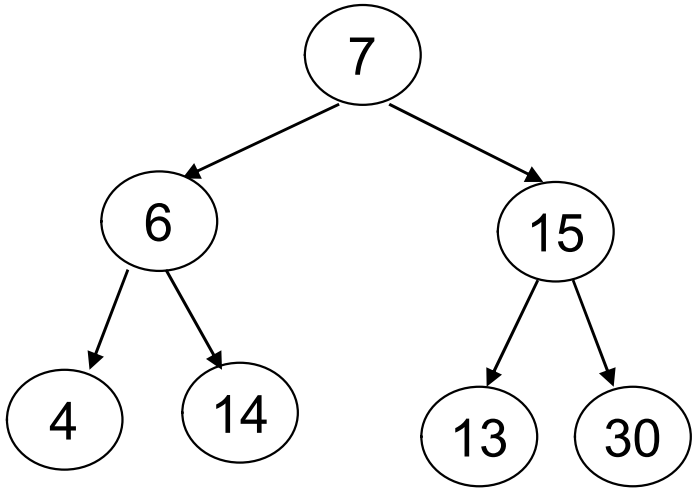
CÂY NHỊ PHÂN CÂN BẰNG

❖ĐỊNH NGHĨA

Cây nhị phân cân bằng hoàn toàn là cây mà tại mỗi nút của nó số nút của cây con trái và số nút của cây con phải chênh lệch không quá 1.



Tính chất:

- Thời gian tìm kiếm xấu nhất:

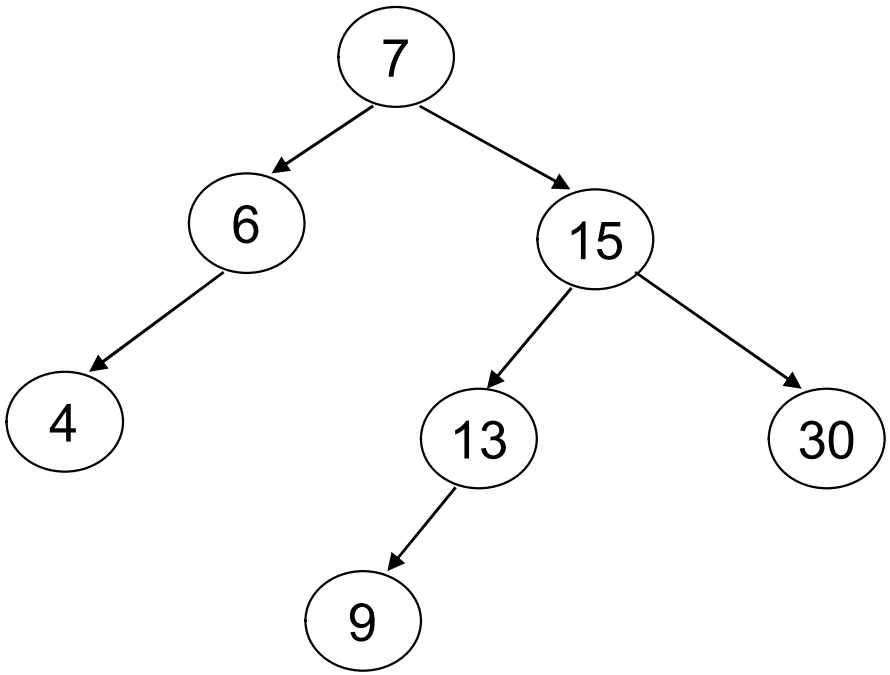
  O(logn)

- Khó duy trì tình trạng cân bằng hoàn toàn.

CÂY NHỊ PHÂN CÂN BẰNG

❖ĐỊNH NGHĨA

Cây nhị phân cân bằng (cây AVL) là cây mà mỗi nút của nó, độ cao của cây con trái và độ cao của cây con phải chênh lệch không quá 1.



Tính chất:

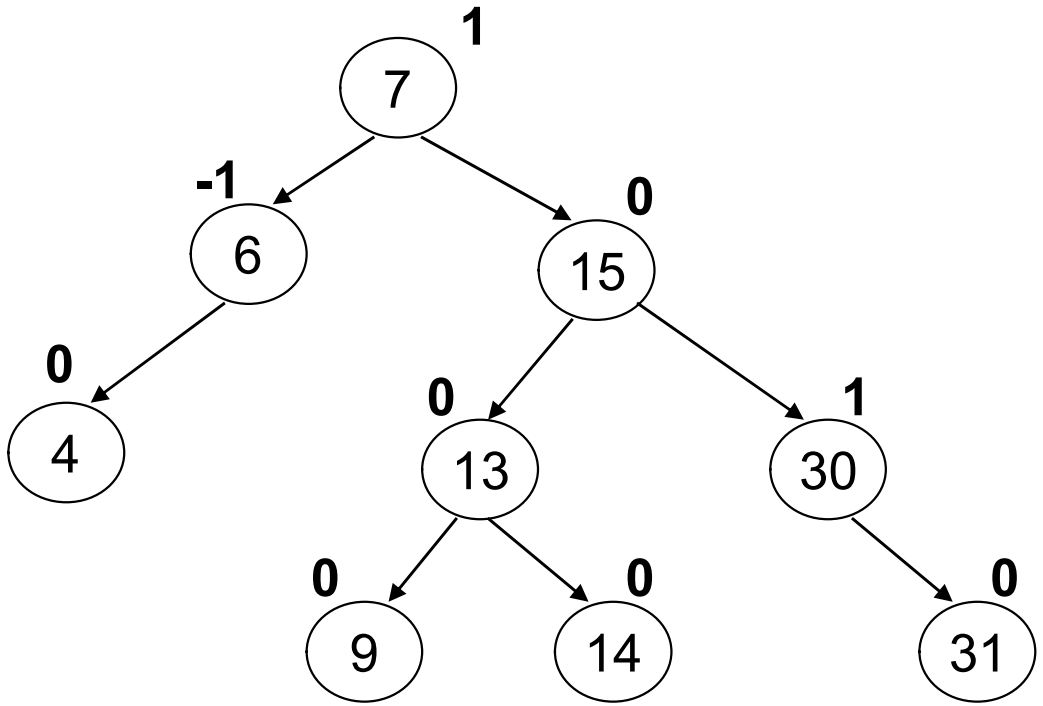
- Cao không quá 45% chiều cao cây cân bằng hoàn toàn
- Chi phí cân bằng cây nhỏ hơn

