

CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

Cây nhị phân

- 1. Giới thiệu cây nhị phân**
- 2. Biểu diễn cây nhị phân**
- 3. Các thao tác trên cây nhị phân**
- 4. Ứng dụng của cây nhị phân**
- 5. Cây nhị phân tìm kiếm**
- 6. Cây nhị phân tìm kiếm cân bằng**

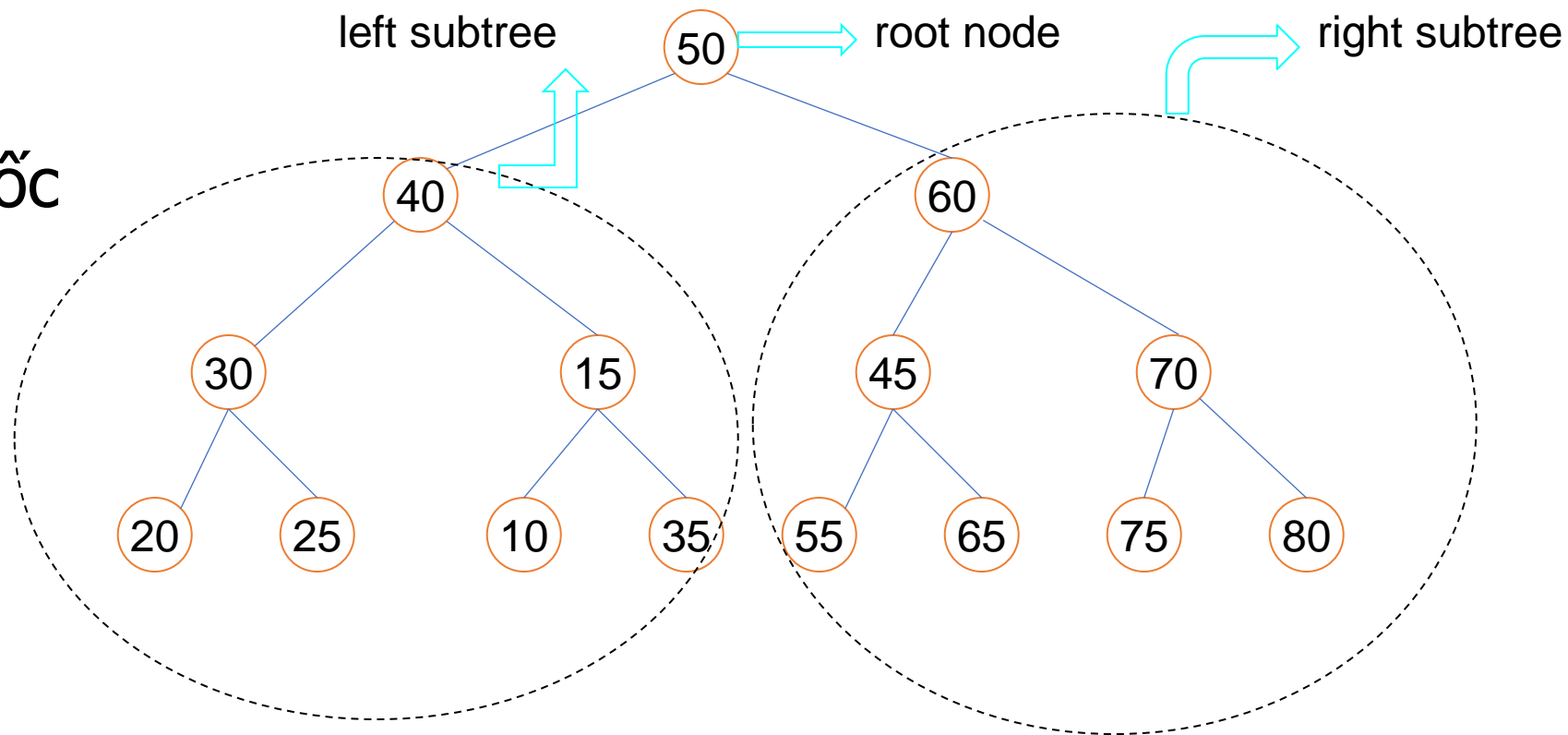
Định nghĩa

Tập hợp hữu hạn các node có cùng kiểu dữ liệu (có thể là tập \emptyset) được phân thành 3 tập con:

Một node gọi là node gốc

Cây con bên trái

Cây con bên phải



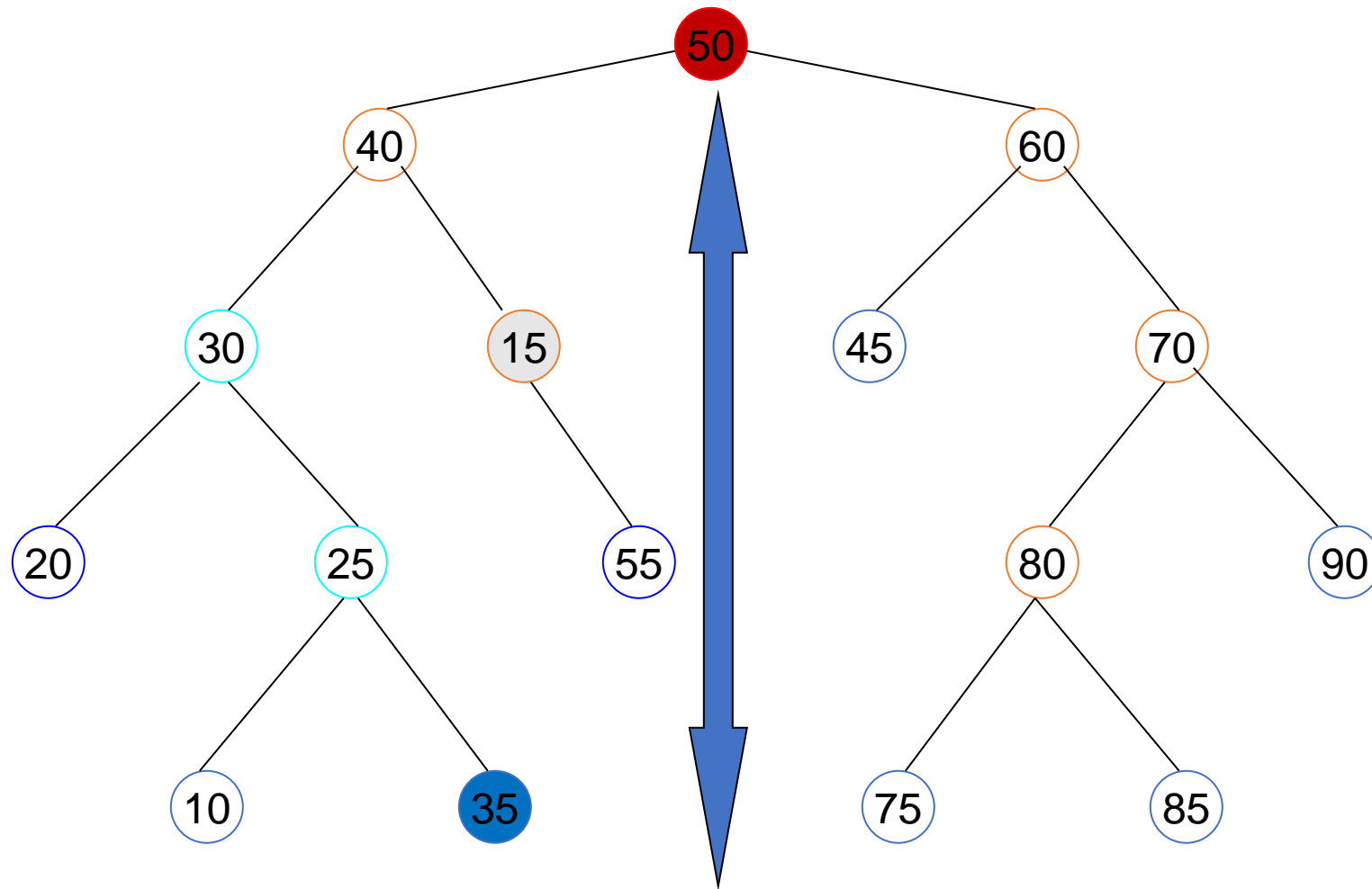
Một số khái niệm

- Node gốc (Root) là node đầu tiên định hình cây.
- Node cha (Father): node A là node cha của node B nếu B hoặc là node con bên trái của node A (left son) hoặc B là node con bên phải của node A (right son).
- Node lá (Leaf): node không có node con trái, không có node con phải.
- Node trung gian (Internal Node): node hoặc có node con trái, hoặc có node con phải, hoặc cả hai hoặc cả hai.
- Node trước (Ancestor): node A gọi là node trước của node B nếu cây con node gốc là A chứa node B.

Một số khái niệm

- Node sau trái (left descendent): node B là node sau bên trái của node A nếu cây con bên trái của node A chứa node B.
- Node sau phải (right descendent): node B là node sau bên phải của node A nếu cây con bên phải của node A chứa node B.
- Node anh em (brother): A và B là anh em nếu cả A và B là node con trái và node con phải của cùng một node cha.
- Bậc của node (degree of node): Số cây con tối đa của node.
- Mức của node (level of node): mức node gốc có bậc là 1, mức của các node khác trên cây bằng mức của node cha cộng thêm 1.
- Chiều sâu của cây (depth of tree): mức lớn nhất của node lá trong cây.

Giới thiệu cây nhị phân



- Node gốc
- Node trung gian
- Node lá
- Độ sâu của cây

Các loại cây nhị phân

Cây lệch trái: Cây chỉ có node con bên trái.

Cây lệch phải: Cây chỉ có node con bên phải.

Cây nhị phân đúng (strickly binary tree): Node gốc và tất cả các node trung gian có đúng hai node con.

Cây nhị phân đầy (complete binary tree): Cây nhị phân đúng và tất cả node lá đều có mức là d .

Cây nhị phân gần đầy (almost complete binary tree): Tất cả node con có mức không nhỏ hơn $d-1$ đều có hai node con. Các node ở mức d đầy từ trái qua phải.

Cây nhị phân hoàn toàn cân bằng: Số node thuộc nhánh con trái và số node thuộc nhánh con phải tại mỗi node chênh lệch nhau không quá 1.

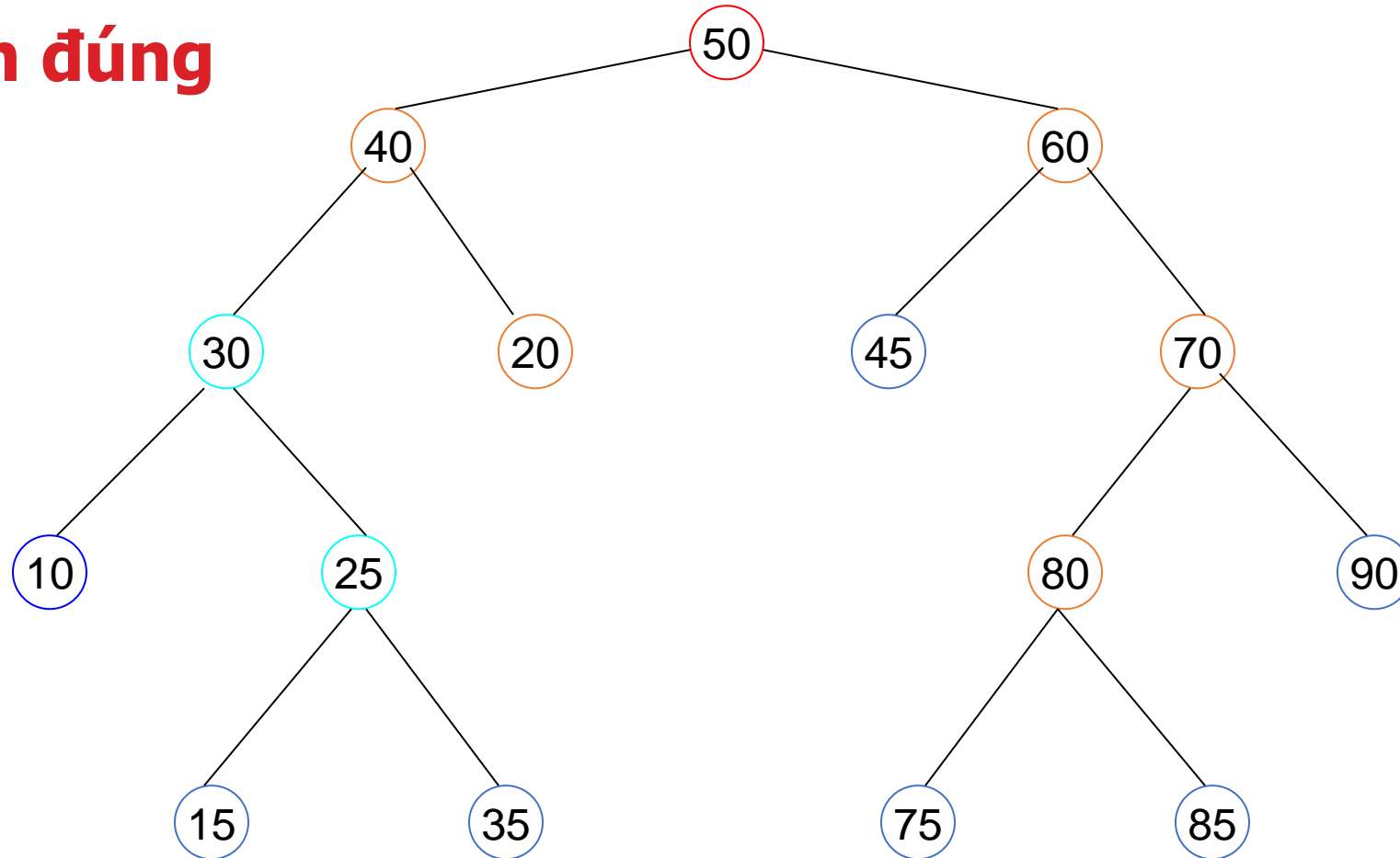
Các loại cây nhị phân

Cây nhị phân tìm kiếm. Cây nhị phân thỏa mãn điều kiện:

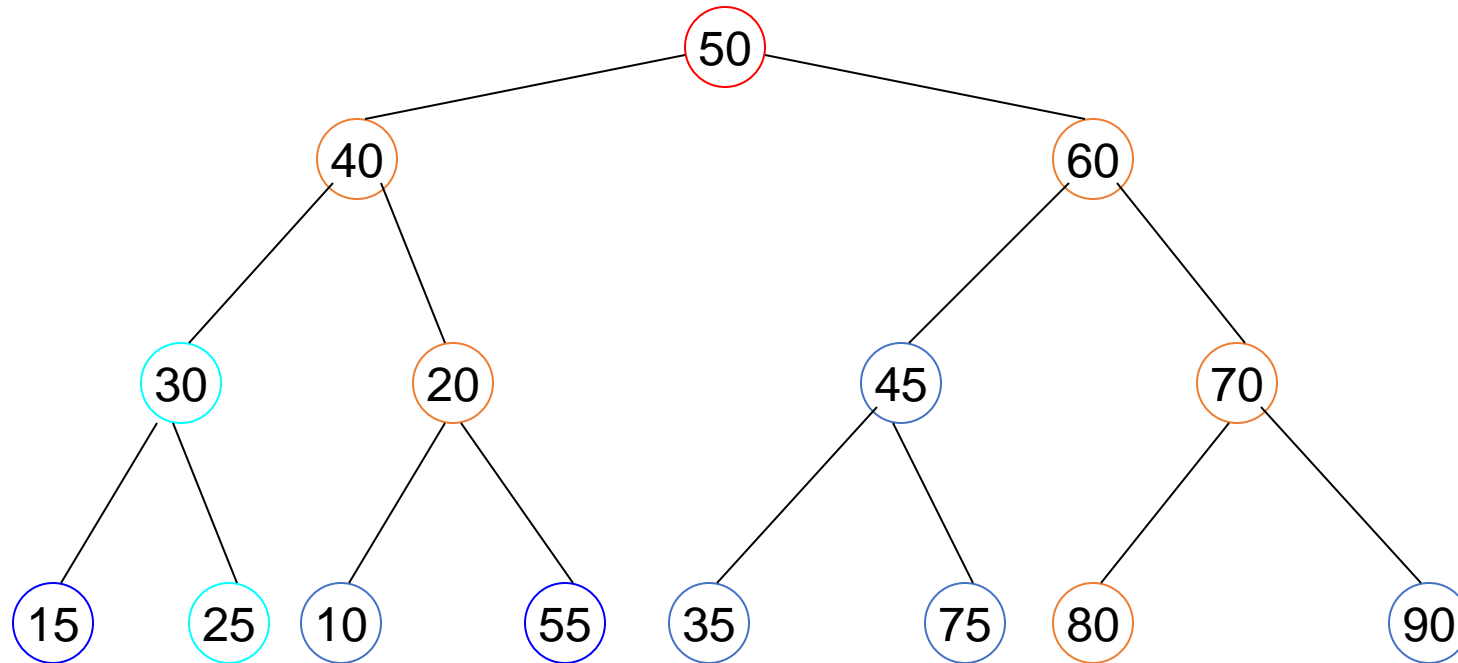
1. Hoặc là rỗng hoặc có một node gốc.
2. Mỗi node gốc có tối đa hai cây con. Nội dung node gốc lớn hơn nội dung node con bên trái và nhỏ hơn nội dung node con bên phải.
3. Hai cây con bên trái và bên phải cũng hình thành nên hai cây tìm kiếm.

