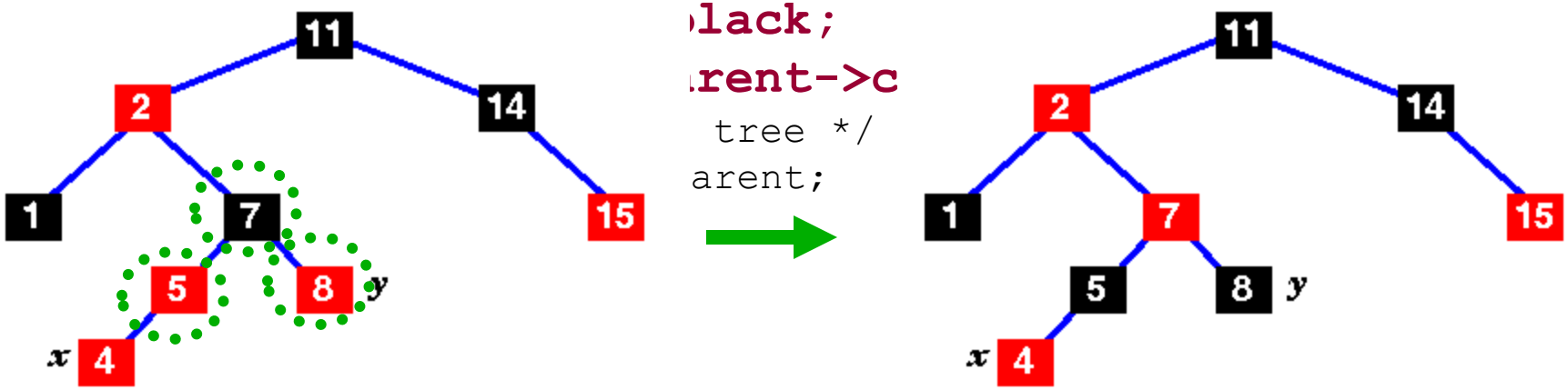
```
while ( (x != T->root) && (x->parent->colour == red) )
  if ( x->parent == x->parent->parent->left ) {
     /* If x's parent is a left, y is x's right 'uncle'
     y = x->parent->parent->right;
     if ( y->colour == red ) {
        /* case 1 - change the colours */
        x->parent->colour = black;
        y->colour = black;
        x->parent->parent->colour = red;
```

Nếu uncle là red, đổi màu của Y cha cuả X và cha của Y



Cây đỏ đen – Phép toán thêm vào(6)

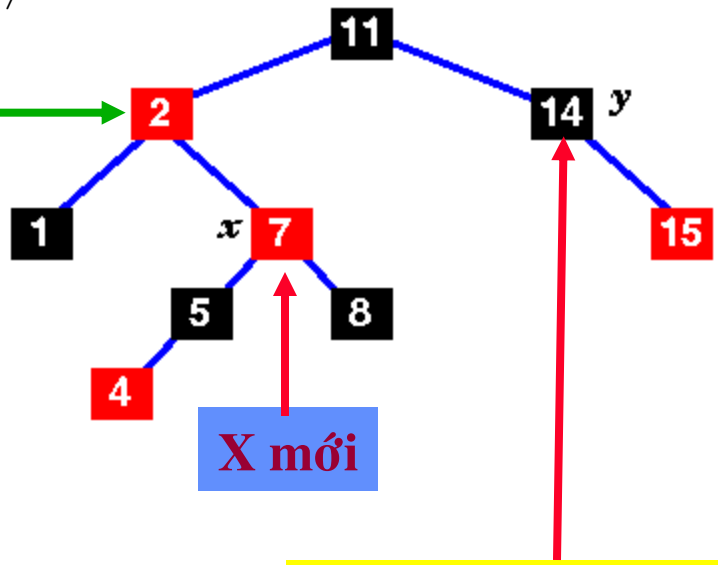
```
while ( (x != T->root) && (x->parent->colour == red) )
  if ( x->parent == x->parent->parent->left ) {
     /* If x's parent is a left, y is x's right 'uncle'
     y = x->parent->parent->right;
     if ( y->colour == red ) {
        /* case 1 - change the colours */
        x->parent->colour = black;
                          lack;
                          rent->c
                           tree */
                          arent;
```



Cây đỏ đen – Phép toán thêm vào(7)

```
while ( (x != T->root) && (x->parent->colour == red) ) {
  if ( x->parent == x->parent->parent->left ) {
    /* If x's parent is a left, y is x's right 'uncle' */
    y = x->parent->parent->right;
    if ( y->colour == red ) {
       /* case 1 - change the colours */
       x->parent->colour = black;
       y->colour = red;
       x = x->parent->parent;
```

Cha của X là node con trái
Chú bác của X là Y có màu là đen

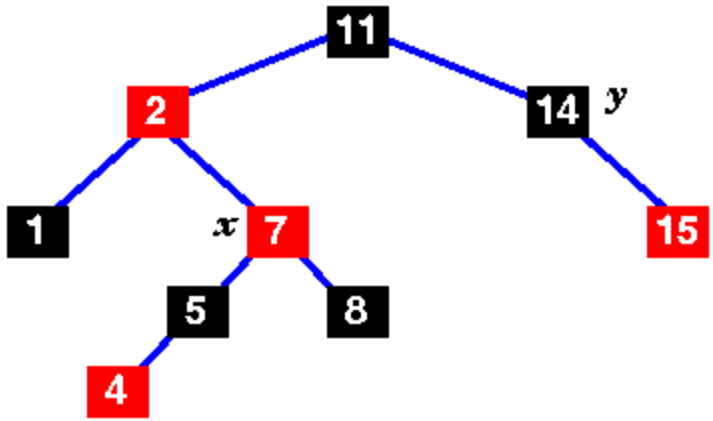


Y là chú bác của X
Bây giờ có màu là đen

Cây đỏ đen – Phép toán thêm vào(8)

```
while ( (x != T->root) && (x->parent->colour == red) ) {
  if ( x->parent == x->parent->parent->left ) {
    /* Nếu cha của X là node trái, Y là chú bác của X */
    y = x->parent->parent->right;
     if ( y->colour == red ) {
       /* case 1 - change the colours */
       x = x->parent->parent;
     else {
          /* Y là node đen */
        if ( x == x->parent->right ) {
           /* và x là node phải */
      /* case 2 – Chuyên X lên trên và xoay */
           x = x->parent;
           left_rotate( T, x );
```