CÂY NHỊ PHÂN CÂN BẰNG

❖ĐỊNH NGHĨA

Chỉ số cân bằng (CSCB) tại một nút p là hiệu số chiều cao cây con phải của p với chiều cao cây con trái của p. Nếu:

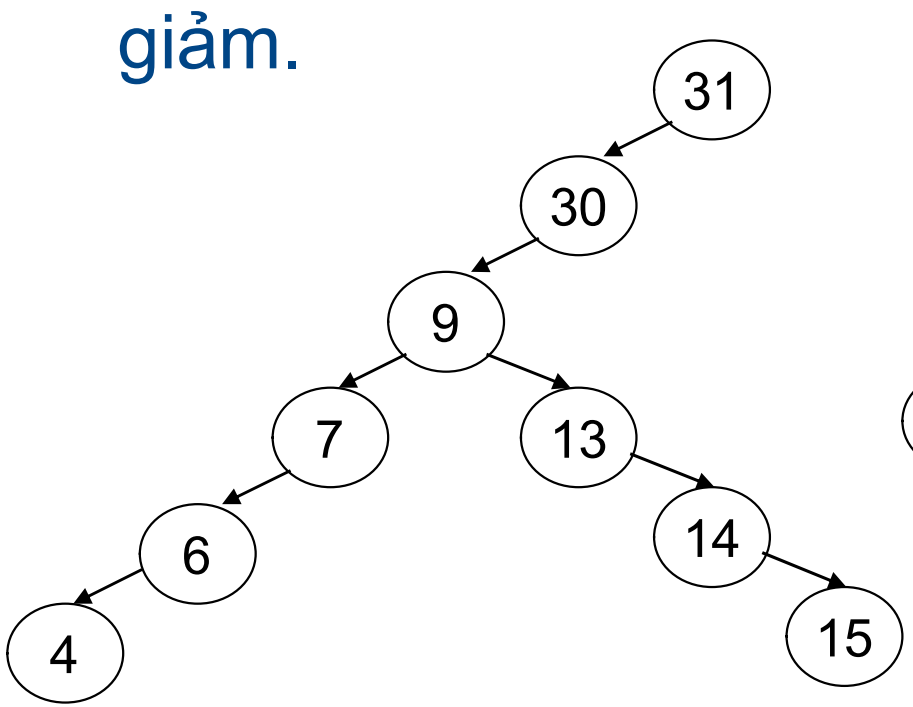
- CSCB(p) = 0: độ cao hai cây con bằng nhau.
- CSCB(p) = -1: cây p bị lệch trái
- CSCB(p) = 1: cây p bị lệch phải

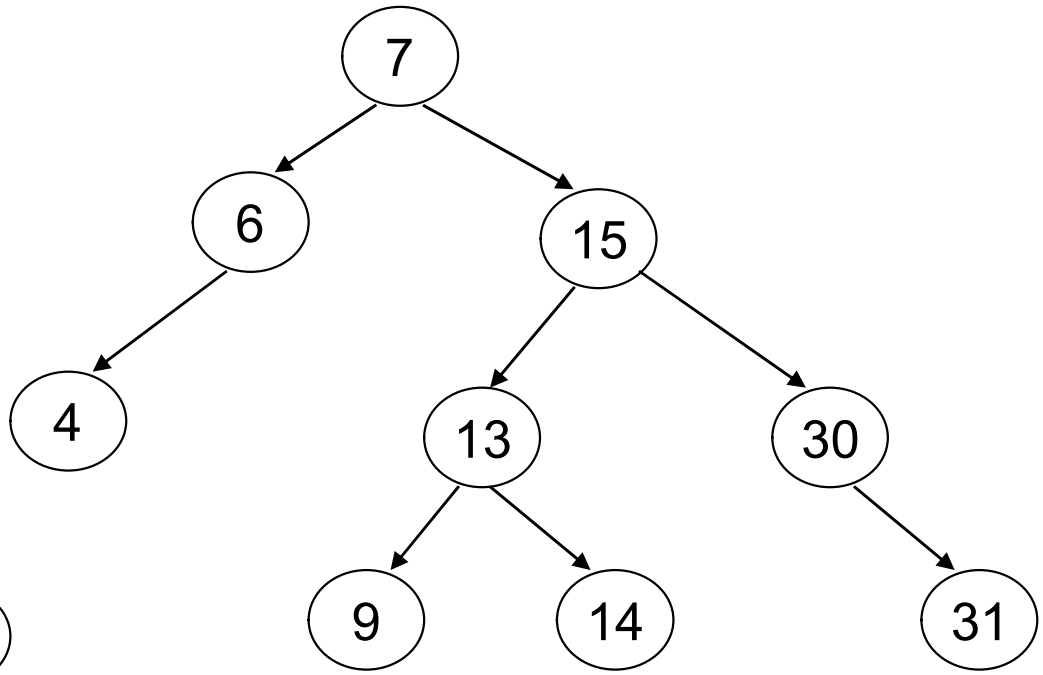


CÂY NHỊ PHÂN CÂN BẰNG

❖ ỨNG DỤNG CỦA CÂY CÂN BẰNG

Cây cân bằng nhằm đảm bảo độ cao của cây nhị phân gần với giá trị $\log_2(n)$ với n là số nút của cây. Vì vậy, các cây nhị phân tìm kiếm nếu là cây nhị phân cân bằng thì chi phí cho việc tìm kiếm sẽ được giảm.