Cây nhị phân tìm kiếm cân bằng: Chiều sâu cây con trái và chiều sâu cây con phải chênh lệch nhau không quá 1.

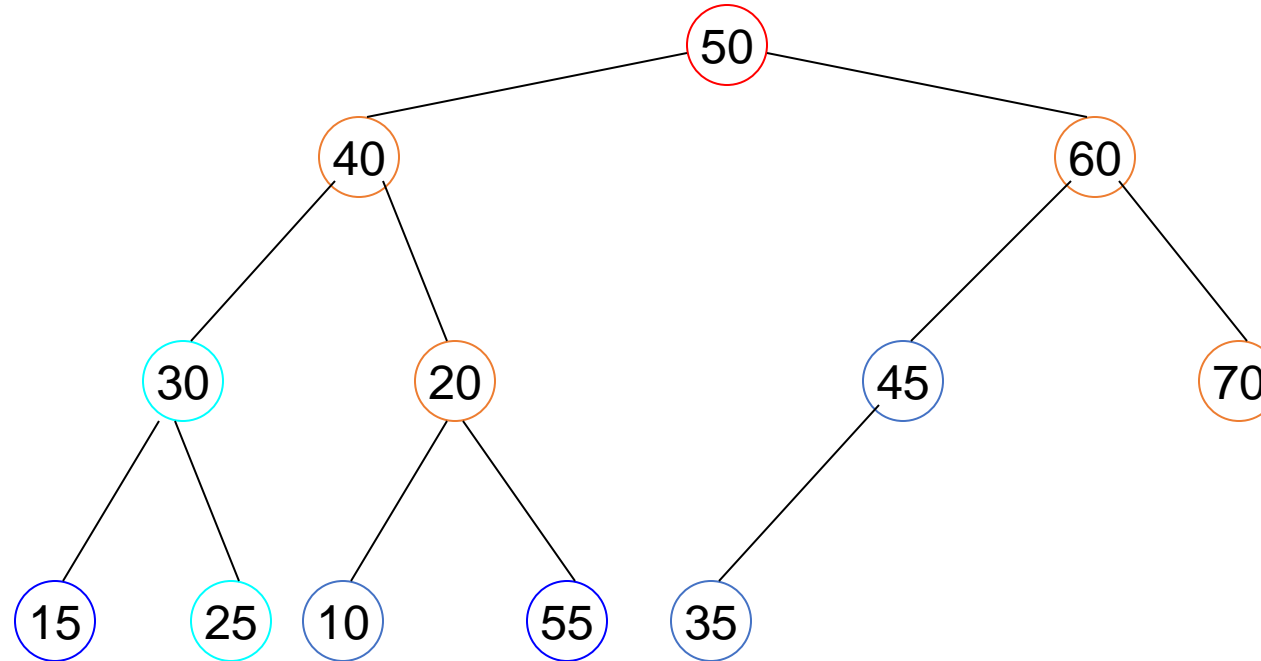
Cây nhị phân đúng



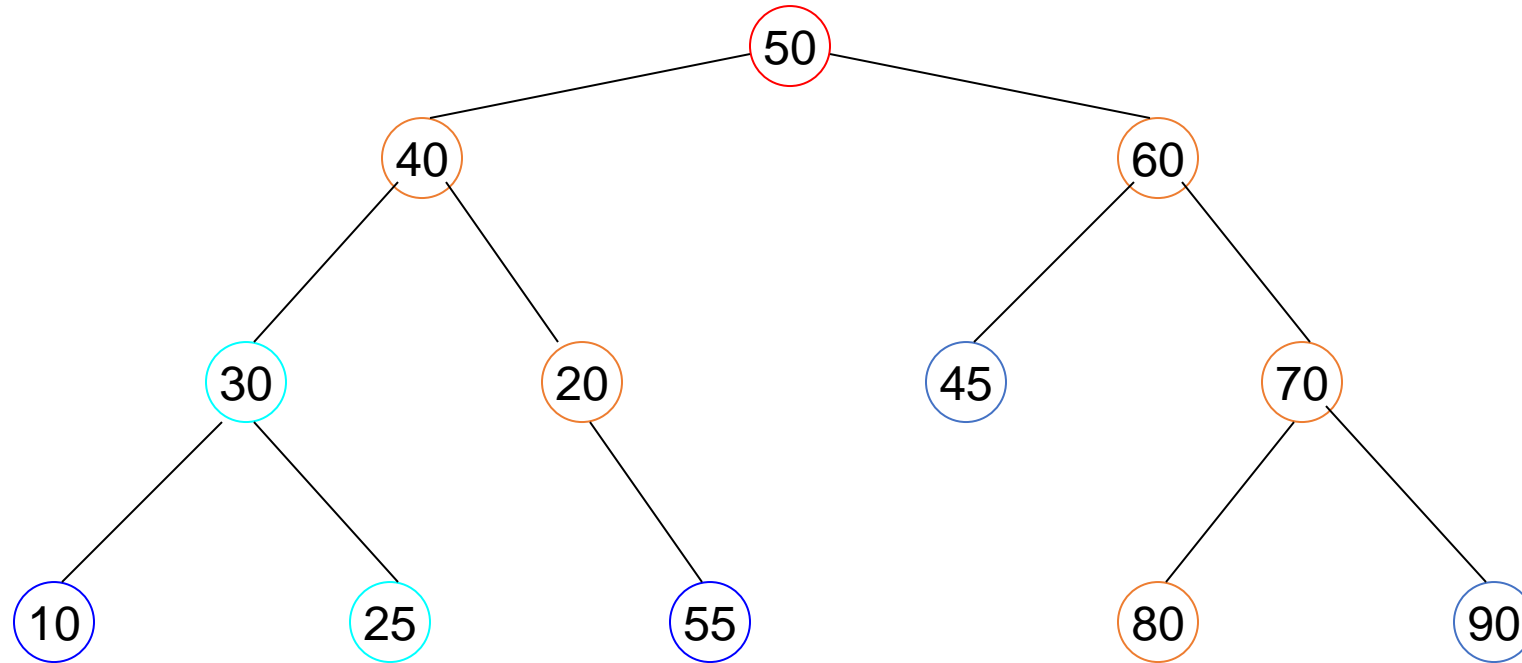
Cây nhị phân đây



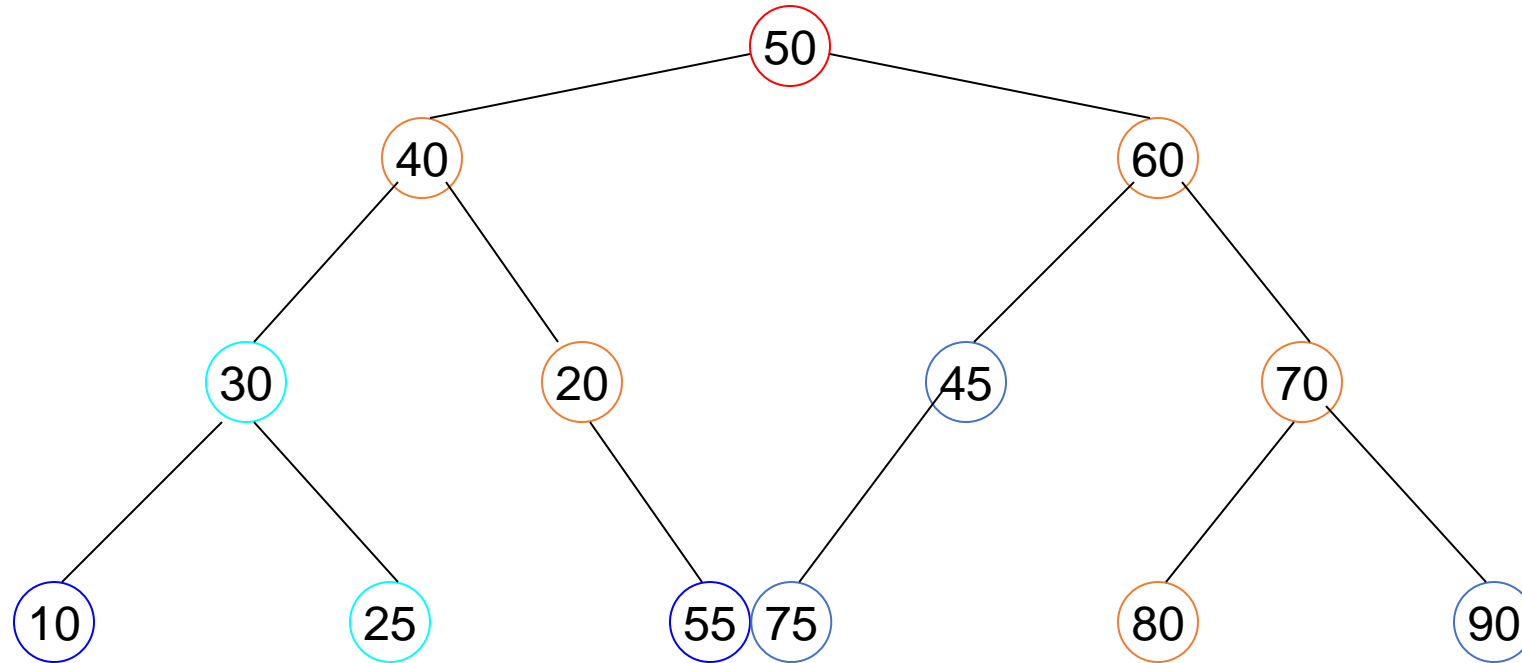
Cây nhị phân gần đây



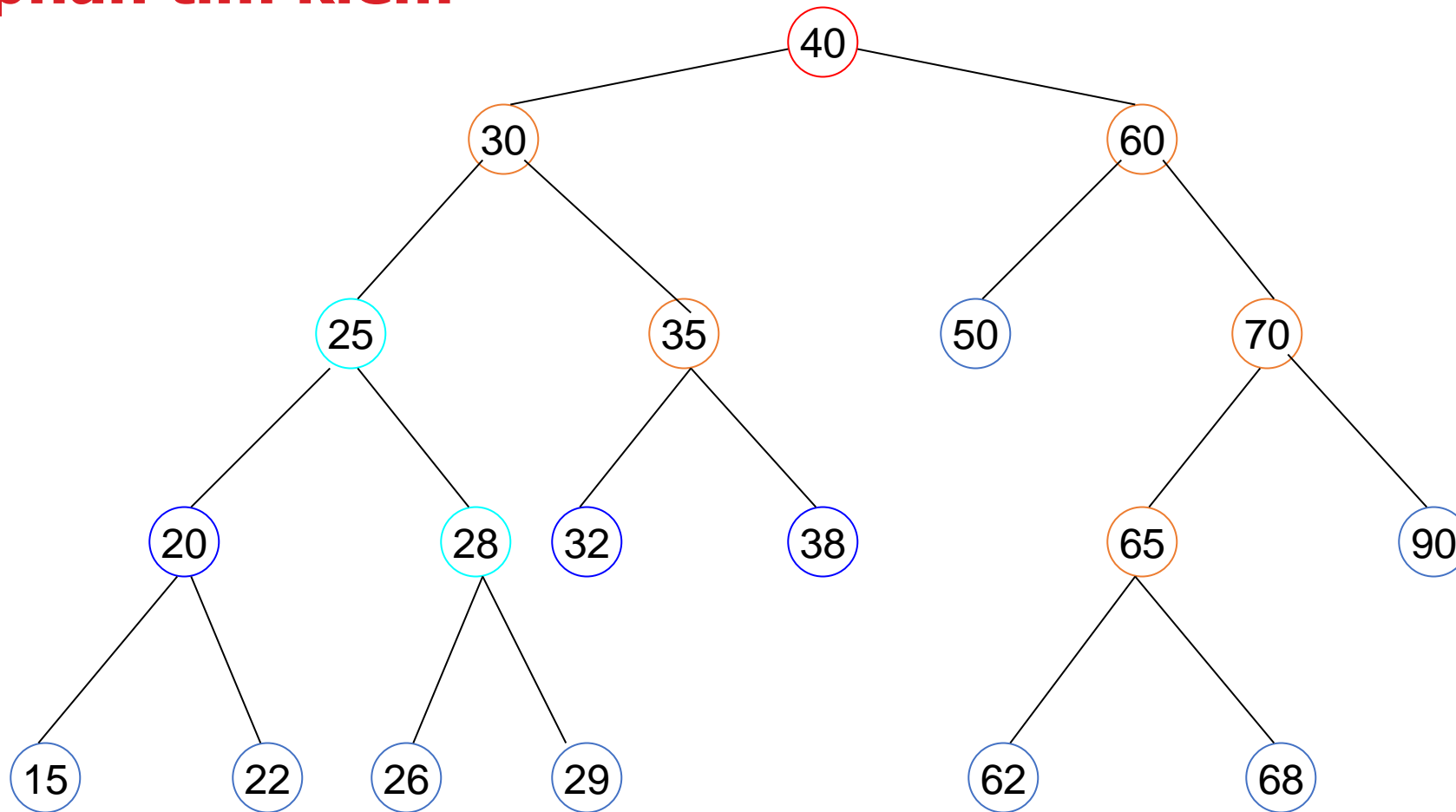
Cây nhị phân cân bằng



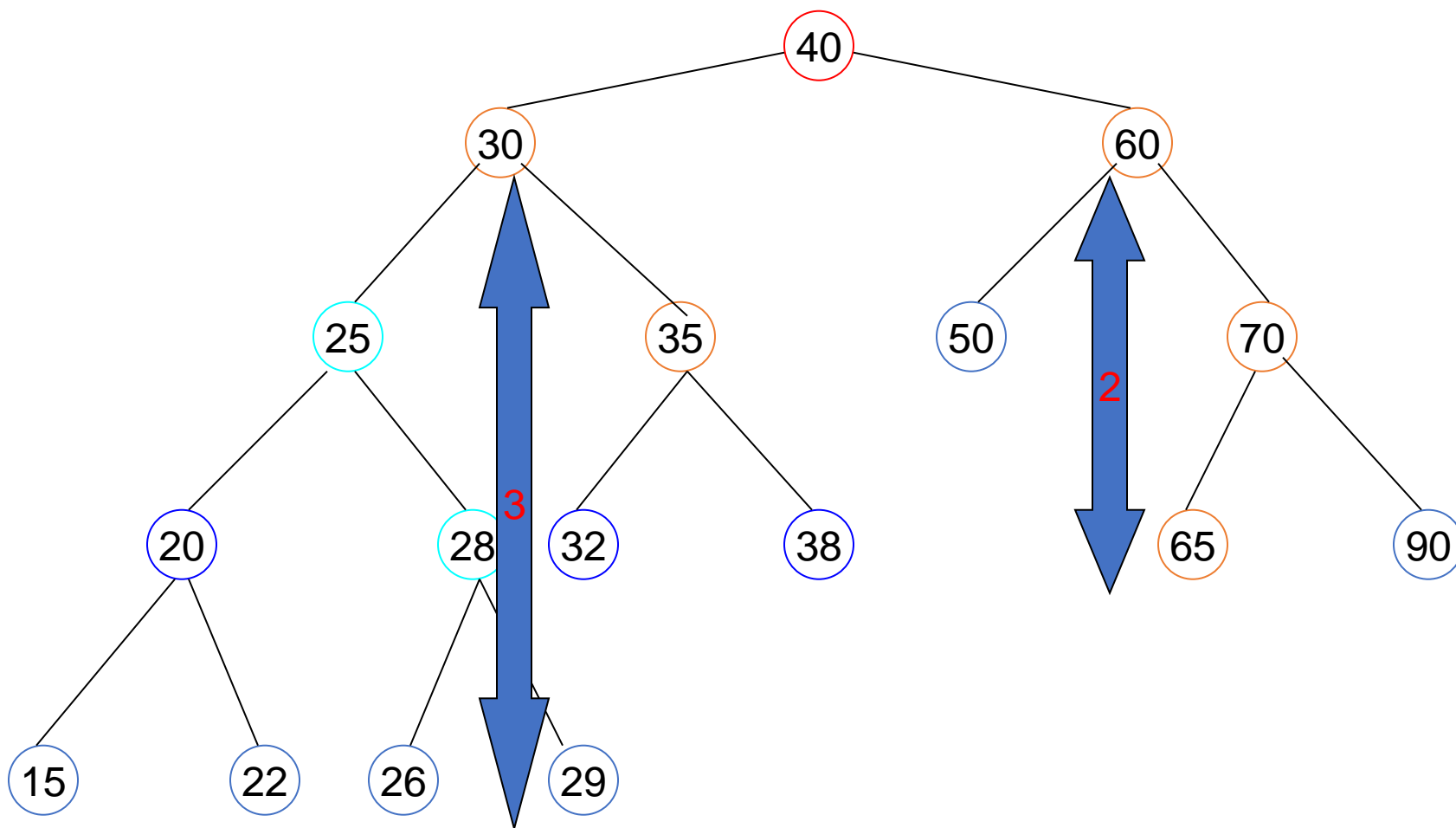
Cây nhị phân cân bằng hoàn toàn



Cây nhị phân tìm kiếm



Cây nhị phân tìm kiếm cân bằng

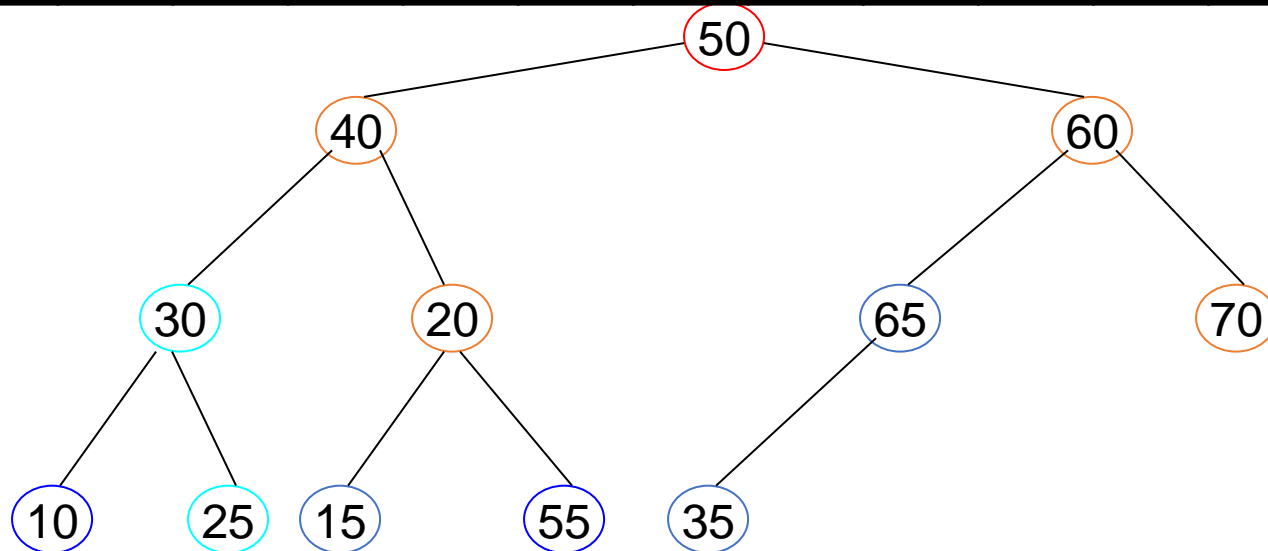


Biểu diễn liên tục sử dụng mảng

Node gốc: Lưu trữ ở vị trí 0.

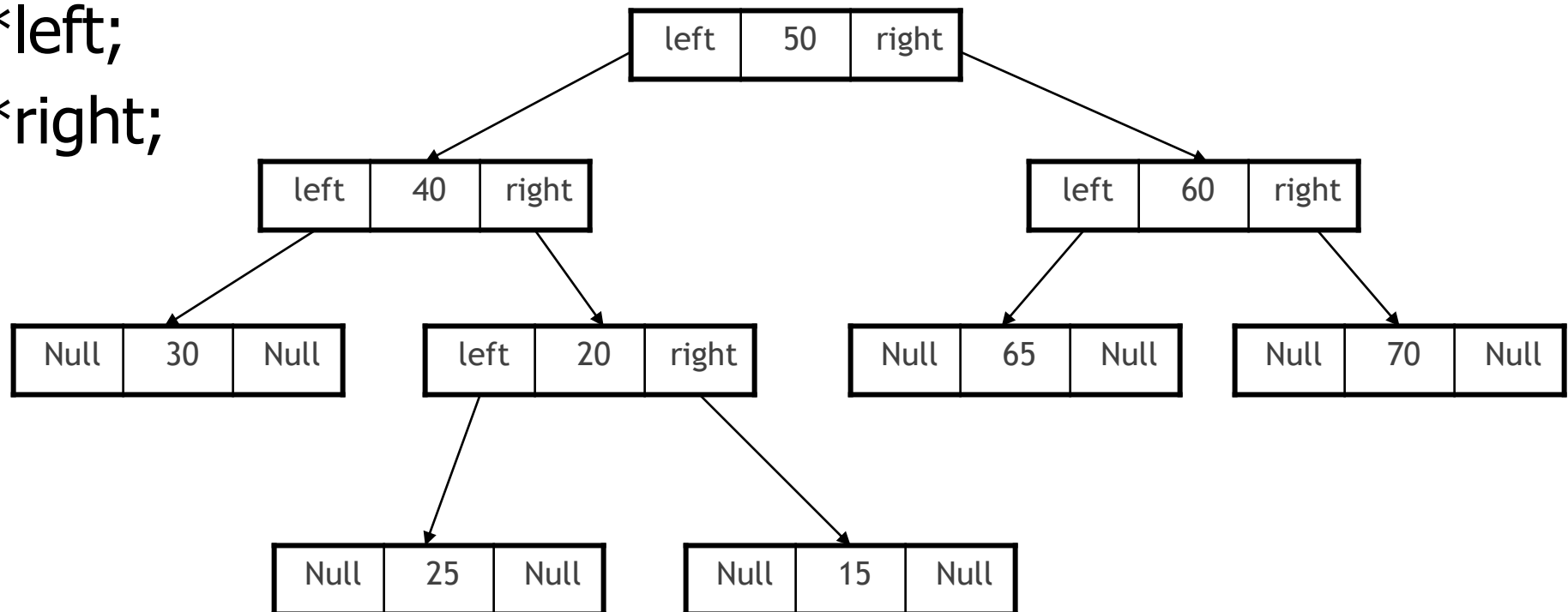
Nếu node cha lưu trữ ở vị trí p thì node con bên trái của nó được lưu trữ ở vị trí $2p+1$, node con phải được lưu trữ ở vị trí $2p+2$.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	...
50	40	60	30	20	65	70	10	25	15	55	35	∅	∅	∅



Biểu diễn rỗi rạc sử dụng danh sách liên kết

```
typedef struct node {  
    Item infor;  
    struct node *left;  
    struct node *right;  
} *Tree;
```



1. Tạo node gốc cho cây.
2. Thêm vào node lá bên trái node p.
3. Thêm vào node lá bên phải node p.
4. Loại bỏ node lá bên trái node p.
5. Loại bỏ node lá bên phải node p.
6. Loại bỏ cả cây.
7. Tìm kiếm node trên cây.
8. Duyệt cây theo thứ tự trước.
9. Duyệt cây theo thứ tự giữa.
10. Duyệt cây theo thứ tự sau.

Khởi tạo cây nhị phân: Tạo lập một cây nhị phân ở trạng thái rỗng.

```
void Init(Tree *T){  
    *T=NULL; //Đưa cây T về trạng thái rỗng  
}
```

Cấp phát miền nhớ cho một node:

```
Tree GetNode(){  
    Tree p;//Khai báo một node kiểu Tree  
    p=new node; //Cấp phát miền nhớ cho node  
    return (p); //Trả lại node được cấp phát  
}
```

Giải phóng một node p cho cây T

```
void FreeNode(Tree p){  
    delete (p); //giải phóng node p  
}
```

Kiểm tra tính rỗng của cây T:

```
int isEmpty(Tree *T){  
    if(*T==NULL) //nếu T rỗng  
        return 1; //trả lại giá trị đúng  
    return 0; //trả lại giá trị sai  
}
```

Tạo một node cho cây T

```
Tree MakeNode( Item x){ //x là giá trị node cần bổ sung vào cây
    Tree p; //Khai báo một node kiểu Tree
    p = GetNode();//cấp phát miền nhớ cho p
    p->infor = x;      //thiết lập thành phần thông tin cho node
    p->left=NULL;      //tạo liên kết trái cho node
    p->right=NULL;     //Tạo liên kết phải cho node
    return (p); //trả lại node được tạo ra
}
```

Tìm node p có nội dung là x trên cây

```
Tree Search(Tree T, Item x) {  
    Tree p;  
    if(T==NULL) //Nếu T rỗng  
