Cấu trúc dữ liệu và thuật toán

Tìm kiếm

Các cây cân bằng động khác và cây đỏ - đen

(Red-Black and Other Dynamically BalancedTrees)

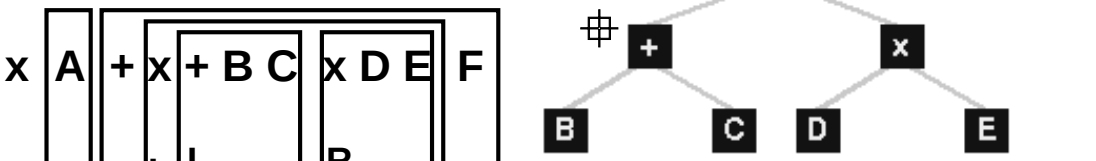
Tìm kiếm – Viếng thăm (Re-visited)

- **Cây nhị phân *O(log n) nếu nó là cây cân bằng***
  - **Cây nhị phân đơn giản tốt cho các mảng tĩnh**
  - **Tần xuất thấp (preferably zero) cho insertions/deletions**

  ***Nhưng* bộ dữ liệu của tôi có thể thay đổi!**
  - **Bộ dữ liệu động**
  - **Cần thiết phải tạo ra cây cân bằng**
- **Đầu tiên, kiểm tra một vài hoạt động của cây cơ bản.**
  - **Hữu ích trong một vài cách!**

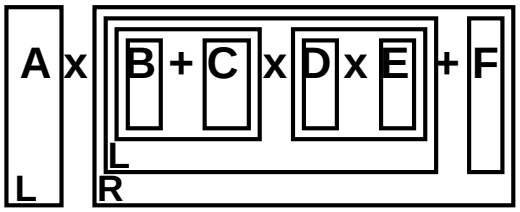
Cây du lịch

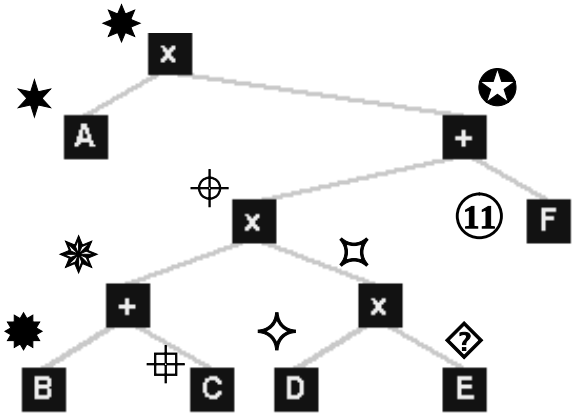
- **Du lịch = viếng thăm tất cả các node của cây**
- **Ba chọn lựa cơ bản**
  - **Thứ tự trước**
    - **Gốc**
    - **Cây con trái**
    - **Cây con phải**



Cây du lịch

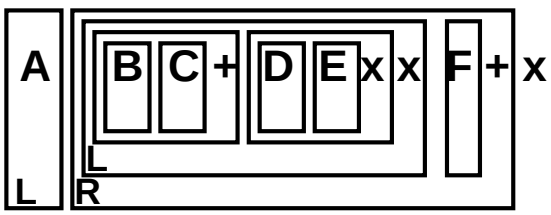
- **Du lịch = viếng thăm tất cả các node của cây**
- **Ba chọ lựa cơ bản**
  - **Thứ tự giữa**
    - **Cây con trái**
    - **Gốc**
    - **Cây con phải**

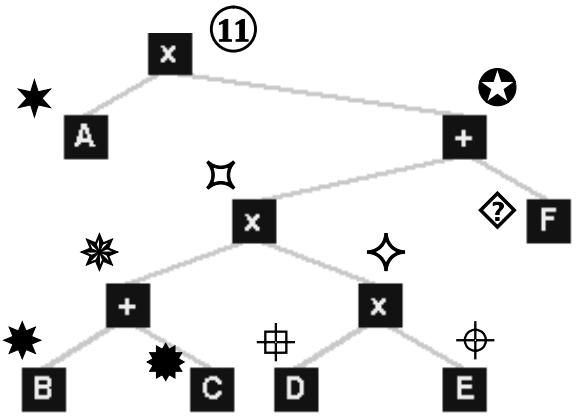




Cây du lịch

- **Du lịch = Viếng thăm các node của cây**
- **Ba chọn lựa cơ bản**
  - **Thứ tự sau**
    - **Cây con trái**
    - **Cây con phải**
    - **Gốc**