CÂY NHỊ PHÂN CÂN BẰNG

CÁC TRƯỜNG HỢP MẤT CÂN BẰNG

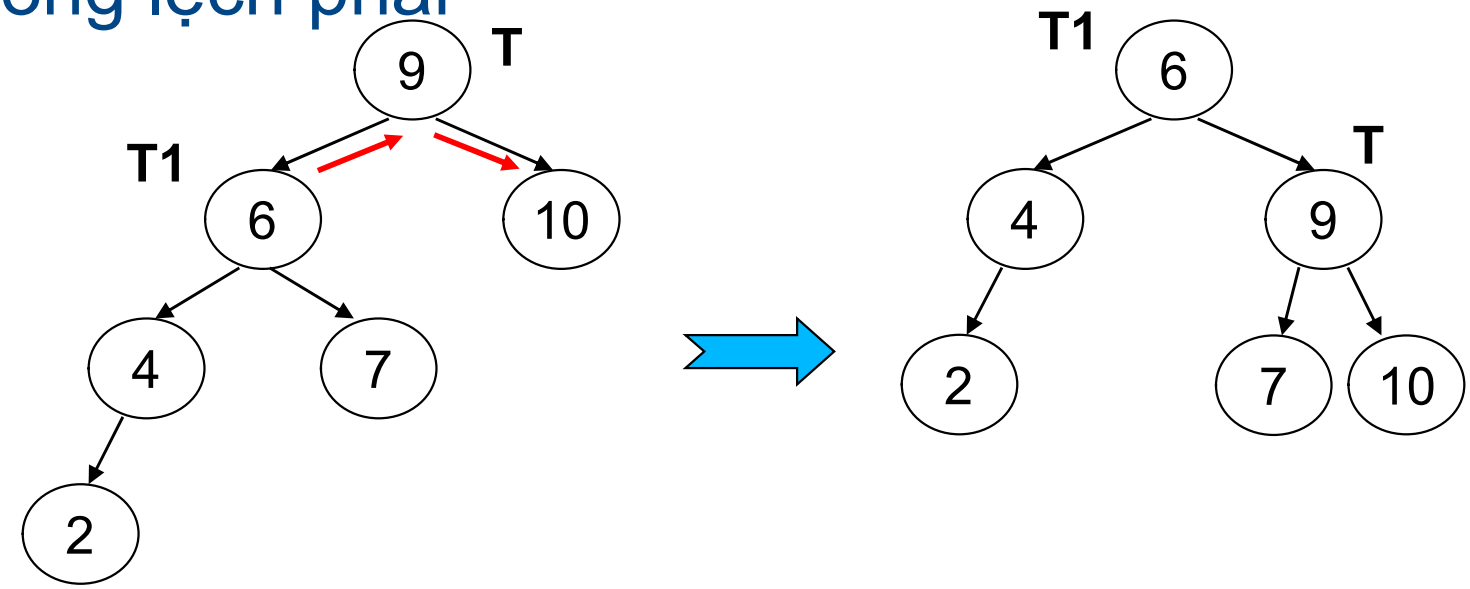
Các trường hợp mất cân bằng xảy ra trong cây AVL khi thêm nút hoặc xóa nút, không xét trường hợp mất cân bằng trong cây nhị phân tìm kiếm tổng quát.

Có 4 trường hợp mất cân bằng:

CÂY NHỊ PHÂN CÂN BẰNG

❖CÁC TRƯỜNG HỢP MẤT CÂN BẰNG

- **Trường hợp 1 (LL)**: cây tại p lệch trái, cây con trái không lệch phải



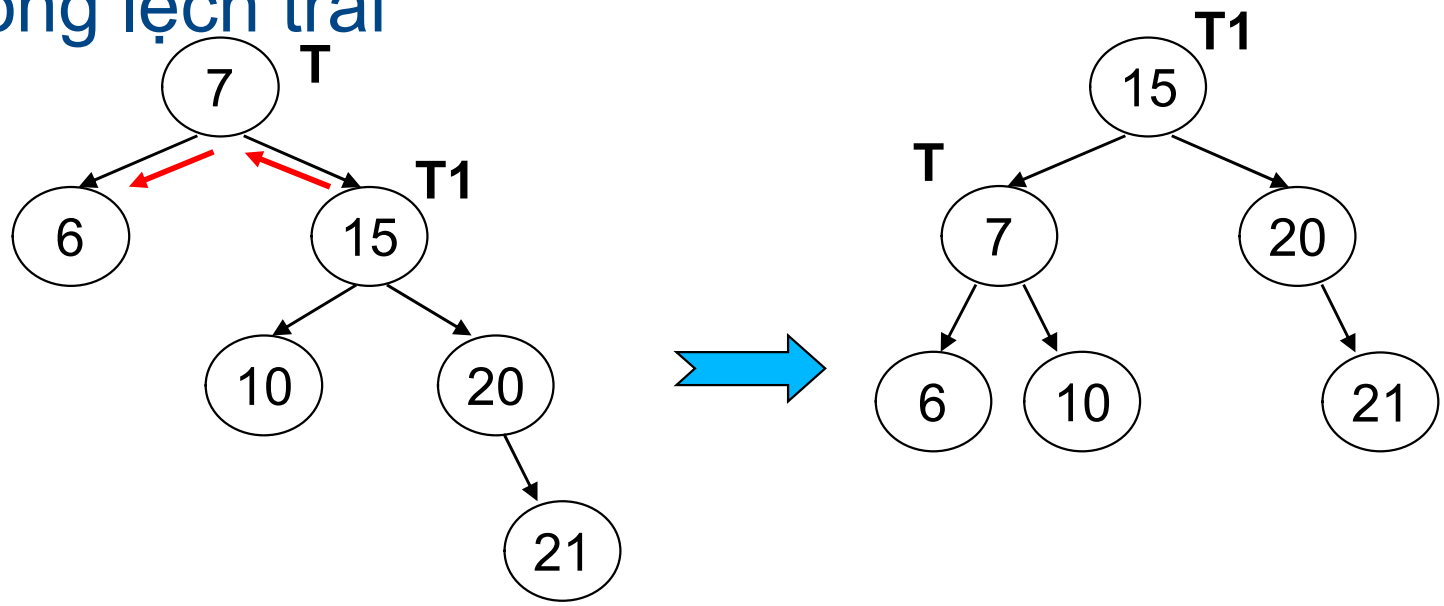
Cần xoay phải tại nút mất cân bằng T:

- Chuyển nút con phải của T1 thành con trái của T
- Chuyển T thành nút con phải của T1 và đặt T1 là gốc

CÂY NHỊ PHÂN CÂN BẰNG

❖CÁC TRƯỜNG HỢP MẤT CÂN BẰNG

- **Trường hợp 2 RR**: cây tại p lệch phải, cây con phải không lệch trái



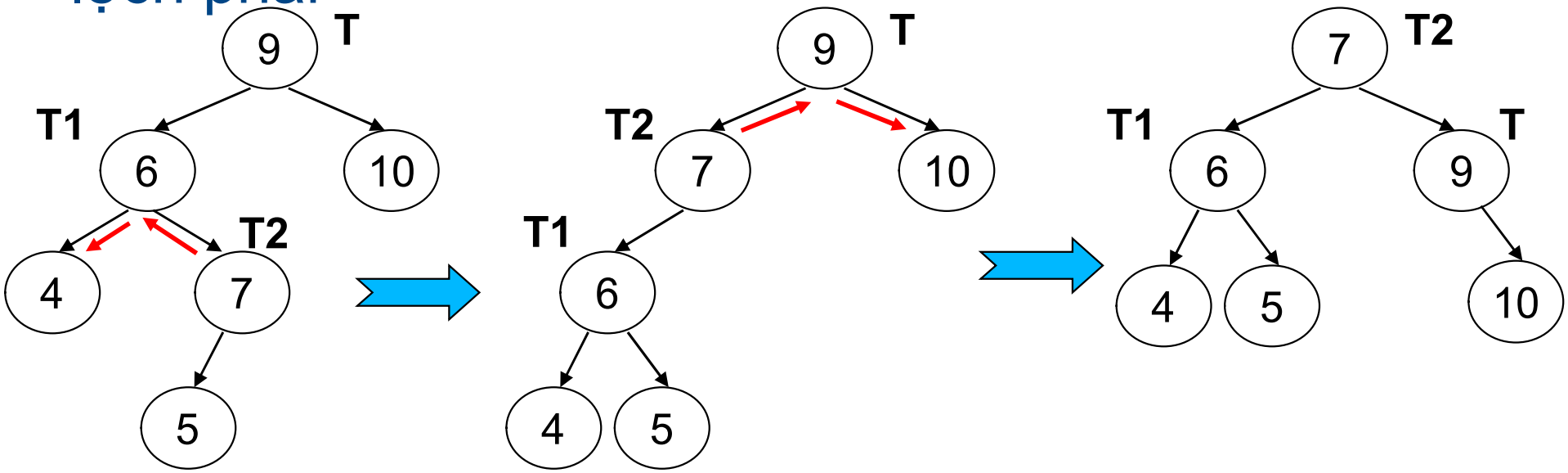
Cần xoay phải tại nút mất cân bằng T:

- Chuyển nút con trái của T1 thành con trái của T
- Chuyển T thành nút con trái của T1 và đặt T1 là gốc