        return NULL; //trả lại giá trị NULL  
    if( T->infor==x)// Nếu node gốc là node cần tìm  
        return T; //trả lại node gốc  
    p = Search(T->left, x); //Tìm ở nhánh cây con trái  
    if ( p ==NULL) // nếu không thấy p ở nhánh cây con bên trái  
        p = Search(T->right,x); // tìm p ở nhánh cây con phải  
    return p;  
}
```

Tạo node gốc cho cây

```
Tree MakeRoot(Tree T, Item x){  
    if(T==NULL) {  
        T = MakeNode(x);  
    }  
    return T;  
}
```

Tạo node lá bên trái node có nội dung là x

```
void Add-Left(Tree T, Item x, Item y ) {  
    Tree p, q;  
    p = Search (T, x); //tìm node có nội dung là x trên cây  
    if (p==NULL ) { //nếu node có nội dung x không có trên cây  
        <thông báo node cha x không có thực>;  
        return; //không thể thêm được node lá trái  
    } else if ( (p -> left) !=NULL )) //nếu p đã có nhánh cây con bên trái  
        <thông báo node cha x có nhánh cây con trái>;  
        return; // không thêm được node lá trái  
    else {  
        q = MakeNode(y); // tạo node lá trái của p là q.  
        p -> left =q; //Node lá bên trái của p là q  
    }  
}
```


Tạo node lá bên phải node có nội dung là x

```
void Add-Right(Tree T, Item x, Item y ) {  
    Tree p, q;  
    p = Search (T, x); //tìm node có nội dung là x trên cây  
    if (p==NULL ) { //nếu node có nội dung x không có trên cây  
        <thông báo node cha x không có thực>;  
        return; //không thể thêm node lá phải  
    }  
    else if ( (p -> right !=NULL ) //nếu p đã có nhánh cây con bên phải  
        <thông báo node cha x có nhánh cây con phải>;  
        return; // không thêm được node lá phải  
    else {  
        q = MakeNode(y); // tạo node lá phải của p là q.  
        p -> right =q; //Node lá bên phải của p là q  
    }  
}
```

Loại bỏ node lá bên trái node có nội dung là x

```
void Remove-Left(Tree T, int x ) {  
    Tree p, q;  
    p = Search (T, x); //tìm node có nội dung là x trên cây  
    if (p==NULL ) { //nếu node có nội dung x không có trên cây  
        <thông báo node cha x không có thực>;  
        return; //không thể loại bỏ được node lá trái  
    } else if ( (p -> left)->right !=NULL || (p -> left)->left !=NULL ) //nếu p có nhánh cây con bên  
trái  
        <thông báo node cha x có nhánh cây con trái>;  
        return; // không thêm loại bỏ được cây con trái  
    else {  
        q = p ->left; // node lá trái là q.  
        p -> left =NULL; //ngắt liên kết cho node p  
        delete(q); //Loại bỏ node lá trái q  
    }  
}
```