Cây du lịch

Thứ tự sau

- **Cây con trái**
- **Cây con phải**
- **Gốc**

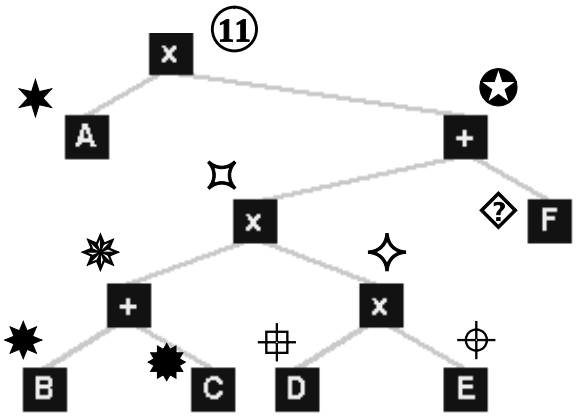
Công thức đảo

(A (((BC+)(DEx) x) F +)x)

- **Công thức đại số**

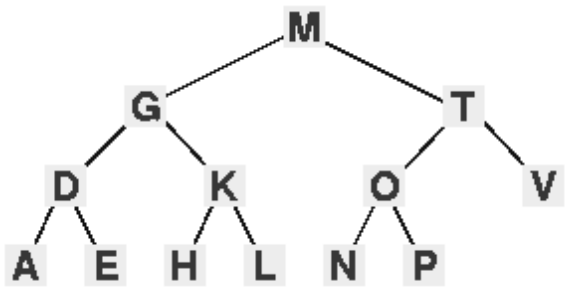
(A x(((B+C) x(DxE))+F))

= **Cây du lịch nào?**



Cây – Tìm kiếm

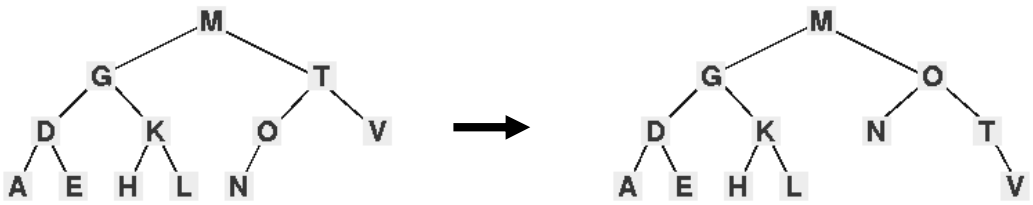
- **Cây tìm kiếm nhị phân**
  - **Tạo một danh sách sắp xếp theo thứ tự giữa**



- **Thư tự :** A D E G H K L M N O P T V

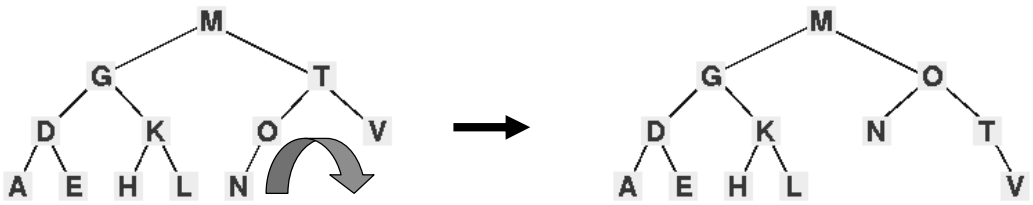
Cây – Tìm kiếm

- **Cây tìm kiếm nhị phân**
  - **Bảo quản thứ tự**
  - **Quan sát thấy rằng: biến đổi này bảo quản cây tìm kiếm**



Cây- tìm kiếm

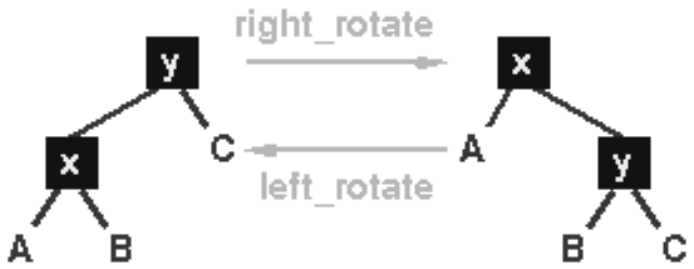
- **Cây tìm kiếm nhị phân**
  - **Bảo quản thứ tự**
  - **Quan sát thấy rằng: biến đổi này bảo quản cây tìm kiếm**