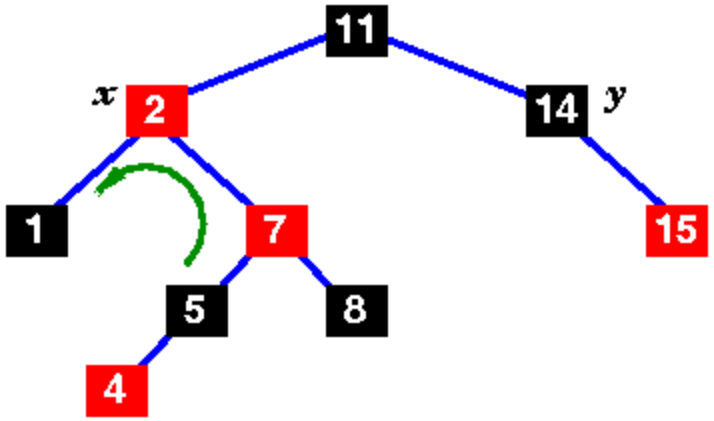
Cha của X là node con trái
Chú bác của X là Y có màu là đen



Cây đỏ đen – Phép toán thêm vào(9)

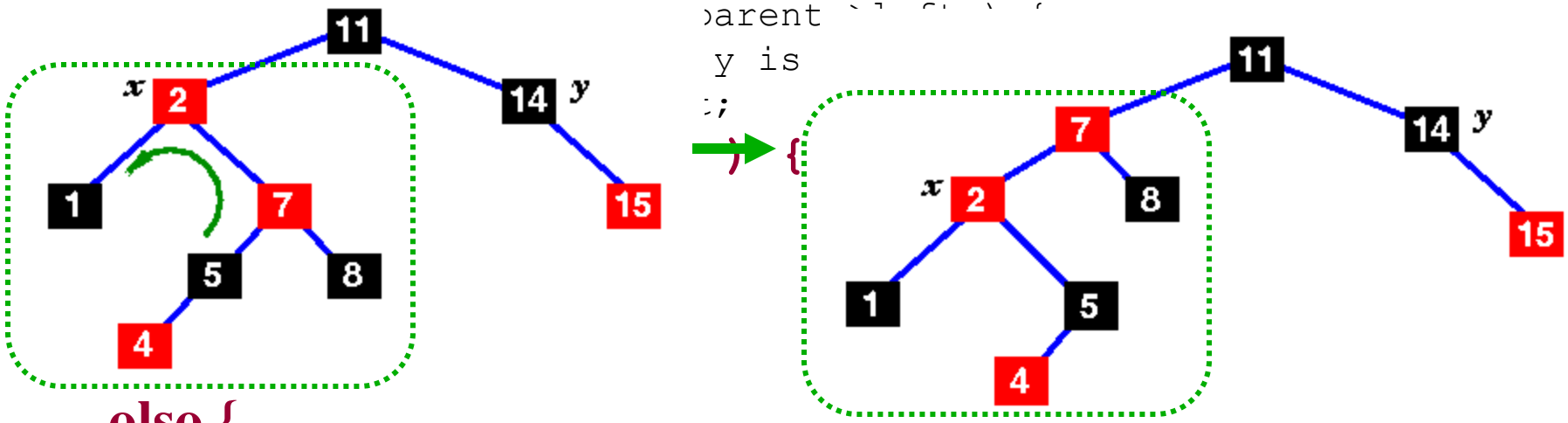
```
while ( (x != T->root) && (x->parent->colour == red) ) {
  if ( x->parent == x->parent->parent->left ) {
    /* If x's parent is a left, y is x
    y = x->parent->parent->right;
     if ( y->colour == red ) {
       /* case 1 - change the colours */
       x = x->parent->parent;
     else {
            /* Y là node đen */
     if ( x == x->parent->right ) {
       /* và x là node phải */
       /* case 2 – Chuyên X lên trên và xoay */
       x = x->parent;
       left_rotate( T, x );
```

.. Di chuyển X lên trên
và thực hiện phép xoay trái...



Cây đỏ đen – Phép toán thêm vào(10)

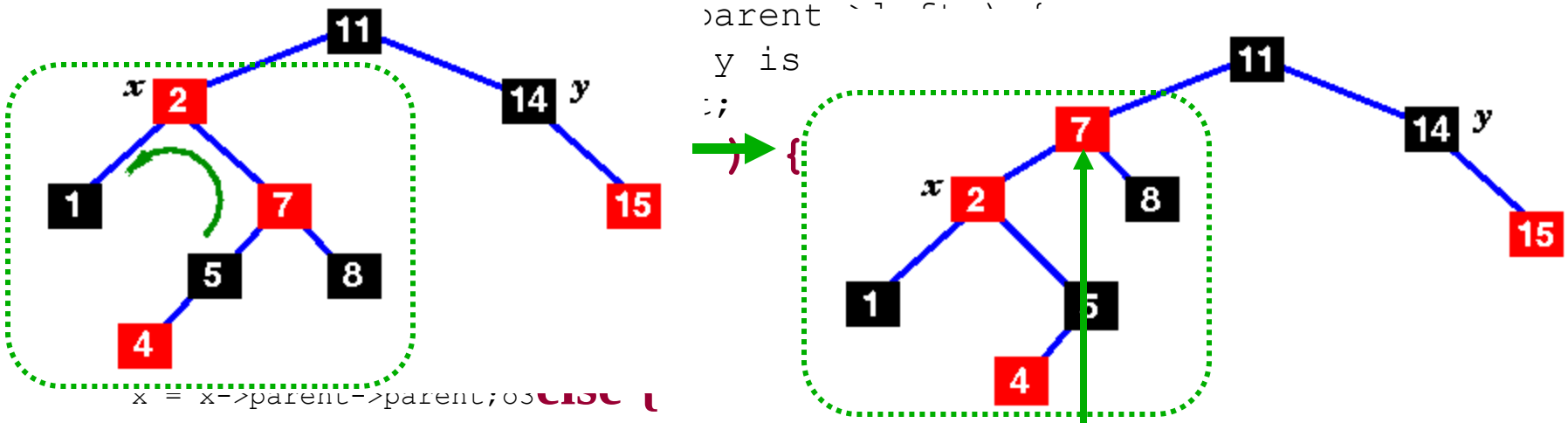
```
while ( (x != T->root) && (x->parent->colour == red) ) {
                                  parent
                                   y is
                                  ;
                                  ) {
```




```
    else {
            /* Y là node đen */
    if ( x == x->parent->right ) {
      /* và x là node phải */
      /* case 2 – Chuyên X lên trên và xoay trái*/
      x = x->parent;
      left_rotate( T, x );
```