CÂY NHỊ PHÂN CÂN BẰNG

❖CÁC TRƯỜNG HỢP MẤT CÂN BẰNG

- **Trường hợp 3 LR**: cây tại p lệch trái, cây con trái lệch phải

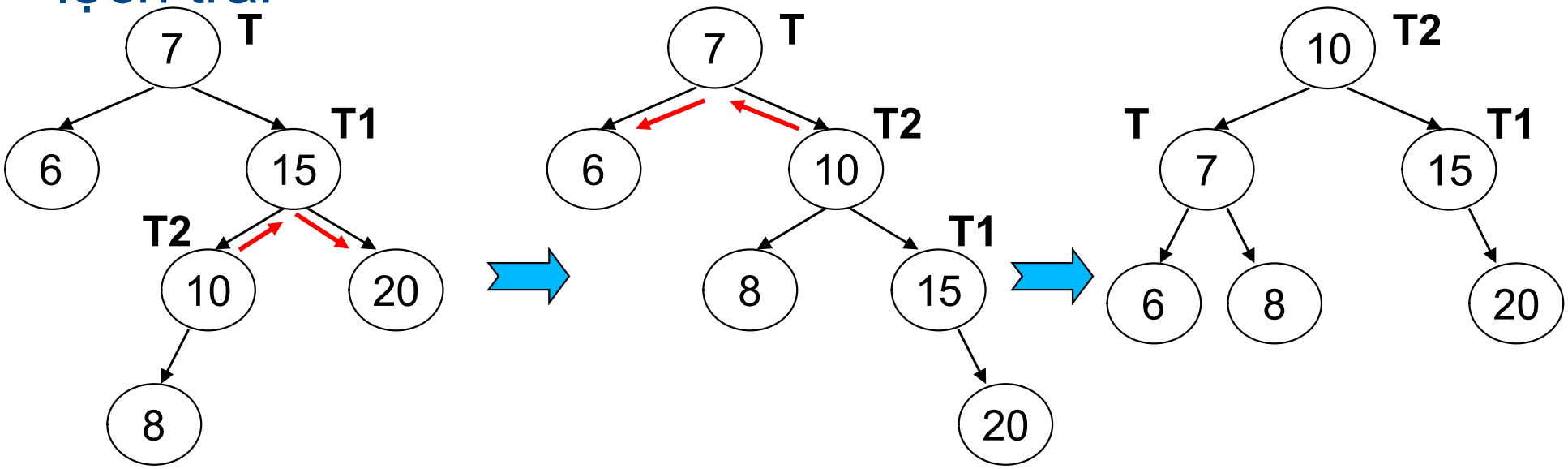


- Thực hiện phép xoay trái tại nút T1:
- Sau đó thực hiện phép xoay phải tại nút T

CÂY NHỊ PHÂN CÂN BẰNG

❖CÁC TRƯỜNG HỢP MẤT CÂN BẰNG

- **Trường hợp 4 RL**: cây tại p lệch phải, cây con trái lệch trái



- Thực hiện phép xoay phải tại nút T1:
- Sau đó thực hiện phép xoay trái tại nút T

CÂY NHỊ PHÂN CÂN BẰNG

❖CÁC THAO TÁC TRÊN CÂY AVL

- Tổ chức dữ liệu
- Cân bằng cây
- Thêm một nút có khóa x
- Hủy một nút có khóa x

CÂY NHỊ PHÂN CÂN BẰNG

❖CÁC THAO TÁC TRÊN CÂY AVL

- **Tổ chức dữ liệu**

```
struct TenDulieu {
    // Dữ liệu cần  quản lý
};
struct AVLNode {
    int balFactor; // chỉ số cân bằng
    TenDulieu key;
    AVLNode *pLeft, *pRight;
};
typedef AVLNode *AVLTREE;
```

CÂY NHỊ PHÂN CÂN BẰNG

❖CÁC THAO TÁC TRÊN CÂY AVL

- **Tổ chức dữ liệu**

```
#define LH -1
#define EH 0
#define RH 1
AVLNode * CreateAVLNode(TenDulieu x) {
    AVLNode *p = new AVLNode;
    if (p)
    {    p->balFactor = EH; p->key = x; p->pLeft = NULL;
    p->pRight = NULL;  }
    return p;
}
```

CÂY NHỊ PHÂN CÂN BẰNG

❖CÁC THAO TÁC TRÊN CÂY AVL

- **Tổ chức dữ liệu**

```
void CreateAVLTree(AVLTREE &T) {
    T = NULL;
}
void RightRotate(AVLTREE &T) {
    AVLNode *T1 = T->pLeft;
    T->pLeft = T1->pRight;  T1->pRight = T;  T = T1;
}
```

CÂY NHỊ PHÂN CÂN BẰNG

❖CÁC THAO TÁC TRÊN CÂY AVL

- **Tổ chức dữ liệu**

```
void LeftRotate(AVLTREE &T) {
    AVLNode *T1 = T->pRight;
    T->pRight = T1->pLeft;  T1->pLeft = T;
    T = T1;
}
```

CÂY NHỊ PHÂN CÂN BẰNG

❖CÁC THAO TÁC TRÊN CÂY AVL

- **Phép xoay trong các trường hợp**

```
void LL(AVLTREE &T) {
        switch(T->pLeft->balFactor) {
         case LH:  T-> balFactor =EH;
                   T->pLeft->balFactor=EH; break;
         case EH:  T->balFactor=LH;
                   T->pLeft->balFactor =RH; break;
        }
    RightRotate(T);
}
```

CÂY NHỊ PHÂN CÂN BẰNG

❖CÁC THAO TÁC TRÊN CÂY AVL

- **Phép xoay trong các trường hợp**

```
void RR(AVLTREE &T) {
   switch(T->pRight-> balFactor) {
   case RH:  T-> balFactor = EH;
             T->pRight-> balFactor = EH; break;
   case EH:  T-> balFactor = RH;
             T->pRight-> balFactor = LH; break;
   }
   LeftRotate(T);
}
```

CÂY NHỊ PHÂN CÂN BẰNG

❖CÁC THAO TÁC TRÊN CÂY AVL

- **Phép xoay trong các trường hợp**

```
void LR(AVLTREE &T) {
    switch(T->pLeft->pRight->balFactor) {
    case LH:   T->balFactor=RH;
               T->pLeft->balFactor=EH; break;
    case EH:   T->balFactor = EH;
               T->pLeft->balFactor=EH; break;
    case RH:   T->balFactor =EH;
               T->pLeft->balFactor= LH; break;
    }
    T->pLeft->pRight->balFactor =EH;
    LeftRotate(T->pLeft); RightRotate(T);
}
```