Loại bỏ node lá bên phải node có nội dung là x

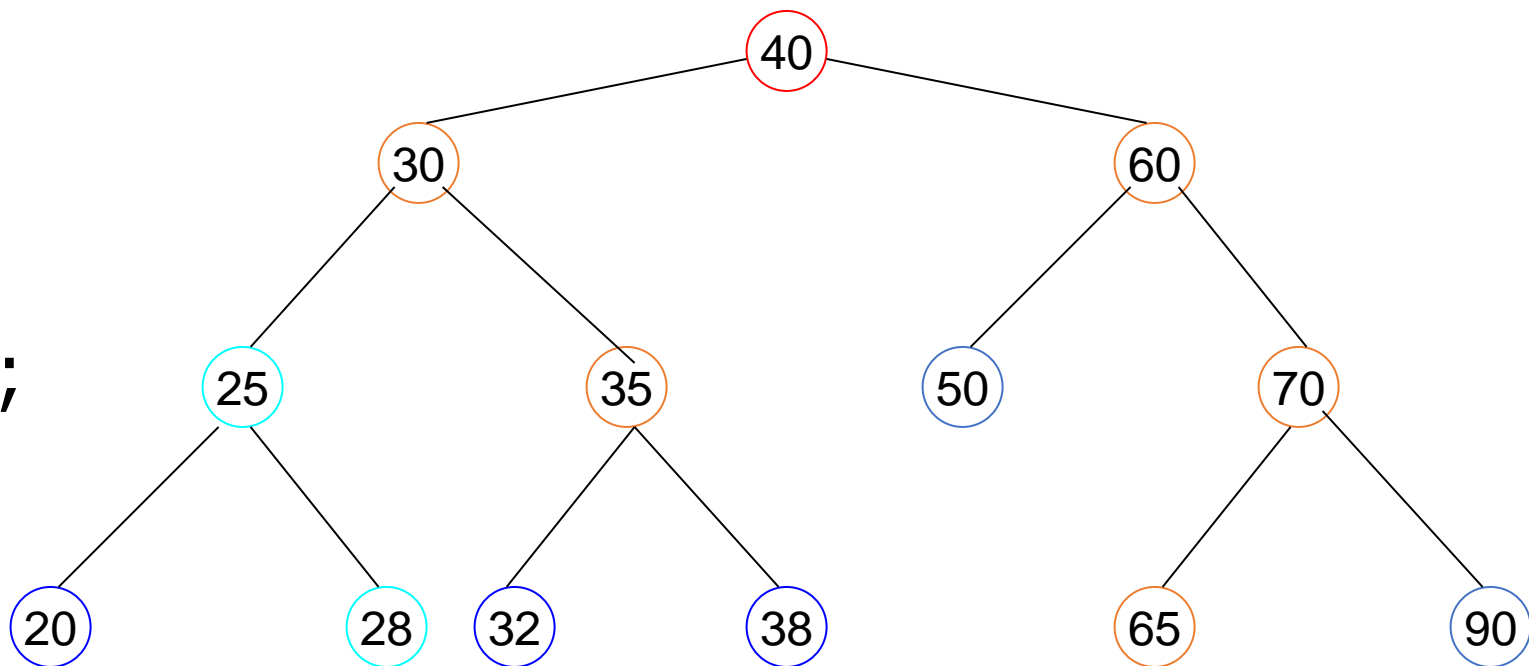
```
void Remove-Right(Tree T, int x ) {  
    Tree p, q;  
    p = Search (T, x); //tìm node có nội dung là x trên cây  
    if (p==NULL ) { //nếu node có nội dung x không có trên cây  
        <thông báo node cha x không có thực>;  
        return; //không thể loại bỏ được node lá phải  
    } //nếu p có nhánh cây con bên phải  
    else if ( (p -> right)->right !=NULL || (p -> right)->left !=NULL ) )  
        <thông báo node cha x có nhánh cây con phải>;  
        return; // không thêm loại bỏ được cây con trái  
    else {  
        q = p ->right; // node lá phải là q.  
        p -> right =NULL; //ngắt liên kết cho node p  
        delete(q); //Loại bỏ node lá phải q  
    }  
}
```

Duyệt cây theo thứ tự trước (Node-Left-Right)

```
void NLR(Tree T) {  
    if (T!=NULL) {  
        <Thăm node>;  
        NLR( T -> left);  
        NLR( T -> right);  
    }  
}
```

Ví dụ.

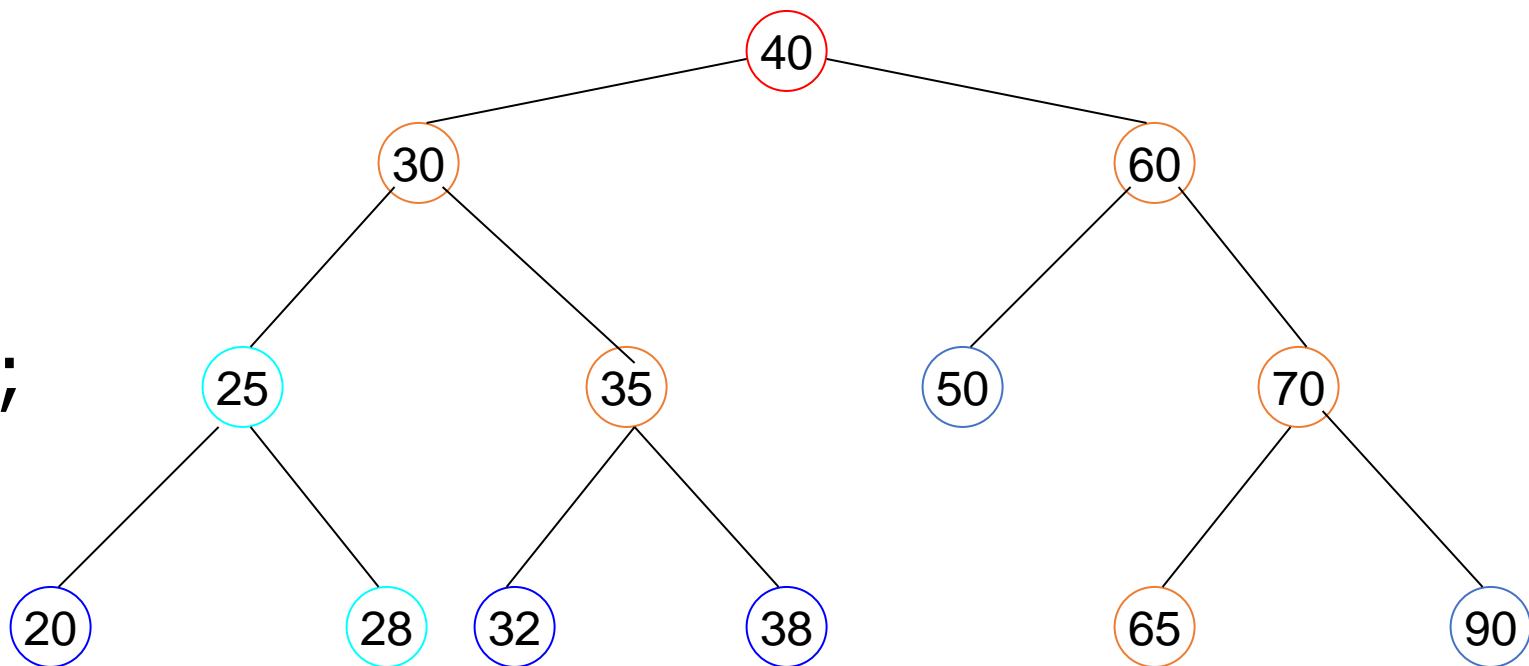
NLR(T) = 40, 30, 25, 20, 28, 35, 32, 38, 60, 50, 70, 65, 90.



Duyệt cây theo thứ tự giữa (Left-Node-Right)

```
void LNR(Tree T) {  
    if (T!=NULL) {  
        LNR( T -> left);  
        <Thăm node>;  
        LNR( T -> right);  
    }  
}
```

Ví dụ.



$LNR(T) = 20, 25, 28, 30, 32, 35, 38, 40, 50, 60, 65, 70, 90.$

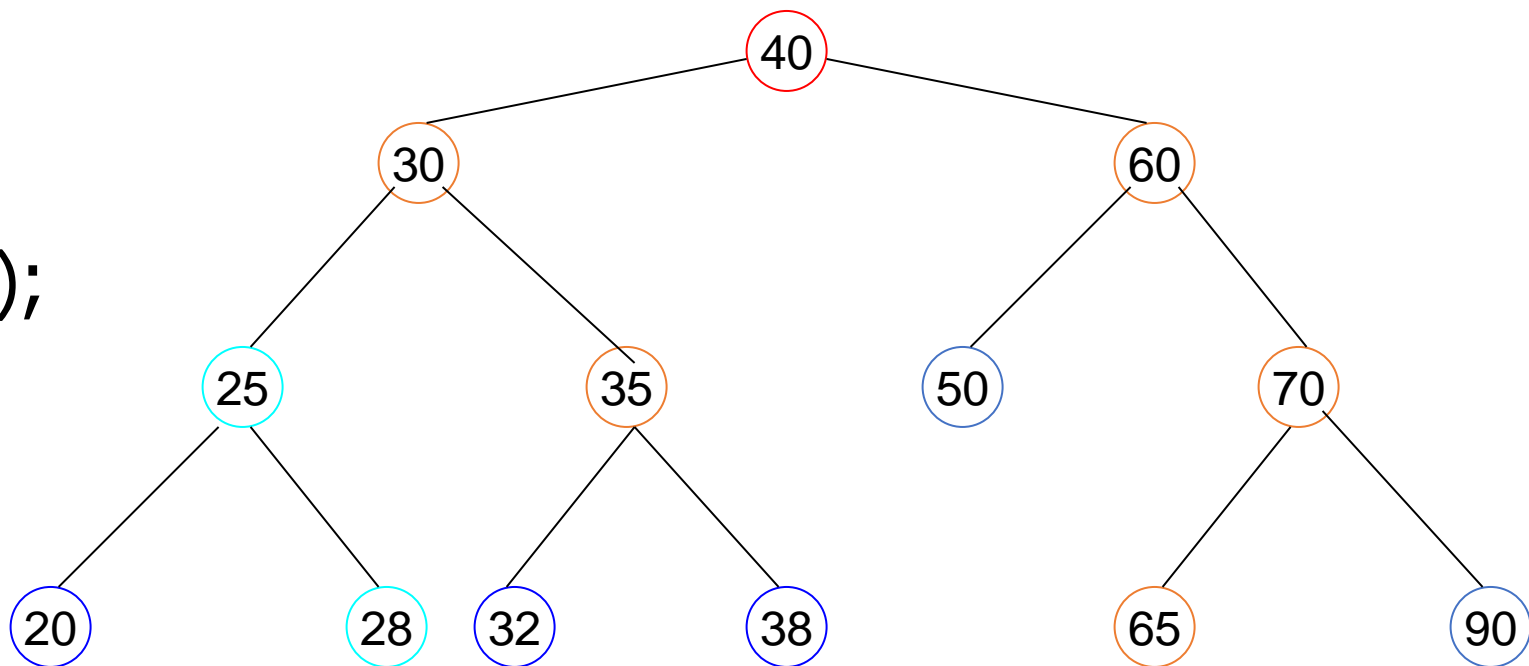
Duyệt cây theo thứ tự sau (Left-Right-Node)

```
void LRN (Tree T ) {  
    if (T!=NULL ) {  
        LRN ( T -> left);  
        LRN ( T -> right);  
        <Thăm node>;  
    }  
}
```

}

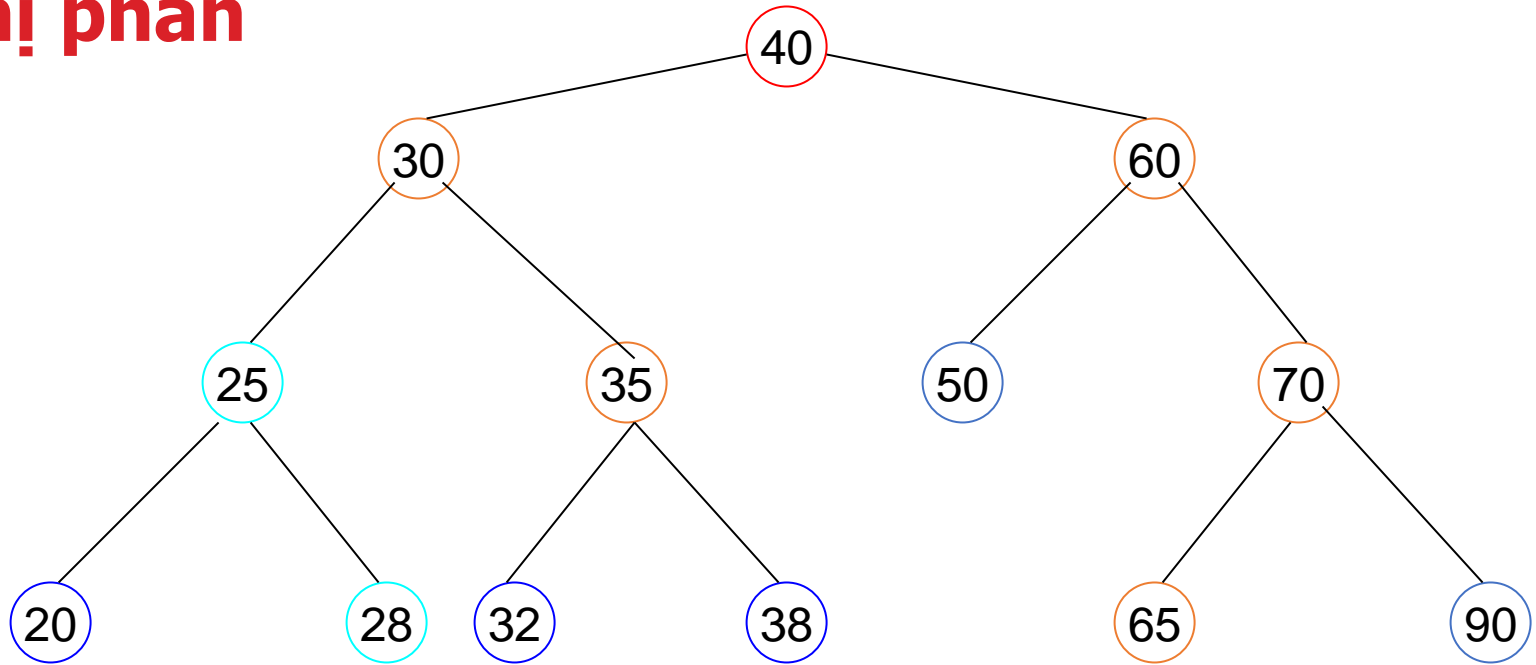
Ví dụ.

LRN (T) = 20, 28, 25, 32, 38, 35, 30, 50, 65, 90, 70, 60, 40.