- **Chúng tôi thực hiện một góc quay đối với cây con về các node T và O.**

Cây - Xoay

- **Cây tìm kiếm nhị phân**
  - **Góc quay có thể thực hiện theo hướng trái hoặc phải (left- or right-rotations)**

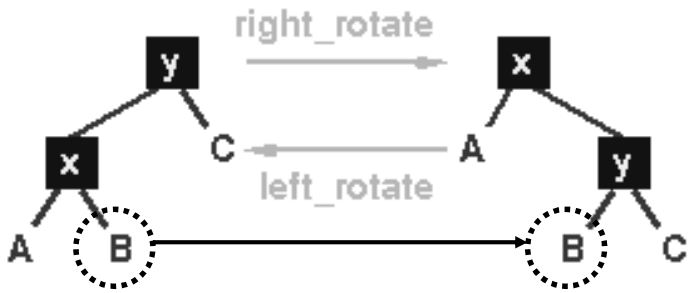


  - **Cho cả hai cây: Du lịch theo thứ tự giữa(inorder) là**

    A x B y C

Cây - Xoay

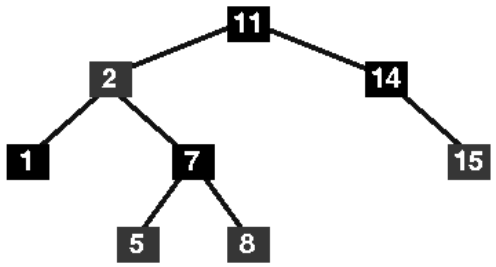
- **Cây tìm kiếm nhị phân**
  - **Góc quay có thể thực hiện theo hướng trái hoặc phải (left- or right-rotations)**



  - **Chú ý rằng: Việc xoay cần thiết phải di chuyển B từ con phải của x đến con trái của y**

Cây -Các cây đỏ đen

- **Một cây Red-Black**
  - **Cây tìm kiếm nhị phân**
  - **Mỗi node có "mầu" red hoặc black**

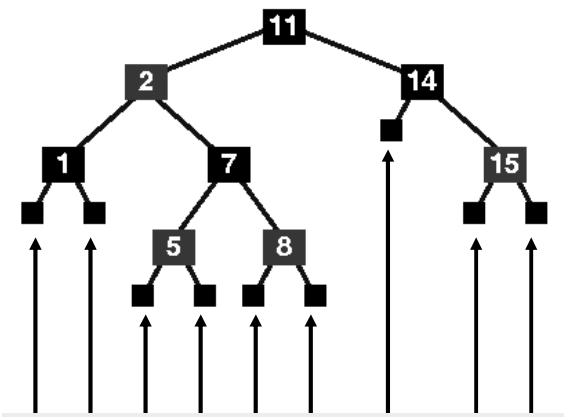


  - **Một cây nhị phân thông thường với các node được tô màu tạo thành cây đỏ- đen**

Cây – Cây Đỏ Đen

- **Một cây Đỏ Đen (A Red-Black Tree)**
  - **Tất cả các node là Đỏ hoặc Đen**
  - **Các lá là Đen (BLACK)**

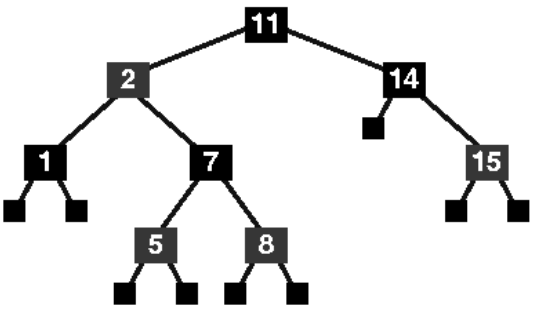
Khi bạn tìm hiểu đoạn code rb-tree, bạn sẽ gặp các node gác (black) được bổ xung như các lá. Chúng không chứa dữ liệu.



