CHƯƠNG 3 – CÂY ĐỎ ĐEN

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.

Giải pháp cho vấn đề này là cây cân bằng AVL. Thời gian tìm kiếm trong cây AVL là O(logN) vì cây là được bảo đảm cân bằng. Tuy nhiên vì phải đi qua cây hai lần để chèn hay xóa một nút, một lần đi xuống để tìm điểm chèn và một lần đi lên để tái cân bằng cây, vì vậy cây AVL là cây đỏ đen không hiệu quả và không thường được sử dụng .

CHƯƠNG 3 – CÂY ĐỎ ĐEN

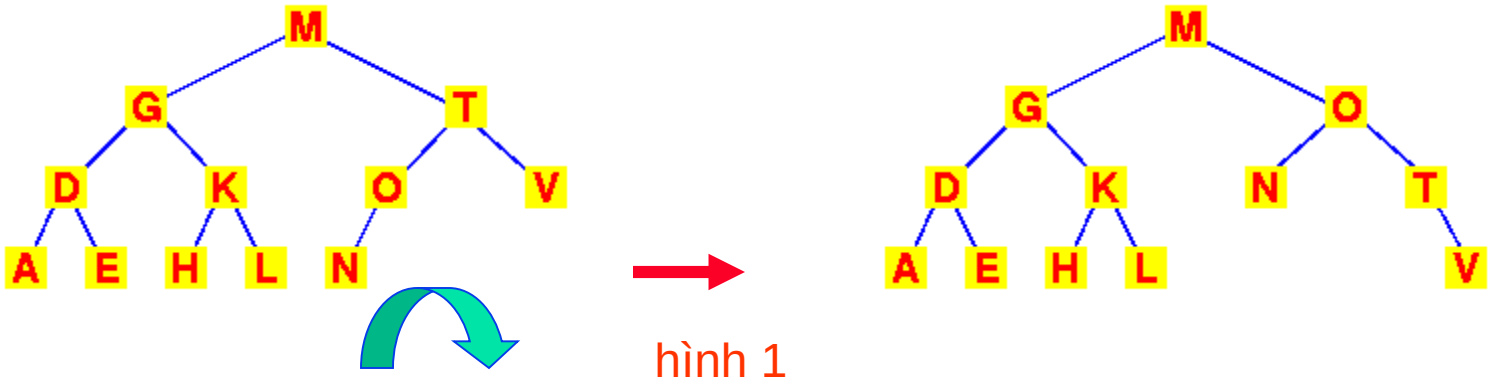
Cây tìm kiếm nhị phân thông thường có những thuận lợi lớn về mặt lưu trữ và truy xuất dữ liệu trong phép toán tìm kiếm thêm vào hay loại bỏ một phần tử. Do đó, cây tìm kiếm nhị phân xem ra là một cấu trúc lưu trữ dữ liệu tốt.

Tuy nhiên trong một số trường hợp cây tìm kiếm nhị phân có một số hạn chế. Nó hoạt động tốt nếu dữ liệu được chèn vào cây theo thứ tự ngẫu nhiên. Tuy nhiên, nếu dữ liệu được chèn vào theo thứ tự đã được sắp xếp sẽ không hiệu quả. Khi các trị số cần chèn đã được sắp xếp thì cây nhị phân trở nên không cân bằng. Khi cây không cân bằng, nó mất đi khả năng tìm kiếm nhanh (hoặc chèn hoặc xóa) một phần tử đã cho.

Phép quay tên 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:

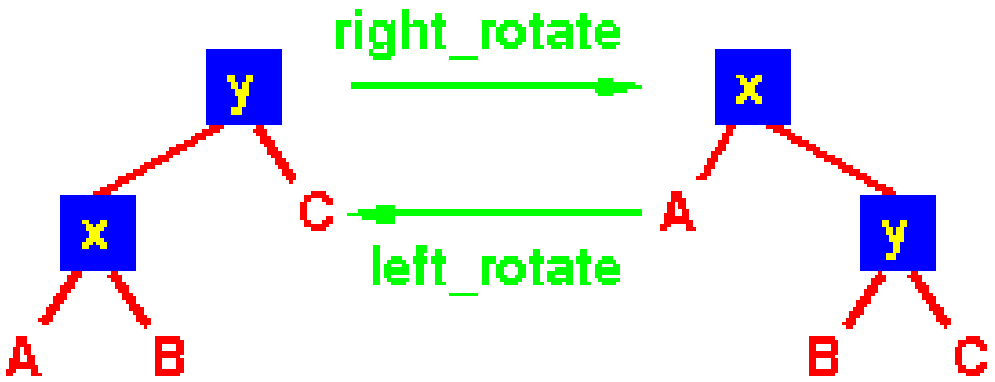


hình 1

Thực hiên phép quay cây con quanh T và O .

Phép quay tên cây BST

- Quay trái và quay phải (left-rotations or right-rotations)

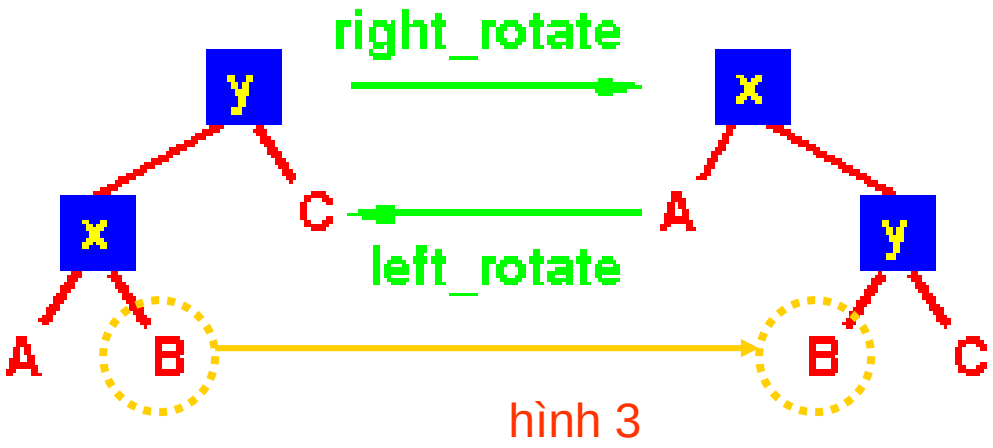


hình 2

- Kết qủa của 2 phép quay trên thứ tự trong phép duyệt cây không bị thay đổi là: A x B y C

Phép quay tên cây BST

- Quay trái và quay phải



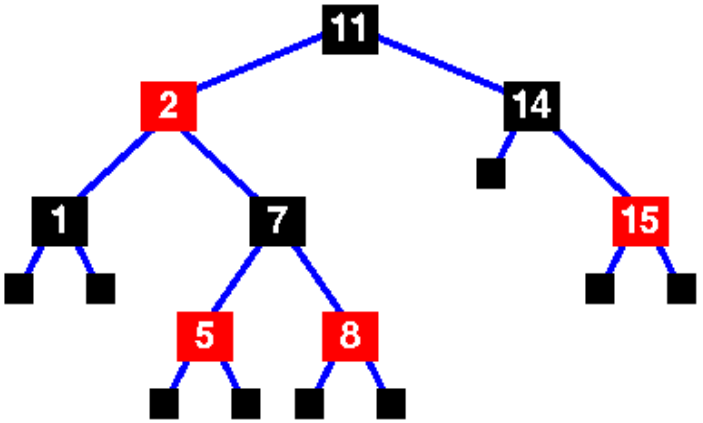
hình 3

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

Cây đỏ đen-Giới thiệu

Một cây BST là cây đỏ đen nếu:

1. Mỗi một nút của cây là đỏ hoặc đen.
2. Tất cả các nút lá là đen.
3. Nếu một nút là đỏ thì cả hai nút con là đen.
4. Đường đi từ nút gôc đến nút lá có cùng số nút đen.
5. Nút gôc có màu đen



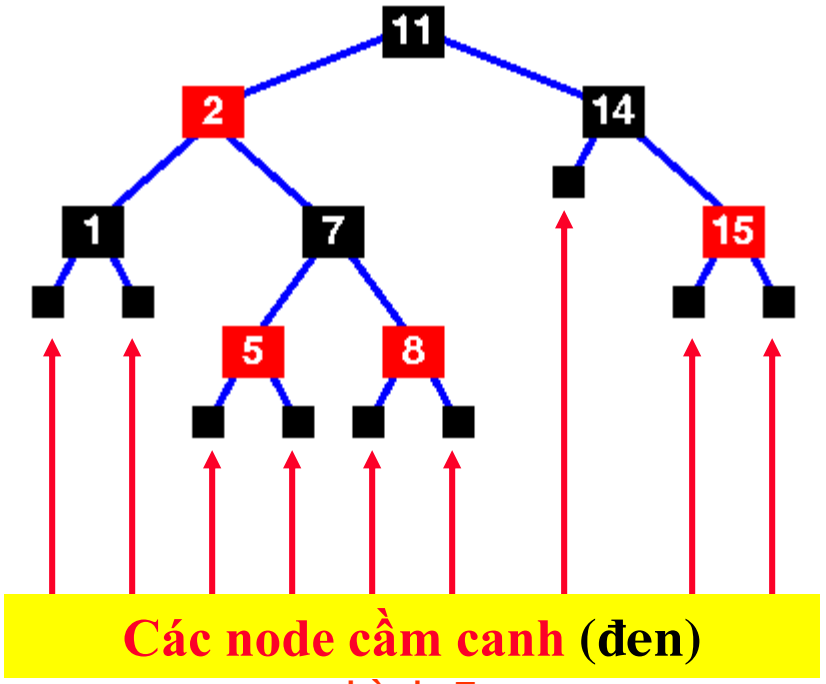
hình 4

Cây đỏ đen-Giới thiệu

Cây đỏ đen là cây BST có:

- Mỗi một node hoặc là node đỏ hoặc node đen
- Tất cả các node lá là node đen

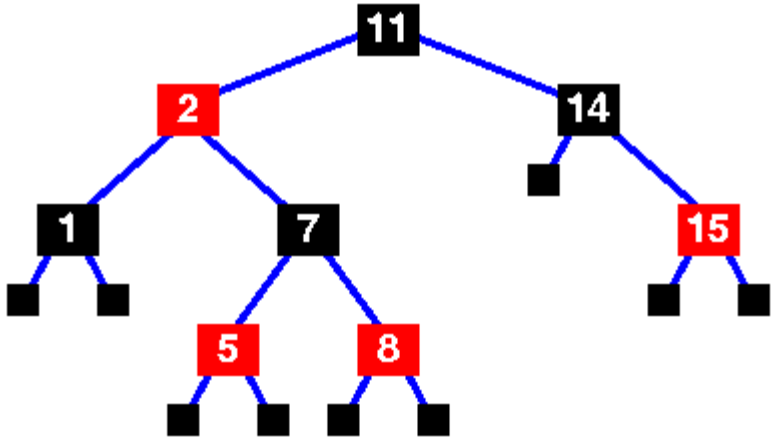
Các node lá không chứa dữ liệu-goi là node cầm canh (đen)



hình 5

Cây đỏ đen-Giới thiệu

- **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**



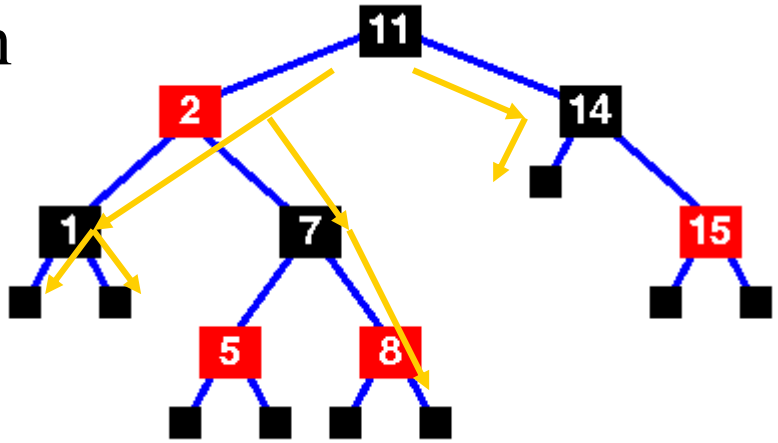
hình 6

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

Cây đỏ đen-Giới thiệu

Cây đỏ đen là cây BST có:

- 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 đế
no



hình 7

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)

Cây đỏ đen-Giới thiệu

Bổ đề

Một cây đỏ đen n- node có

height <= 2 log*(n+1)*

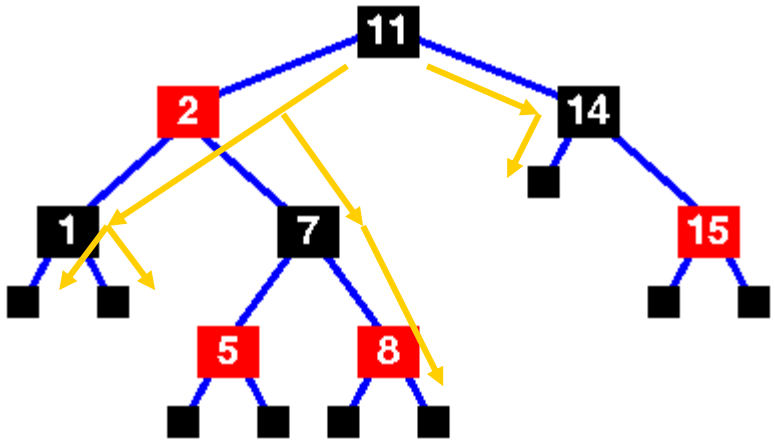
height- Chiều cao của cây

Tính chất:,

height <= 2 * bh(x)

Thời gian tìm kiếm:

O(log *n)*



hình 8

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

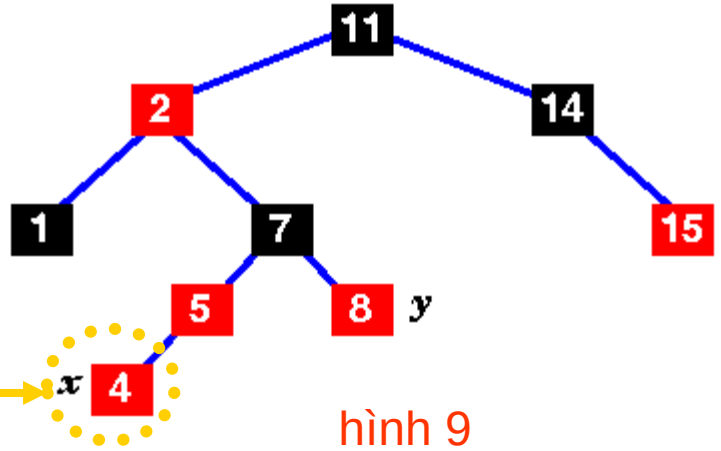
Cây đỏ đen-Phép toán thêm vào

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;
   while ( (x != T->root) && (x->parent-
      if ( x->parent == x->parent->parent
          /*
          y
          if ( y->colour == red ) {
              /* case 1 - change the colours */
              x->parent->colour = black;
              y->colour = black;
              x->parent->parent->colour = red;
              /* Move x up the tree */
              x = x->parent->parent;
```

Node thêm vào là 4-màu đỏ



hình 9

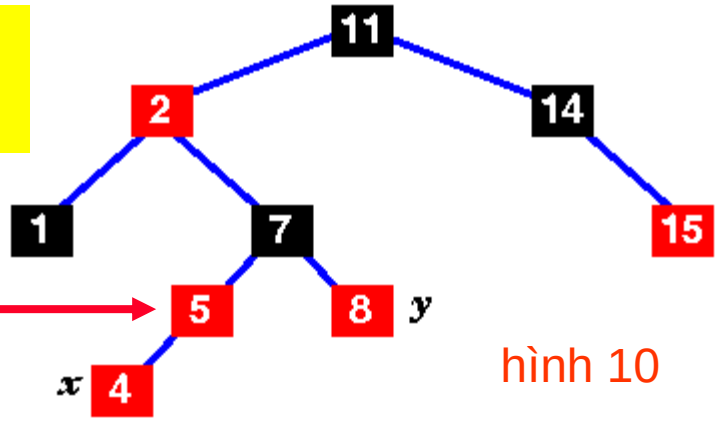
Cây đỏ đen-Phép toán thêm vào *(tt)*

```
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) )
```

Trong khi X không phải là gốc và cha của X là màu đỏ

```
        if ( y->colour == red )
            /* case 1 - change
            x->parent->colour =
            y->colour = black;
            x->parent->parent->
            /* ..... */
            x = x->parent->parent;
```

x->parent



hình 10

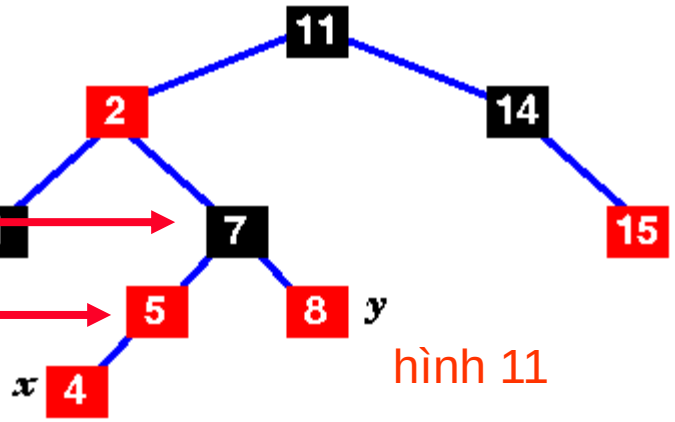
Cây đỏ đen-Phép toán thêm vào *(tt)*