1. Tính kích thước cây nhị phân

```
int Size(Tree T) {  
    if (T==NULL)  
        return 0;  
    else  
        return(Size(T->left) + 1 + Size(T->right));  
}
```



2. Xác định 2 cây nhị phân giống nhau hay không

```
int identicalTrees(Tree T1, Tree T2) {  
    if (T1==NULL && T2==NULL) //nếu cả hai cây T1 và T2 đều rỗng  
        return 1; //rõ ràng chúng giống nhau  
    if (T1!=NULL && T2!=NULL){ //nếu cả hai cây khác rỗng  
        return (T1->data == T2->data &&  
                identicalTrees(T1->left, T2->left) &&  
                identicalTrees(T1->right, T2->right) );  
    }  
    return 0; //một trong hai cây khác rỗng  
}
```

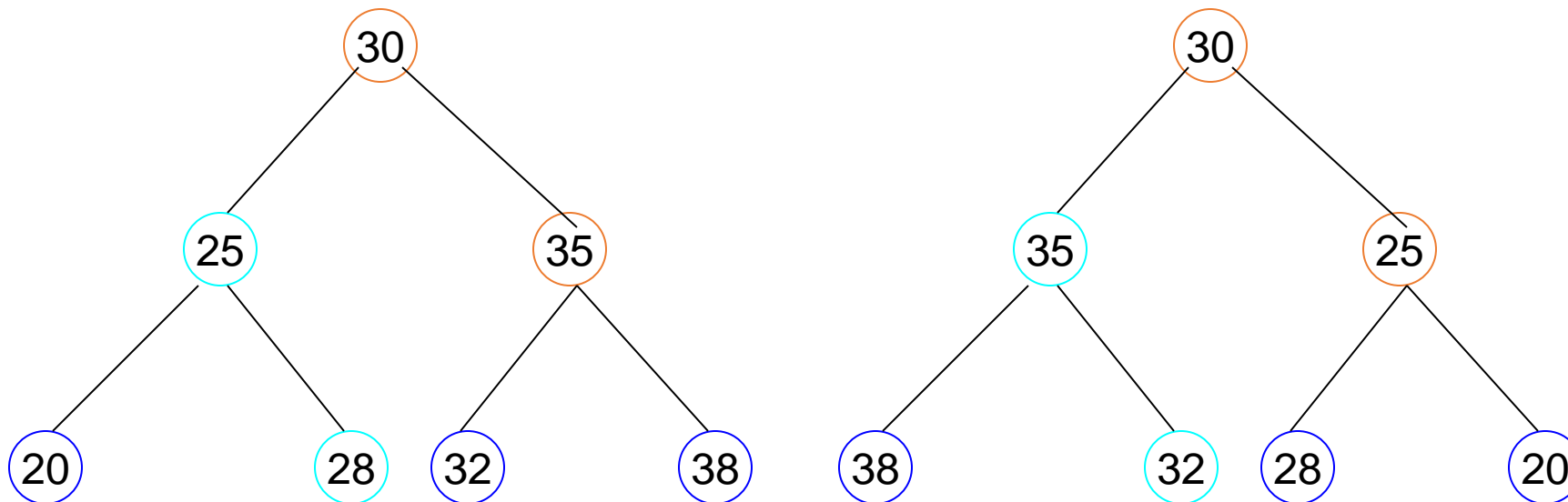

3. Tìm độ cao của cây nhị phân

```
int maxDepth(Tree *T) {  
    if (T==NULL)  
        return -1;  
    else {  
        int lDepth = maxDepth(T->left); //Tìm độ cao của cây con trái  
        int rDepth = maxDepth(T->right); //Tìm độ cao của cây con phải  
        if (lDepth > rDepth)  
            return(lDepth + 1);  
        else  
            return(rDepth + 1);  
    }  
}
```

4. Cây phản chiếu

```
void mirror(struct node* node) {  
    if (node==NULL)    return; //Nếu cây rỗng thì không phải làm gì  
    else { //Nếu cây không rỗng  
        struct node* temp; //Sử dụng node trung gian temp  
        mirror(node->left); //Gọi đến cây phản chiếu bên trái  
        mirror(node->right); //Gọi đến cây phản chiếu bên phải  
        temp = node->left; node->left = node->right;  
        node->right = temp; //Tráo đổi hai node trái và phải  
    }  
}
```

4. Cây phản chiếu

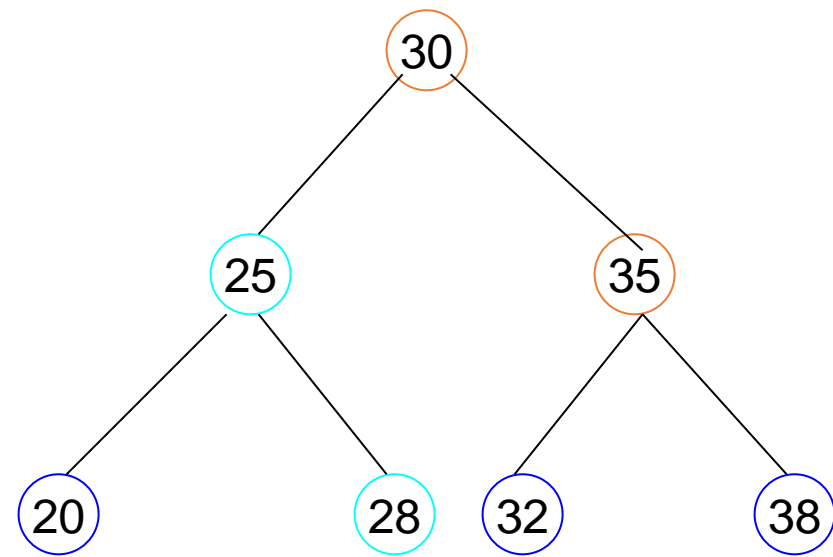


5. Tìm đường đi từ gốc đến lá

```
void printPathsRecur(struct node* node, int path[], int pathLen) {  
    if (node==NULL) return; //Nếu cây rỗng thì không phải làm gì  
    path[pathLen] = node->data; pathLen++; //Ghi nhận đường đi bắt đầu từ  
    gốc  
    if (node->left==NULL && node->right==NULL) //Node lá  
        printArray(path, pathLen); //Đường đi từ node gốc đến node lá  
    else {  
        printPathsRecur(node->left, path, pathLen); //Đi tiếp sang cây bên trái  
        printPathsRecur(node->right, path, pathLen); //Đi tiếp sang bên phải  
    }  
}
```

6. Đếm số node lá trên cây

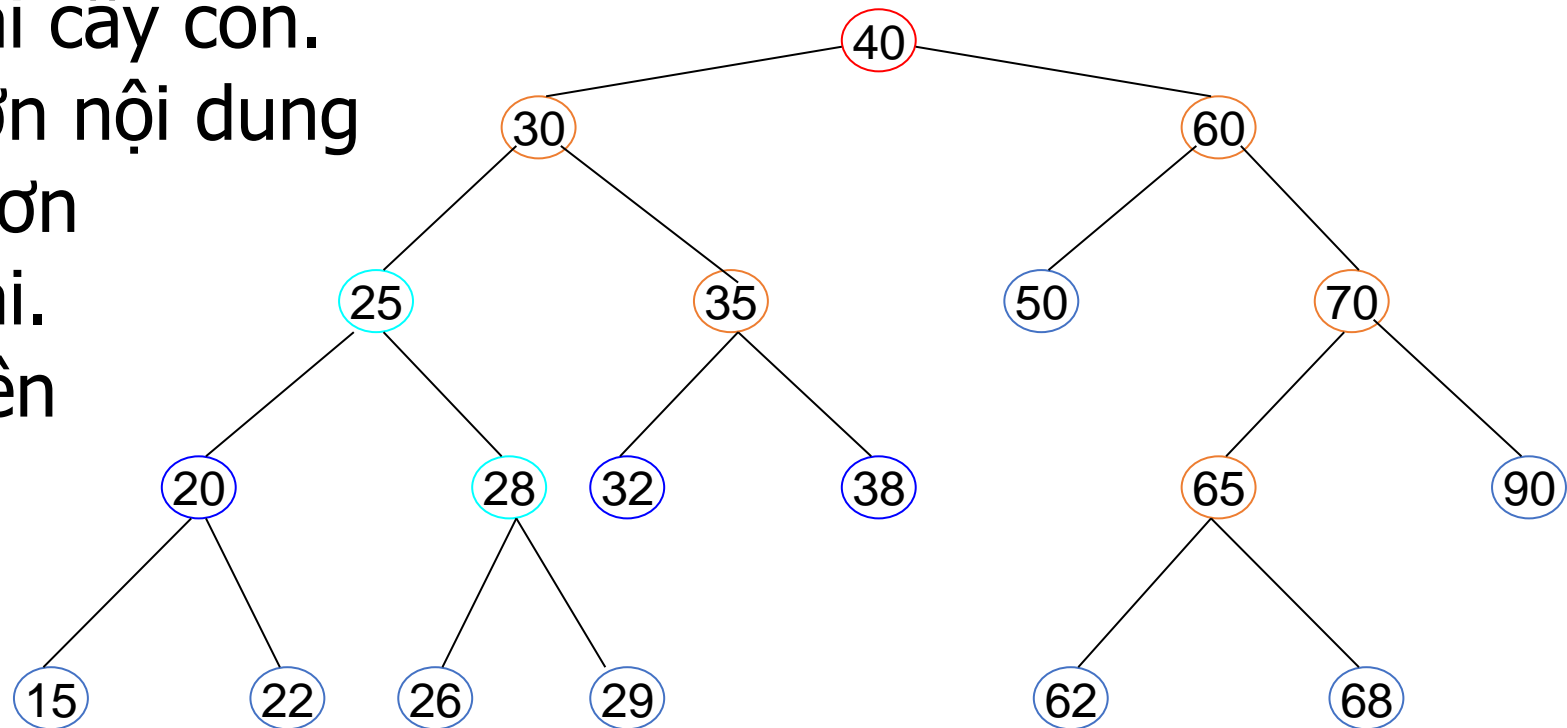
```
unsigned int getLeafCount(struct node* node) {  
    if(node == NULL) //Nếu cây rỗng  
        return 0; //Số node lá là 0  
    if(node->left == NULL && node->right == NULL) //Nếu cây có một node  
        return 1; //Số node lá là 1  
    else //Nếu cây có ít nhất một cây con  
        return getLeafCount(node->left) + getLeafCount(node->right);  
}
```



Cây nhị phân tìm kiếm

Cây nhị phân thỏa mãn điều kiện:

- Hoặc là rỗng hoặc có một node gốc.
- Mỗi node gốc có tối đa hai cây con.
- Nội dung node gốc lớn hơn nội dung node con bên trái và nhỏ hơn nội dung node con bên phải.
- Hai cây con bên trái và bên phải cũng hình thành nên hai cây tìm kiếm.



Biểu diễn cây nhị phân tìm kiếm

Giống như cây nhị phân thông thường.

```
typedef struct node {  
    Item infor; //Thông tin của node  
    struct node *left; //Con trỏ sang cây con trái  
    struct node *right; //Con trỏ sang cây con phải  
} *Tree;
```

Các thao tác trên cây nhị phân tìm kiếm

1. Tạo node gốc cho cây.
2. Thêm vào node vào cây tìm kiếm.
3. Loại bỏ node trên cây tìm kiếm.
4. Tìm kiếm node trên cây.
5. Xoay trái cây tìm kiếm
6. Xoay phải cây tìm kiếm
7. Duyệt cây theo thứ tự trước.
8. Duyệt cây theo thứ tự giữa.
9. Duyệt cây theo thứ tự sau.

Khởi tạo cây nhị phân tìm kiếm:

```
void Init(Tree *T){  
    *T=NULL; //Đưa cây T về trạng thái rỗng  
}
```

Cấp phát miền nhớ cho một node:

```
Tree GetNode(){  
    Tree p;//Khai báo một node kiểu Tree  
    p=new node; //Cấp phát miền nhớ cho node  
    return (p); //Trả lại node được cấp phát  
}
```

Giải phóng một node p cho cây T

```
void FreeNode(Tree p){  
    delete (p); //giải phóng node p  
}
```

Kiểm tra tính rỗng của cây T

```
int isEmpty(Tree *T){  
    if(*T==NULL) //nếu T rỗng  
        return 1; //trả lại giá trị đúng  
    return 0; //trả lại giá trị sai  
}
```

Tạo một node cho cây T

```
Tree MakeNode( Item x){ //x là giá trị node cần bổ sung vào cây
    Tree p; //Khai báo một node kiểu Tree
    p = GetNode();//cấp phát miền nhớ cho p
    p->infor = x;      //thiết lập thành phần thông tin cho node
    p->left=NULL;      //tạo liên kết trái cho node
    p->right=NULL;     //Tạo liên kết phải cho node
    return (p); //trả lại node được tạo ra
}
```

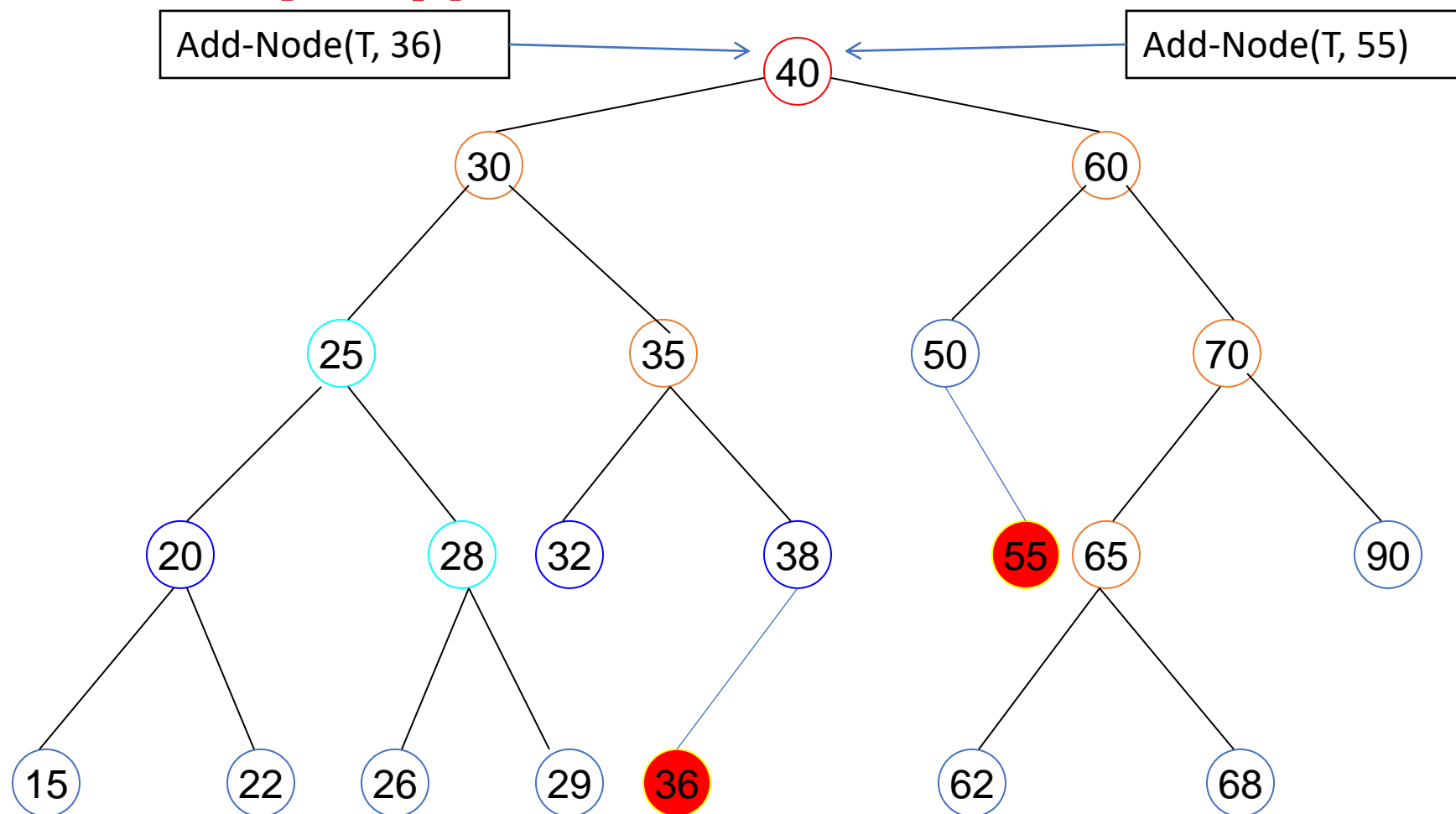
Tìm node có nội dung là x trên cây tìm kiếm T:

```
Tree Search(Tree T, Item x){  
    Tree p = T; //p trở đến node gốc của cây  
    if ( p!=NULL ) { //nếu p không phải là NULL  
        if ( x < T -> infor )      p = Search( T->left, x);  
        else p = Search(T ->right, x);  
    }  
    return(p);  
}
```