Các node gác (black)

Cây – Cây Đỏ Đen

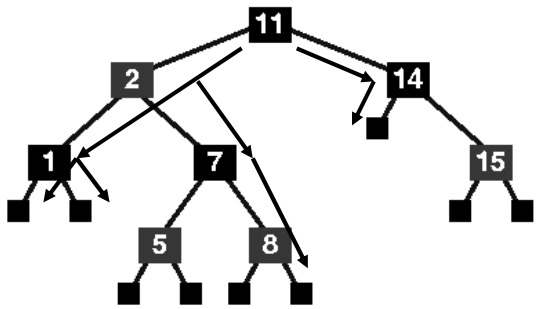
- **Một cây Đỏ Đen (A Red-Black Tree)**
  - **Tất cả các node là Đỏ hoặc Đen**
  - **Các lá là Đen (BLACK)**
  - **Nếu một node là RED, thì cả hai con của nó là BLACK**



Điều này có nghĩa: không có Một nhánh nào tồn tại hai node Đỏ kế liền nhau.
(Nhưng bất cứ các node BLACK nào cũng có thể kế liền nhau.)

Cây – Cây Đỏ Đen

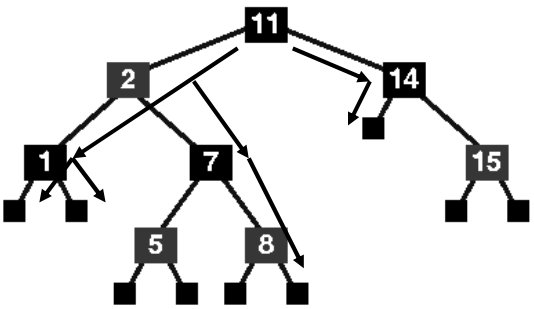
- **Một cây Đỏ Đen (A Red-Black Tree)**
  - **Tất cả các node là Đỏ hoặc Đen**
  - **Các lá là Đen (BLACK)**
  - **Nếu một node là RED, thì cả hai con của nó là BLACK**
  - **Mọi nhánh từ một node đến một lá chứa cùng số lượng các node BLACK**



Tính từ gốc(root), có 3 node BLACK trên tất cả các đường

Cây – Cây Đỏ Đen

- **Một cây Đỏ Đen (A Red-Black Tree)**
  - **Tất cả các node là Đỏ hoặc Đen**
  - **Các lá là Đen (BLACK)**
  - **Nếu một node là RED, thì cả hai con của nó là BLACK**
  - **Mọi nhánh từ một node đến một lá chứa cùng số lượng các node BLACK**



Chiều dài của nhánh này chính là Chiều cao các node đen của cây

Cây – Cây Đỏ Đen

- *Lemma*

  Một RB-Tree với n node có

  $height \leq 2 \log(n+1)$

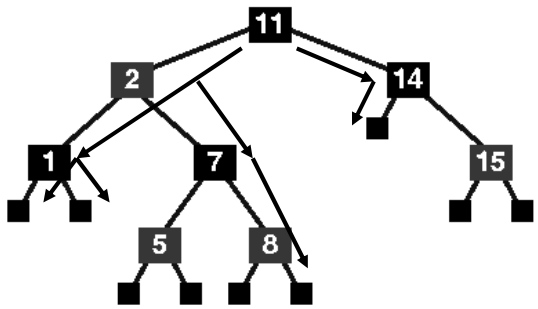
  - *Proof* .. See Cormen

- **Bản chất,**

  $height \leq 2$ black height

- **Thời gian tìm kiếm**

  $O(\log n)$



Cây – Cây Đỏ Đen

- *Cấu trúc dữ liệu*
  - Như chúng tôi đã biết, Các node trong cây red-black cần biết cha mẹ của chúng,
  - Do đó, chúng tôi cần cấu trúc dữ liệu này

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

Giống như một Cây nhị phân, Bổ xung thêm hai Thuộc tính đánh Dấu

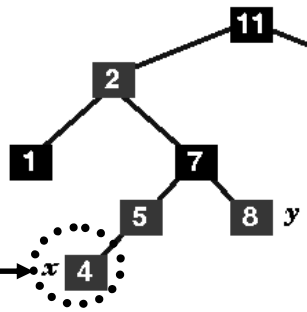
Cây - *Insertion*

- **Chèn thêm một new node**
  - **Yêu cầu một re-balance của cây**

```
rb_insert( Tree T, node x ) {
    /* Insert in the tree in the usual way */
    tree_insert( T, x );
    /* Now restore the red-black property */
    x->colour = red;
```