```
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 cây con 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



hình 11

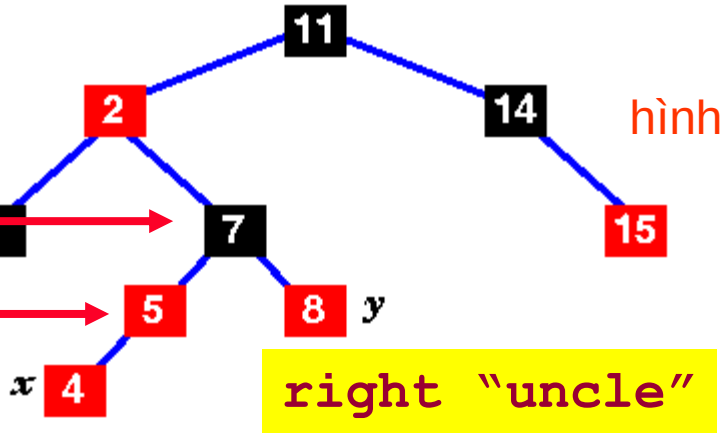
Cây đỏ đen-Phép toán thêm vào *(tt)*

```
/* Now restore the red-black property */
x->colour = red;
while ((x != T->root)&&(x->parent->colour == red))
{
    if ( x->parent == x->parent->parent->left ) {
        /* Y là node chú bác "uncle" bên phải của X */
        /* case 1 - change th
        x->parent->colour = b
        x->parent->parent->co
        /* e tree
        x = x->parent->parent
```

Y là node chú bác bên phải của X

x->parent->parent

x->parent

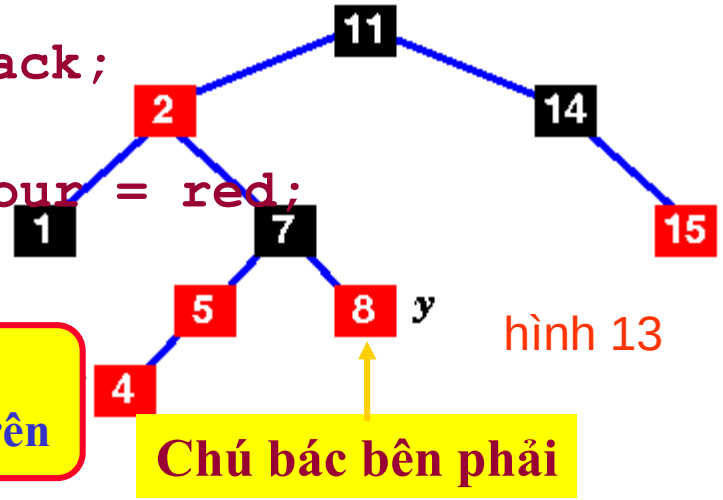


hình 12

right "uncle"

Cây đỏ đen-Phép toán thêm vào *(tt)*

```
    while ( (x != T->root) && (x->parent->colour == red)
) {
        if ( x->parent == x->parent->parent->left ) {
            /* Y là node chú bác "uncle" bên phải của X */
            y = x->parent->parent->right;
            if ( y->colour == red ) {
                /* case 1 - Đổi màu */
                x->parent->colour = black;
                y->colour = black;
                x->parent->parent->colour = red;
                x->parent->parent
```



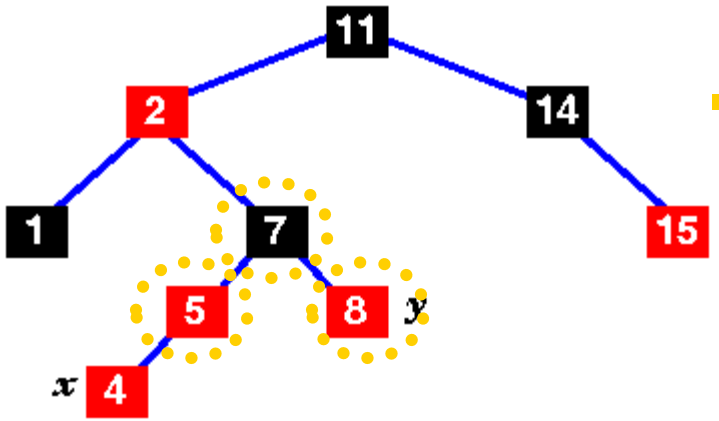
hình 13

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

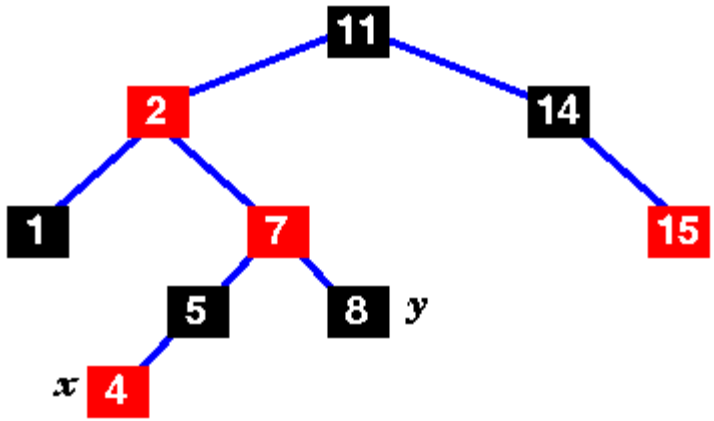
Cây đỏ đen-Phép toán thêm vào *(tt)*

```
while ( (x != T->root) && (x->parent->colour == red) )
{
    if ( x->parent == x->parent->parent->left ) {
        /* Y là node chú bác "uncle" bên phải của X */
        y = x->parent->parent->right;
        if ( y->colour == red ) {
            /* TH 1-Đổi màu X */
            x->parent->colour = black;
            y->colour = black;
            x->parent
            /* Chuyển
                                    >pare
```

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



hình 14



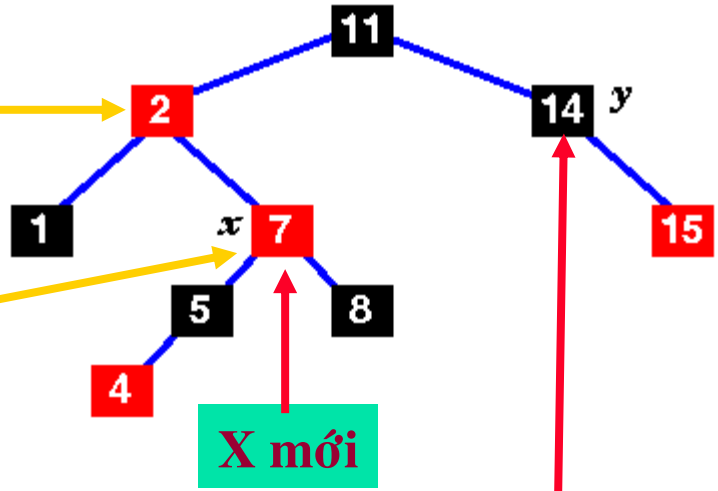
hình 15

Cây đỏ đen-Phép toán thêm vào *(tt)*

```
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 - Đổi màu */
      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

Chuyển X lên trên



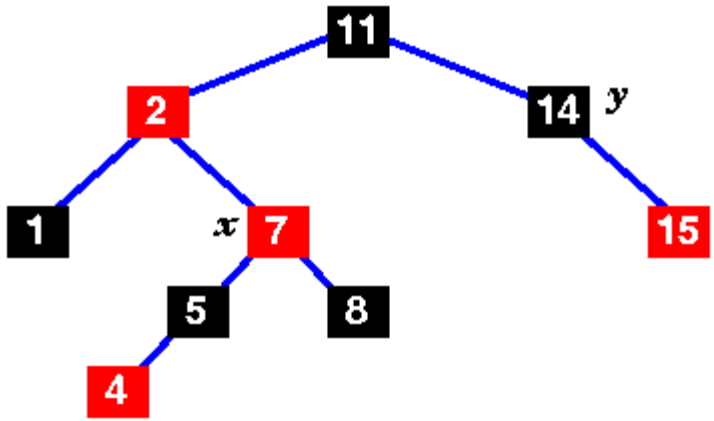
Y là chú bác của X
Bây giờ có màu là đen

hình 16

Cây đỏ đen-Phép toán thêm vào *(tt)*

```
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

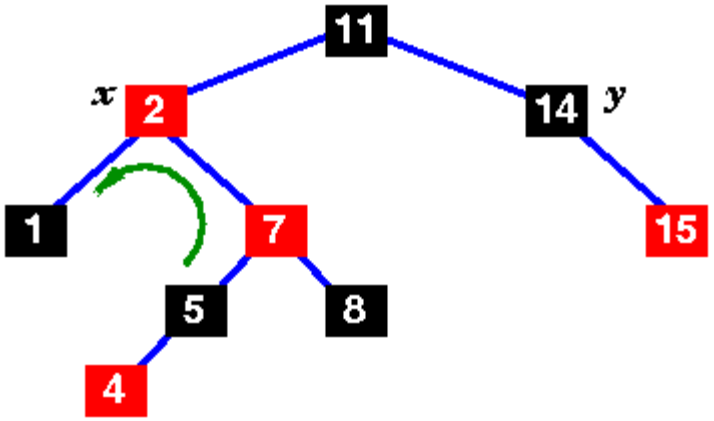


hình 17

Cây đỏ đen-Phép toán thêm vào *(tt)*

```
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 );
```

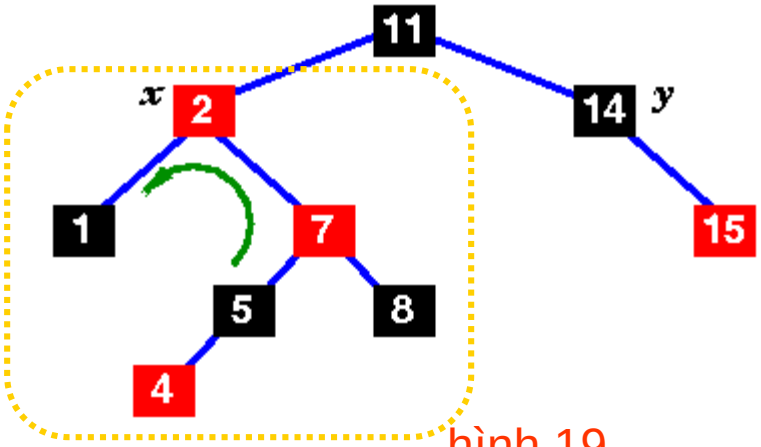
.. Di chuyển X lên trên
và thực hiện phép xoay trái...



