

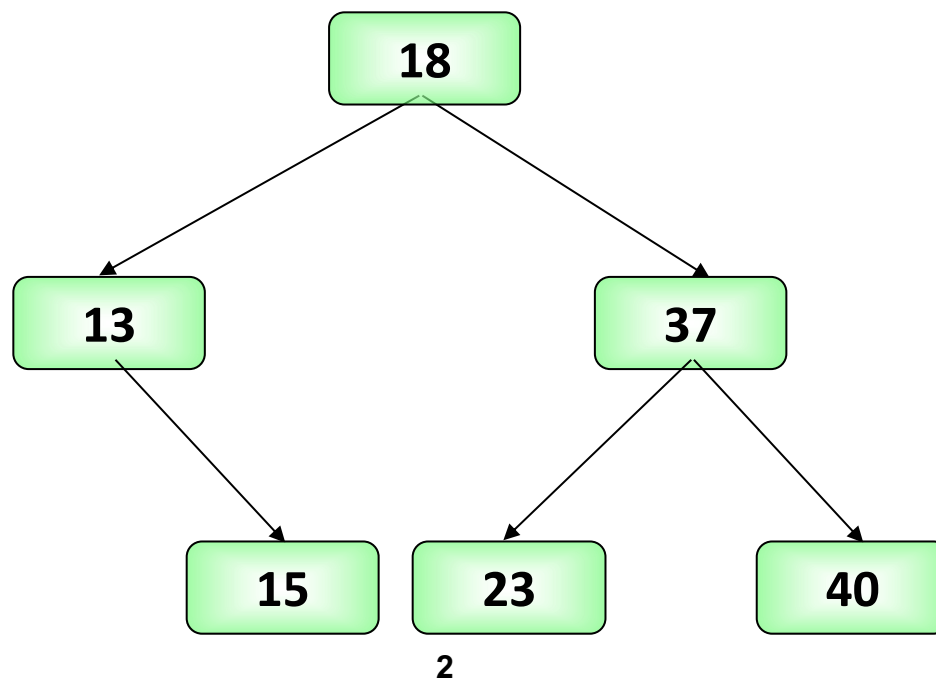
# NỘI DUNG

## CÂY NHỊ PHÂN TÌM KIẾM

# Định nghĩa cây nhị phân tìm kiếm

- Cây nhị phân
- Bảo đảm nguyên tắc bố trí khoá tại mỗi nút:
  - Các nút trong cây trái nhỏ hơn nút hiện hành
  - Các nút trong cây phải lớn hơn nút hiện hành

Ví dụ:



# Ưu điểm của cây nhị phân tìm kiếm

- Nhờ trật tự bố trí khóa trên cây :
  - Định hướng được khi tìm kiếm
- Cây gồm N phần tử :
  - Trường hợp tốt nhất  $h = \log_2 N$
  - Trường hợp xấu nhất  $h = N$
  - Tình huống xảy ra trường hợp xấu nhất ?

# Cấu trúc dữ liệu của cây nhị phân tìm kiếm

- *Cấu trúc dữ liệu của 1 nút*

```
typedef struct tagTNode
```

```
{
```

```
    int    Key; //trường dữ liệu là 1 số nguyên
```

```
    struct tagTNode *pLeft;
```

```
    struct tagTNode *pRight;
```

```
}TNode;
```

- *Cấu trúc dữ liệu của cây*

```
typedef TNode *TREE;
```

# Các thao tác trên cây nhị phân tìm kiếm

- Tạo 1 cây rỗng
- Tạo 1 nút có trường Key bằng x
- Thêm 1 nút vào cây nhị phân tìm kiếm
- Xoá 1 nút có Key bằng x trên cây
- Tìm 1 nút có khoá bằng x trên cây

# Tạo cây rỗng

- Cây rỗng -> địa chỉ nút gốc bằng NULL

```
void CreateTree(TREE &T)
```

```
{
```

```
    T=NULL;
```

```
}
```

# Tạo 1 nút có Key bằng x

```
TNode *CreateTNode(int x)
{
    TNode *p;
    p = new TNode; //cấp phát vùng nhớ động
    if(p==NULL)
        exit(1); // thoát
    else
    {
        p->key      = x; //gán trường dữ liệu của nút = x
        p->pLeft   = NULL;
        p->pRight  = NULL;
    }
    return p;
}
```

