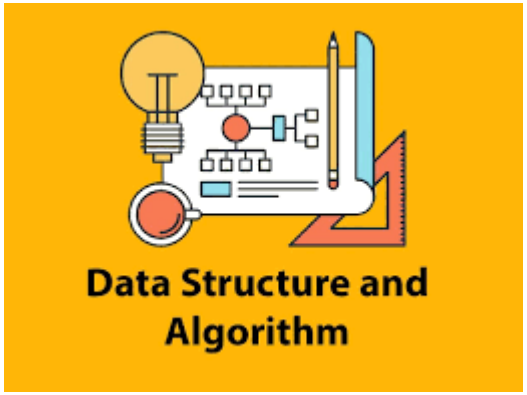
CẤU TRÚC DỮ LIỆU & GIẢI THUẬT

NGÀY 8: CÂY NHỊ PHÂN



Giảng viên: Th.S Bùi Văn Kiên

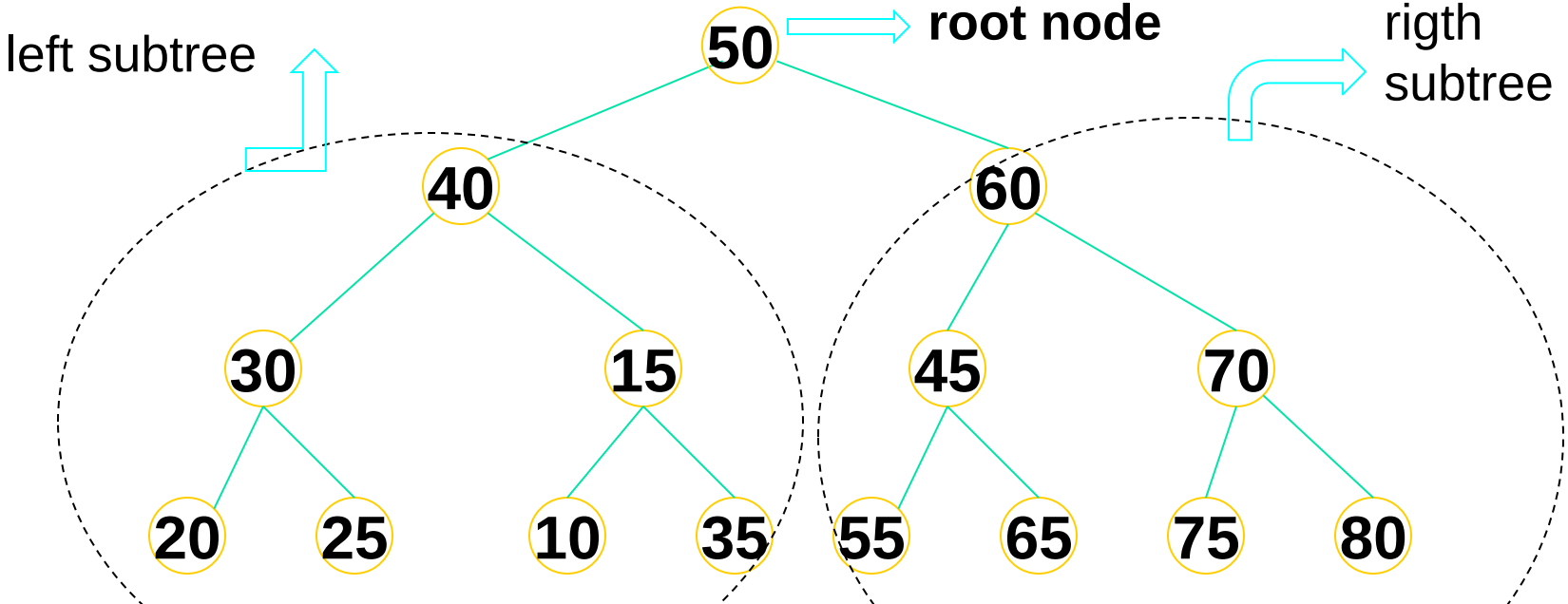
Nội dung

1. Định nghĩa và khái niệm
2. Biểu diễn cây nhị phân
3. Các thao tác trên cây nhị phân
4. Các bài toán trên cây nhị phân
5. Cây nhị phân tìm kiếm
6. Cây nhị phân tìm kiếm cân bằng (AVL)

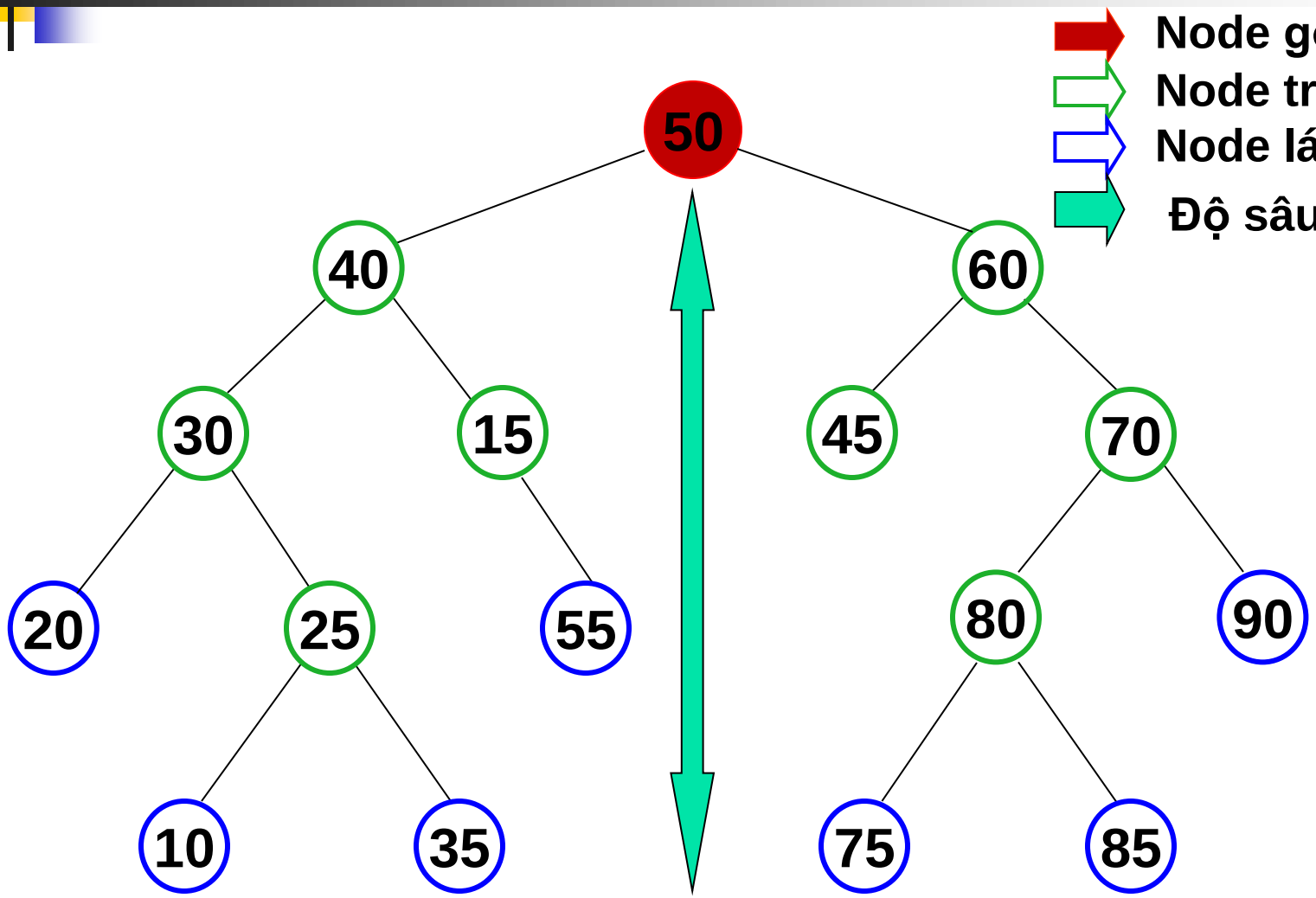
1. Khái niệm cây nhị phân

Định nghĩa:

- Tập hợp hữu hạn các node có cùng kiểu dữ liệu (có thể là tập ∅) được phân thành 3 tập con:
- Một node gọi là node gốc (root).
- Cây con bên trái (left subtree)
- Cây con bên phải (right subtree)



1. Khái niệm cây nhị phân



1. Khái niệm 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.

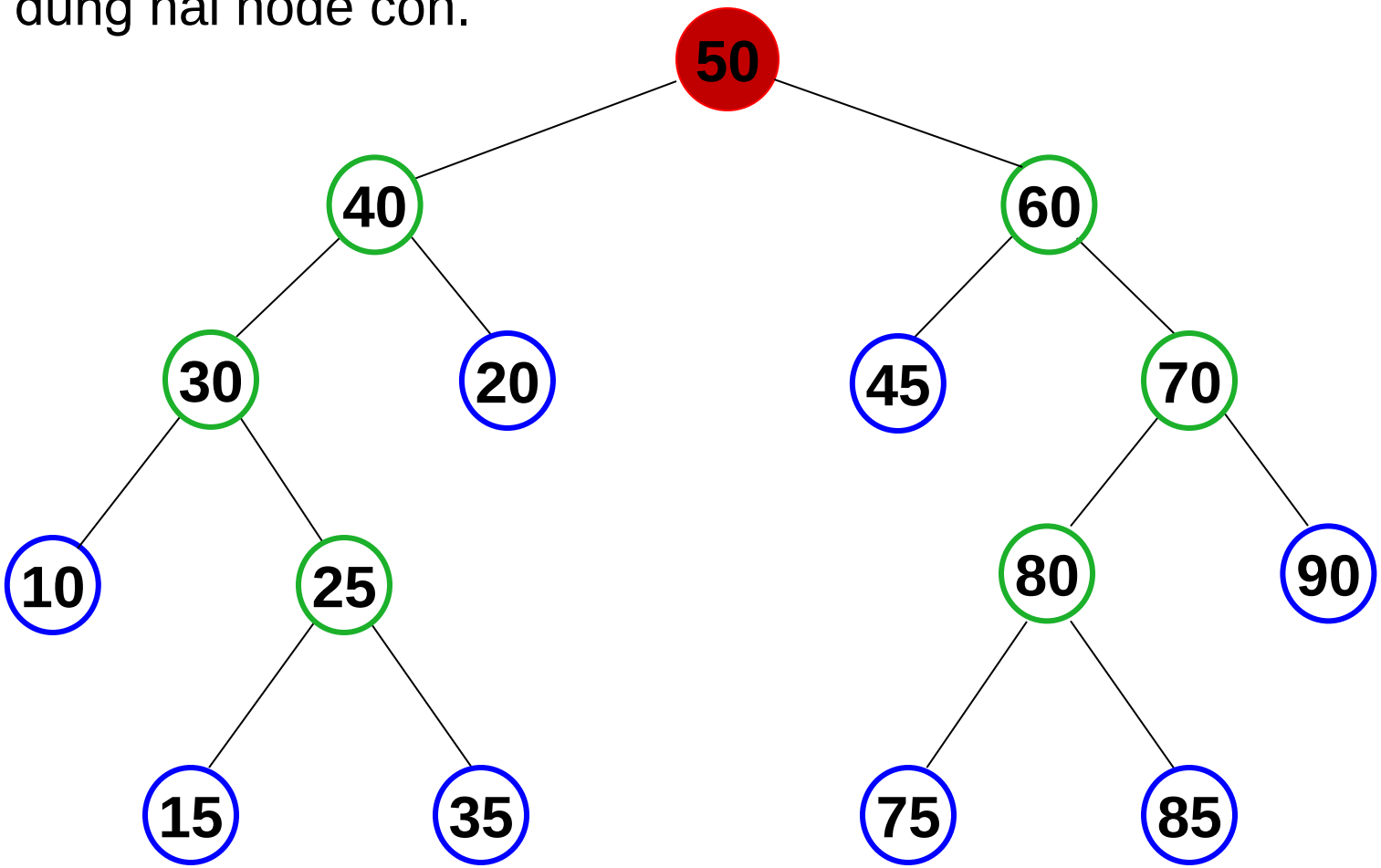
- Cây nhị phân gần đầy (almost complete binary tree):
  - Tất cả node con có mức không nhỏ hơn d-1 đều có hai node con.
  - Các node ở mức d đầy từ trái qua phải.
- Cây nhị phân hoàn toàn cân bằng. Số node thuộc nhánh cây con trái và số node thuộc nhánh cây con phải chênh lệch nhau không quá 1.

1. Khái niệm cây nhị phân

- 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. Giá trị node gốc lớn hơn nội dung node con bên trái và nhỏ hơn giá trị node con bên phải.
  - Hai cây con bên trái và bên phải cũng hình thành nên hai cây tìm kiếm.
- Cây nhị phân tìm kiếm hoàn toàn cân bằng. Chiều sâu cây con trái và chiều sâu cây con phải chênh lệch nhau không quá 1.

1. Phân loại cây nhị phân

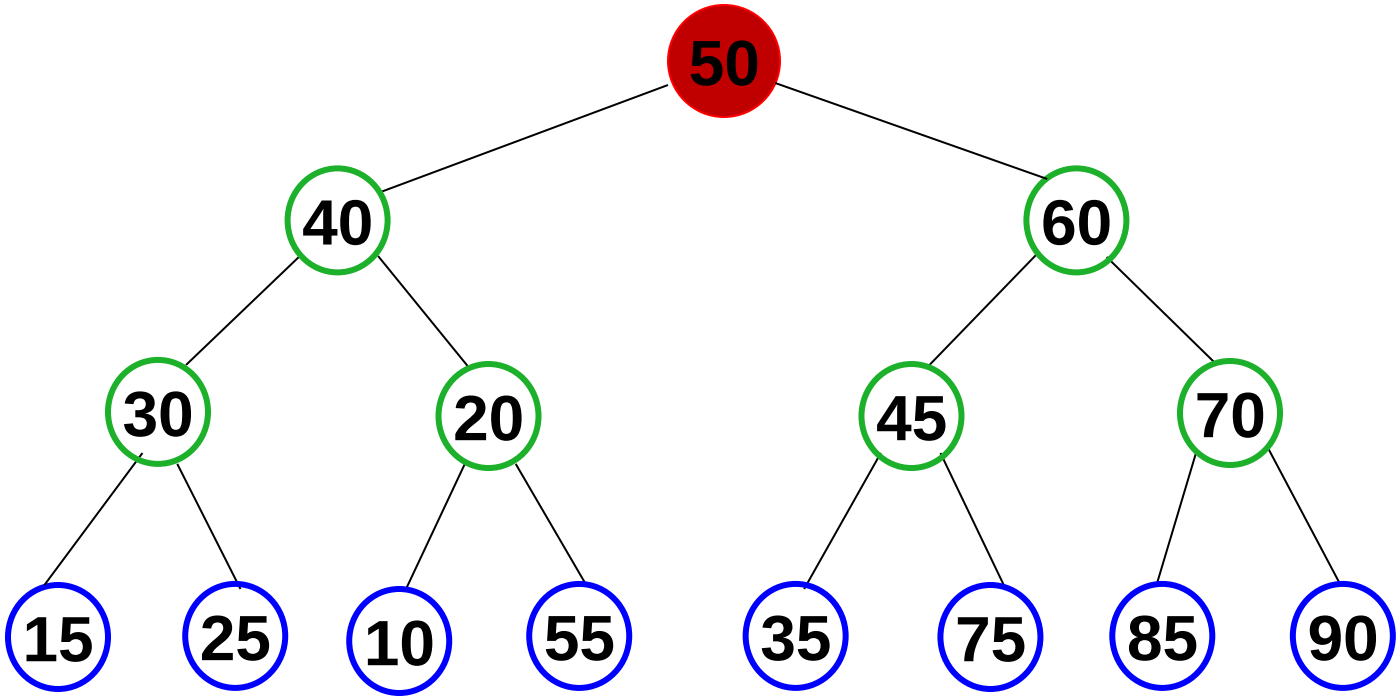
- Cây nhị phân đúng: Node gốc và tất cả các node trung gian có đúng hai node con.