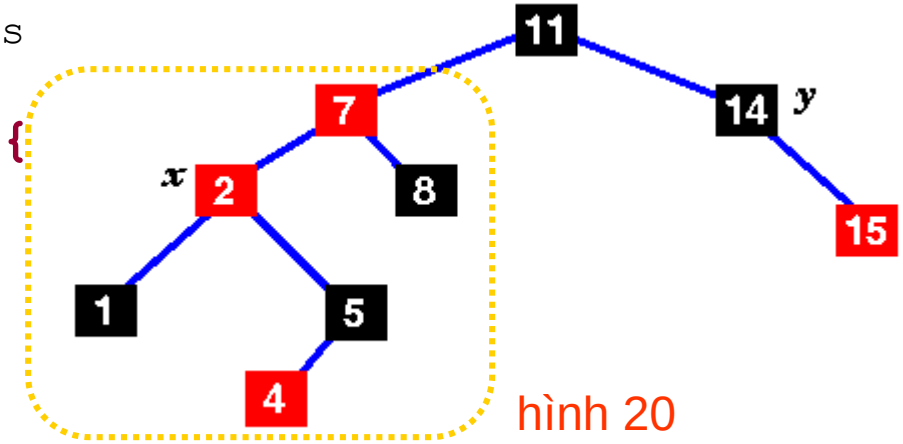
hình 18

Cây đỏ đen-Phép toán thêm vào *(tt)*

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



hình 19

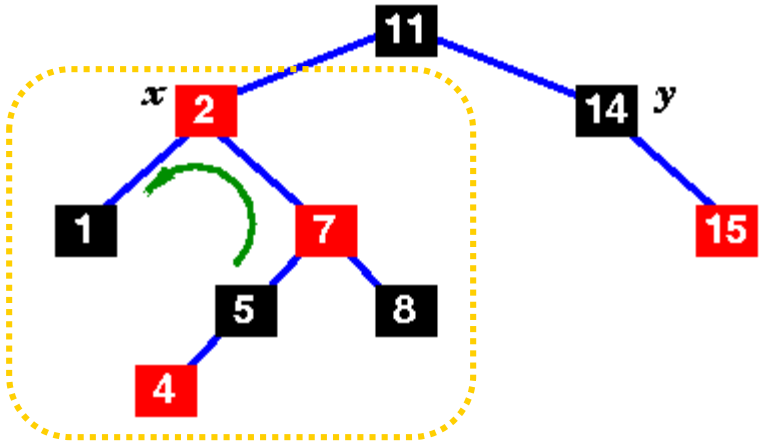


hình 20

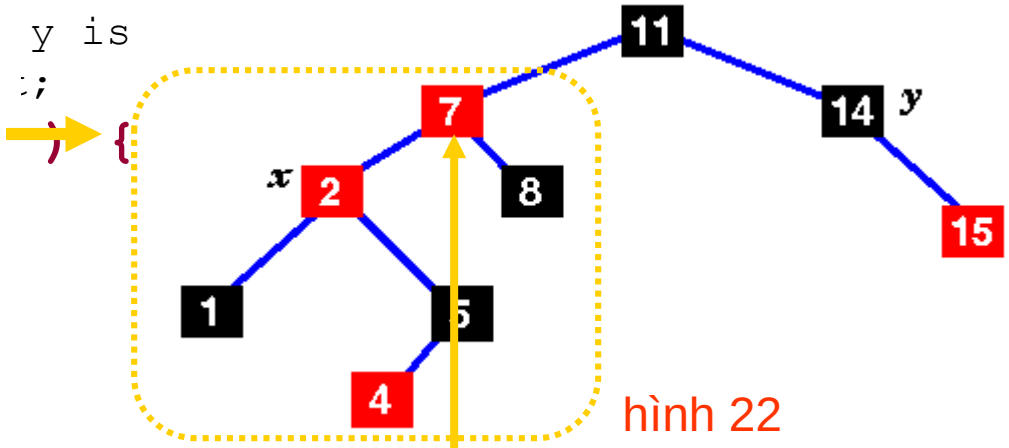
```
else {
        /* Y là node đen */
if ( x == x->parent->right ) {
  /* và x là node phải */
  /* case 2 – Chuyên X lên trên và xoay trái*/
  x = x->parent;
  left_rotate( T, x );
```

Cây đỏ đen-Phép toán thêm vào *(tt)*

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



hình 21



hình 22

```
        /* 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 );
```

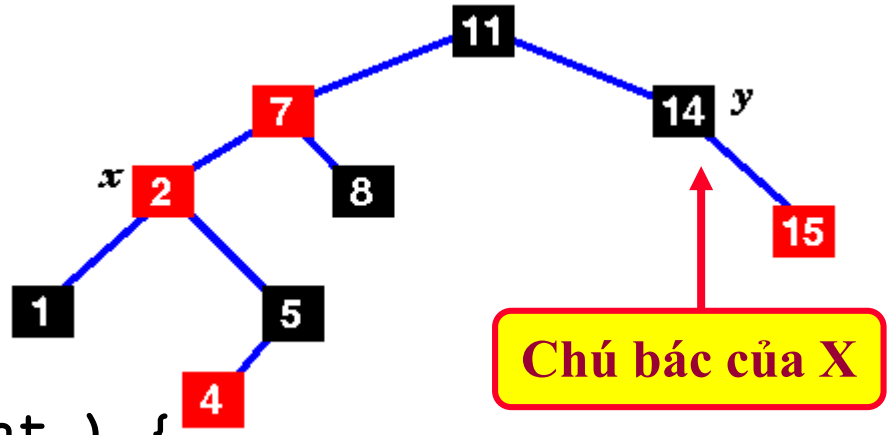
.. Bây giờ cha của X là đỏ ...

Cây đỏ đen-Phép toán thêm vào *(tt)*

```
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 -*/
                                    x;
                                    = red;
x = x->parent->parent;
   else {
     /* Y là node đen */
     if ( x == x->parent->right ) {
         /* and x is to the right */
         /* case 2 - move x up and rotate */
         x
         l
```

.. Chú bác của X là đen ..

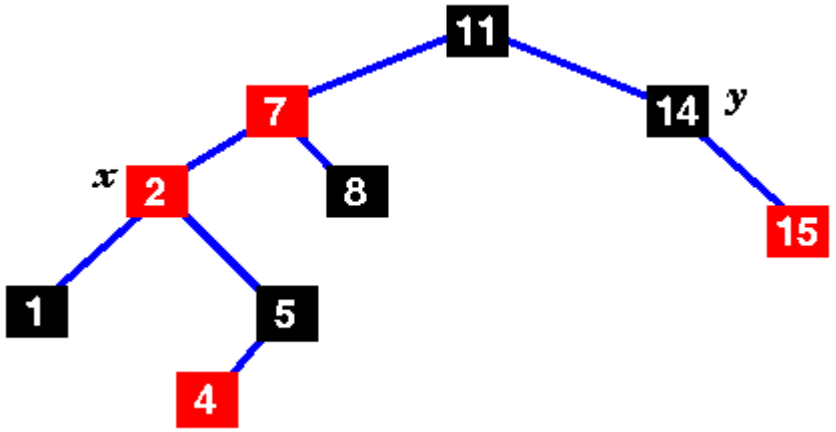
.. X là cây con bên trái..



hình 23

Cây đỏ đen-Phép toán thêm vào *(tt)*

```
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 );
```



hình 24

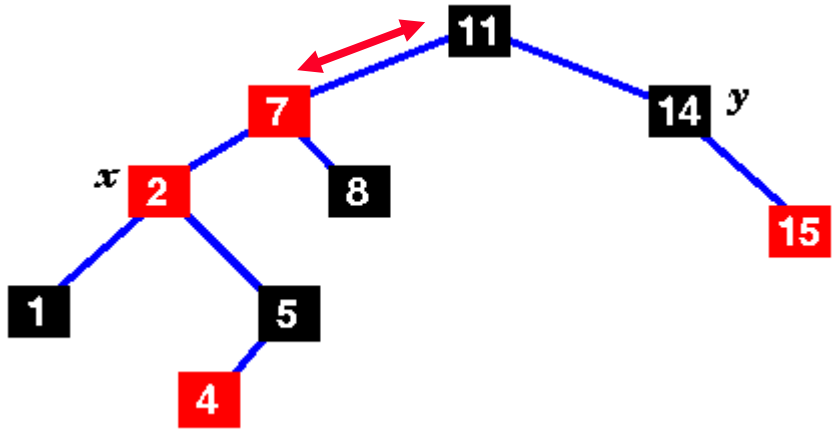
```
   else { /* trường hợp 3 */
     x->parent->colour = black;
     x->parent->parent->colour = red;
     right_rotate( T, x->parent->parent );
     }
```

Cây đỏ đen-Phép toán thêm vào *(tt)*

```
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 ch
      x = x->
   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 );
```

..Đổi màu và xoay..