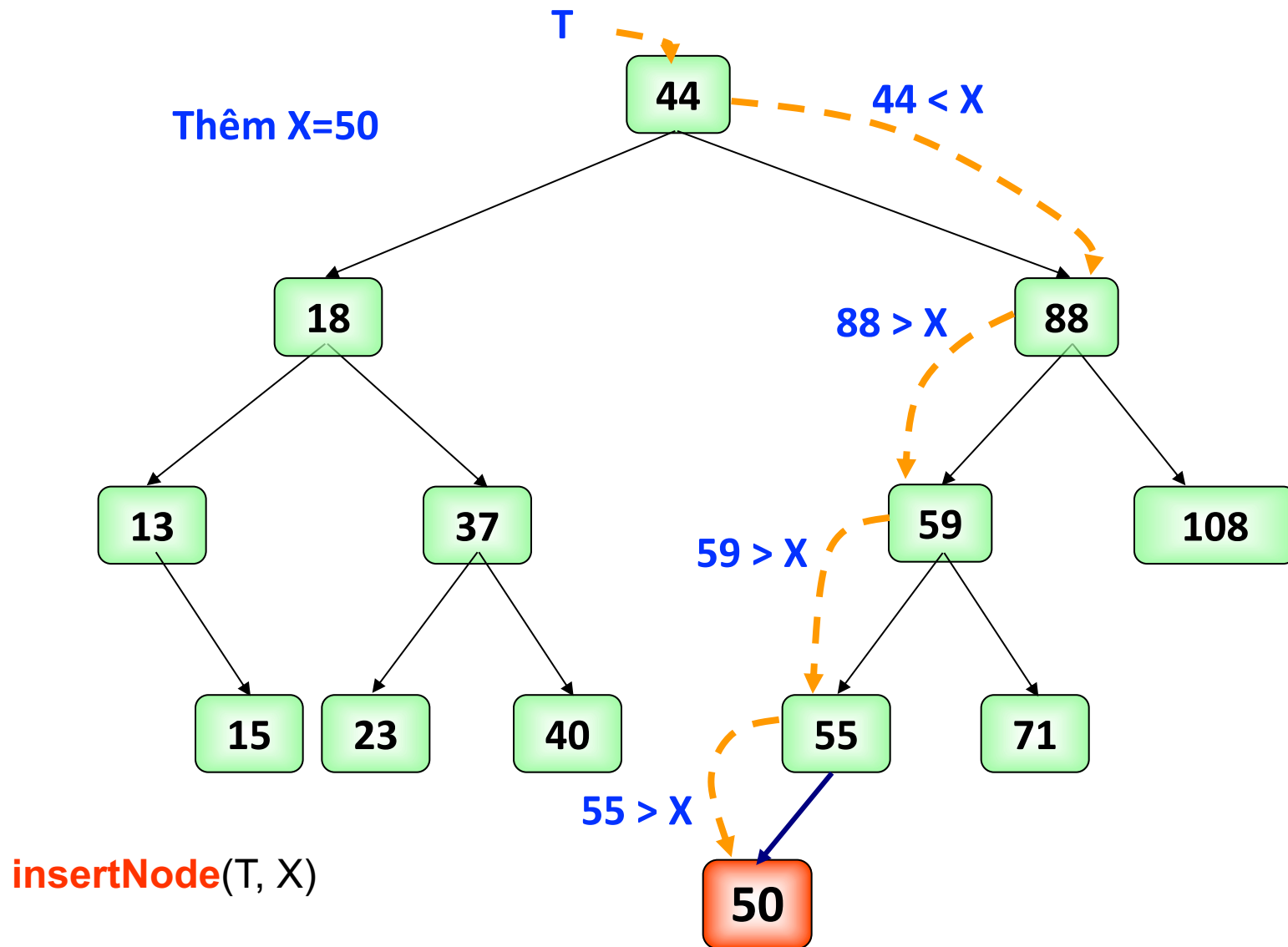
# Thêm một nút x

**Rằng buộ**c: Sau khi thêm cây đảm bảo là cây nhị phân tìm kiếm.

```
int insertNode(TREE &T, Data X)
{
    if (T != NULL)
    {
        if (T->Key == X)    return 0;
        if (T->Key > X)    return insertNode(T->pLeft, X);
        else                return insertNode(T->pRight, X);
    }
    T = CreateTNode(X);
    return 1;
}
```