Cây đỏ đen – Phép toán thêm vào(11)

```
while ( (x != T->root) && (x->parent->colour == red) ) {
```



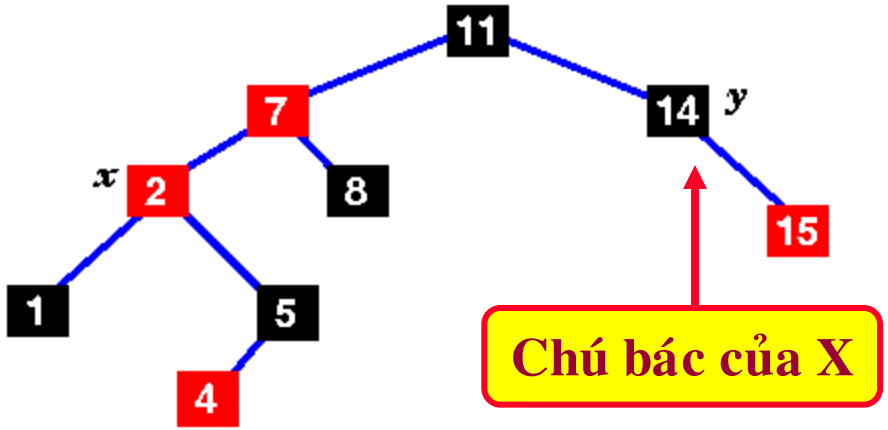

```
    /* Y là node đen */
    if ( x == x->parent->right ) {
      /* và x là node phải */
      /* case 2 – Chuyên X lên trên và xoay */
      x = x->parent;
      left_rotate( T, x );
```

.. Bây giờ cha của X là đỏ ...

Cây đỏ đen – Phép toán thêm vào(12)

```
while ( (x != T->root) && (x->parent->colour == red) ) {
  if ( x->parent == x->parent->parent->left ) {
  /*Cha của X là bên trái, Y là chú bác của X, Y là node phải */
  y = x->parent->parent->right;
  if ( y->colour == red ) {
        /* case 1 - change the colours */
        y->colour = black;
        x->parent->colour = red;
        /* Move x up the tree */
        x = x->parent->parent;
    else {
      /* Y là node đen */
      if ( x == x->parent->right ) {
          /* and x is to the right */
          x = x->parent;
          left_rotate( T, x );
```

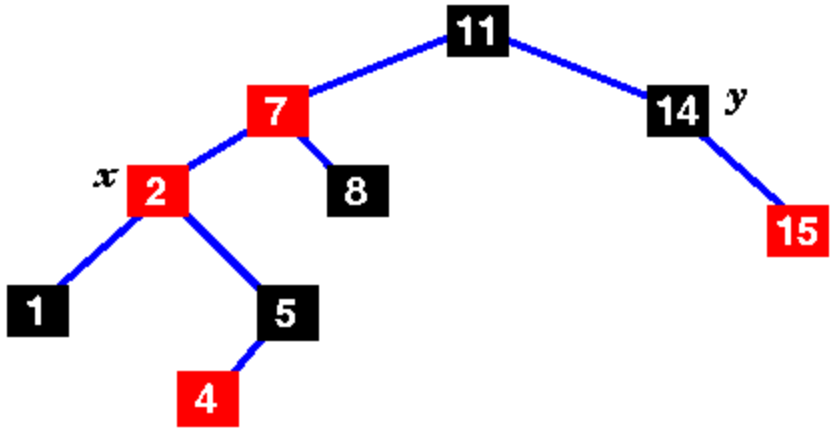
.. Chú bác của X là đen ..



.. Và X la node bên trái..

Cây đỏ đen – Phép toán thêm vào(13)

```
while ( (x != T->root) && (x->parent->colour == red) ) {
  if ( x->parent == x->parent->parent->left ) {
y = x->parent->parent->right;
  if ( y->colour == red ) {
        /* TH 1 – chuyển màu */
        x->parent->colour = black;
        y->colour = black;
        x->parent->parent->colour = red;
        /* Di chuyển X lên phía trên */
        x = x->parent->parent;
     else {
     /* Y là node đen */
     if ( x == x->parent->right ) {
           /* and x is to the right */
           /* TH 2 – Chuyển X lên trên và xoay trái */
           x = x->parent;
           left_rotate( T, x );
```



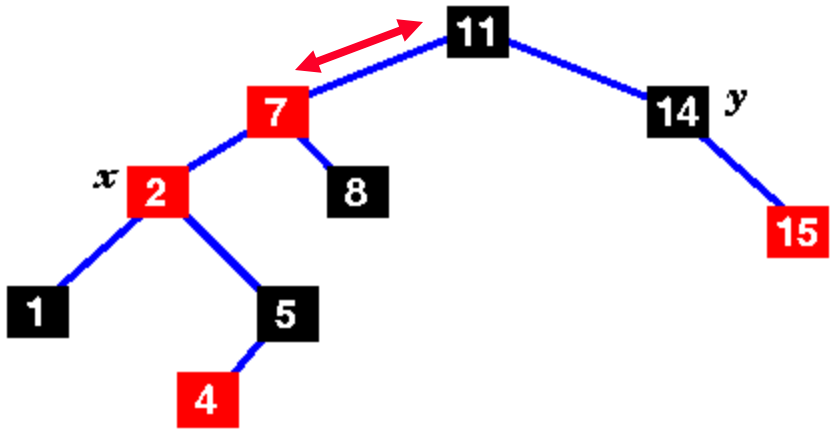

```
   else { /* trường hợp 3 */
     x->parent->colour = black;
     x->parent->parent->colour = red;
     right_rotate( T, x->parent->parent );
     }
```