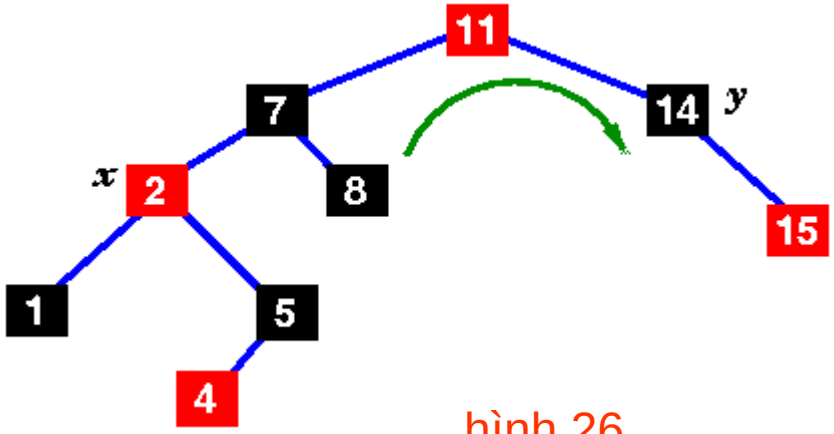


hình 25

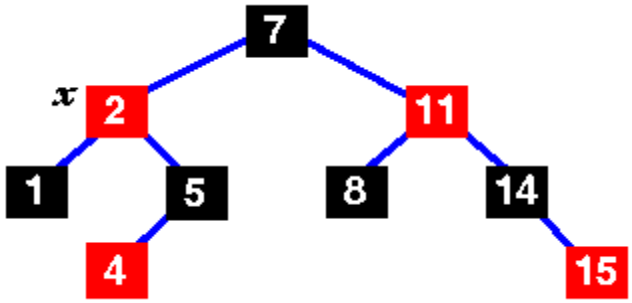
```
   else {  /* Trường hợp 3 */
      x->parent->colour = black;
      x->parent->parent->colour = red;
      right_rotate( T, x->parent->parent );
      }
```

Cây đỏ đen-Phép toán thêm vào *(tt)*

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



hình 26



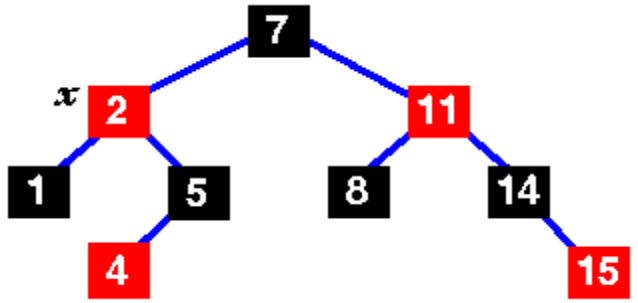
hình 27

```
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 *(tt)*

```
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



hình 28

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

Cây đỏ đen-Phép toán loại bỏ

Các thao tác chính

Thực hiện xoá trên cây BST

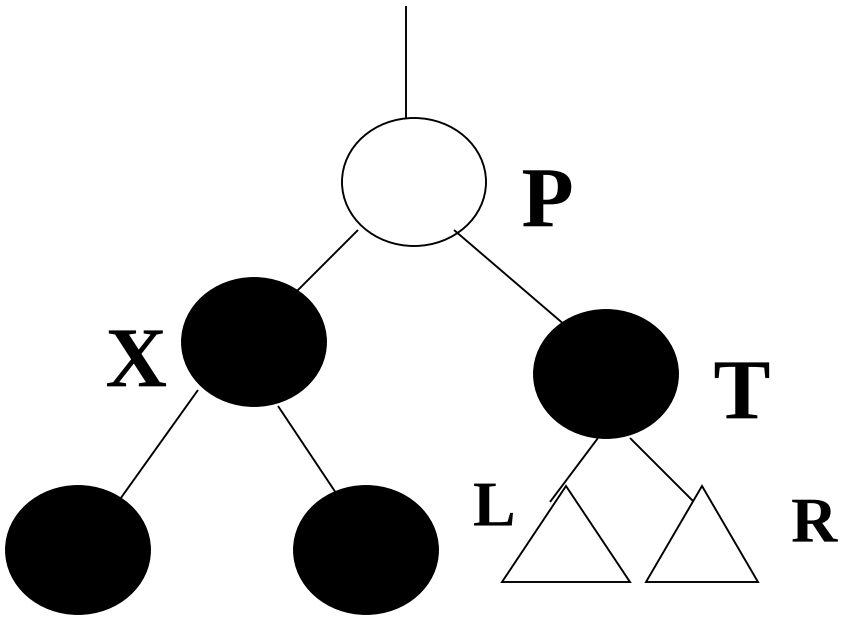
- *Nếu node xoá là lá thì xoá nó*
- *Nếu node xoá có một cây con thì thay nó bằng cây con của nó.*
- *Nếu node xoá có 2 cây con, Thay giá trị của nó bằng node thế mạng và gọi đệ quy.*

- Nếu node xoá là node đỏ, không ảnh hưởng đến cấu trúc cây đỏ đen
- Nếu node xoá là node đen
  - *Nếu node xoá không phải là node gốc, việc xoá nó có thể làm thay đổi chiều cao đen.*

Cây đỏ đen-Phép toán loại bỏ *(tt)*

Một số ký hiệu:

- X là node loại bỏ
- T là anh em của X
- P là cha của X
- R là con phải của T
- L là con trái của T

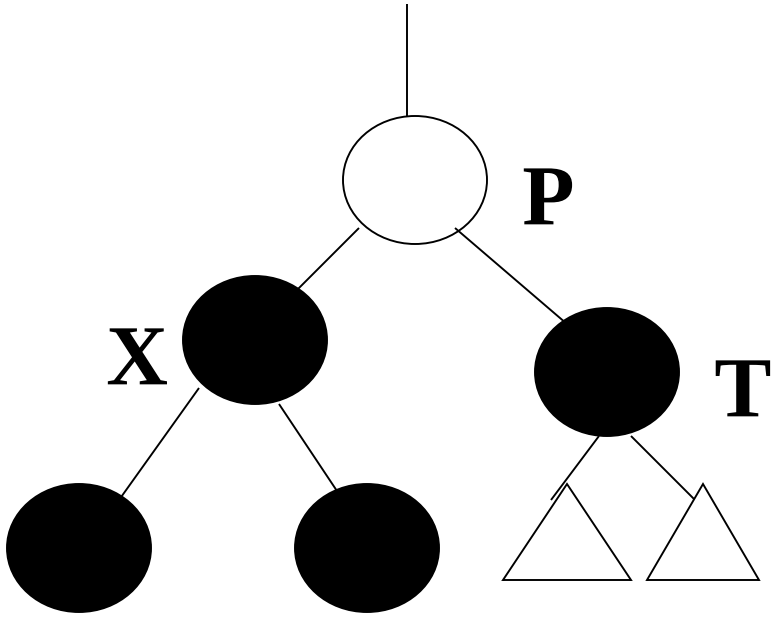


hình 29

Cây đỏ đen-Phép toán loại bỏ *(tt)*

Duyệt cây, đổi node X thành node đỏ, khi đó:

- P cũng là node đỏ
- T là đen (vì P là đỏ, con của nó phải là đen)



hình 30

Bước 1- Xem xét node gốc

Nếu cả 2 node con của root là đen:

- Đổi node root thành đỏ.
- Chuyển X đến node con thích hợp của root.
- Xử lý đến bước 2.

Ngược lại chọn root là X và xử lý đến bước 2B.

Bước 2 – Trường hợp chính

Quá trình này được tiếp tục cho đến khi node được xóa .

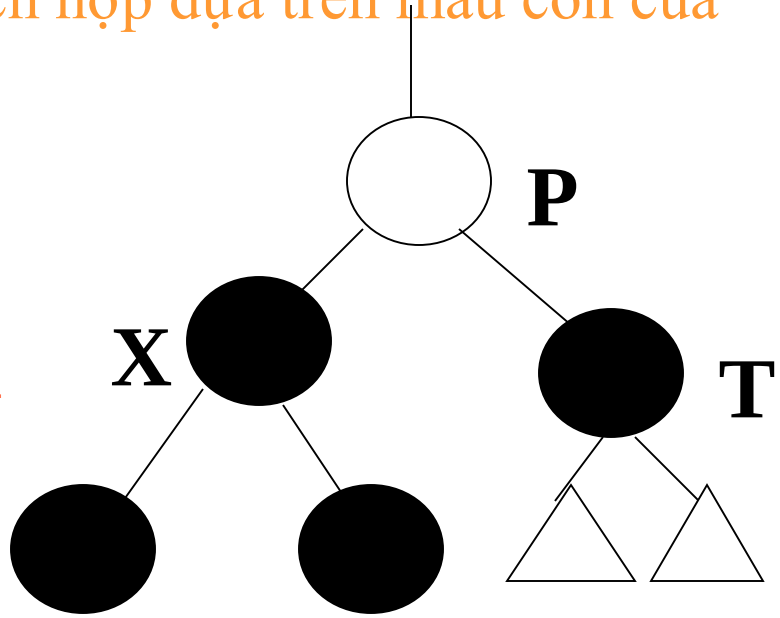
X là đen, P là đỏ, T là đen

Đổi màu X là Red, tiếp theo đổi màu các node khác và sử dụng các phép xoay thích hợp dựa trên màu con của X và T.