# Minh họa thêm 1 phần tử vào cây



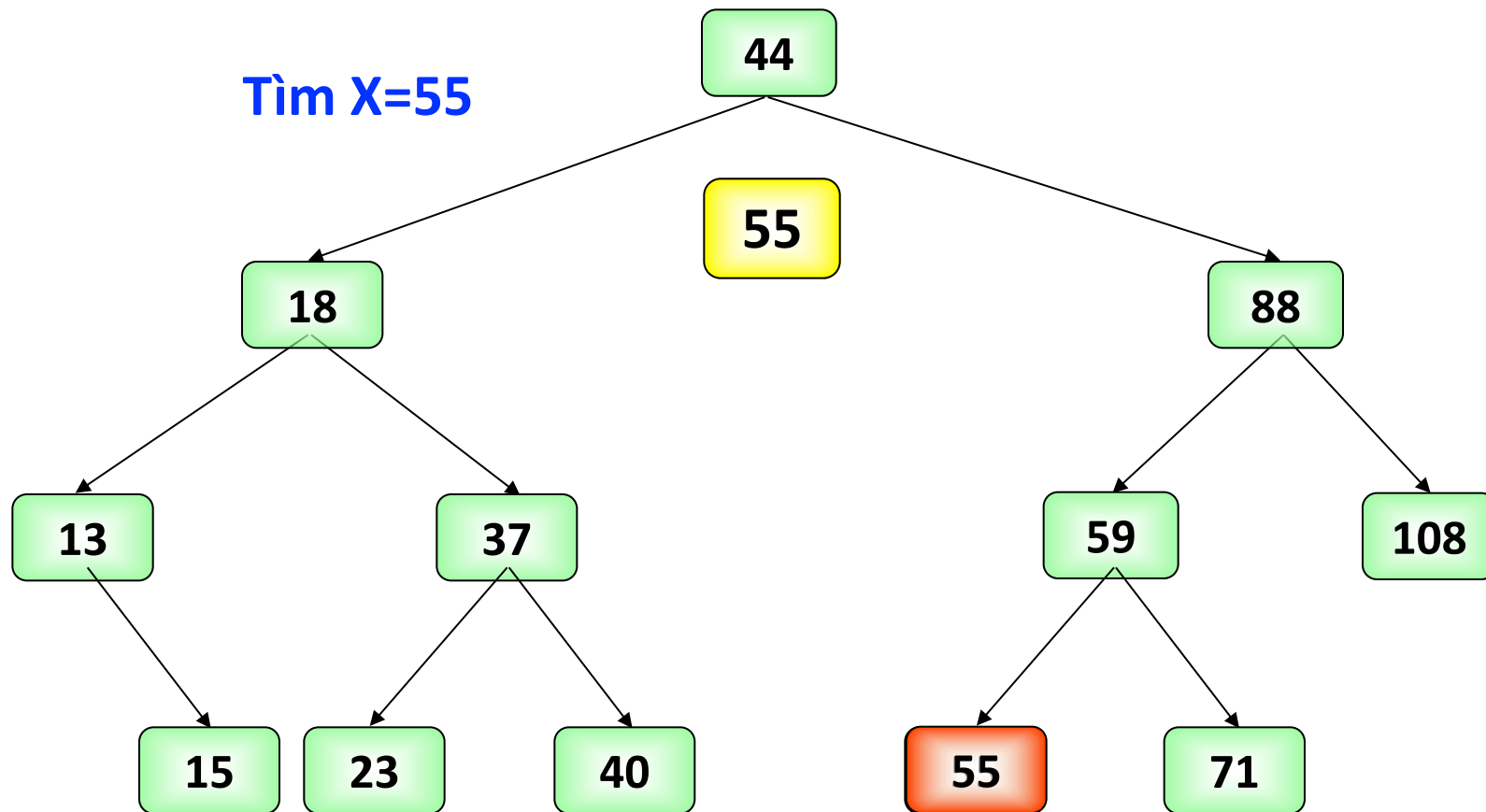
# Tìm nút có khoá bằng x (dùng đệ quy)

```
TNode *SearchTNode(TREE T, int x)
{
    if(T!=NULL)
    {
        if(T->key==x)
            return T;
        else if(x>T->key)
            return SearchTNode(T->pRight,x);
        else
            return SearchTNode(T->pLeft,x);
    }
    return NULL;
}
```