Cây đỏ đen – Phép toán thêm vào(14)

```
while ( (x != T->root) && (x->parent->colour == red) ) {
   if ( x->parent == x->parent->parent->left ) {
y = x->parent->parent->right;
 if ( y->colour == red ) {
      /* TH 1 – chuyển màu */
      x->parent->colour = black;
      y->colour = black;
      x->parent->parent->colour = red;
      /*
      x
   els
   /* Y là node đen */
   if ( x == x->parent->right ) {
      /* and x is to the right */

          /* TH 2 – Chuyển X lên trên và xoay trái */
      x = x->parent;
      left_rotate( T, x );
```

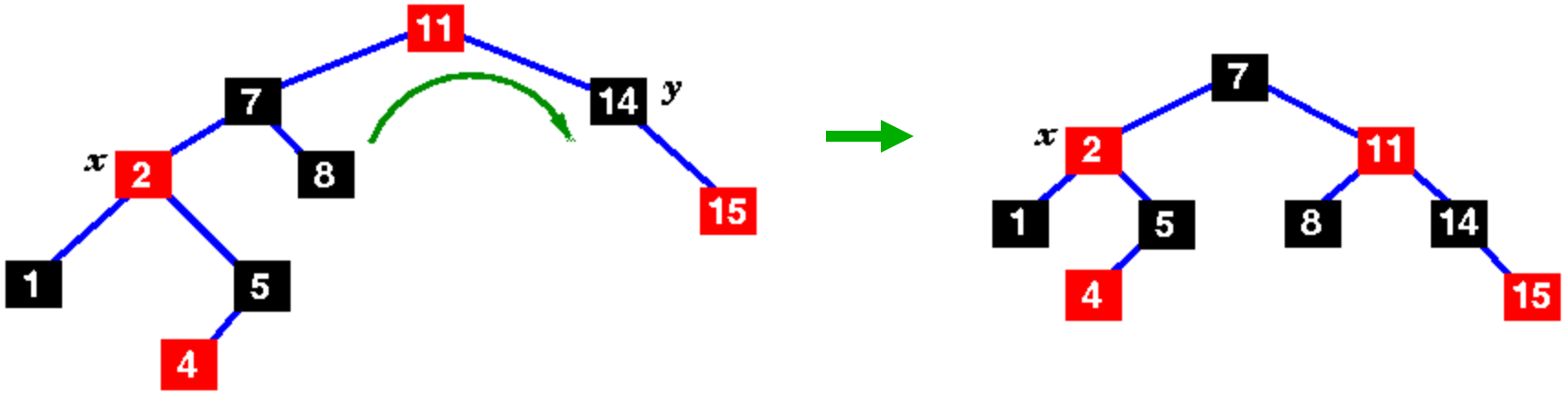
.. Chuyển màu và xoay ..




```
   else {  /* Trường hợp 3 */
      x->parent->colour = black;
      x->parent->parent->colour = red;
      right_rotate( T, x->parent->parent );
      }
```

Cây đỏ đen – Phép toán thêm vào(15)

```
while ( (x != T->root) && (x->parent->colour == red) ) {
   if ( x->parent == x->parent->parent->left ) {
y = x->parent->parent->right;
```

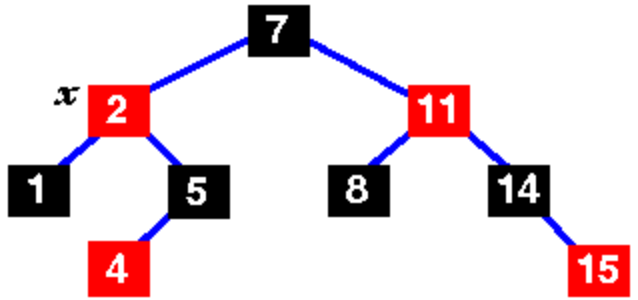



```
x = x->parent;
left_rotate( T, x );
else { /* Trường hợp 3 */
   x->parent->colour = black;
   x->parent->parent->colour = red;
   right_rotate( T, x->parent->parent );
   }
```

Cây đỏ đen – Phép toán thêm vào(16)

```
while ( (x != T->root) && (x->parent->colour == red) ) {
  if ( x->parent == x->parent->parent->left ) {
  /* If x's parent is a left, y is x's right 'uncle' */
  y = x->parent->parent->right;
  if ( y->colour == red ) {
        /* case 1 - change the colours */
        x->parent->colour = black;
        y->
        x->
        /*
        x = x->parent->parent;
     else {
     /* y is a black node */
     if ( x == x->parent->right ) {
           /* and x is to the right */
           /* case 2 - move x up and rotate */
           x = x->parent;
           left_rotate( T, x );
```

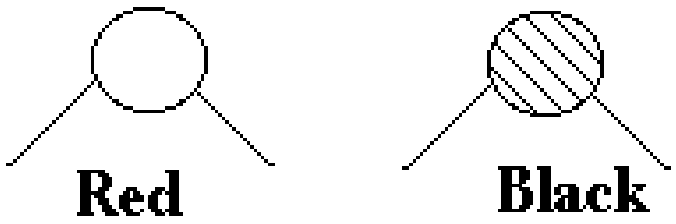
Kết thúc việc thêm vào cây




```
   else { /* Trường hợp 3 */
      x->parent->colour = black;
      x->parent->parent->colour = red;
      right_rotate( T, x->parent->parent );
      }
```

Phép toán loại bỏ trên Cây đỏ đen(1)