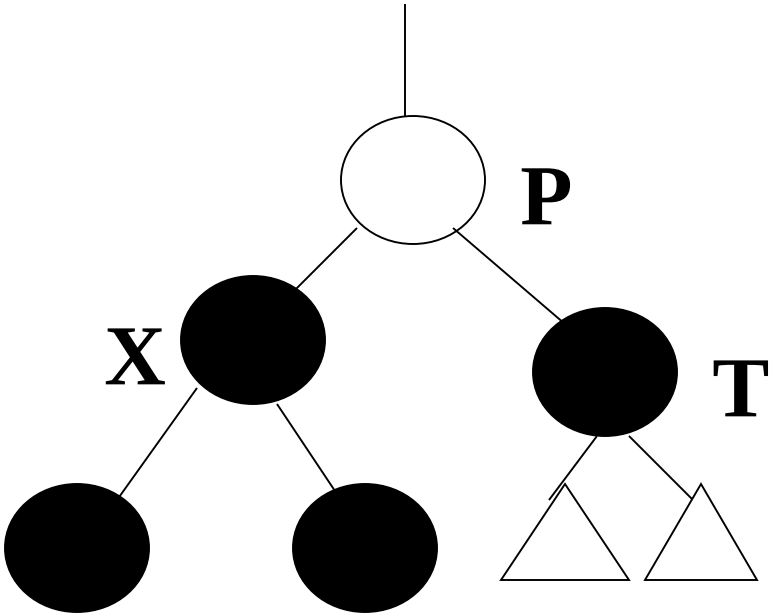
2A. X có 2 con đen.

2B. X có ít nhất một con đỏ.

hình 31



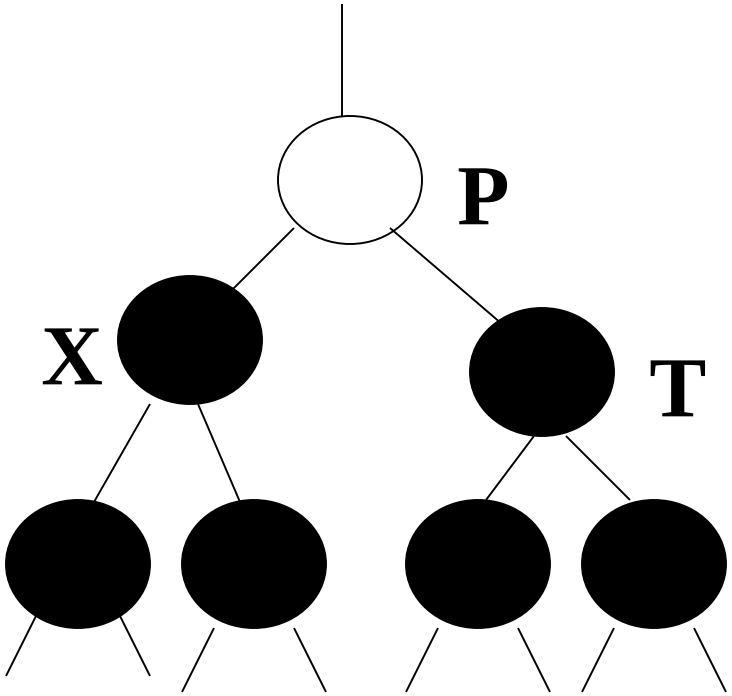
Trường hợp 2A-X có 2 cây con đen



hình 32

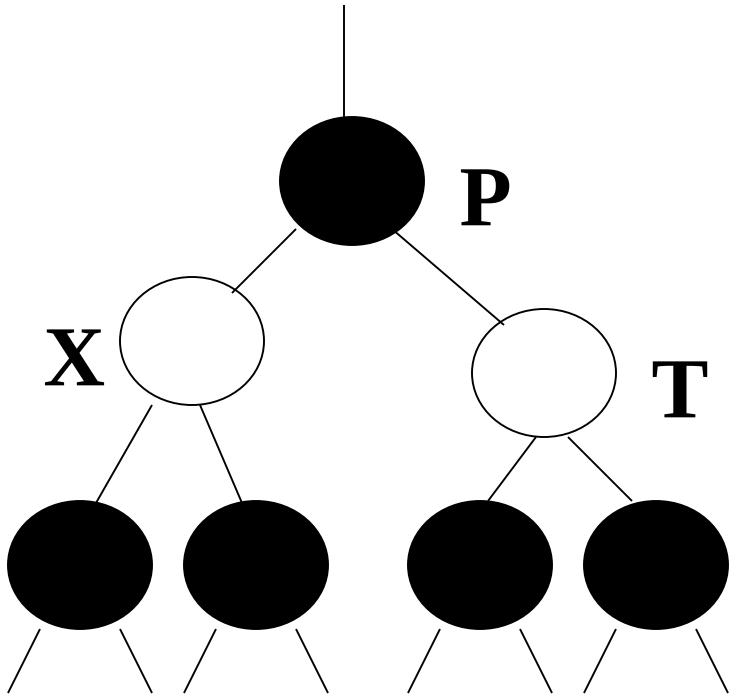
2A1. T có 2 con đen
2A2. T có con trái là đỏ
2A3. T có con phải là đỏ.
(nếu cả 2 con của T là đỏ có thể sử dụng 2A2 hoặc 2A3)

Trường hợp 2A1 - X và T có 2 con đen



hình 33

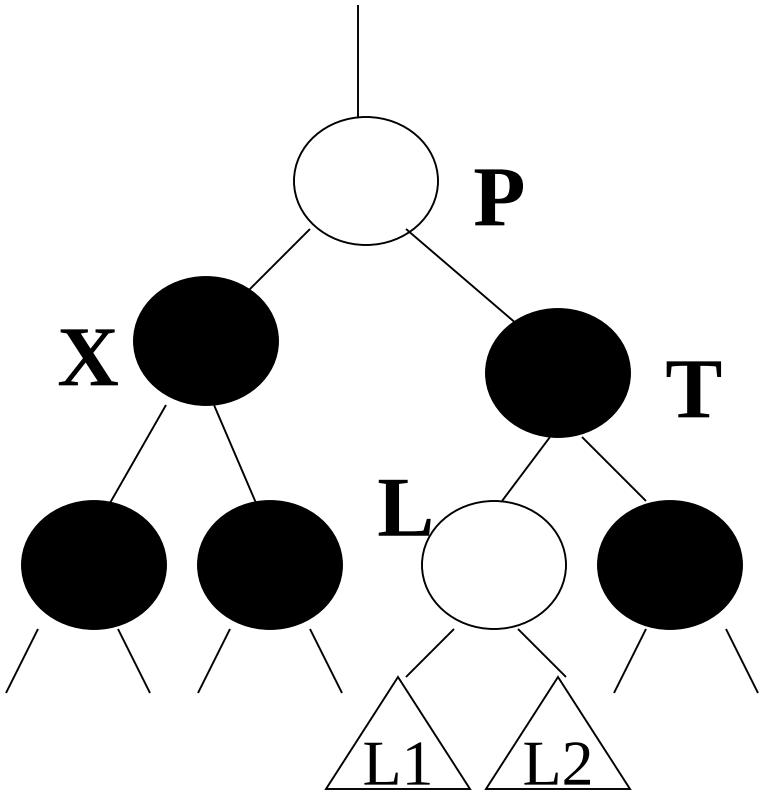
Đổi màu X , P và T



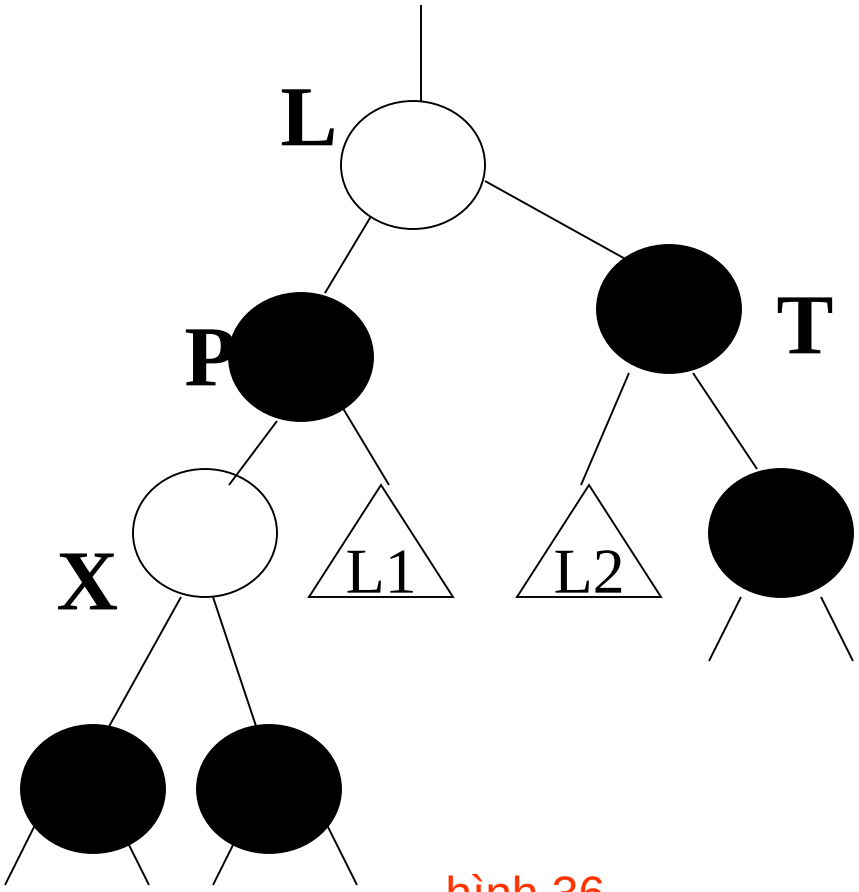
hình 34

Trường hợp 2A2

X có 2 cây con đen và con trái của T là đỏ
Thực hiện phép Xoay L quanh T, L quanh P
Đổi màu X và P sau đó quá trình tiếp tục đi xuống.