Tìm nút có khoá bằng x (không dùng đệ quy)

```
TNode * searchNode(TREE Root, Data x)
{
    Node *p = Root;
    while (p != NULL)
    {
        if(x == p->Key)        return p;
        else if(x < p->Key)    p = p->pLeft;
        else                  p = p->pRight;
    }
    return NULL;
}
```

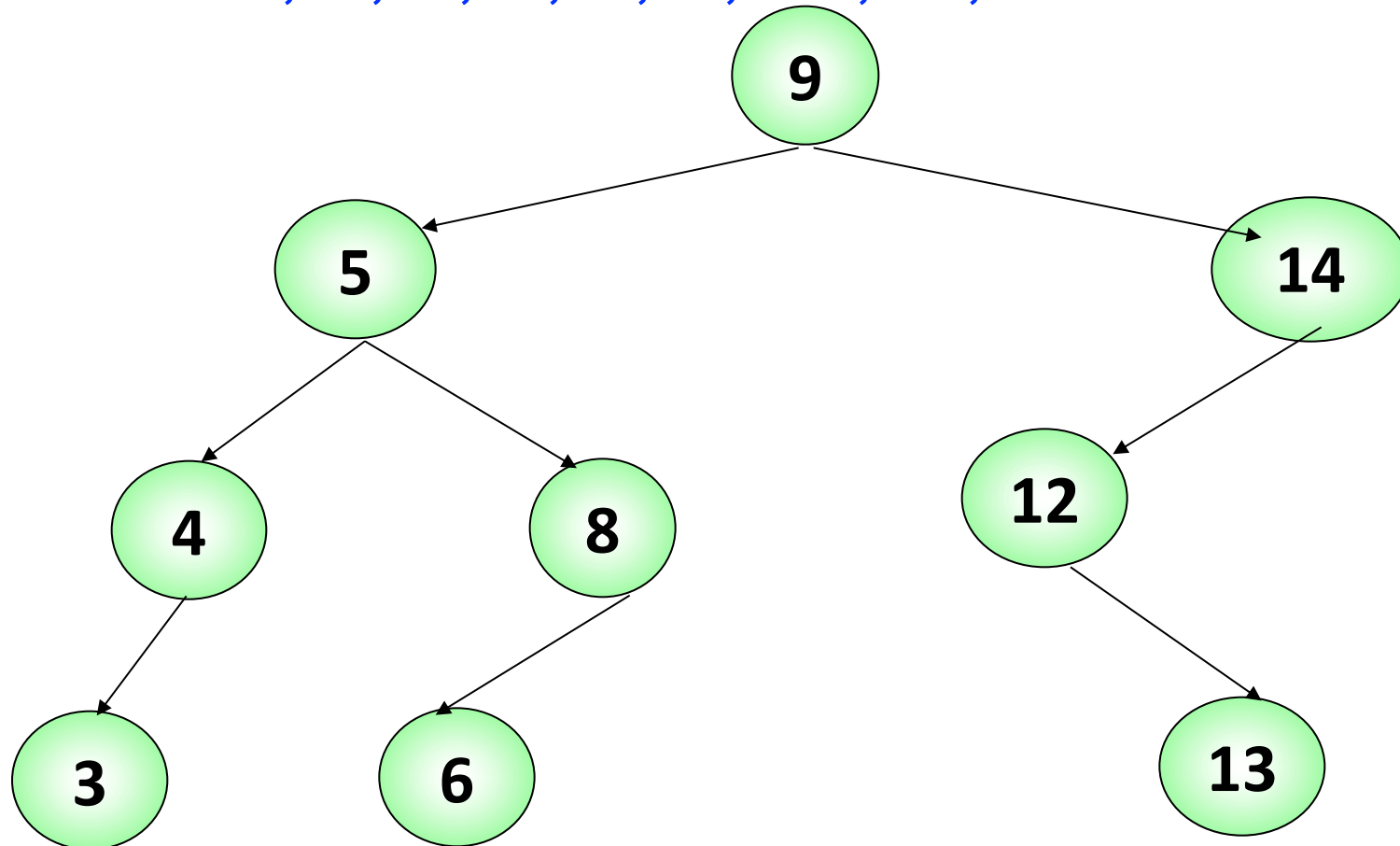
# Minh họa tìm một nút



Tìm thấy  $X=55$

# Minh họa thành lập 1 cây từ dãy số

9, 5, 4, 8, 6, 3, 14, 12, 13

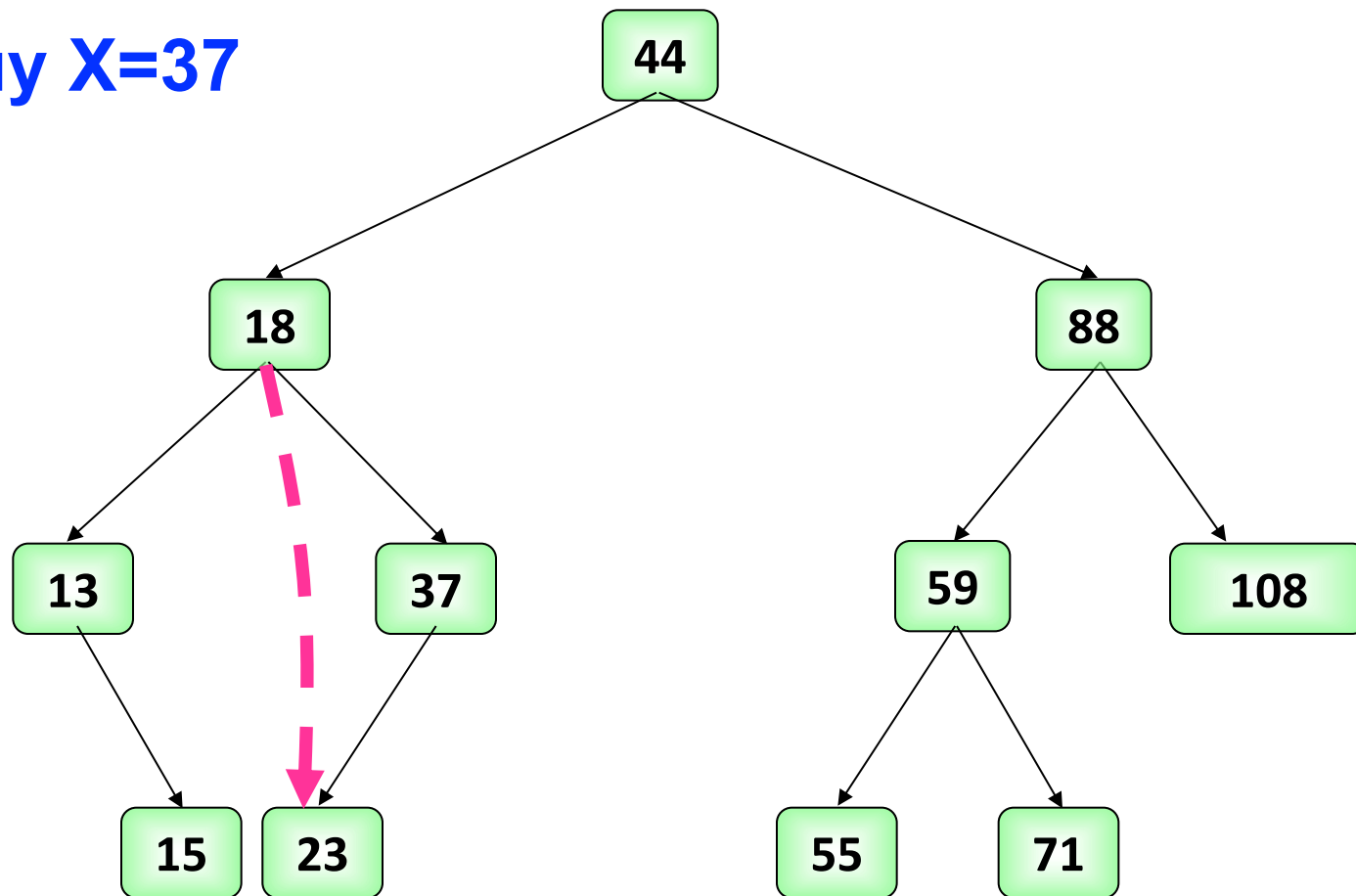