CÂY NHỊ PHÂN CÂN BẰNG

❖CÁC THAO TÁC TRÊN CÂY AVL

- **Phép xoay trong các trường hợp**

```
void RL(AVLTREE &T) {
    switch(T->pRight->pLeft-> balFactor)
    {     case RH:   T-> balFactor = LH;
                     T->pRight-> balFactor = EH; break;
    case EH:   T-> balFactor = EH;
                     T->pRight-> balFactor = EH; break;
    case LH:   T-> balFactor = EH;
                     T->pRight-> balFactor = RH; break;
    }
    T->pRight->pLeft-> balFactor = EH;
    RightRotate(T->pRight); LeftRotate(T);
}
```

CÂY NHỊ PHÂN CÂN BẰNG

❖CÁC THAO TÁC TRÊN CÂY AVL

- **Cân bằng cây - Cân bằng lệch trái**: trả về 1 nếu chiều cao cây không đổi, trả về 2 nếu chiều cao cây có thay đổi

```
int BalanceLeft(AVLTREE &T) {
    switch(T->pLeft->balFactor) {
    case LH: LL(T); return 2;
    case EH: LL(T); return 1;
    case RH: LR(T); return 2;
    }
    return 0;
}
```

CÂY NHỊ PHÂN CÂN BẰNG

❖CÁC THAO TÁC TRÊN CÂY AVL

- **Cân bằng cây - Cân bằng lệch phải**: trả về 1 nếu chiều cao cây không đổi, trả về 2 nếu chiều cao cây có thay đổi

```
int BalanceRight(AVLTREE &T) {
    switch(T->pRight->balFactor) {
    case LH: RL(T); return 2;
    case EH: RR(T); return 1;
    case RH: RR(T); return 2;
    }
    return 0;
}
```

CÂY NHỊ PHÂN CÂN BẰNG

❖CÁC THAO TÁC TRÊN CÂY AVL

- **Thêm một nút có khóa x:** Khi thêm một nút, cần đảm bảo tính chất cân bằng của cây bằng cách:

  B1) Thêm một nút có khóa x vào cây tương tự như cây nhị phân tìm kiếm

  B2) Xét các nút từ vị trí nút cha của nút thêm vào về gốc để xác định nút p mất cân bằng.

  B3) Cân bằng tại nút p.

CÂY NHỊ PHÂN CÂN BẰNG

❖CÁC THAO TÁC TRÊN CÂY AVL

- **Thêm một nút có khóa x:** trả về -1 nếu không đủ bộ nhớ, 0 nếu đã tồn tại nút x, 1 nếu thêm thành công và cây không thay đổi chiều cao, 2 nếu thêm thành công và cây thay đổi chiều cao.

```
int Compare(TenDulieu x, TenDulieu x);
int AddAVLNode(AVLTREE &T, TenDulieu x) {
    int ret;
    if (T == NULL) {
    T = CreateAVLNode(x);
    if (T== NULL) return -1; else return 2;
    }
```

CÂY NHỊ PHÂN CÂN BẰNG

```
if (Compare(T->key, x) == 0) return 0;
if (Compare(T->key, x) > 0) {
    ret = AddAVLNode(T->pLeft, x);
    if (ret < 2) return ret;
    switch(T->balFactor) {
        case RH: T->balFactor = EH; return 1;
        case EH: T->balFactor = LH; return 2;
        case LH: BalanceLeft(T); return 1;
    }
}
```

CÂY NHỊ PHÂN CÂN BẰNG

```
    else {
        ret = AddAVLNode(T->pRight, x);
        if (ret < 2) return ret;
        switch(T->balFactor) {
            case LH: T->balFactor = EH; return 1;
            case EH: T->balFactor = RH; return 2;
            case RH: BalanceRight(T); return 1;
        }
    }
    return 0;
}
```

CÂY NHỊ PHÂN CÂN BẰNG

❖CÁC THAO TÁC TRÊN CÂY AVL

- **Hủy một nút có khóa x:** Khi hủy một nút, cần đảm bảo tính chất cân bằng của cây bằng cách:

  B1) Hủy nút có khóa x trên cây tương tự như cây nhị phân tìm kiếm.

  B2) Xét các nút từ vị trí nút cha của nút bị hủy về gốc để xác định nút p mất cân bằng.

  B3) Cân bằng tại nút p.

  B4) Tiếp tục tìm nút mất cân bằng từ nút p về nút gốc và cân bằng các nút này

CÂY NHỊ PHÂN CÂN BẰNG

❖CÁC THAO TÁC TRÊN CÂY AVL

- **Hủy một nút có khóa x:** 0 nếu không tồn tại nút x, 1 nếu hủy thành công và cây không thay đổi chiều cao, 2 nếu hủy thành công và cây thay đổi chiều cao.

```
int AVLSearchStandFor(AVLTREE &p, AVLTREE &q);
int DeleteAVLNode(AVLTREE &T, TenDulieu x) {
    int ret;
    AVLNode *p;
    if (T == NULL) return 0;
    if (Compare(T->key, x) > 0) {
        ret = DeleteAVLNode(T->pLeft, x);
        if (ret < 2) return ret;
```

CÂY NHỊ PHÂN CÂN BẰNG

```
    switch (T->balFactor) {
        case LH: T->balFactor = EH; return 2;
        case EH: T->balFactor = RH; return 1;
        case RH: return BalanceRight(T);
    }
}
```

CÂY NHỊ PHÂN CÂN BẰNG

```
if (Compare(T->key, x) < 0) {
    ret = DeleteAVLNode(T->pRight, x);
    if (ret < 2) return ret;
    switch (T->balFactor) {
        case RH: T->balFactor = EH; return 2;
        case EH: T->balFactor = LH; return 1;
        case LH: return BalanceLeft(T);
    }
}
```