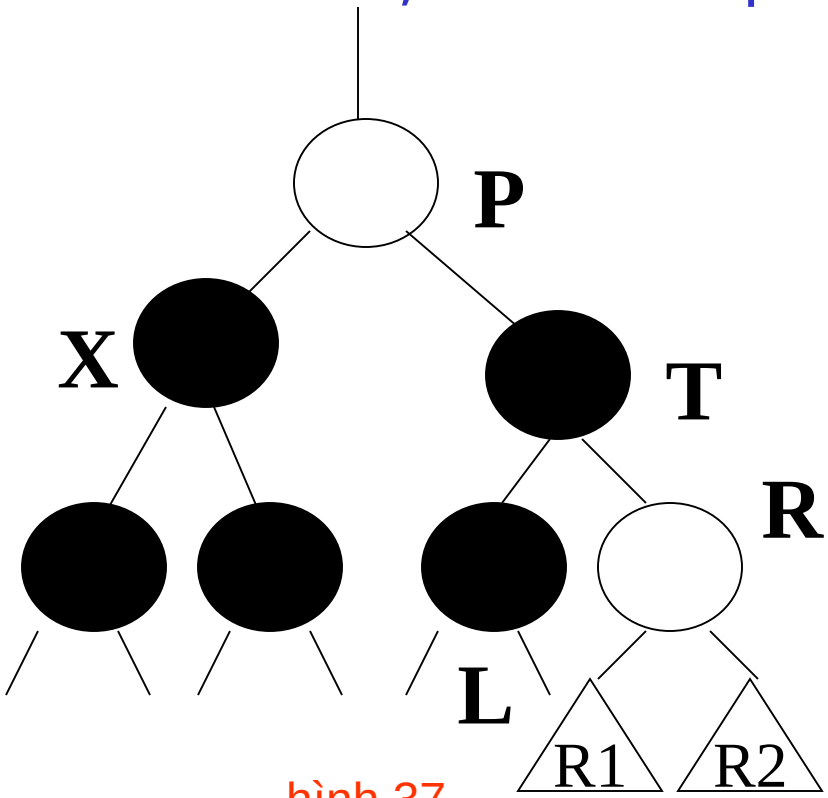
hình 35



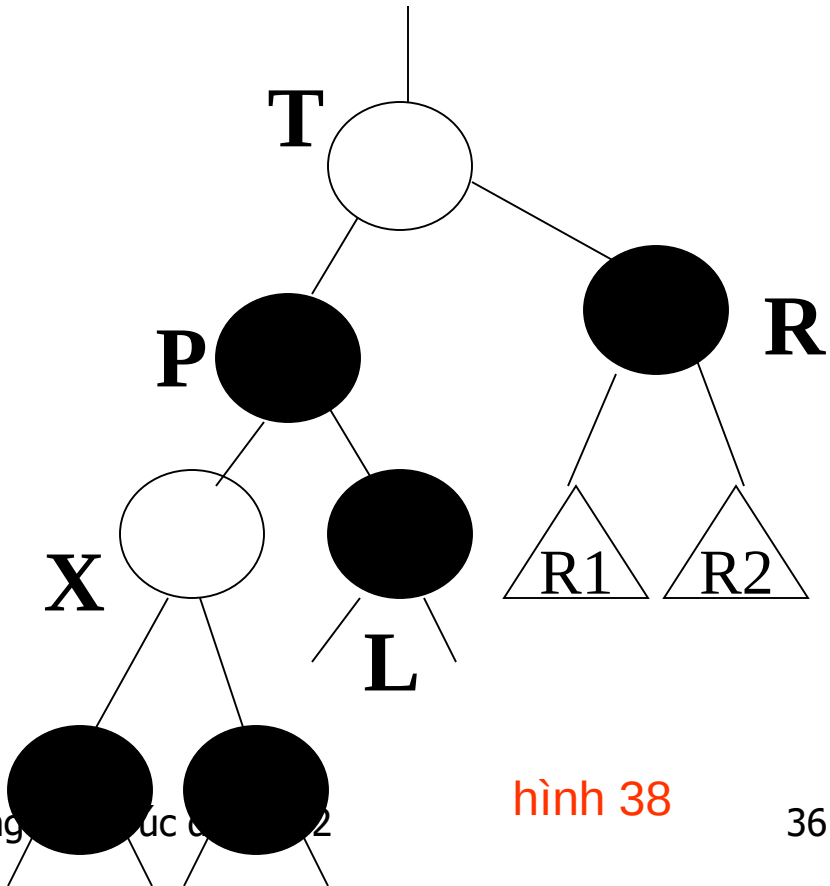
hình 36

Trường hợp 2A3

X có 2 con đen và T có con phải là đỏ:
Thực hiện phép quanh T quanh P.
Đổi màu X, P và T và quá trình tiếp tục đi xuống.



hình 37



hình 38

Trường hợp 2B .X có ít nhất một con là đỏ

Tiếp tục giảm chiều cao đen của cây đến mức kế tiếp.

Nếu X mới là đỏ tiếp tục quá trình này theo hướng đi xuống của cây.

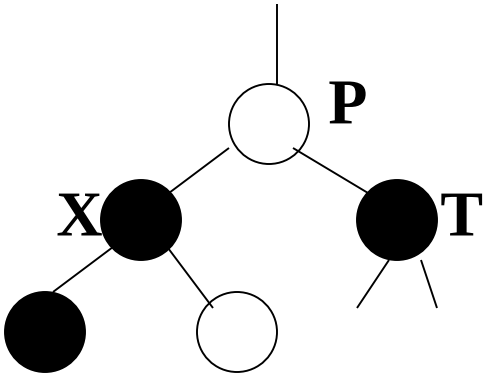
Nếu X mới là đen (T là đỏ, P là đen)

Quanh T quanh P.

Đổi màu P và T.

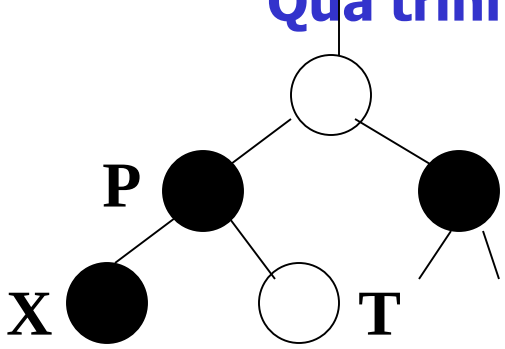
Quay trở lại bước 2 (Trường hợp chính)

Trường hợp 2B



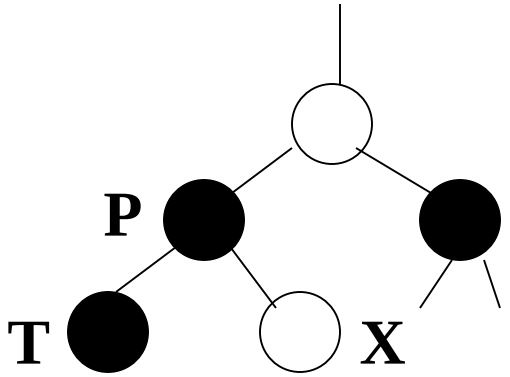
hình 39

Quá trình xữ lý theo hướng xuống bên dưới cây.