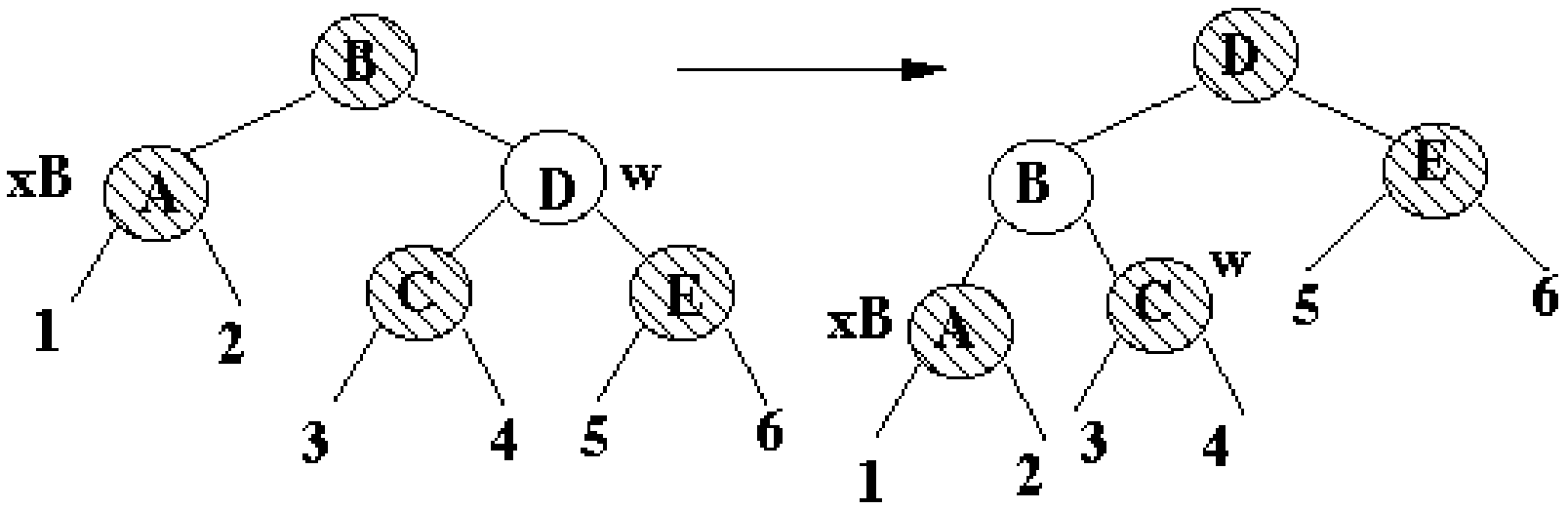
1. **Tìm nút để xóa.**
2. **Xoá nút đó như trên cây BST.**
3. **Nút bị xóa sẽ tối đa 1 nút con.**
4. **Nếu nút bị xoá màu đỏ thì cây vẫn là cây đỏ đen.**
5. **Nếu nút bị xóa màu đen và là nút gốc thì dừng.**
6. **Nếu nút bị xóa màu đen và không là nút gốc thì:**
    1. **Gọi x là nút con của nút bị xóa**
    2. **Nếu nút x màu đỏ thì chuyển thành màu đen và dừng**
    3. **Nếu nút x màu đen thì cân bằng lại cây đỏ đen:**



Phép toán loại bỏ trên Cây đỏ đen(2)

Trường hợp 1:

Nút anh em w của x màu đỏ:



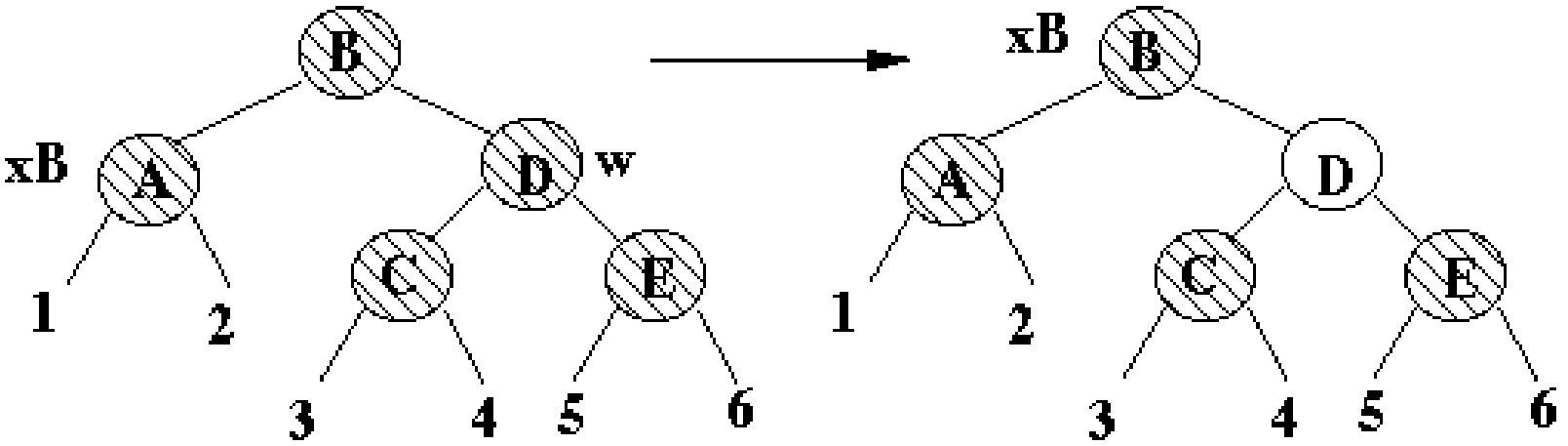
-> trường hợp 2b

Phép toán loại bỏ trên Cây đỏ đen(3)

Trường hợp 2a:

Nút anh em w màu đen

Nút cha mẹ màu đen



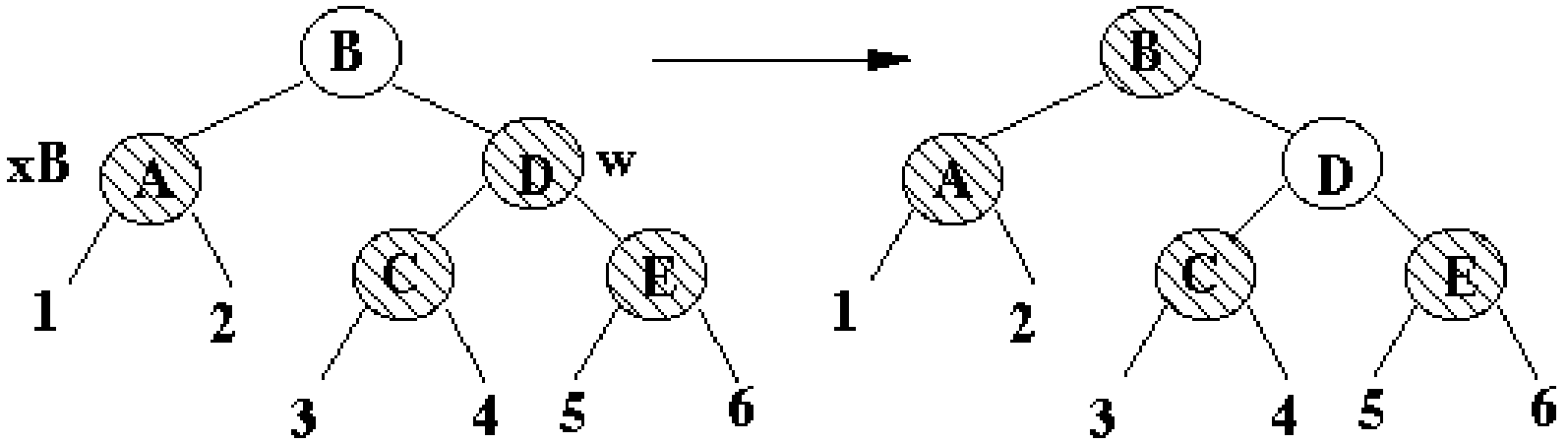
Giảm chiều cao đen của nút x một

Phép toán loại bỏ trên Cây đỏ đen(4)

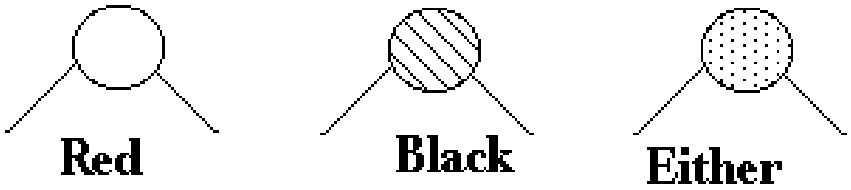
Trường hợp 2b:

Nút anh em w màu đen

Nút cha mẹ màu đỏ



Phép toán loại bỏ trên Cây đỏ đen(5)

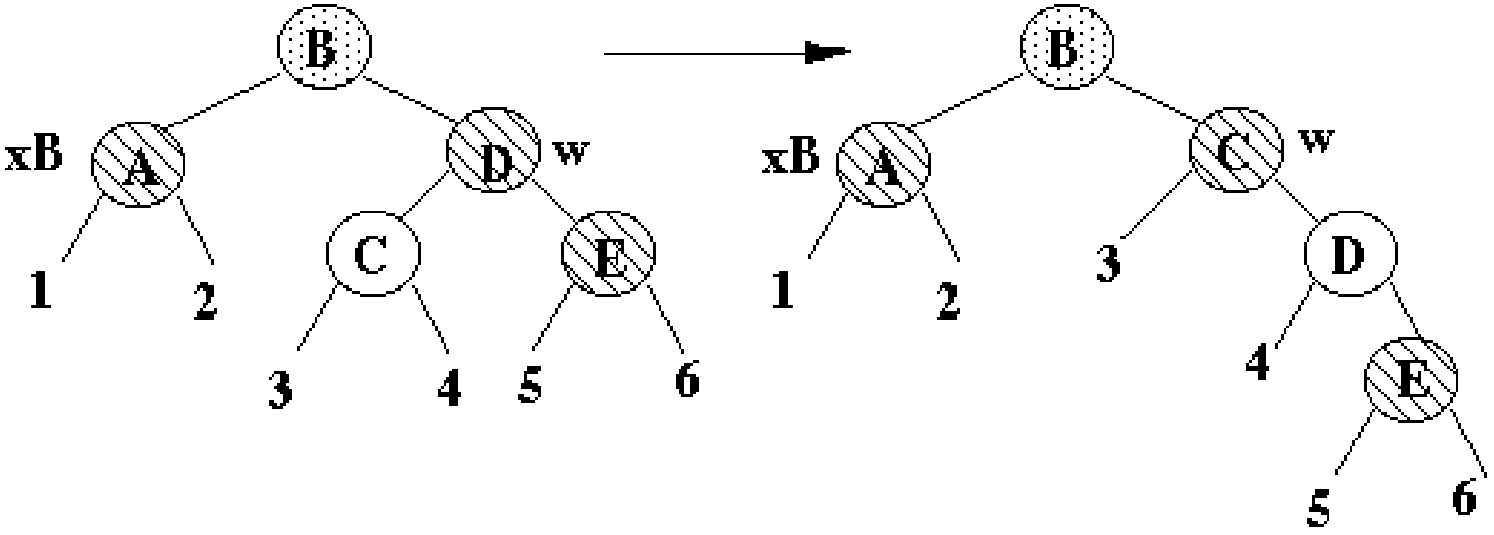


Trường hợp 3:

Nút anh em w màu đen

Nút con trái của nút anh em màu đỏ

Nút con phải của nút anh em màu đen



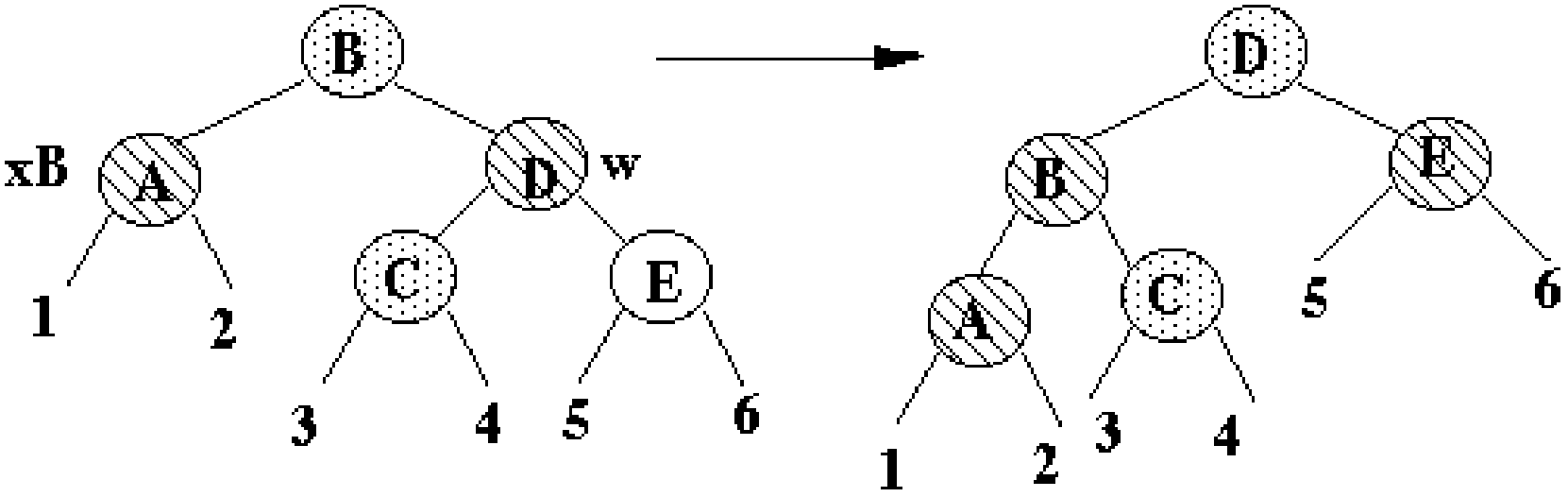
-> Trường hợp 4

Phép toán loại bỏ trên Cây đỏ đen(6)

Trường hợp 4:

Nút anh em w màu đen

Nút con phải của nút anh em màu đỏ



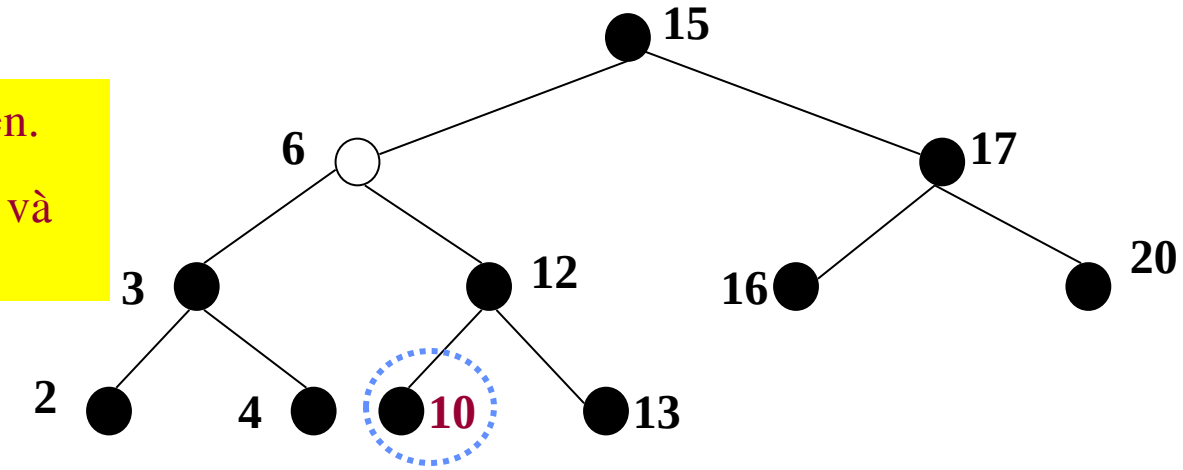
Phép toán loại bỏ trên Cây đỏ đen(7)

- **Xóa thì mất thời gian O(lg(n))**
- **Số lần xoay tối đa là 3**

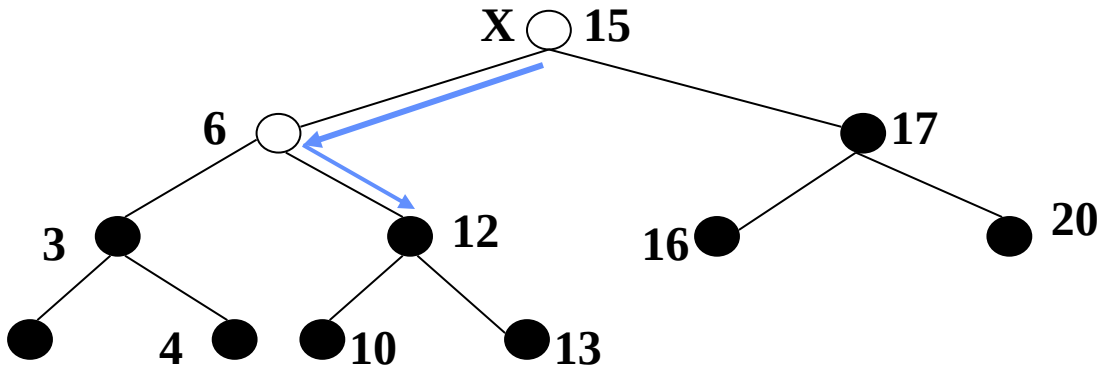
Ví dụ- Xoá 10

Bước 1 – Root không có 2 con đen.

Đổi màu root là đỏ , đặt X = root và xữ lý đến *bước 2*

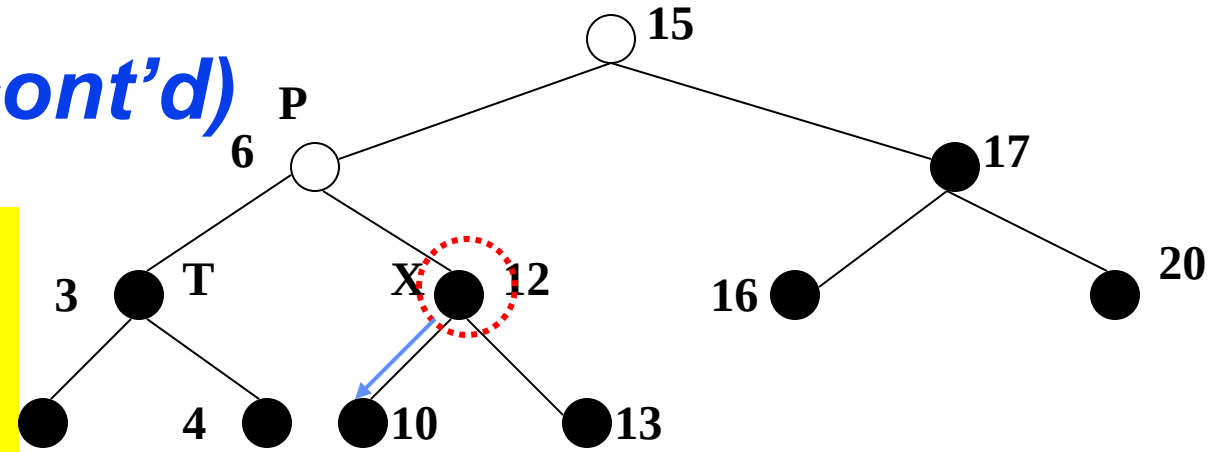


X có ít nhất là một con đỏ (TH 2B). Thao tác đi xuống đến 6. Lúc này 6 là đỏ(TH 2B1), tiếp tục đi xuống đến 12.