CÂY NHỊ PHÂN CÂN BẰNG

```
p = T;
if (T->pLeft == NULL) {  T = T->pRight; ret = 2;   }
else
    if (T->pRight == NULL) {     T = T->pLeft; ret = 2; }
    else {
        ret = AVLSearchStandFor(p, T->pRight);
        if (ret < 2) return ret;
        switch (T->balFactor) {
            case RH: T->balFactor = EH; return 2;
            case EH: T->balFactor = LH; return 1;
            case LH: return BalanceLeft(T);
        }
    }
```

CÂY NHỊ PHÂN CÂN BẰNG

```
    delete p;
    return ret;
}
```

CÂY NHỊ PHÂN CÂN BẰNG

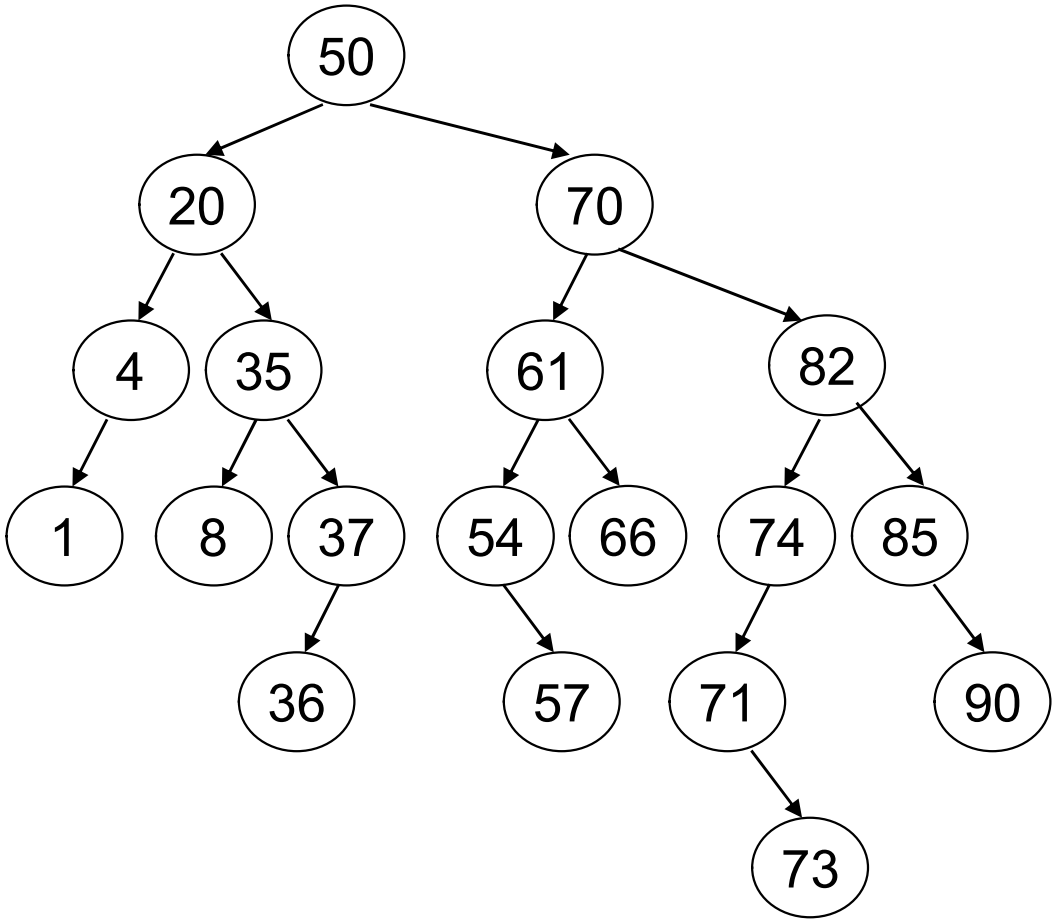
```
int AVLSearchStandFor(AVLTREE &p, AVLTREE &q) {
    int ret;
    if (q->pLeft) {
        ret = AVLSearchStandFor(p, q->pLeft);
        if (ret < 2) return ret;
        switch (q->balFactor) {
            case LH: q->balFactor = EH; return 2;
            case EH: q->balFactor = RH; return 1;
            case RH: return BalanceRight(q);
        }
    }
```

CÂY NHỊ PHÂN CÂN BẰNG

```
    else {
        p->key = q->key; p = q; q = q->pRight;
        return 2;
    }
    return 0;
}
```