1. Phân loại cây nhị phân

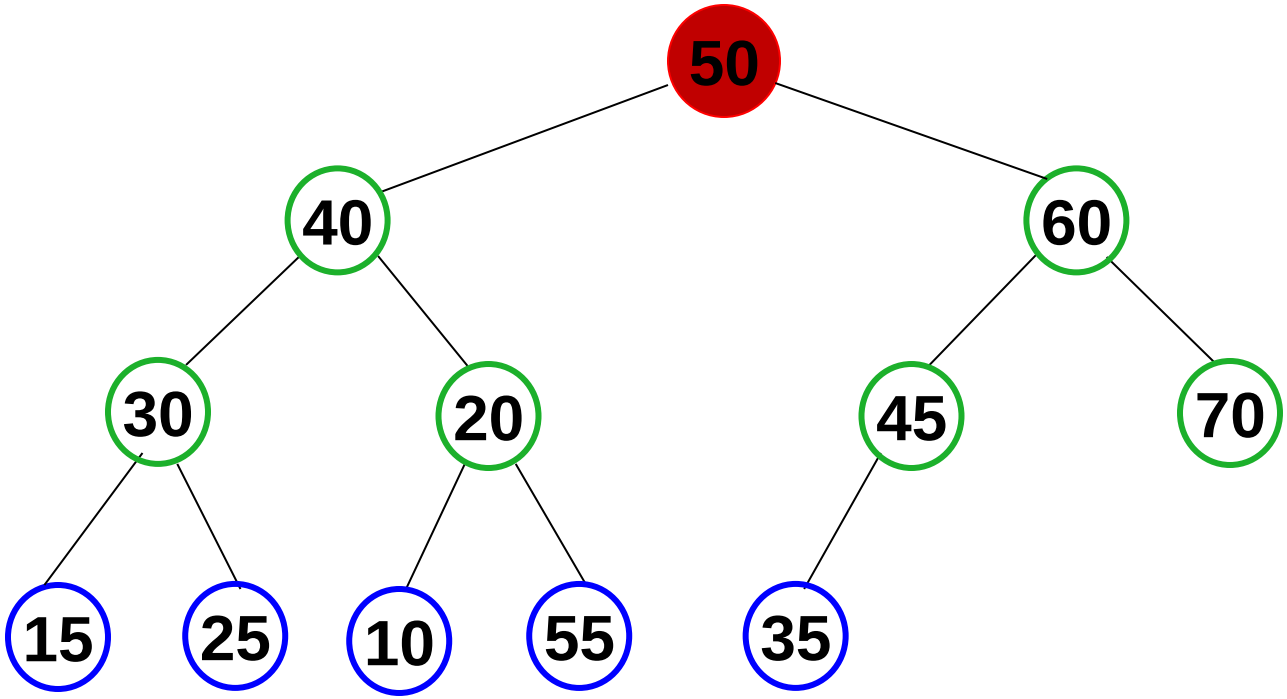
- Cây nhị phân đầy:

Là 1 cây nhị phân đúng và tất cả node lá đều có mức là d.



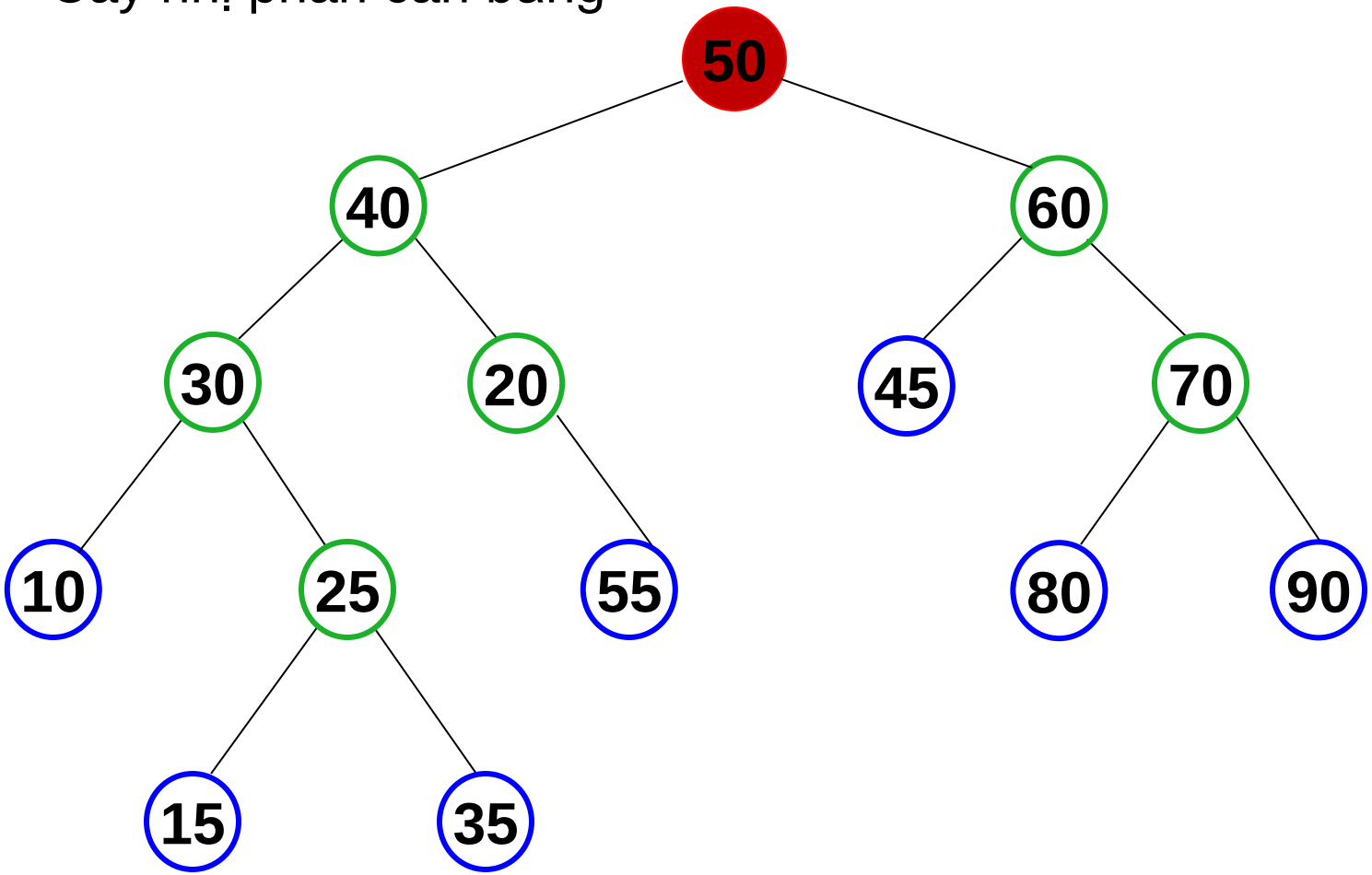
1. Phân loại cây nhị phân

- Cây nhị phân gần đầy
  - 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.