Chèn node 4 Tô màu red

Nhãn của node *x*



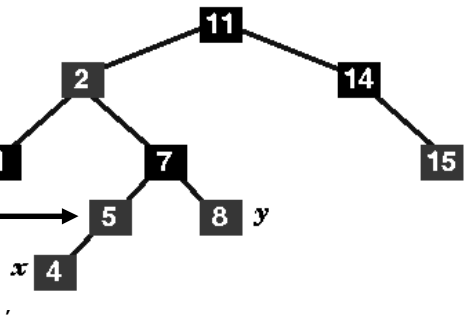

```
                T->root) &&
                ent == x->pa
                's parent is
                parent->pare
                >colour == r
            / case 1 - cha
                ->colo
                = bla
                ->parent->colour = red;
        /* Move x up the tree */
        x = x->parent->parent;
```

Cây - *Insertion*

```
rb_insert( Tree T, node x ) {
    /* Insert in the tree in the usual way */
    tree_insert( T, x );
     /* 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 l
                                    ent->
                                    red )
                                    ange
            x->parent->colour =
                                    lack;
                                    rent->
            /* Move x up the tr
            x = x->parent->pare
```

Trong khi chưa đi tới root và cha của x là red

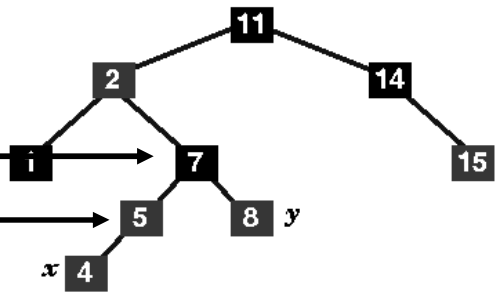
x->parent



Cây - Insertion

```
rb_insert( Tree T, node x ) {
    /* Insert in the tree in the usual way */
    tree_insert( T, x );
    /* 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,
          if ( y->colour == red ) {
              /* case 1 - change th
              x = x->parent->parent
              x->parent->colour =
              /* move x up the tree
              x = x->parent->parent
```

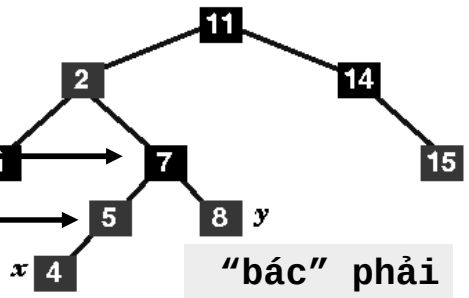
Nếu x nằm ở bên trái ông của nó



Trees - Insertion

```
/* 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;
     red ) {
     ange th
     x->parent->colour = b
     v->colour = black;
     x->parent->parent
     /* Move x up the tree
     x = x->parent
```

y là bác phải của x

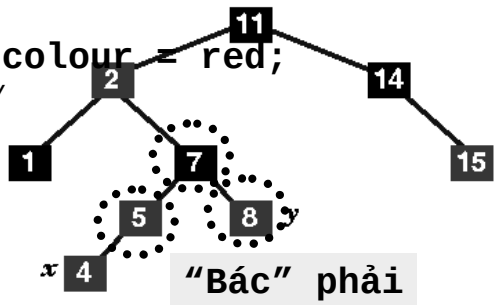


"bác" phải

Trees - Insertion

```
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;
                                    ree */
                                    nt;
```

Nếu bác là red, đổi mầu của y, ông và cha.