# Hủy 1 nút có khoá bằng X trên cây

- Hủy 1 phần tử trên cây phải đảm bảo điều kiện ràng buộc của Cây nhị phân tìm kiếm
- Có 3 trường hợp khi hủy 1 nút trên cây
  - TH1: X là nút lá
  - TH2: X chỉ có 1 cây con (cây con trái hoặc cây con phải)
  - TH3: X có đầy đủ 2 cây con
- TH1: Xoá nút lá mà không ảnh hưởng đến các nút khác trên cây
- TH2: Trước khi xoá X ta móc nối cha của X với con duy nhất của X.
- TH3: Dùng cách xoá gián tiếp

# Minh họa hủy phần tử x có 1 cây con

Hủy  $X=37$



# Hủy 1 nút có 2 cây con

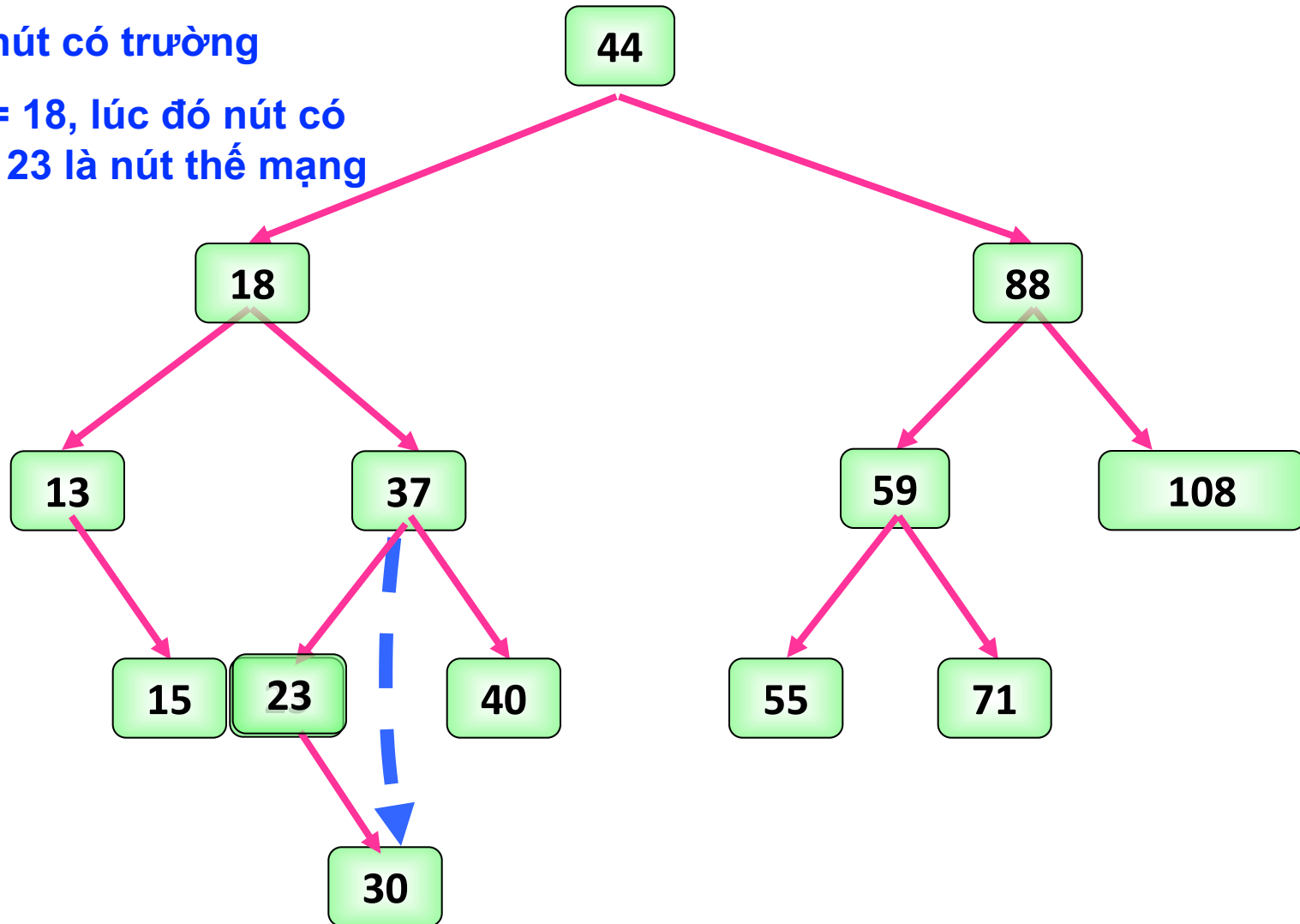
- Ta dùng cách hủy gián tiếp, do X có 2 cây con
- Thay vì hủy X ta tìm phần tử thế mạng Y. Nút Y có tối đa 1 cây con.
- Thông tin lưu tại nút Y sẽ được chuyển lên lưu tại X.
- Ta tiến hành xoá hủy nút Y (xoá Y giống 2 trường hợp đầu)
- Cách tìm nút thế mạng Y cho X: Có 2 cách
  - C1: Nút Y là nút có khoá nhỏ nhất (trái nhất) bên cây con phải X
  - C2: Nút Y là nút có khoá lớn nhất (phải nhất) bên cây con trái của X