hình 40

Nếu đi xuống đến node con Đen(2B2): Quanh T quanh P, Đổi màu P và T, Quay lại bước 2



hình 41

Nếu đi xuống đến node con đỏ (2B1) Tiếp tục xữ lý đi xuống

Bước 3

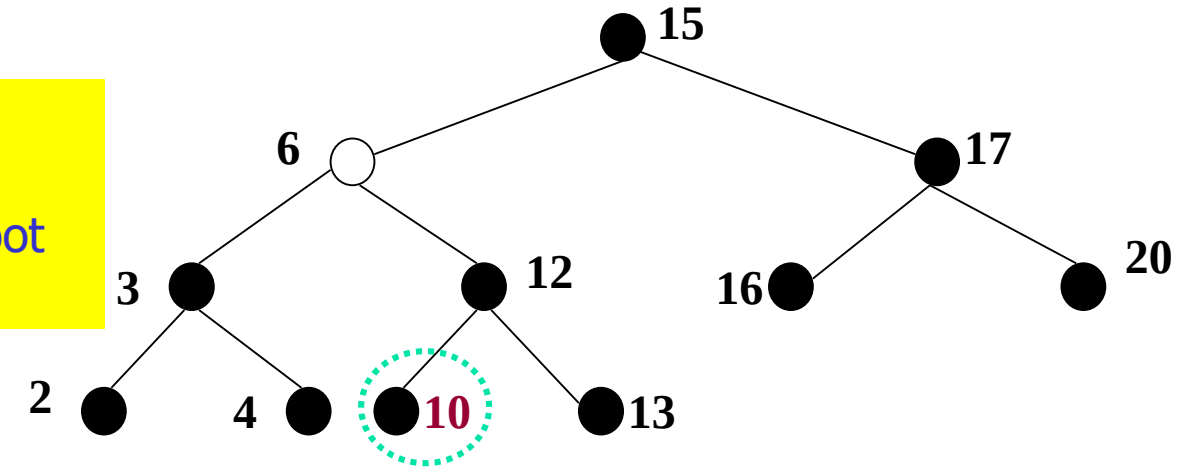
Cuối cùng tìm thấy node để xóa-là node lá hoặc là node có một node con khác NULL là node lá.
Xoá node thích hợp như là node lá đỏ.

Bước 4

Chuyển màu node gốc thành màu đen

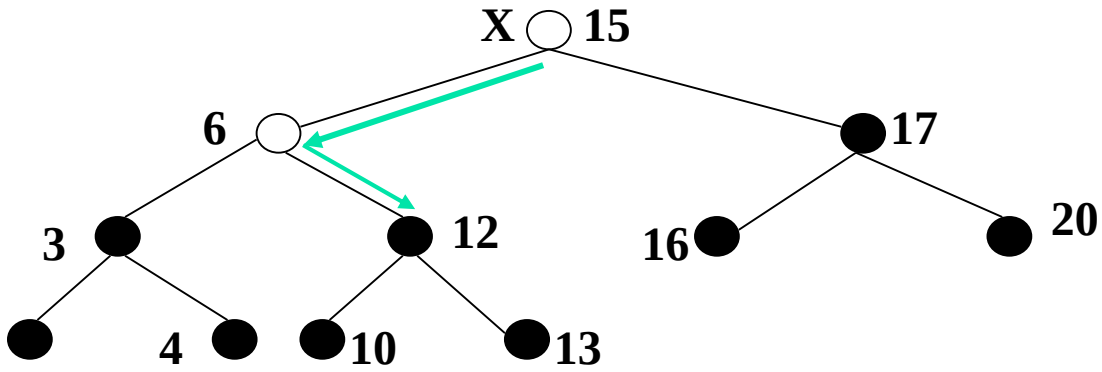
Ví dụ- xóa 10

Bước 1 – Root không có 2 con đen.
Đổi màu root là đỏ , đặt X = root và xữ lý đến *bước 2*



hình 41

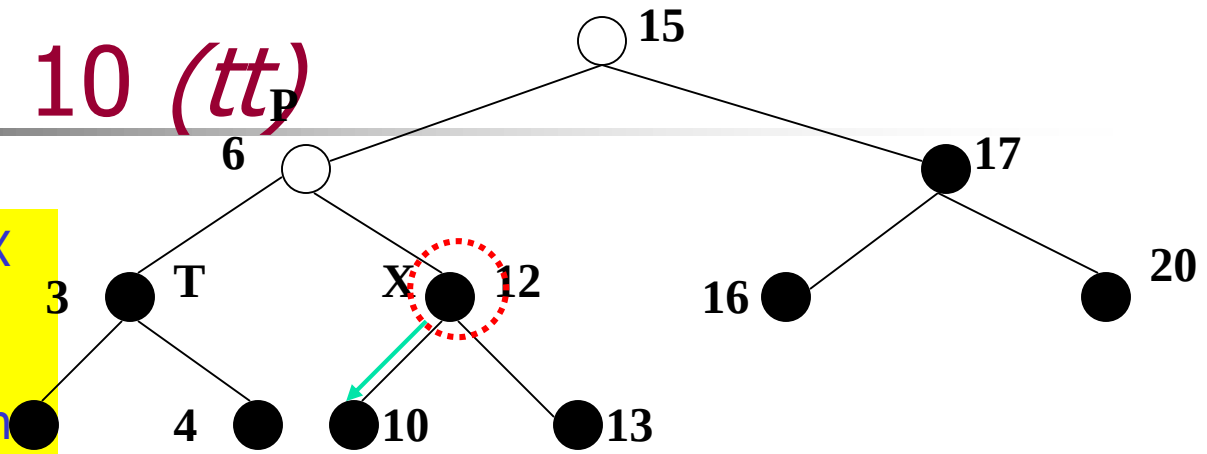
X có ít nhất là một con đỏ (TH 2B). Thao tác đi xuống đến 6. Lúc này 6 là đỏ(TH 2B1), tiếp tục đi xuống đến 12.



hình 42

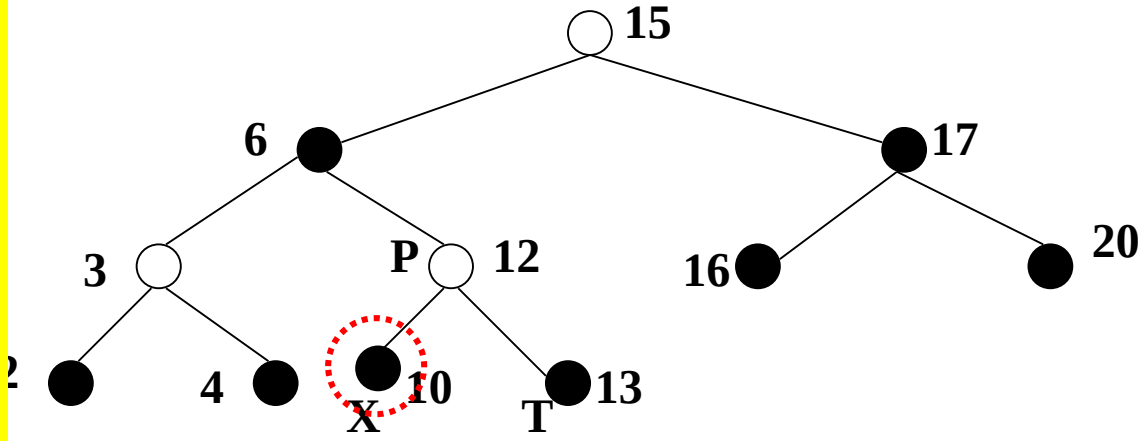
Ví dụ- xóa 10 *(tt)*

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.



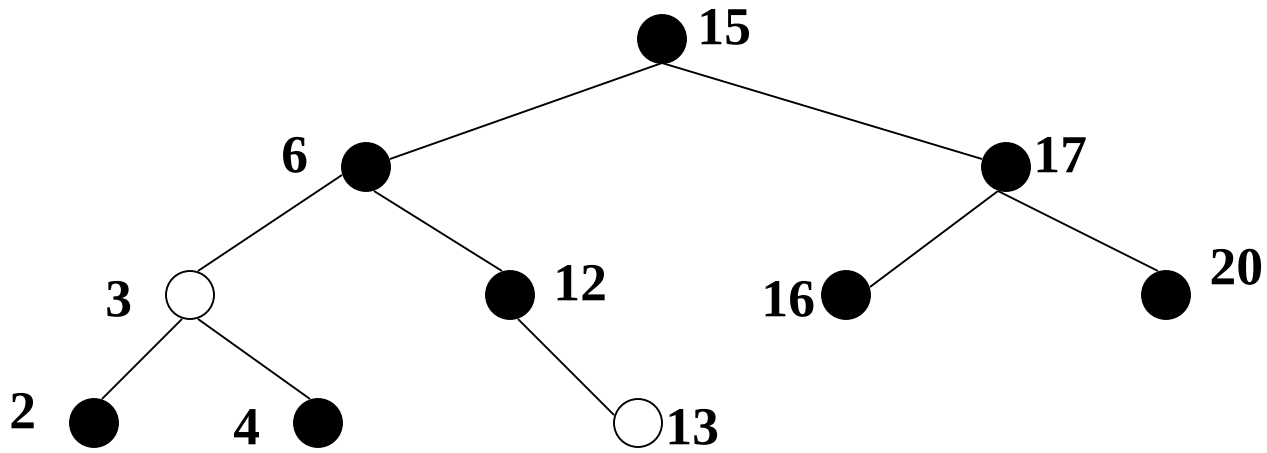
hình 43

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



hình 44

Cây sau khi loại bỏ 10



hình 45

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.

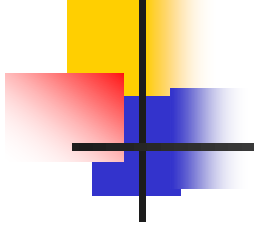
Điều bất lợi duy nhất là việc lưu trữ thêm thuộc tính màu và liên kết đến node cha của một node trên cây.

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.

Hiệu quả của cây đỏ đen *(tt)*

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).



Cám ơn