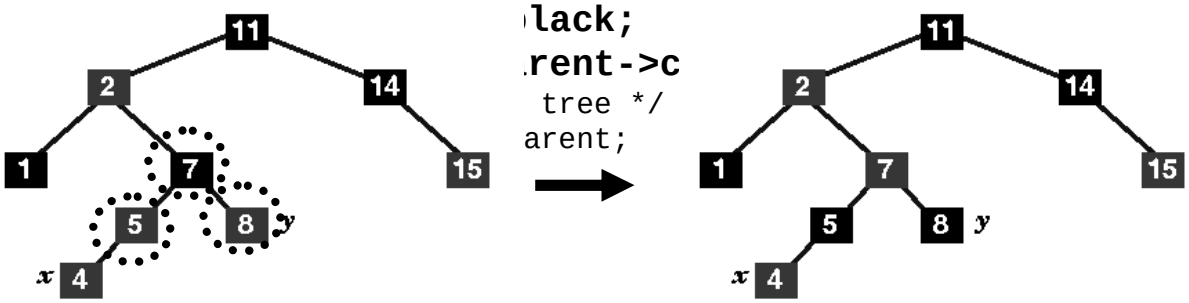


`x->parent->parent`

"Bác" phải

Cây - Insertion

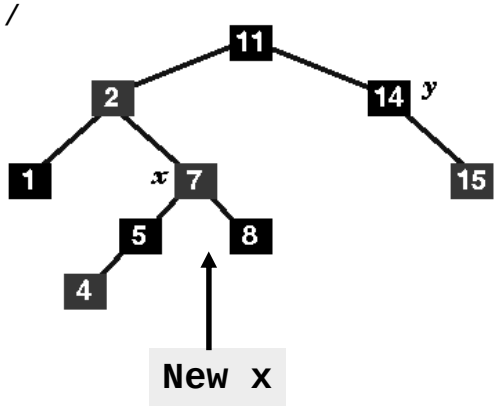
```
while ( (x != T->root) && (x->parent->colour == red) )
  if ( x->parent == x->parent->parent->left ) {
     /* If x's parent is a left, y is x's right 'uncle'
     y = x->parent->parent->right;
     if ( y->colour == red ) {
        /* case 1 - change the colours */
        x->parent->colour = black;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



Cây - *Insertion*

```
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;
```

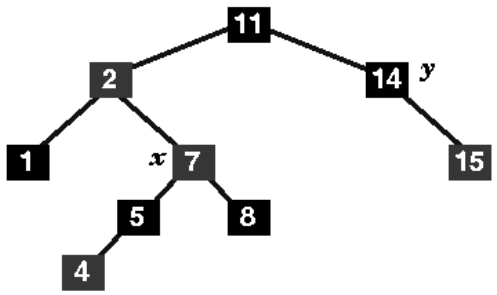
Cha của x lại nằm bên trái,
Đánh dấu bác của x
Nhưng bác là black tại thời điểm này



Cây - Insertion

```
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 ) {
```

.. Nhưng bác là black tại thời điểm này và x nằm bên phải Cha của nó

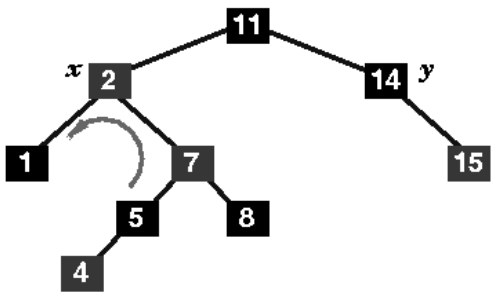



```
        /* y is a black node */
        if ( x == x->parent->right ) {
           /* and x is to the right */
           /* case 2 - move x up and rotate */
           x = x->parent;
           left_rotate( T, x );
```

Cây - *Insertion*

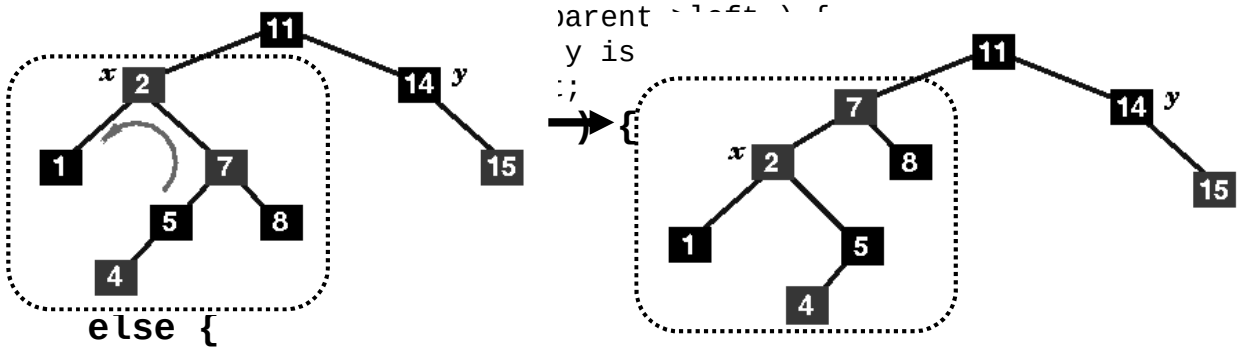
```
while ( (x != T->root) && (x->parent->colour == red) ) {
  if ( x->parent == x->parent->parent->left ) {
    /* If x's parent is a left, y is x
    y = x->parent->parent->right;
     if ( y->colour == red ) {
       /* case 1 - change the colours */
       x->parent->colour = black;
       y->colour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

.. Vì thế chuyển x lên và quay x như một gốc ...