1. Phân loại cây nhị phân

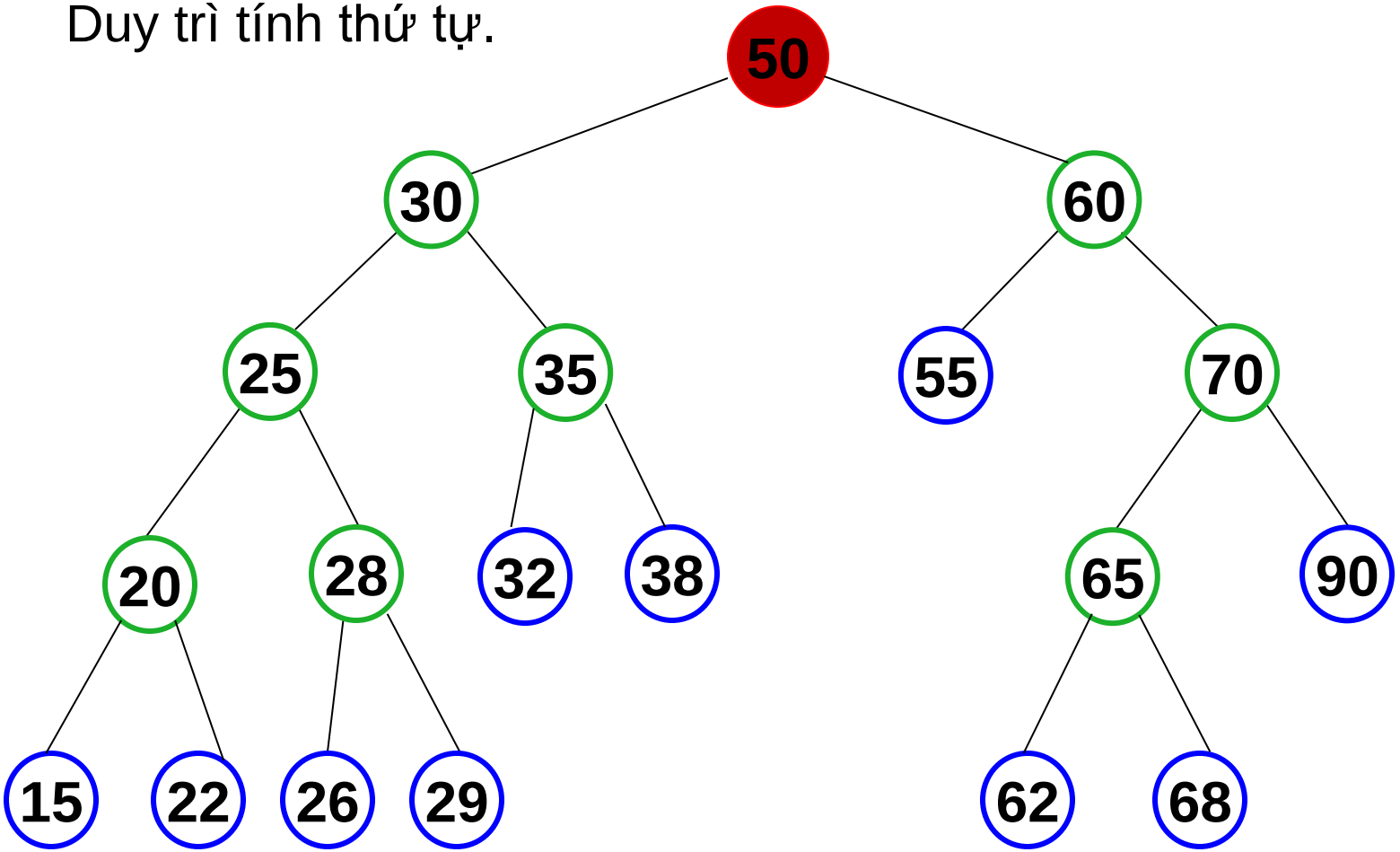
- Cây nhị phân cân bằng



1. Phân loại cây nhị phân

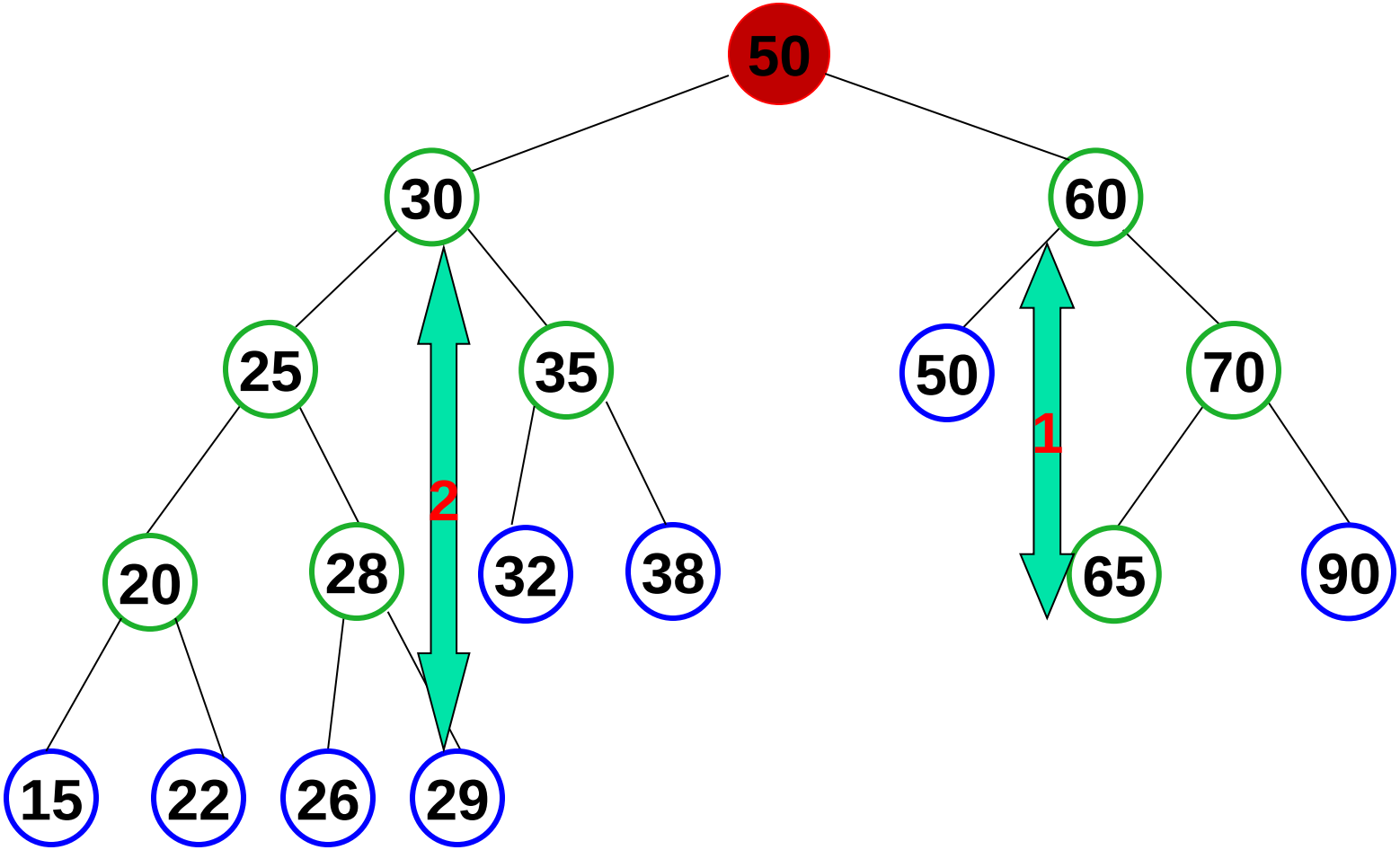
- Cây nhị phân tìm kiếm

  Duy trì tính thứ tự.



1. Phân loại cây nhị phân

- Cây nhị phân tìm kiếm cân bằng:



2. Biểu diễn cây nhị phân

A. Biểu diễn liên tục sử dụng mảng:

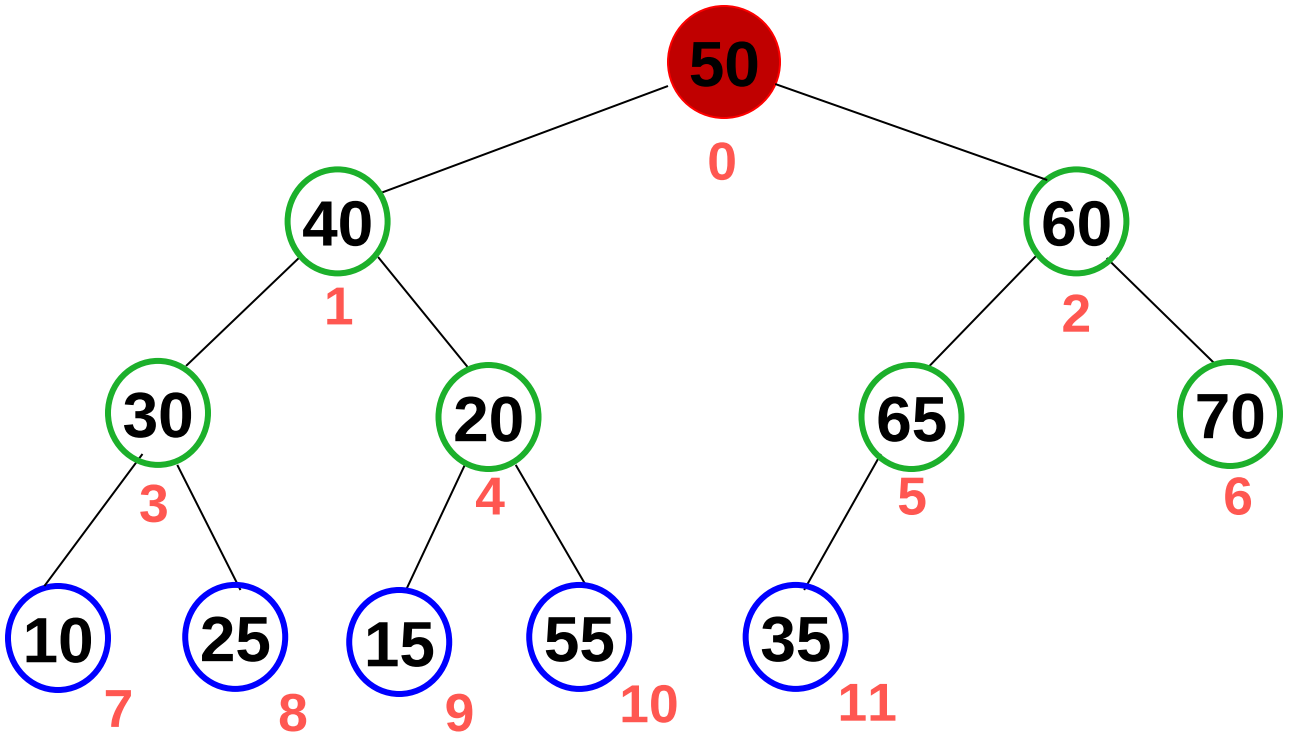
- Node gốc: Lưu trữ ở vị trí 0.
- Nếu node cha lưu trữ ở vị trí p thì node con bên trái của nó được lưu trữ ở vị trí 2p+1, node con phải được lưu trữ ở vị trí 2p+2.
- Ví dụ:

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	∅	∅	∅	...	...	...	...	...

2. Biểu diễn cây nhị phân

- Node gốc: Lưu trữ ở vị trí 0, cha p, trái 2p+1, phải 2p+2

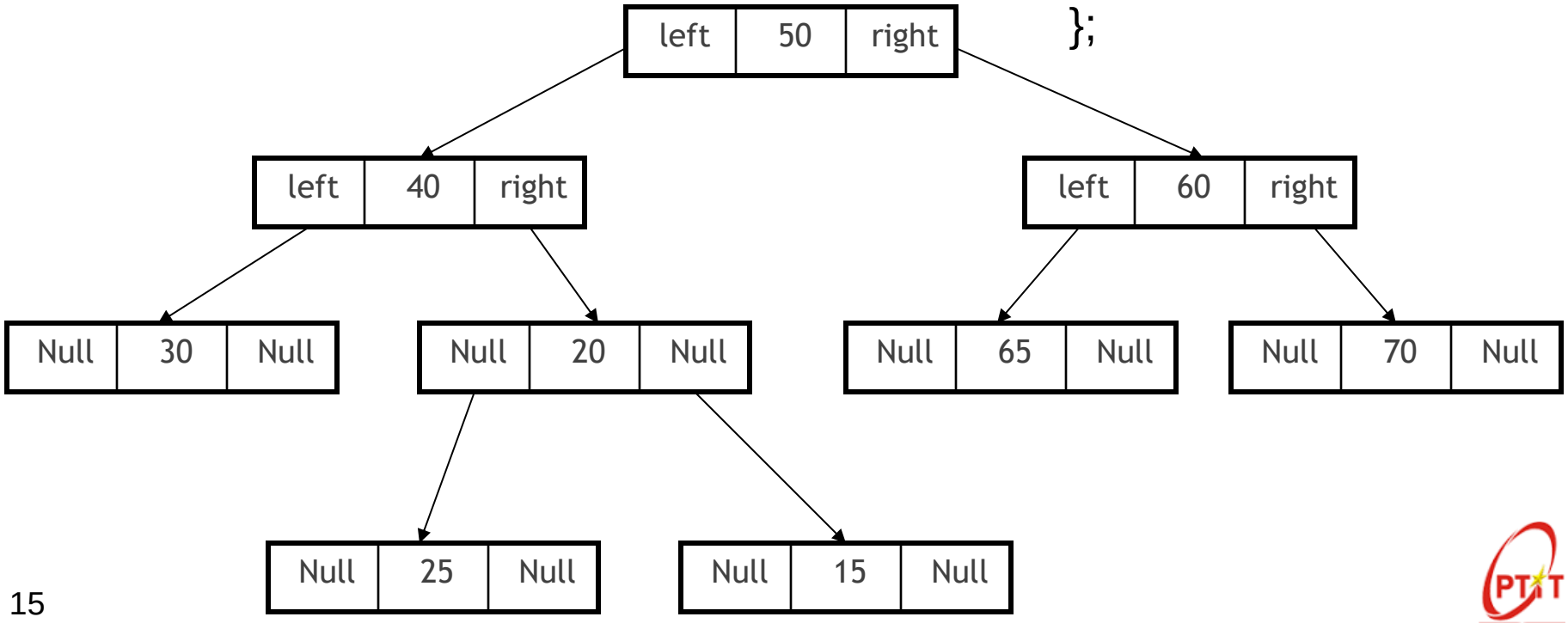
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	...
50	40	60	30	20	65	70	10	25	15	55	35	∅	∅	∅	...	...	...	...	...



2. Biểu diễn cây nhị phân

B. Biểu diễn liên tục sử dụng DSLK:

```
struct Node {
        Item key;
        Node *left;
        Node *right
};
```



3. Các thao tác trên cây nhị phân

- Tạo node gốc cho cây.
- Thêm vào node lá bên trái node p.
- Thêm vào node lá bên phải node p.
- Loại bỏ node lá bên trái node p.
- Loại bỏ node lá bên phải node p.
- Loại bỏ cả cây.
- **Tìm kiếm node trên cây.**
- **Duyệt cây theo thứ tự trước.**
- **Duyệt cây theo thứ tự giữa.**
- **Duyệt cây theo thứ tự sau.**
- **Đọc dữ liệu input bằng danh sách cạnh**

3. Các thao tác trên cây nhị phân

Khai báo struct node bằng danh sách liên kết

```
struct Node {
    int key;
    Node* left;
    Node* right;
    Node(int item) {
        key = item;
        left = NULL;
        right = NULL;
    }
};
```

Khởi tạo:

```
Node* root = NULL;
```

3. Các thao tác trên cây nhị phân

Khởi tạo cây bằng cách gán trực tiếp con trái, phải

```
Node* root = new Node(50);
root->left = new Node(30);
root->right = new Node(70);
root->left->left = new Node(20);
root->left->right = new Node(40);
root->right->left = new Node(60);
root->right->right = new Node(80);
```

3. Các thao tác trên cây nhị phân

Tìm kiếm:

```
Node* Search (Node*  T,  int  x) {
        Node* p;
        if( T->key == x)   {          // Nếu node gốc là node cần tìm
                cout << "YES" << endl;
                return T;             //trả lại node gốc
        }
        if(T==NULL)                   //Nếu T rỗng
                return NULL;          //trả lại giá trị NULL
        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;
}
```

3. Các thao tác trên cây nhị phân

Tạo node lá bên trái node có nội dung là x:

```
void  Add-Left(Node* T, int x, int y ) {
      Node* 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 = new Node(y);                // tạo node lá trái của p là q.
            p -> left = q;                  //Node lá bên trái của p là q
       }
}
```

3. Các thao tác trên cây nhị phân

Tạo node lá bên phải node có nội dung là x:

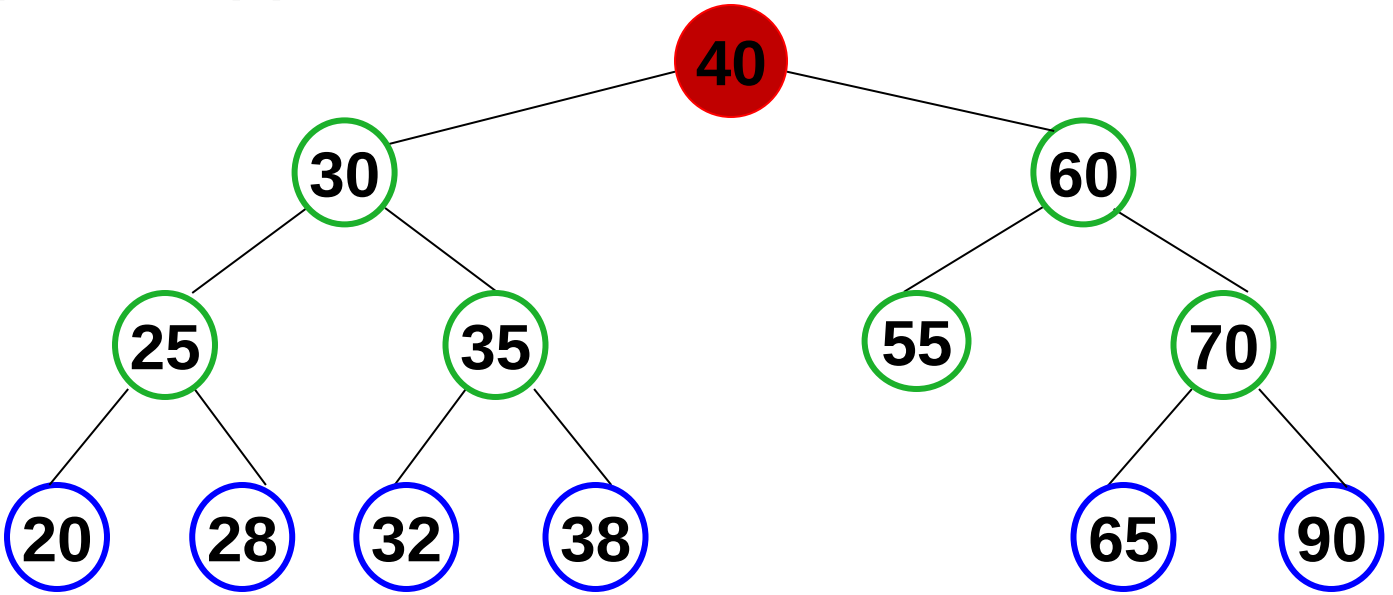
```
void  Add-Left(Node* T, int x, int y ) {
      Node* p, q;
      p = Search (T, x);                    //tìm node có nội dung là x trên cây
      if (p==NULL ) {                       //nếu node có nội dung x không có trên cây
            <thông báo node cha x không có thực>;
            return;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

3. Các thao tác trên cây nhị phân

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

```
void  preOrder(Node* T ) {
      if (T != NULL ) {
            <Thăm node>;          // Cout << T->data << endl;
            preOrder(T -> left);    //duyệt thứ tự giữa sang nhánh cây con bên trái
            preorder(T -> right); //duyệt thứ tự giữa sang nhánh cây con bên phải
      }
}
```

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

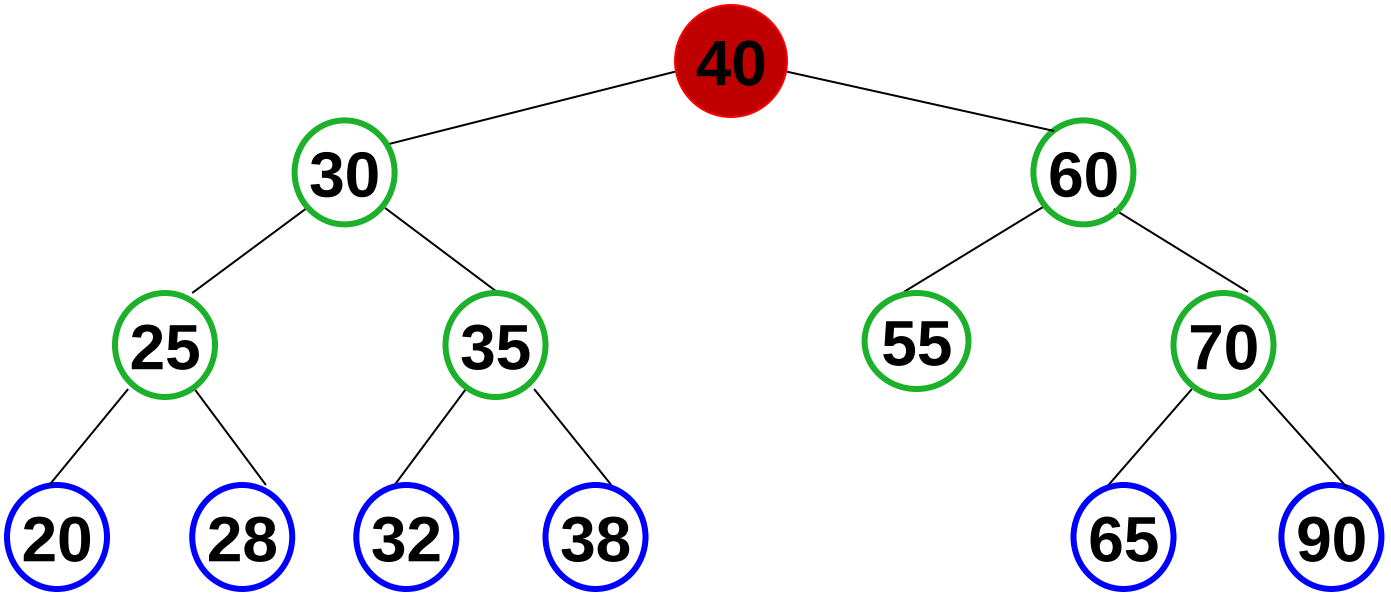


3. Các thao tác trên cây nhị phân

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

```
void  inOrder(Node* T ) {
      if (T!=NULL ) {
            preOrder(T -> left);     //duyệt thứ tự giữa sang nhánh cây con bên trái
            <Thăm node>;
            preorder(T -> right);  //duyệt thứ tự giữa sang nhánh cây con bên phải
      }
}
```

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

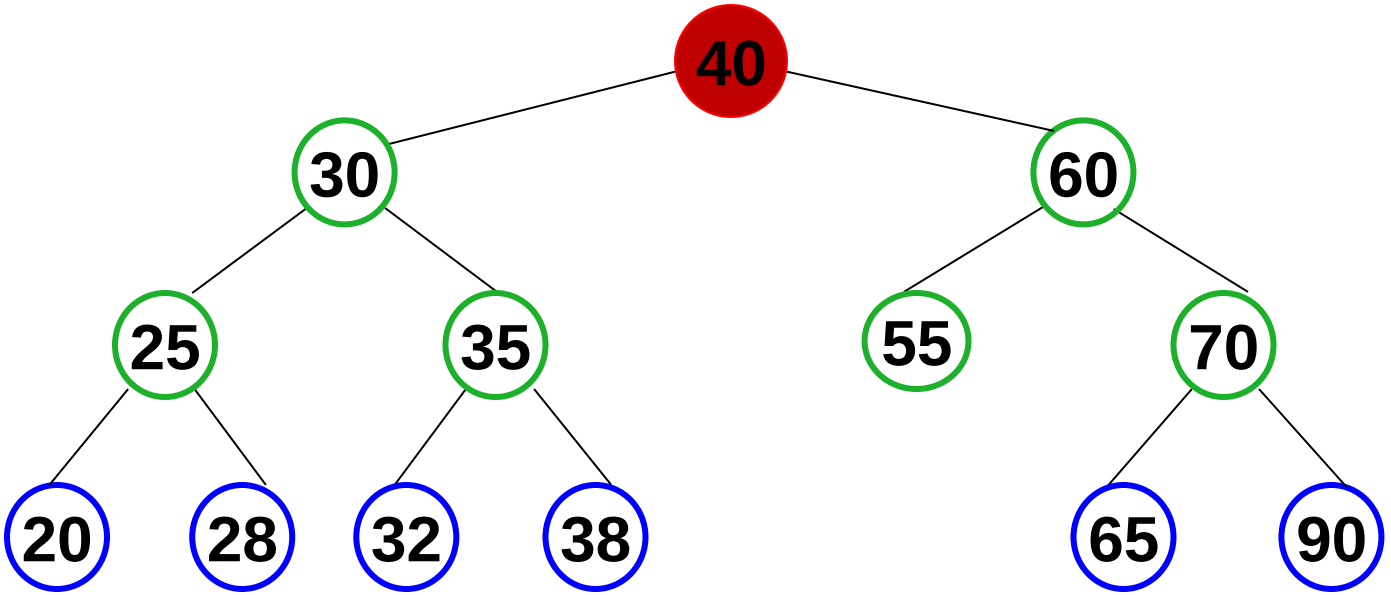


3. Các thao tác trên cây nhị phân

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