# Minh họa hủy phần tử X có 2 cây con

Xoá nút có trường

Key = 18, lúc đó nút có  
khoá 23 là nút thế mạng



# Cài đặt thao tác xóa nút có trường Key = x

```
void DeleteNodeX1(TREE &T,int x)
{
    if(T!=NULL)
    {
        if(T->Key < x)    DeleteNodeX1(T->Right, x);
        else if(T->Key>x) DeleteNodeX1(T->Left,  x);
        else //tim thấy Node có trường dữ liệu = x
        {
            TNode *p = T;
            if (T->Left ==NULL) T = T->Right;
            else if(T->Right==NULL) T=T->Left;
            else
                ThayThe1(p, T->Right);// tìm bên cây con phải

            delete p;
        }
    }
    else
        printf("Khong tim thay phan can xoa tu");
}
```

# Hàm tìm phần tử thể mạng

```
void ThayThe1(TREE &p, TREE &T)
{
    if(T->Left!=NULL)
        ThayThe1(p,T->Left);
    else
    {
        p->Key = T->Key;    // Gán lại giá trị node bị
        xoá
        p      = T;        // Trỏ đến node bị xoá thực sự
        T      = T->Right;  // Cô lập node bị xoá
    }
}
```

# Bài tập

- Cài đặt để nhập một danh sách các số nguyên vào Cây NPTK cho đến khi gặp số 0.
- In ra tình trạng cây.
- Tìm giá trị X (nhập từ bàn phím) xem có trên cây hay không?
- Nếu có X, hãy xoá nó khỏi cây và in ra tình trạng cây.
  - Định nghĩa      - Duyệt cây
  - Tạo cây, node, insert node
  - Tìm x    - Xoá node

# Bài tập

```
• void main( ){  
  TREE T; CreateTree(T);  
  while( 1 ) {  
    X = ?  
    if (X == 0) break;  
    InsertNode(T, X);  
  }  
  DuyetLNR( T );  
  int h = DoCaoCay(T );  
  printf("Do cao cua cay%d", h);
```

# Bài tập

$X = ?$

```
TNode p = SearchNode( T, X );
```

```
if (p == NULL)
```

```
    printf("Ko co");
```

```
else {
```

```
    DeleteNode(T, X);
```

```
    DuyetLNR( T );
```

```
}
```