Cây - Insertion

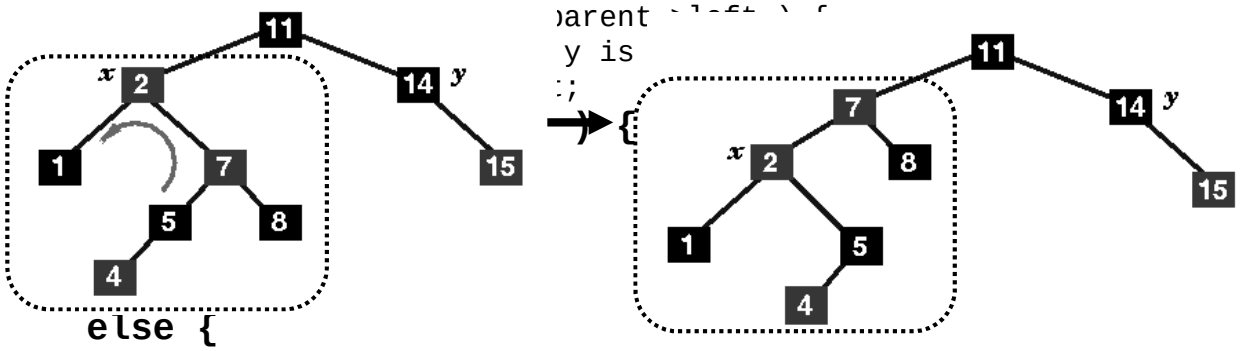
```
while ( (x != T->root) && (x->parent->colour == red) ) {
                                  parent
                                  y is
                                  ;;
```




```
    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 );
```

Cây - Insertion

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




```
    else {
        /* y is a black node */
        if ( x == x->parent->right ) {
           /* and x is
           /* case 2 -
           x = x->parent;
           left_rotate( T, x );
```

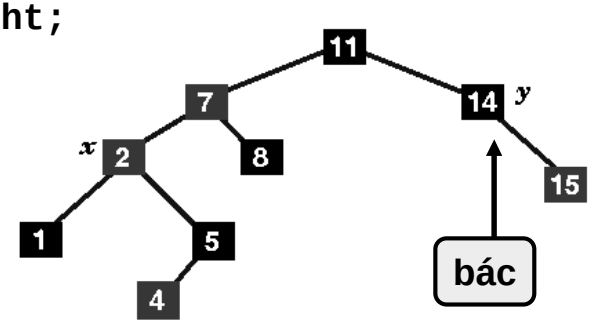
.. Nhưng cha của x vẫn là red ...

Cây - *Insertion*

```
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 */
     y->colour = black;
     x->parent->colour = red;
     x->parent->parent->colour = red;
     /* Move x up the tree */
     x = x->parent->parent;
  else {
    /* y is a black node */
    if ( x == x->parent->right ) {
       /* and x is to the right */
       left_rotate( T, x );
```

.. Bác là black ..

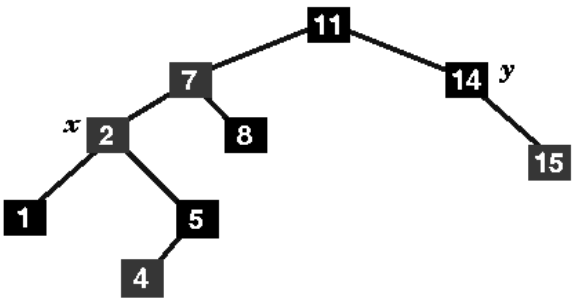
.. và x chuyển *về trái* cha của nó



Cây - *Insertion*

```
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;
        /* Move x up the tree */
```

.. Vì thế chúng ta có trường hợp Cuối cùng ..

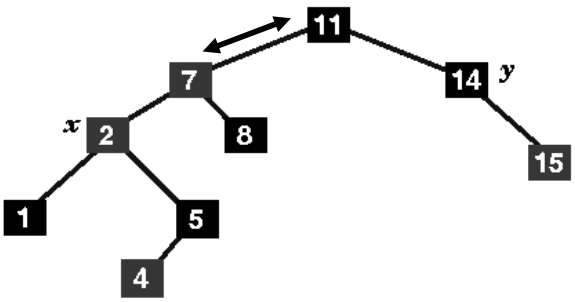



```
        /* case 2 - move x up and rotate */
        x = x->parent;
        left_rotate( T, x );
  else { /* case 3 */
    x->parent->colour = black;
    x->parent->parent->colour = red;
    right_rotate( T, x->parent->parent );
    }
```