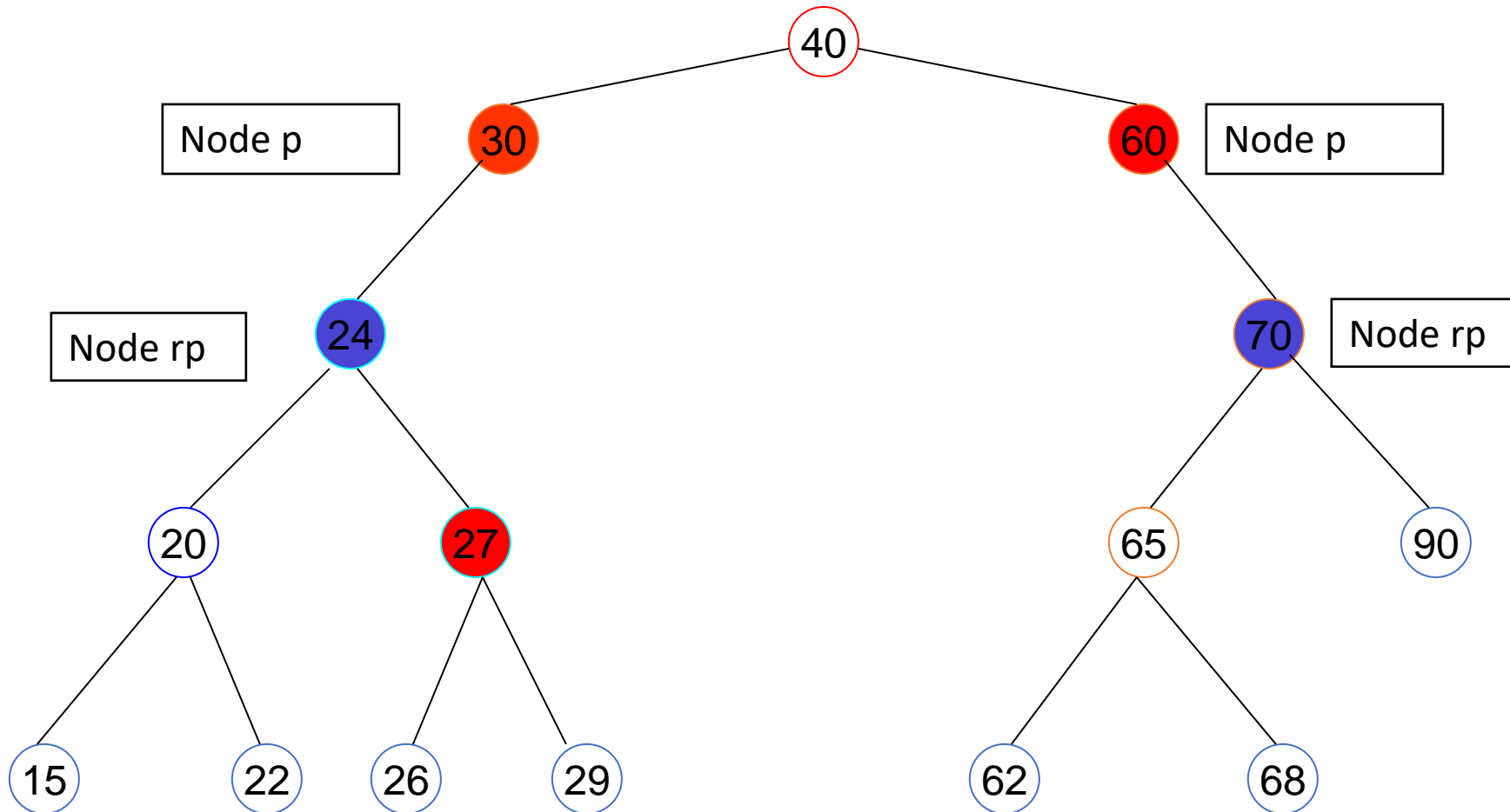
Tìm node có nội dung là x trên cây tìm kiếm T (không đệ quy):

```
Tree Search(Tree T, Item x){  
    Tree p = T; //p trở đến node gốc của cây  
    while (p != NULL) { //Lặp trong khi p khác NULL  
        if (x == p->infor)      return (p); //Nếu x trùng với p  
        else if (x < p->infor) p = p -> left; //nếu x nhỏ hơn p  
        else p = p -> right; // nếu x lớn hơn p  
    }  
    return(p);  
}
```

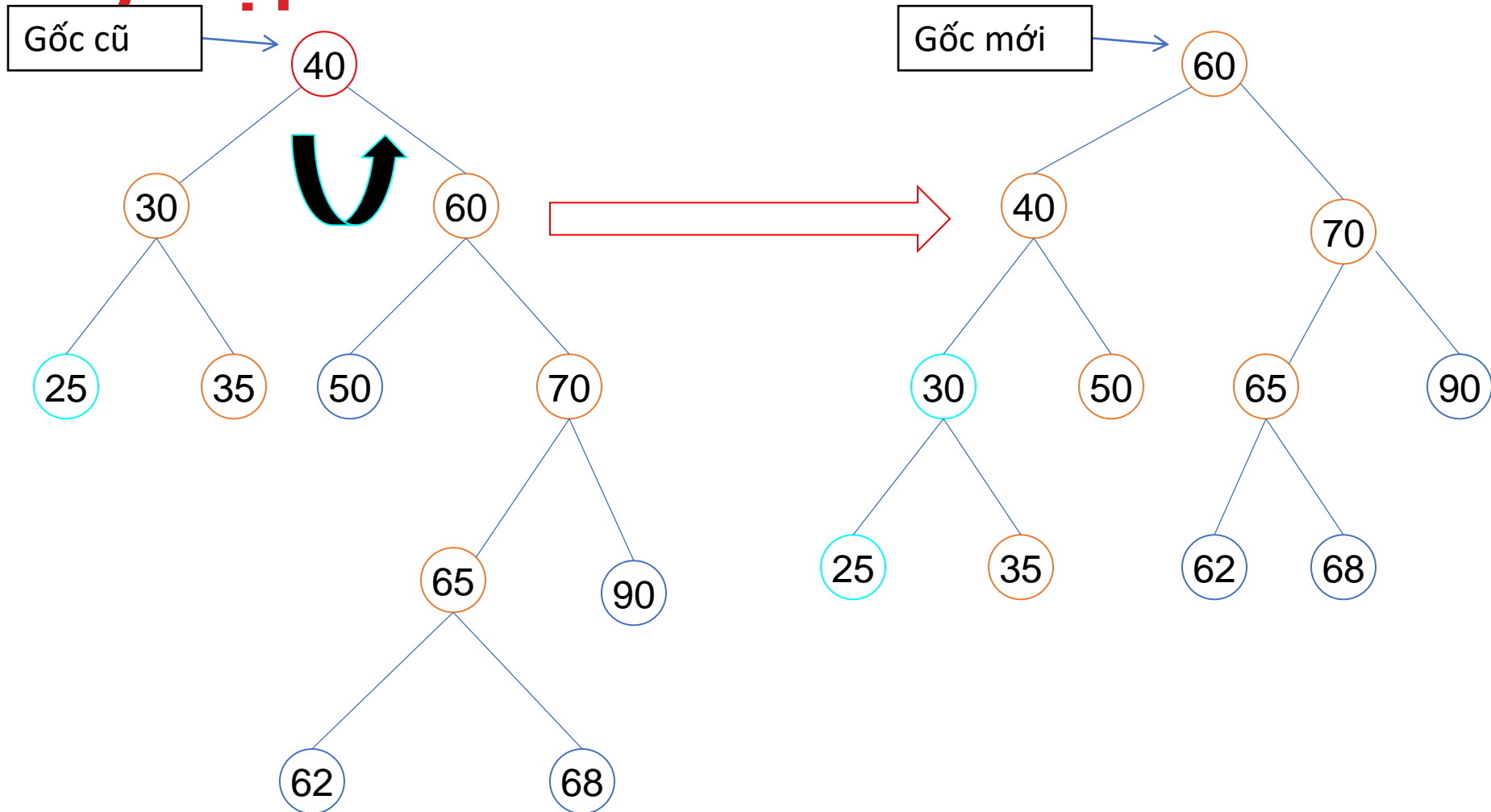
Thêm node vào cây nhị phân tìm kiếm



Loại node trên cây nhị phân tìm kiếm



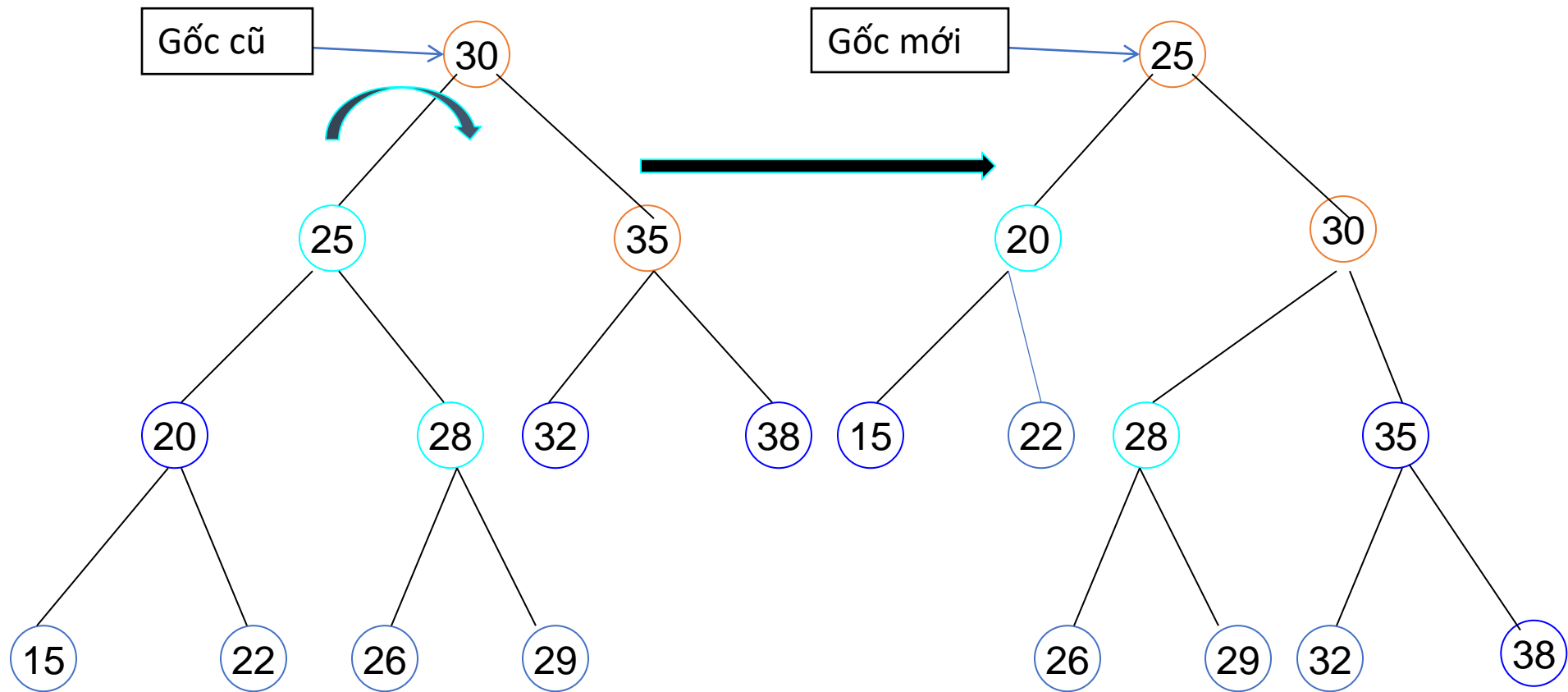
Xoay trái cây nhị phân tìm kiếm



Xoay trái cây nhị phân tìm kiếm

```
Tree RotateLeft (Tree T ) {  
Tree p = T; // p trở đến node gốc của cây  
if ( T==NULL)  
    <Cây rỗng>;  
else if (T -> right == NULL)  
    <T không có cây con phải>;  
else {  
    p = T -> right; //p trở đến cây con phải  
    T -> right = p ->left; //T trở đến node trái của p  
    p -> left = T; //Thiết lập liên kết trái cho p  
}  
return(p);  
}
```

Xoay phải cây nhị phân tìm kiếm



Xoay phải cây nhị phân tìm kiếm

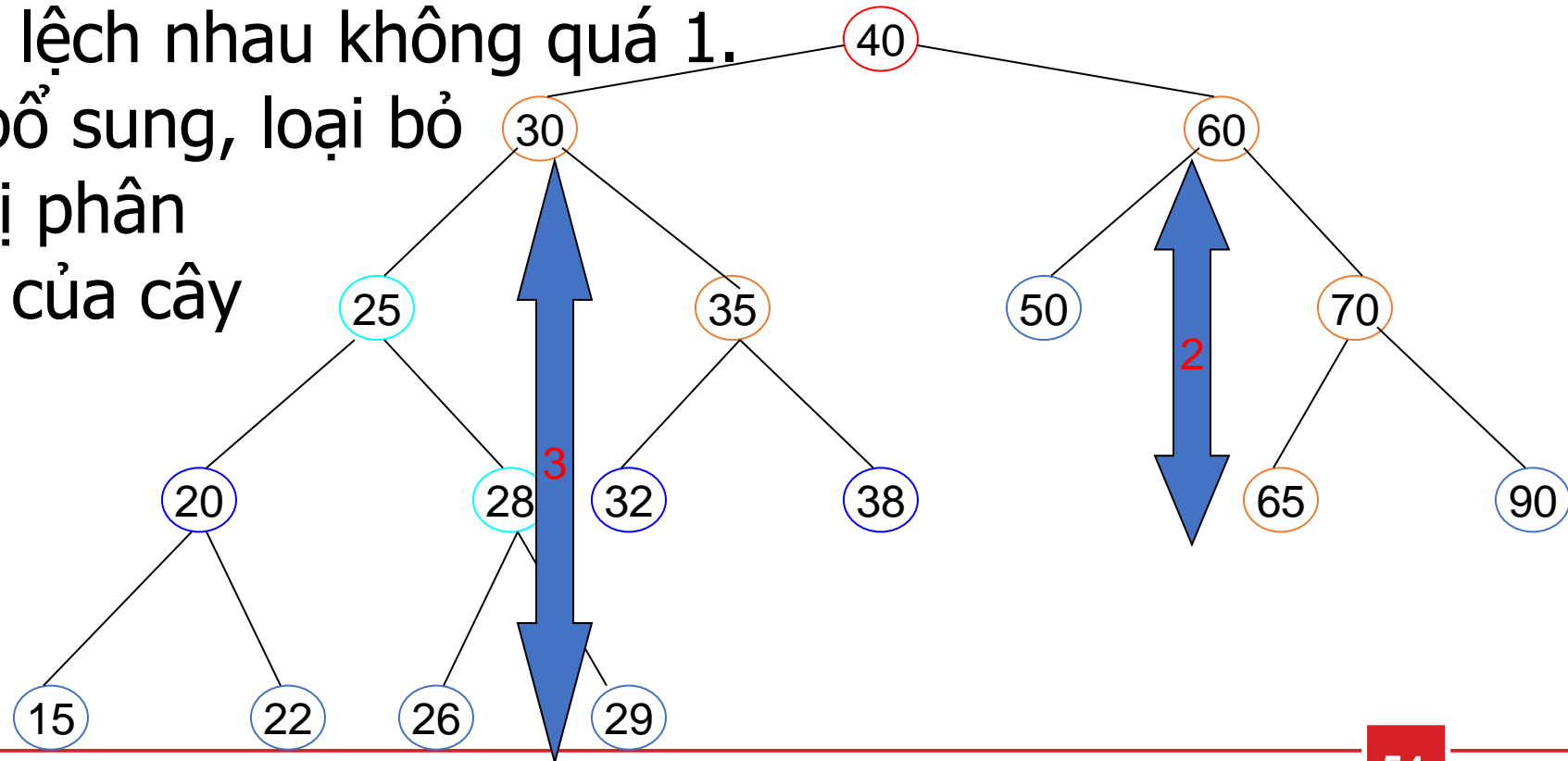
```
Tree RotateRight (Tree T ) {  
Tree p = T; // p trỏ đến node gốc của cây  
if ( T==NULL)  
    <Cây rỗng>;  
else if (T -> left == NULL)  
    <T không có cây con trái>;  
else {  
    p = T -> left; //p trỏ đến cây con trái  
    T -> left = p ->right; //T trỏ đến node phải của p  
    p -> right = T; //Thiết lập liên kết phải cho p  
}  
return(p);  
}
```

Khái niệm

Cây nhị phân tìm kiếm tự cân bằng. (Adelson-Velsky and Landis)

Cây tìm kiếm cân bằng có tính chất độ cao của cây con bên trái và độ cao cây con bên phải chênh lệch nhau không quá 1.