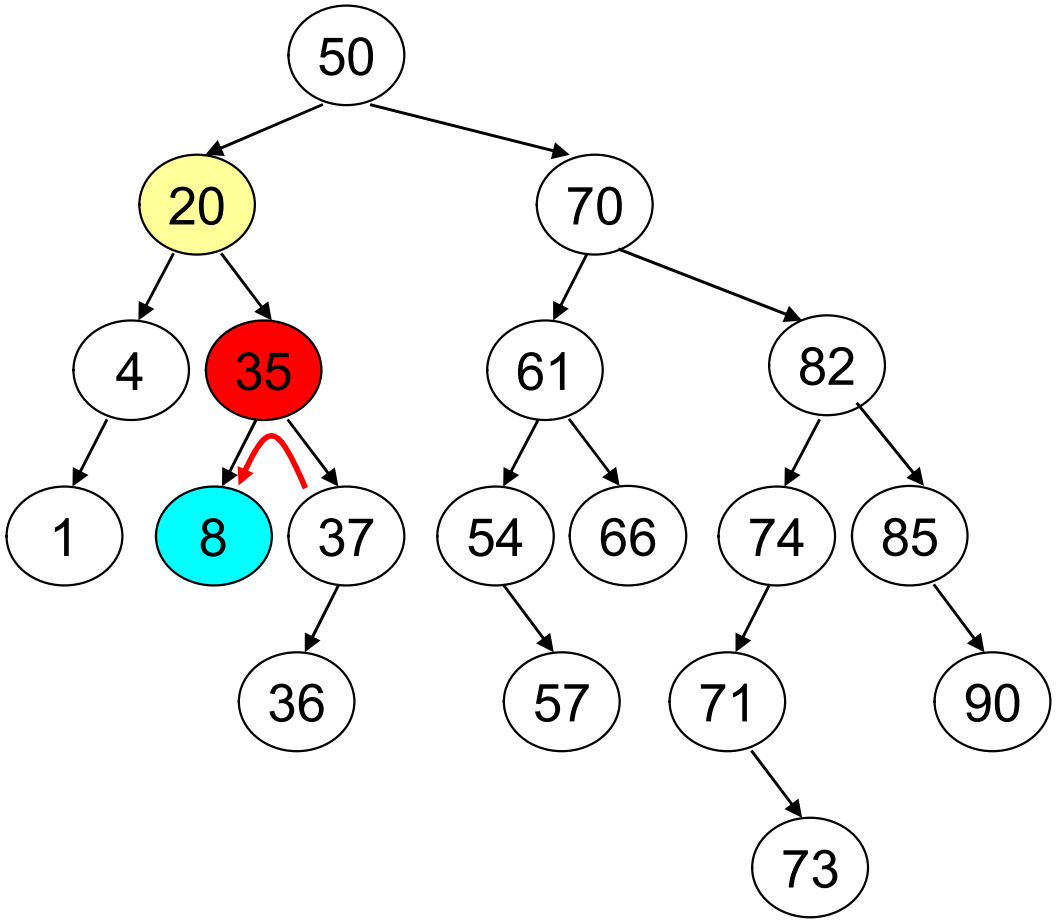
CÂY NHỊ PHÂN CÂN BẰNG

Xóa nút x=20



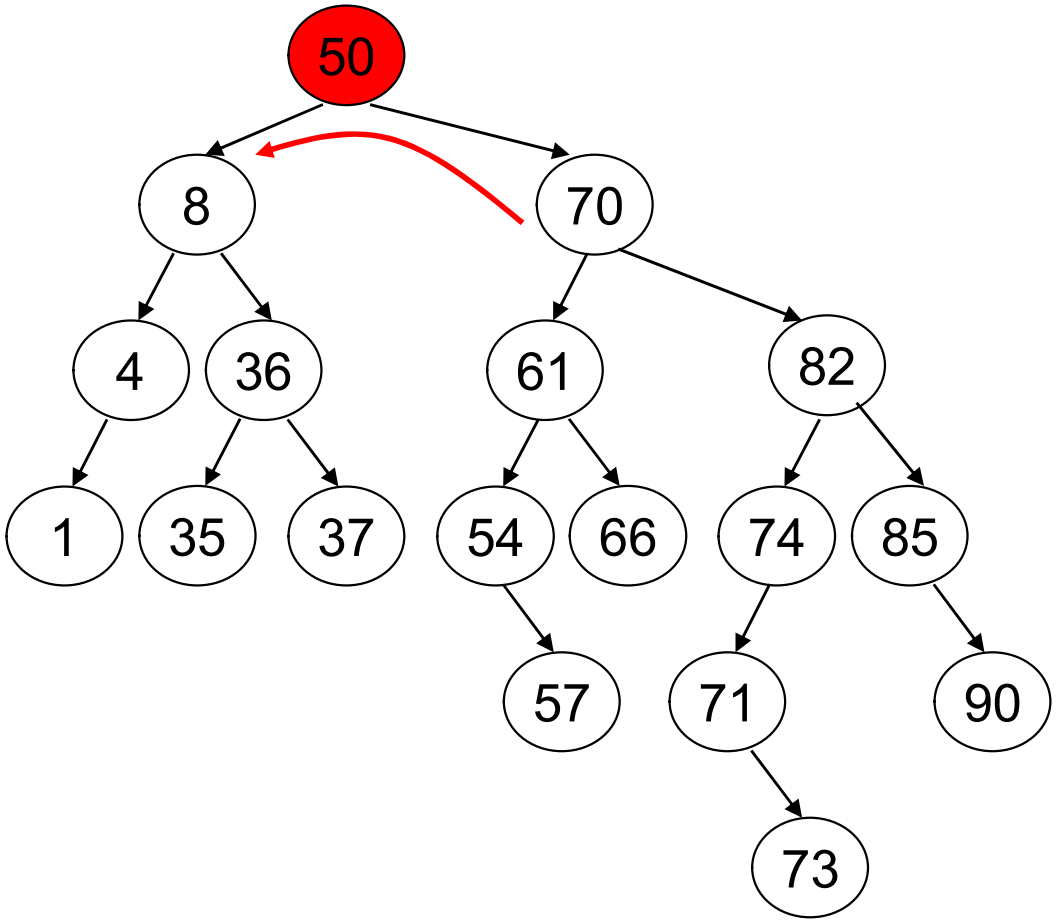
CÂY NHỊ PHÂN CÂN BẰNG

Xóa nút x=20



CÂY NHỊ PHÂN CÂN BẰNG

Xóa nút x=20



CÂY NHỊ PHÂN CÂN BẰNG

Xóa nút x=20