```
void  postOrder(Node* T ) {
      if (T!=NULL ) {
            preOrder(T -> left);     //duyệt thứ tự giữa sang nhánh cây con bên trái
            preorder(T -> right);  //duyệt thứ tự giữa sang nhánh cây con bên phải
            <Thăm node>;
      }
}
```

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



3. Các thao tác trên cây nhị phân

Duyệt cây theo thứ tự độ cao

```
void  levelOrder(Node* T ) {
        ..........
}
```

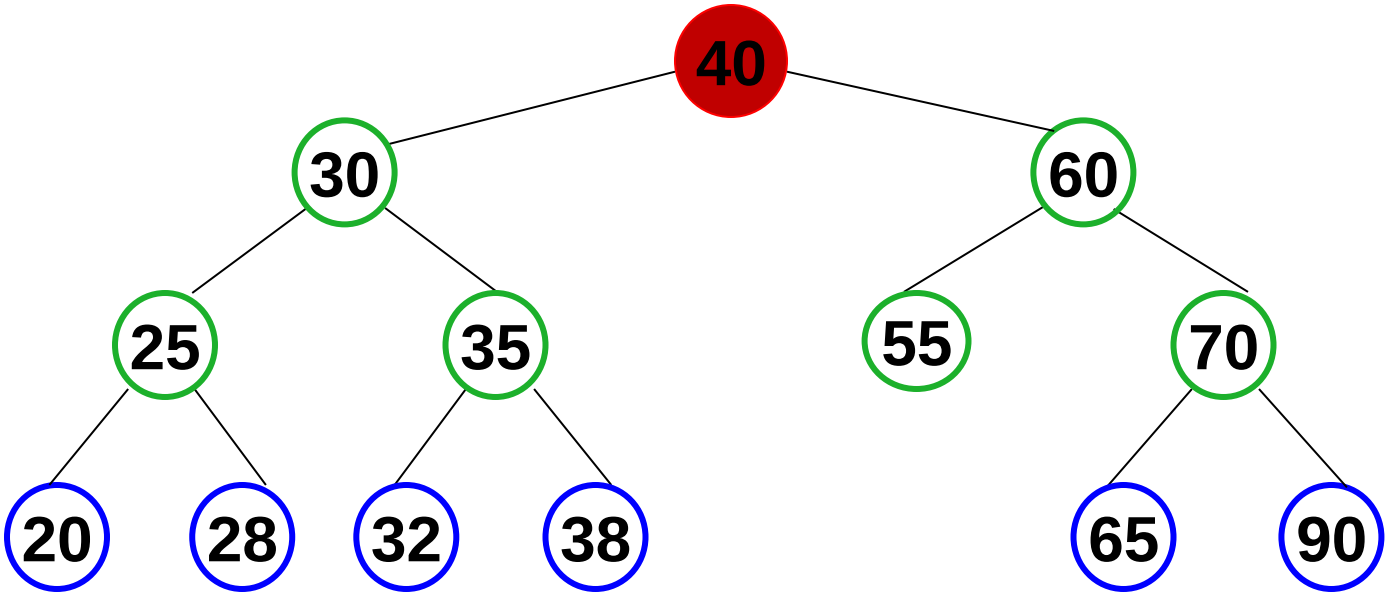
levelOrder(T)

40

30 60

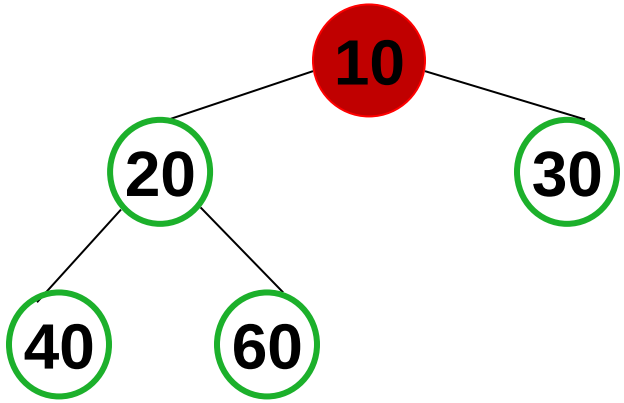
25 35 55 70

20 28 32 38 65 90



3. Các thao tác trên cây nhị phân

Xử lý input đầu vào với danh sách cạnh:
Dòng đầu tiên là số N là số lượng cạnh của cây;
Dòng tiếp theo đưa vào N bộ ba (u, v, x), trong đó
u là node cha,
v là node con,
X = R nếu v là con phải, x = L nếu v là con trái;
u, v, x được viết cách nhau một vài khoảng trống.
Input liệt kê các cạnh theo từng tầng.



Input:	Output phép duyệt preorder:
2 1 2 R 1 3 L	1 3 2
4 10 20 L 10 30 R 20 40 L 20 60 R	10 20 40 60 30

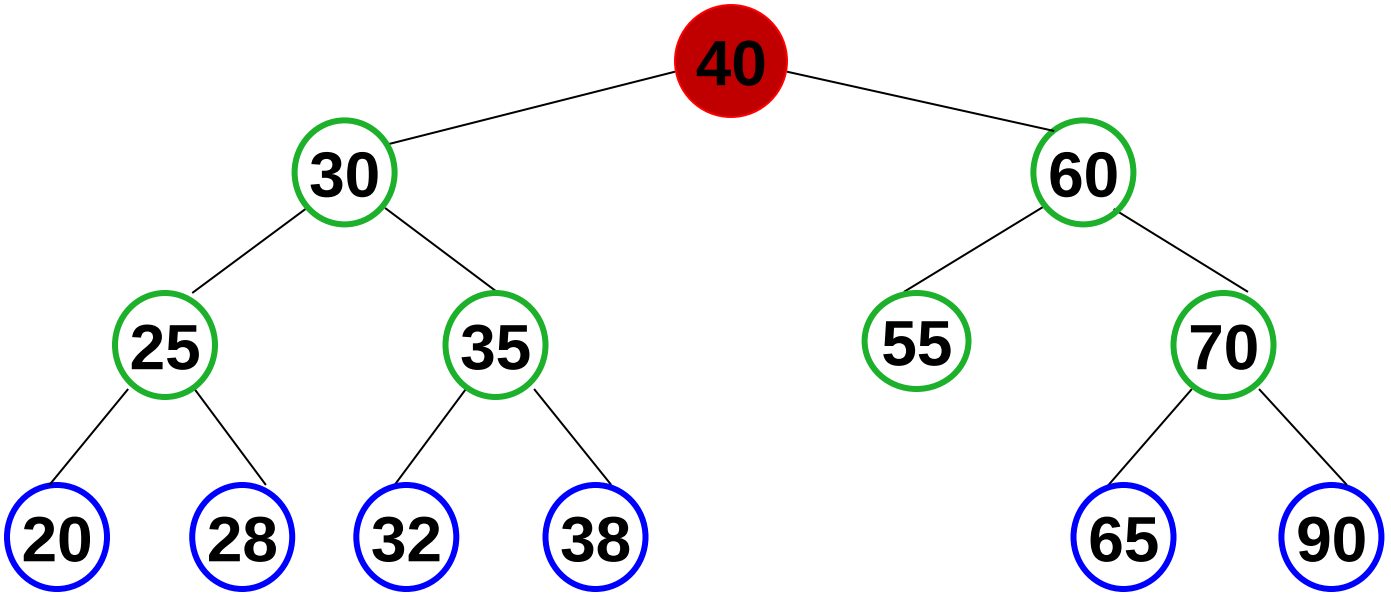
4. Các bài toán trên cây nhị phân

BÀI TOÁN:

- Tính kích thước cây nhị phân
- Xác định 2 cây nhị phân có giống nhau hay không
- Tìm độ cao của cây nhị phân
- Cây phản chiếu
- Tìm đường đi từ gốc đến nút lá
- Đếm số nút lá trên cây

4.1 Tính kích thước cây nhị phân

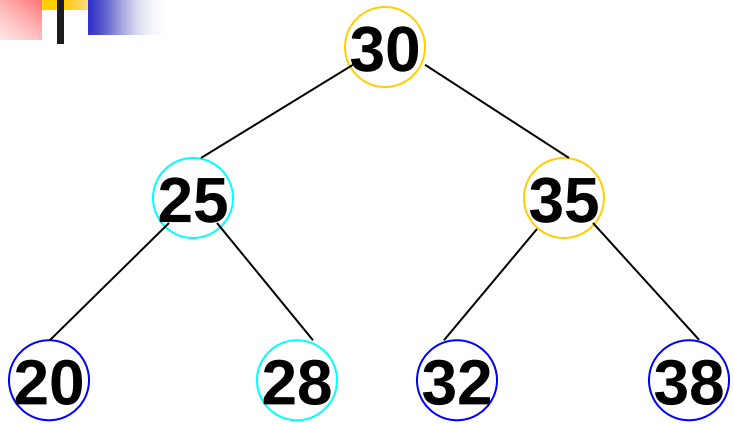
```
int Size(Node* T) {
        if (T==NULL)
                return 0;
        else
                return(Size(T->left) + 1 + Size(T->right));
}
```



4.3 Tìm độ cao của cây nhị phân