Cho phép ta tìm kiếm, bổ sung, loại bỏ node nhanh hơn cây nhị phân thông thường vì độ cao của cây luôn là $O(\log(n))$.



Cây nhị phân tìm kiếm cân bằng

$lh(p)$, $rh(p)$ là chiều sâu của cây con trái và chiều sâu của cây con phải.

$bf(p) = lh(p) - rh(p)$ là chỉ số cân bằng của node p

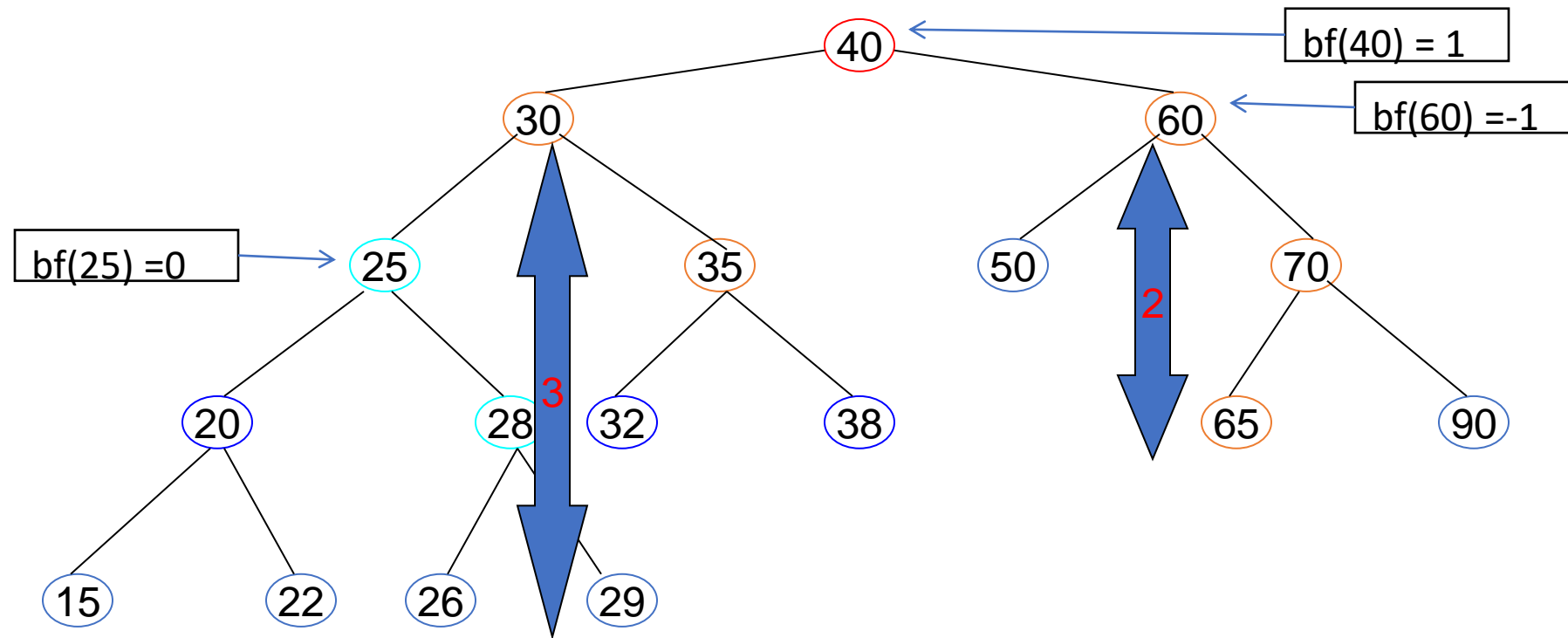
Các trường hợp:

$lh(p) = rh(p) \leftrightarrow bf(p) = 0$: node p cân bằng tuyệt đối.

$lh(p) = rh(p) + 1 \leftrightarrow bf(p) = 1$: node p lệch về phía trái.

$lh(p) = rh(p) - 1 \leftrightarrow bf(p) = -1$: node p lệch về phía phải.

Cây nhị phân tìm kiếm cân bằng



Thêm node vào cây nhị phân tìm kiếm cân bằng

Bước 1. Thực hiện thêm node w vào cây tìm kiếm giống như cây thông thường.

Bước 2. Đi lên từ nút w và tìm nút không cân bằng đầu tiên.

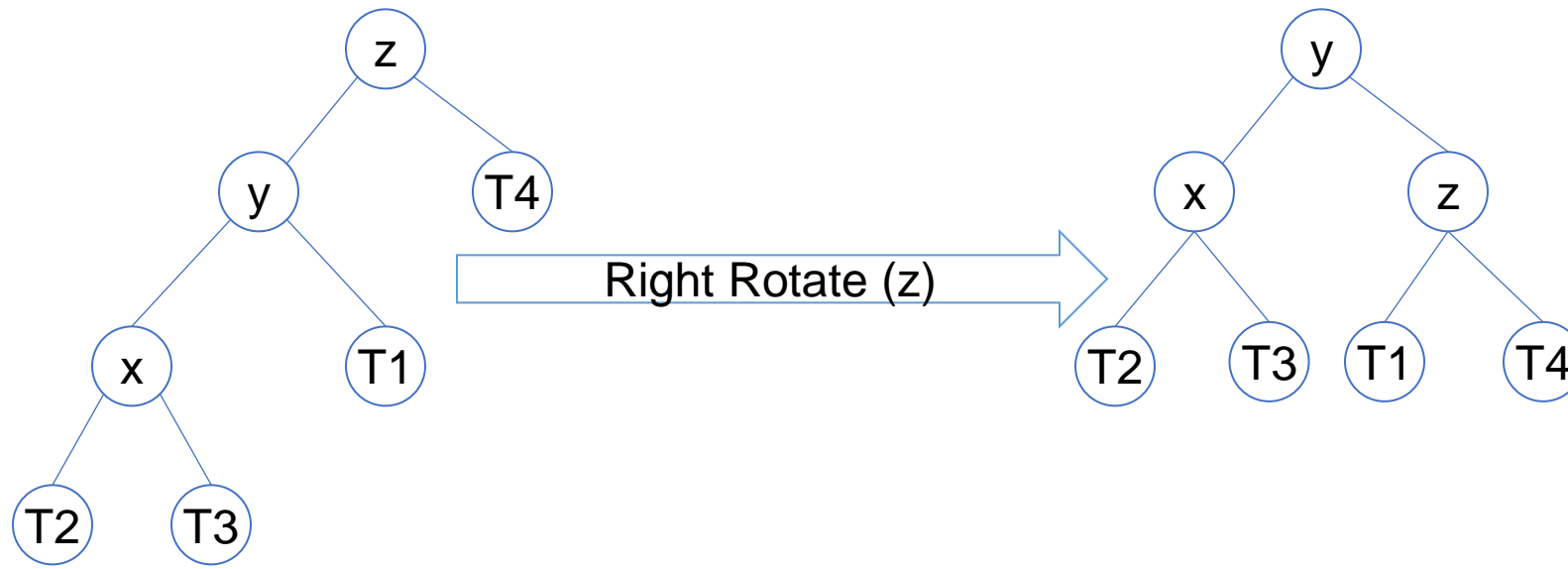
Gọi z là nút không cân bằng đầu tiên, y là con của z đi trên đường từ w đến z và x là cháu của z đi trên đường từ w đến z .

Bước 3. Cân bằng lại cây bằng các phép quay thích hợp tại cây con gốc.

Thêm node vào cây nhị phân tìm kiếm cân bằng

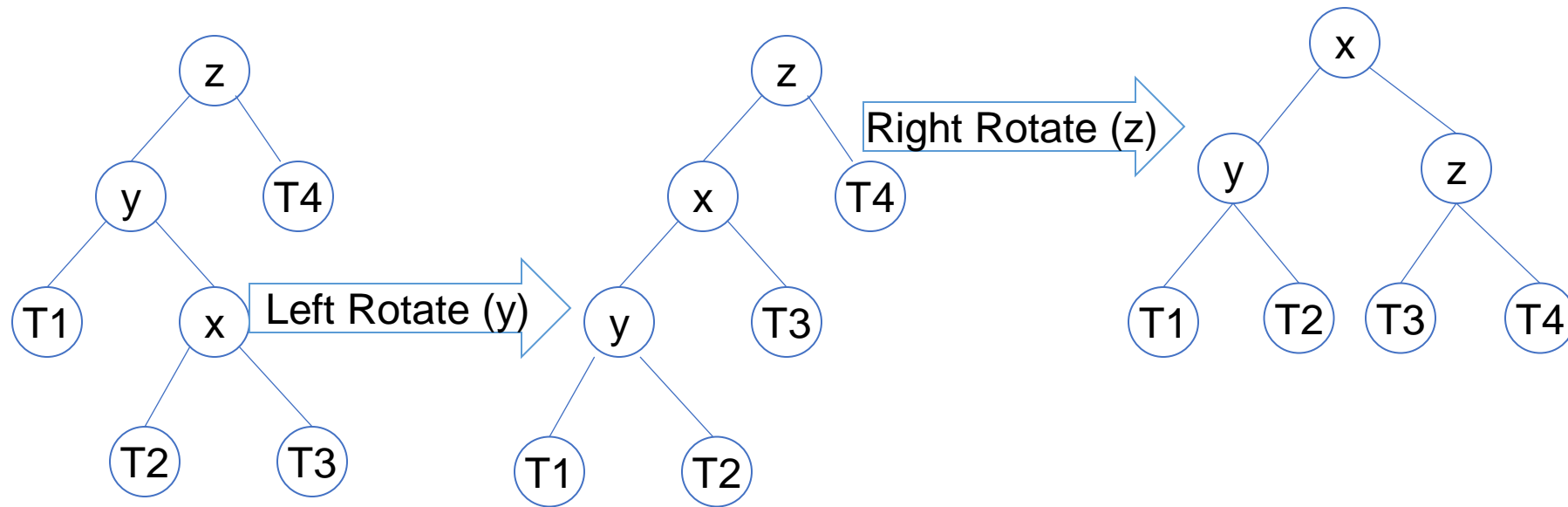
Giả sử T1, T2, T3, T4 là các cây con gốc z, khi đó các phép cân bằng lại (re-balance) được mô tả như sau:

Trường hợp left-left-case: (Node y là node con trái của z và x là node con trái của y).



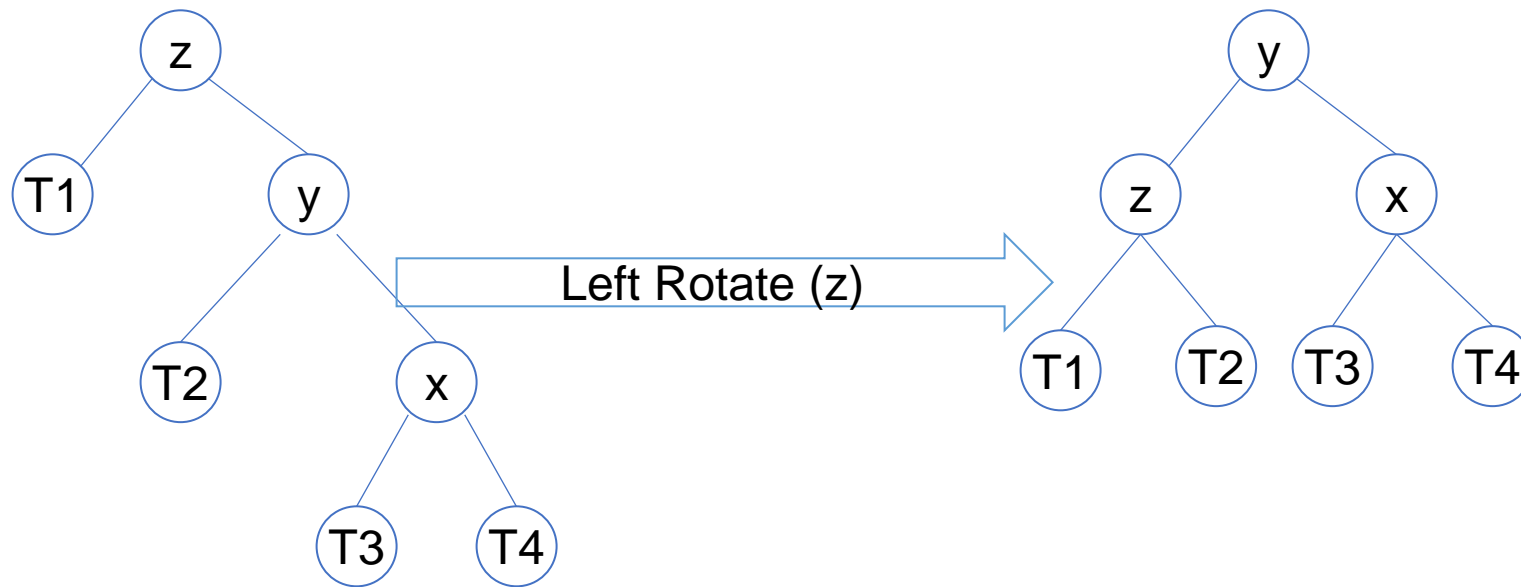
Thêm node vào cây nhị phân tìm kiếm cân bằng

Trường hợp left-right-case: Node y là node con trái của z và x là node con phải của y.



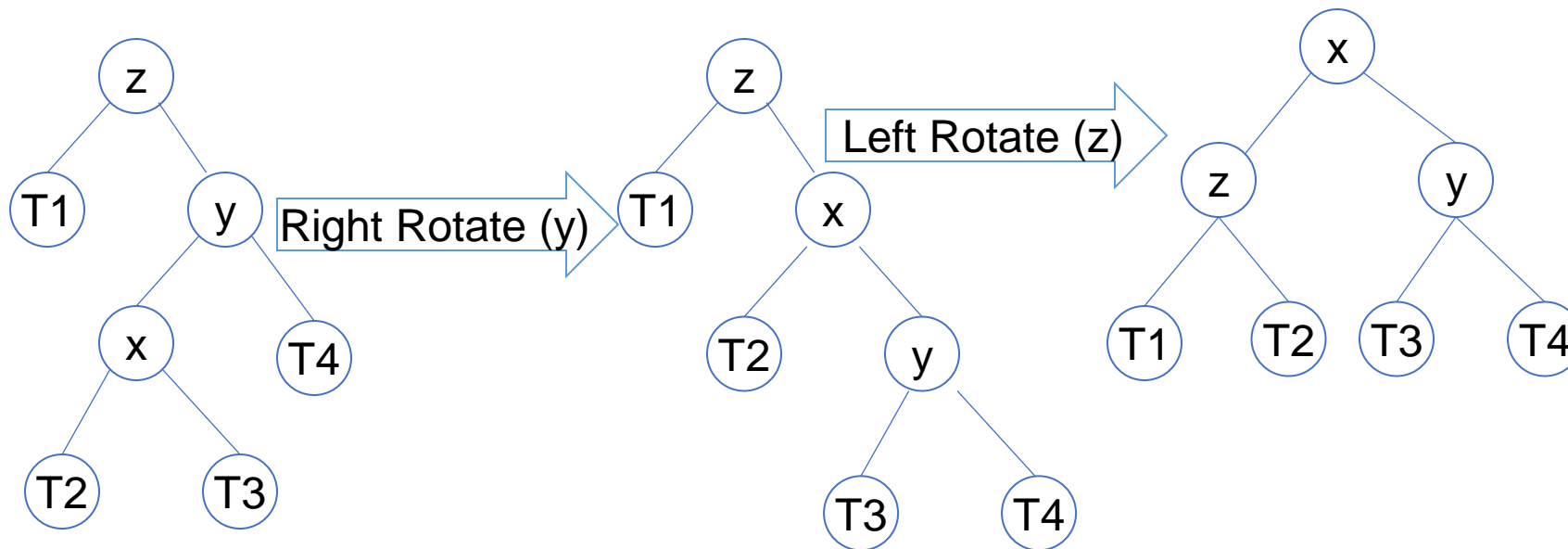
Thêm node vào cây nhị phân tìm kiếm cân bằng

Trường hợp right-right-case: Node y là node con phải của z và x là node con phải của y.