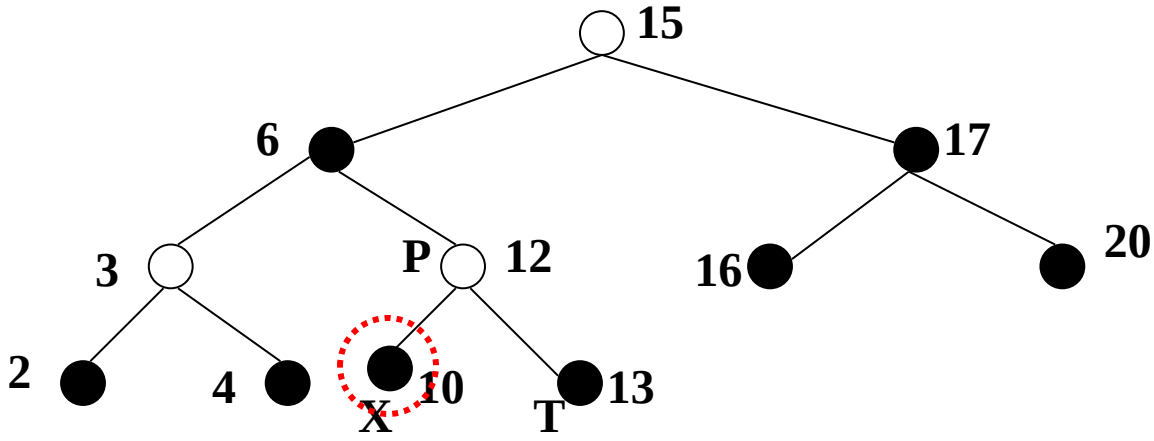
Example 2 (cont'd)

X có 2 con đen, anh em của X (3) cũng có 2 con đen.
TH 2A1– đổi màu X, P và T, tiếp tục thao tác đi xuống đến 10.

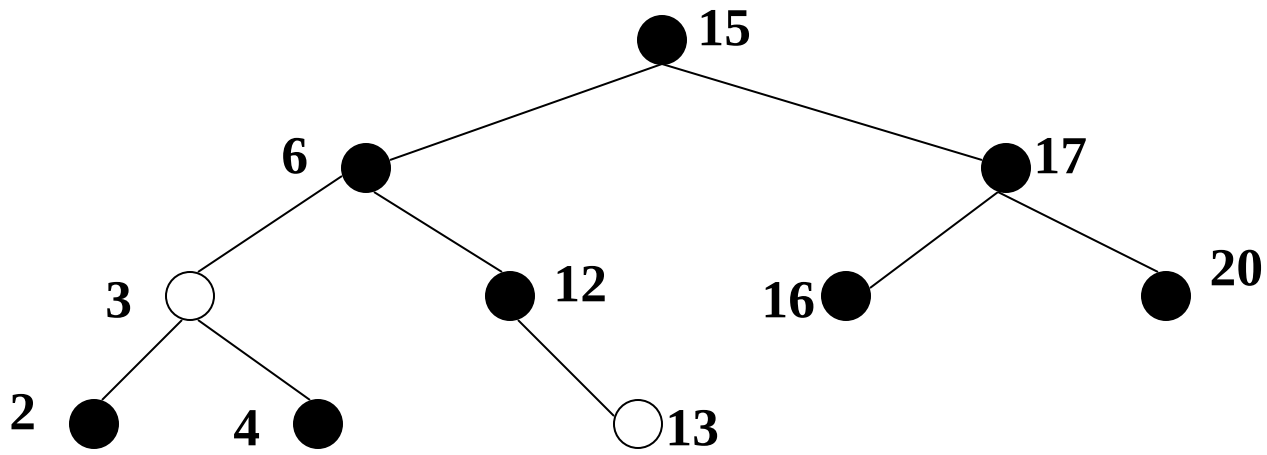


Bây giờ X là node lá được xóa, nhưng nó là đen, quay lại bước 2.
X có 2 con đen và T có 2 con đen(TH 2A1)

Đổi màu X, P và T.
Bước 3 – Bây giờ xoá 10 như là lá.
Bước 4 – Đổi màu Root là đen



Cây sau khi loại bỏ 10



Hiệu quả của cây đỏ đen

- **Giống như cây BST, cây đỏ đen có thể cho phép việc tìm kiếm, chèn và xóa trong thời gian O(log2N). Thời gian tìm kiếm là gần như bằng nhau đối với hai loại cây.**
- **Điều bất lợi duy nhất là việc lưu trữ thêm thuộc tính màu và liên kết đến node cha của một node trên cây.**
- **Theo Sedgewick, trong thực tế tìm kiếm trên cây đỏ đen mất khoảng log2N phép so sánh, và có thể chứng minh rằng nó không lớn hơn 2*log2N phép so sánh.**
- **Thời gian chèn và xóa tăng dần bởi một hằng số vì việc phải thực thi phép lật màu và quay trên đường đi xuống và tại những điểm chèn. Trung bình một phép chèn cần khoảng chừng một phép quay. Do đó, chậm hơn phép chèn trong cây BST.**
- **Bởi vì trong hầu hết các ứng dụng, có nhiều thao tác tìm kiếm hơn là chèn và xóa, có lẽ không có nhiều bất lợi về thời gian khi dùng cây đỏ đen thay vì cây nhị phân thường. Dĩ nhiên, điều thuận lợi là trong cây đỏ đen, dữ liệu đã sắp xếp không làm giảm hiệu suất O(N).**

Red-black trees - Analysis

- **Addition**
  - **Insertion** **Comparisons** $O(\log n)$
  - **Fix-up**
    - **At every stage,**
      x moves up the tree
      at least one level $O(\log n)$
  - **Overall** $O(\log n)$
- **Deletion**
  - **Also** $O(\log n)$
- **More complex**
- **... *but* gives $O(\log n)$ behaviour in dynamic cases**