Cây - *Insertion*

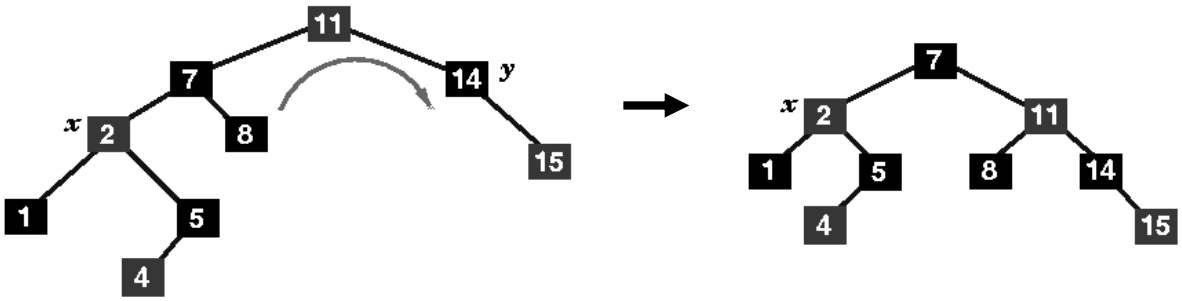
```
while ( (x != T->root) && (x->parent->colour == red) ) {
  if ( x->parent == x->parent->parent->left ) {
  /* If x's parent is a left, y is x's right 'uncle' */
  y = x->parent->parent->right;
  if ( y->colour == red ) {
        /* case 1 - change the colours */
        x->parent->colour = black;
        y->colour = black;
        x->parent->parent->colour = red;
        /* Move x up the tree */
        x = x->
     else {
     /* y is a
     if ( x ==                                    {
           /*                                     */
           /* case 2 - move x up and rotate */
           x = x->parent;
           left_rotate( T, x );
    else { /* case 3 */
      x->parent->colour = black;
      x->parent->parent->colour = red;
      right_rotate( T, x->parent->parent );
      }
```

.. Đổi mầu
Và quay ..



Cây - Insertion

```
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' */
```

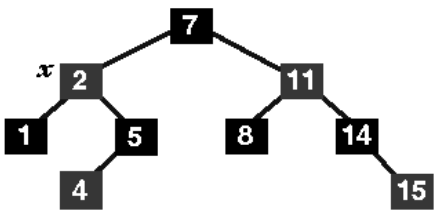



```
  else { /* case 3 */
    x->parent->colour = black;
    x->parent->parent->colour = red;
    right_rotate( T, x->parent->parent );
    }
```

Cây - *Insertion*

```
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;
```

Bây giờ, đây là cây red-black .. Vì thế, chúng ta kết thúc!

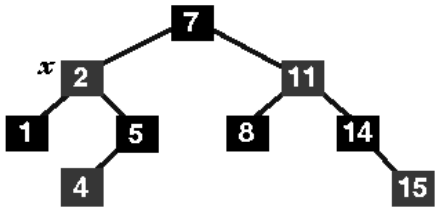



```
    el
    /* y is a black node */
    if ( x == x->parent->right ) {
          /* and x is to the right */
          /* case 2 - move x up and rotate */
          x = x->parent;
          left_rotate( T, x );
     else { /* case 3 */
       x->parent->colour = black;
       x->parent->parent->colour = red;
       right_rotate( T, x->parent->parent );
       }
```

Cây - *Insertion*

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

Các trường hợp là tương đương khi cha nằm ở bên Phải của ông!



Cây Red-black – Phân tích

- **Addition**
  - **Insertion** **So sánh** $O(\log n)$
  - **Fix-up**
    - **Với tưng giai đoạn, x di chuyển trong cây tới một mức đỉnh của nó** $O(\log n)$
  - **Overall** $O(\log n)$
- **Deletion**
  - **cũng** $O(\log n)$

Cây đỏ đen – Cái gì bạn cần biết?

- **Các yêu cầu bạn cần nắm:**
  - **Thuật toán tồn tại có liên quan**
  - **Định nghĩa thế nào là cây đỏ đen**
  - **Khi nào sử dụng nó**
    - *ie* **Những bài toán nào, nó có thể giải đáp?**
  - **Độ phức tạp của nó**
  - **Nó hoạt động như thế nào**
  - ***Nơi nà, nó có thể ứng dụng***
    - **Làm thế nào để chuyển nó vào ứng dụng của bạn.**