Thêm node vào cây nhị phân tìm kiếm cân bằng

Trường hợp right-left-case: Node y là node con phải của z và x là node con trái của y.



Loại node trên cây nhị phân tìm kiếm cân bằng

Bước 1. Thực hiện loại node w như cây tìm kiếm thông thường.

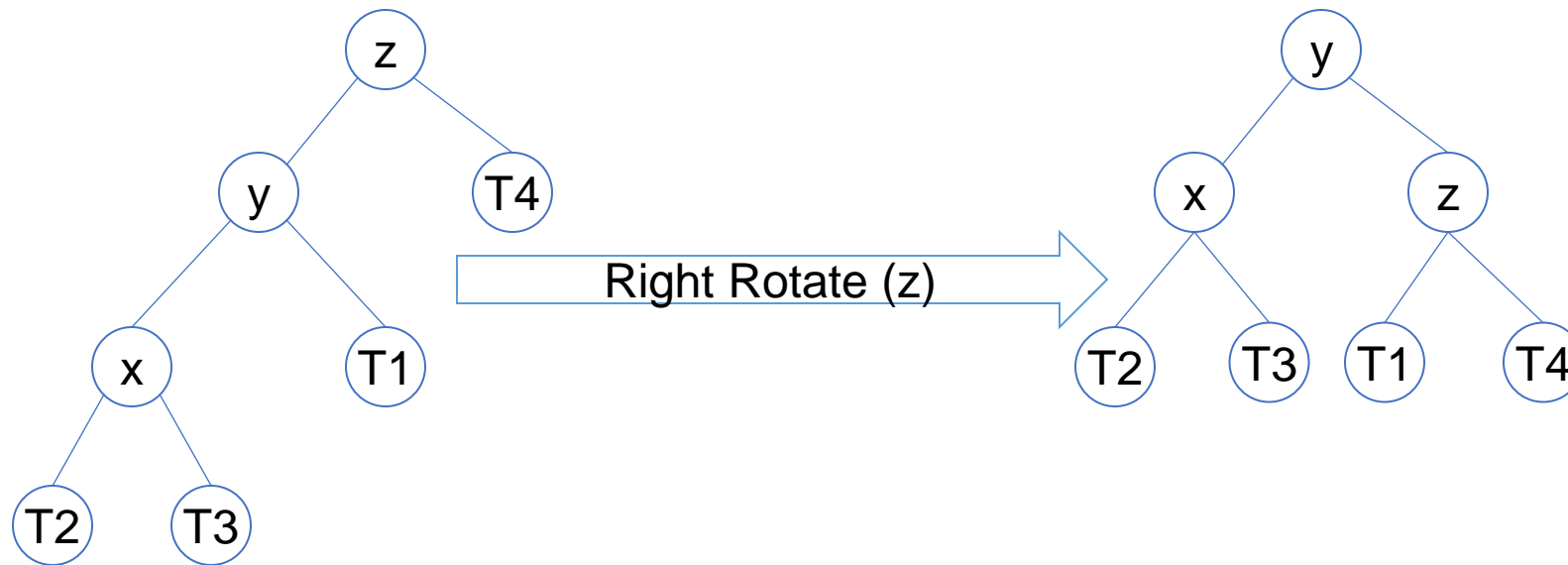
Bước 2. Đi lên tìm nút không cân bằng đầu tiên (z). y là cây con có chiều cao lớn hơn của z và x là con có chiều cao lớn hơn của y . Lưu ý rằng các định nghĩa của x và y khác với phần thêm node.

Bước 3. Cân bằng lại cây bằng các phép quay thích hợp.

Loại node trên cây nhị phân tìm kiếm cân bằng

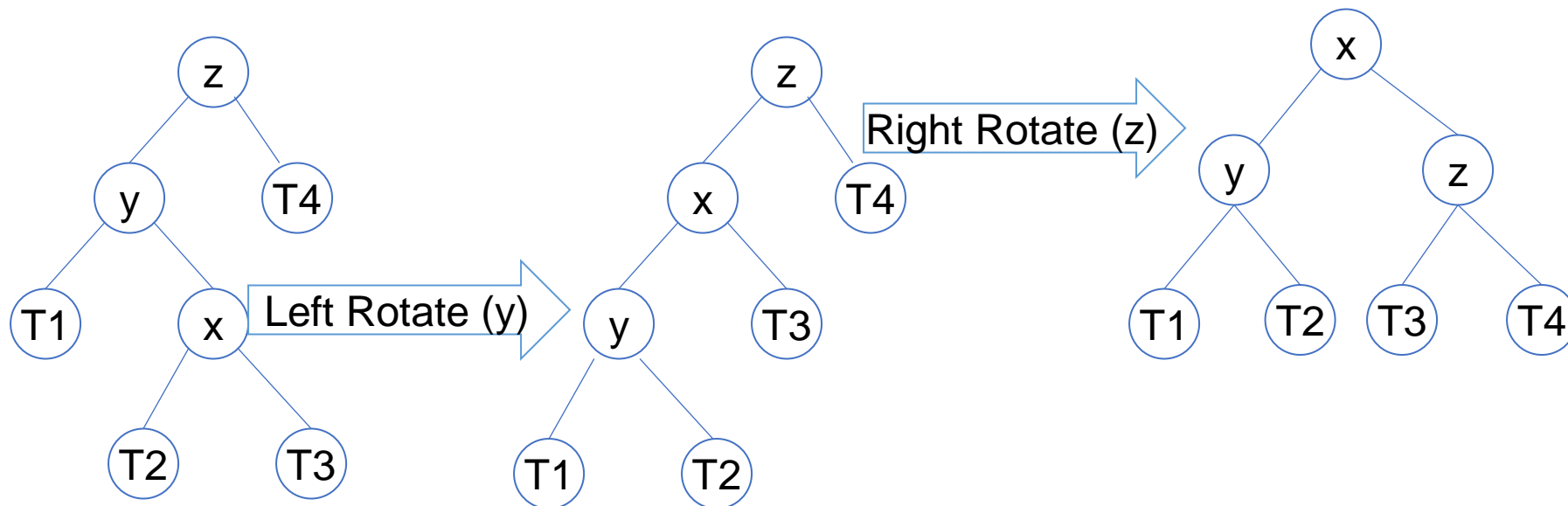
Giả sử T1, T2, T3, T4 là các cây con gốc z, khi đó các phép cân bằng lại (re-balance) được mô tả như sau:

Trường hợp left-left-case: y là con trái của z và x là con trái của y



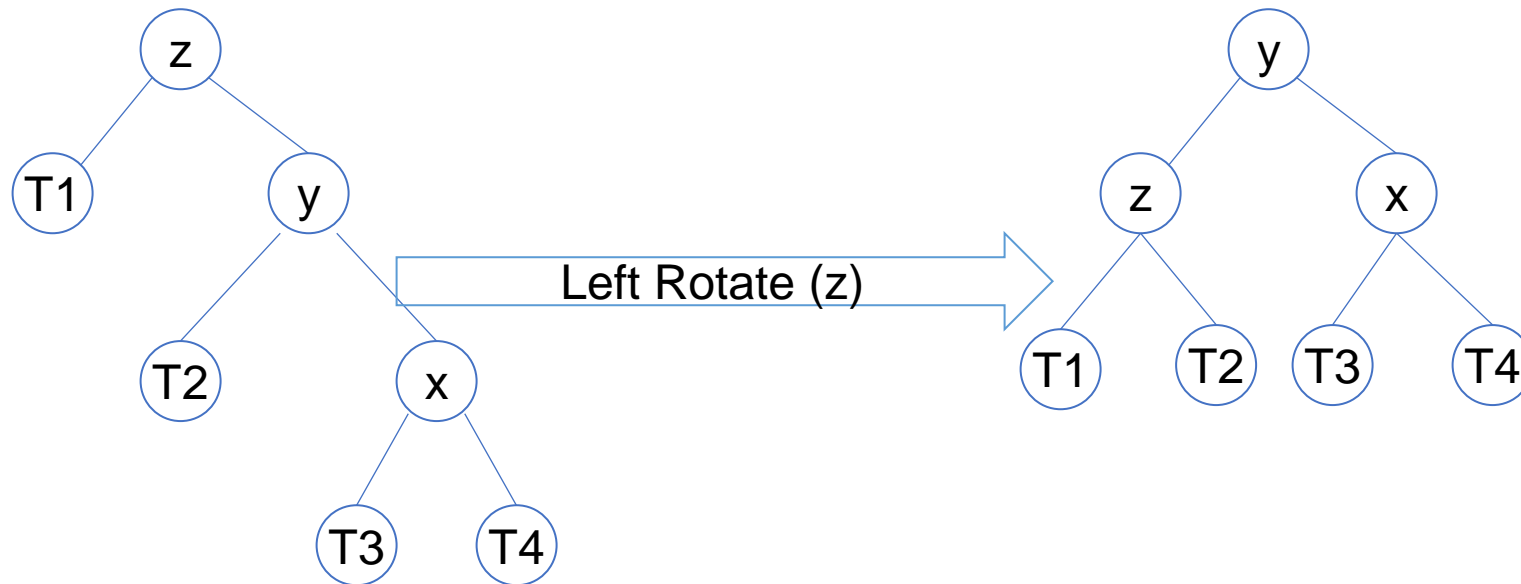
Loại node trên cây nhị phân tìm kiếm cân bằng

Trường hợp left-right-case: y là con bên trái của z và x là con bên phải của y



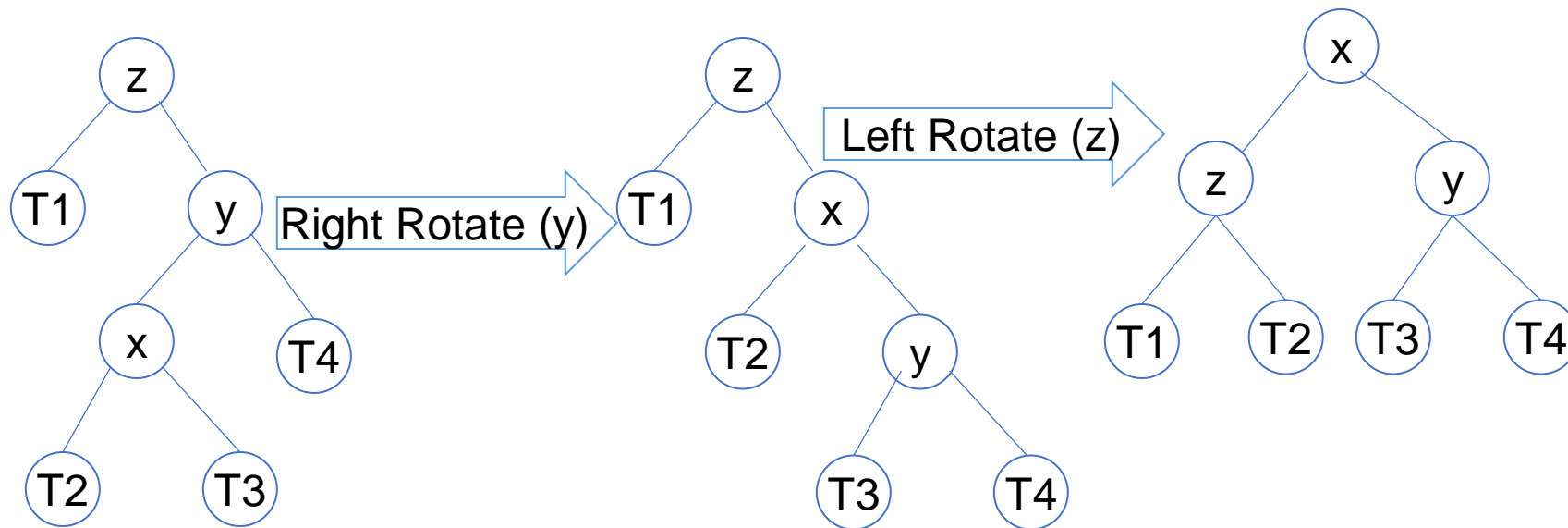
Loại node trên cây nhị phân tìm kiếm cân bằng

Trường hợp right-right-case: y là con bên phải của z và x là con bên phải của y



Loại node trên cây nhị phân tìm kiếm cân bằng

Trường hợp right-left-case: y là con bên phải của z và x là con bên trái của y



Cài đặt cây nhị phân tìm kiếm cân bằng

Khai báo node cây AVL:

```
struct node {  
    int key;//thành phần dữ liệu  
    struct node *left; //thành con trỏ đến cây con trái  
    struct node *right; //thành con trỏ đến cây con phải  
    int height; // chiều cao cây  
};
```

Phép xoay phải (rightRotation)

```
struct node *rightRotate(struct node *y) {  
    struct node *x = y->left; //x trở đến node bên trái của y  
    struct node *T2 = x->right; //T2 là cây con phải của x->right  
    //Thực hiện quay phải tại y  
    x->right = y; y->left = T2;  
    //Cập nhật lại chiều cao  
    y->height = max(height(y->left), height(y->right))+1;  
    x->height = max(height(x->left), height(x->right))+1;  
    return x;  
}
```

Phép xoay trái (leftRotation)

```
struct node *leftRotate(struct node *x) {  
    struct node *y = x->right;  
    struct node *T2 = y->left;  
    // thực hiện quay trái  
    y->left = x; x->right = T2;  
    // cập nhật độ cao  
    x->height = max(height(x->left), height(x->right))+1;  
    y->height = max(height(y->left), height(y->right))+1;  
    // trả lại gốc mới root  
    return y;  
}
```

Thêm node vào cây nhị phân tìm kiếm cân bằng

```
struct node* insert(struct node* node, int key) {  
    //Bước 1. Thực hiện thêm node trên cây tìm kiếm thông thường  
    if (node == NULL) return(newNode(key));  
    if (key < node->key) node->left = insert(node->left, key);  
    else node->right = insert(node->right, key);  
    //Bước 2. Cập nhật độ cao của node trước  
    node->height = max(height(node->left), height(node->right)) + 1;  
    //Bước 3. Cân bằng cây  
    int balance = getBalance(node);  
    if (balance > 1 && key < node->left->key) return rightRotate(node);  
    if (balance < -1 && key > node->right->key) return leftRotate(node);  
    if (balance > 1 && key > node->left->key) {  
        node->left = leftRotate(node->left); return rightRotate(node);  
    }  
    if (balance < -1 && key < node->right->key) {  
        node->right = rightRotate(node->right);  
        return leftRotate(node);  
    }  
    return node;  
}
```



THANK YOU FOR YOUR ATTENTION!