```
int maxDepth (Node * T) {
    if (T==NULL)
        return 0;
    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
        // TODO ...
    }
}
```

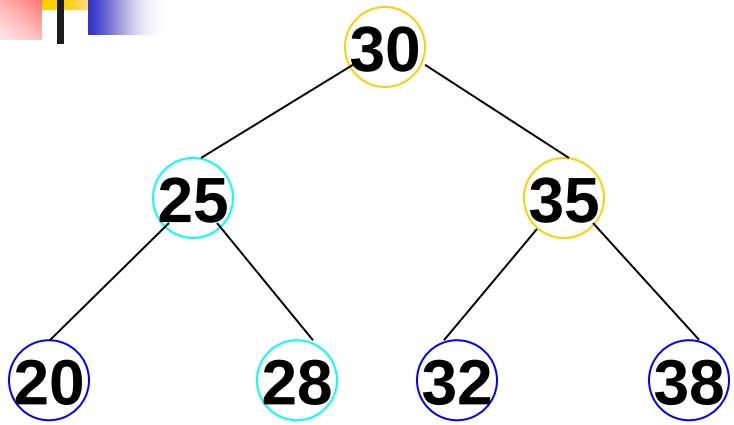
4.5 Tìm đường đi từ nút gốc → lá



Cho một nút lá có xuất hiện trong cây. Yêu cầu in ra đường đi từ gốc → lá.

Gợi ý: Sử dụng DFS + mảng parent[]

4.6 Đếm số nút lá trên cây




```
int getLeafCount (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
        // TODO ...
}
```

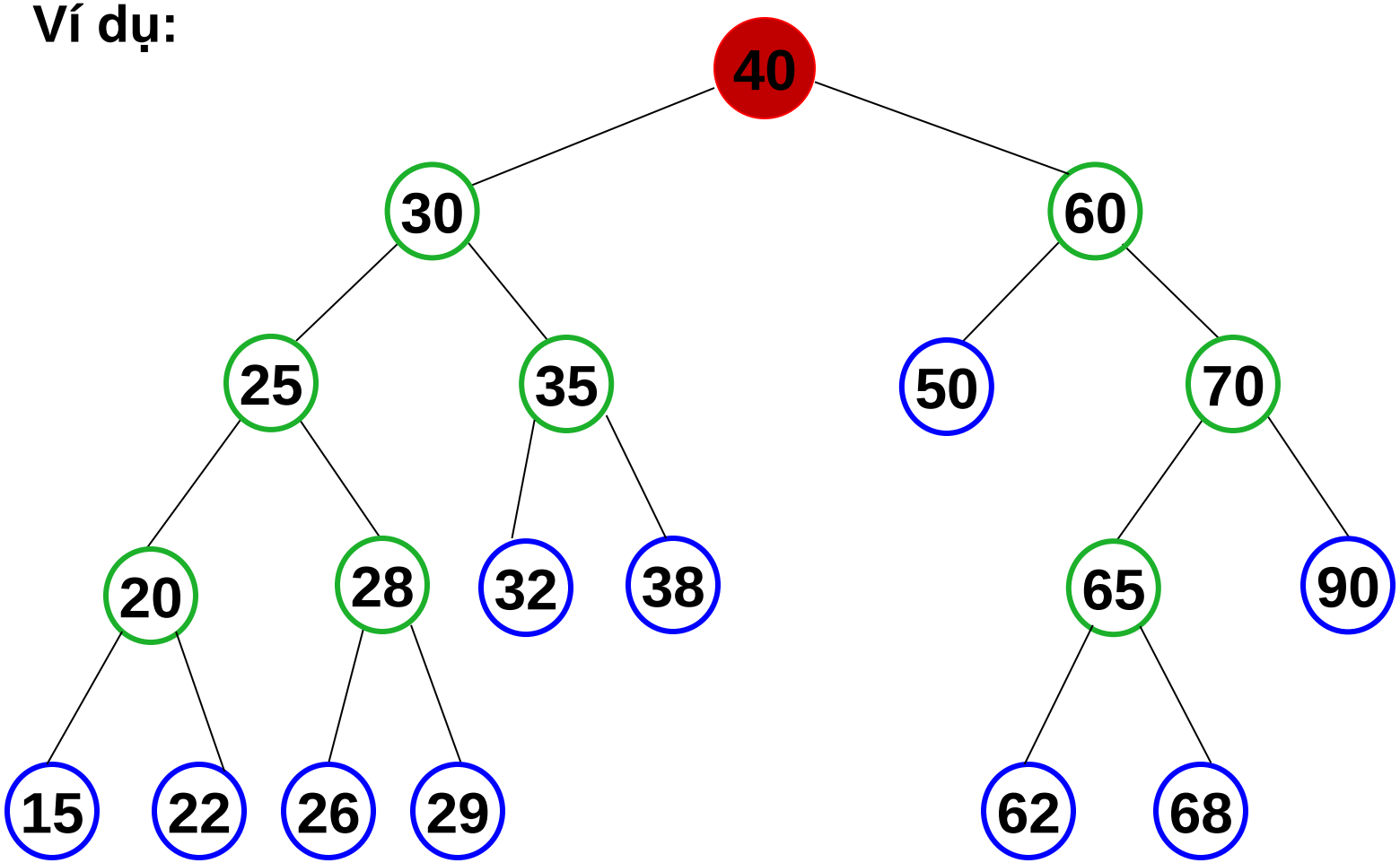
5. Cây nhị phân tìm kiếm

Là cây nhị phân thỏa mãn các điều kiện sau:

- Hoặc là rỗng hoặc có một node gốc.
- Mỗi node gốc có tối đa hai cây con.
- **Tính thứ tự:**
  - 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.

5. Cây nhị phân tìm kiếm

Ví dụ:



5. Cây nhị phân tìm kiếm

Các thao tác

- Tạo node gốc cho cây.
- Thêm vào node vào cây tìm kiếm.
- Loại bỏ node trên cây tìm kiếm.
- Tìm kiếm node trên cây.
- Xoay trái cây tìm kiếm
- Xoay phải cây tìm kiếm
- Duyệt cây theo thứ tự trước.
- Duyệt cây theo thứ tự giữa.
- Duyệt cây theo thứ tự sau.

```
typedef Node {
        Item key;
        Node *left;
        Node *right
};
```

5. Cây nhị phân tìm kiếm

Khai báo struct node bằng danh sách liên kết

```
struct Node {
   int key;
   Node* left;
   Node* right;
   Node(int item) {
      key = item;
      left = NULL;
      right = NULL;
   }
};
```

Khởi tạo:

```
Node* root = NULL;
```

5. Cây nhị phân tìm kiếm

Thêm phần tử:

Kiểm tra giá trị được chèn (giả sử X) với giá trị của nút hiện tại (key) chúng ta đang ở:

- Nếu X nhỏ hơn key thì chuyển sang cây con bên trái.
- Nếu không thì di chuyển tới cây con bên phải.
- Khi đã đến nút lá, hãy chèn X vào bên phải hoặc bên trái dựa trên mối quan hệ giữa X và giá trị của nút lá.

5. Cây nhị phân tìm kiếm

Thêm phần tử:

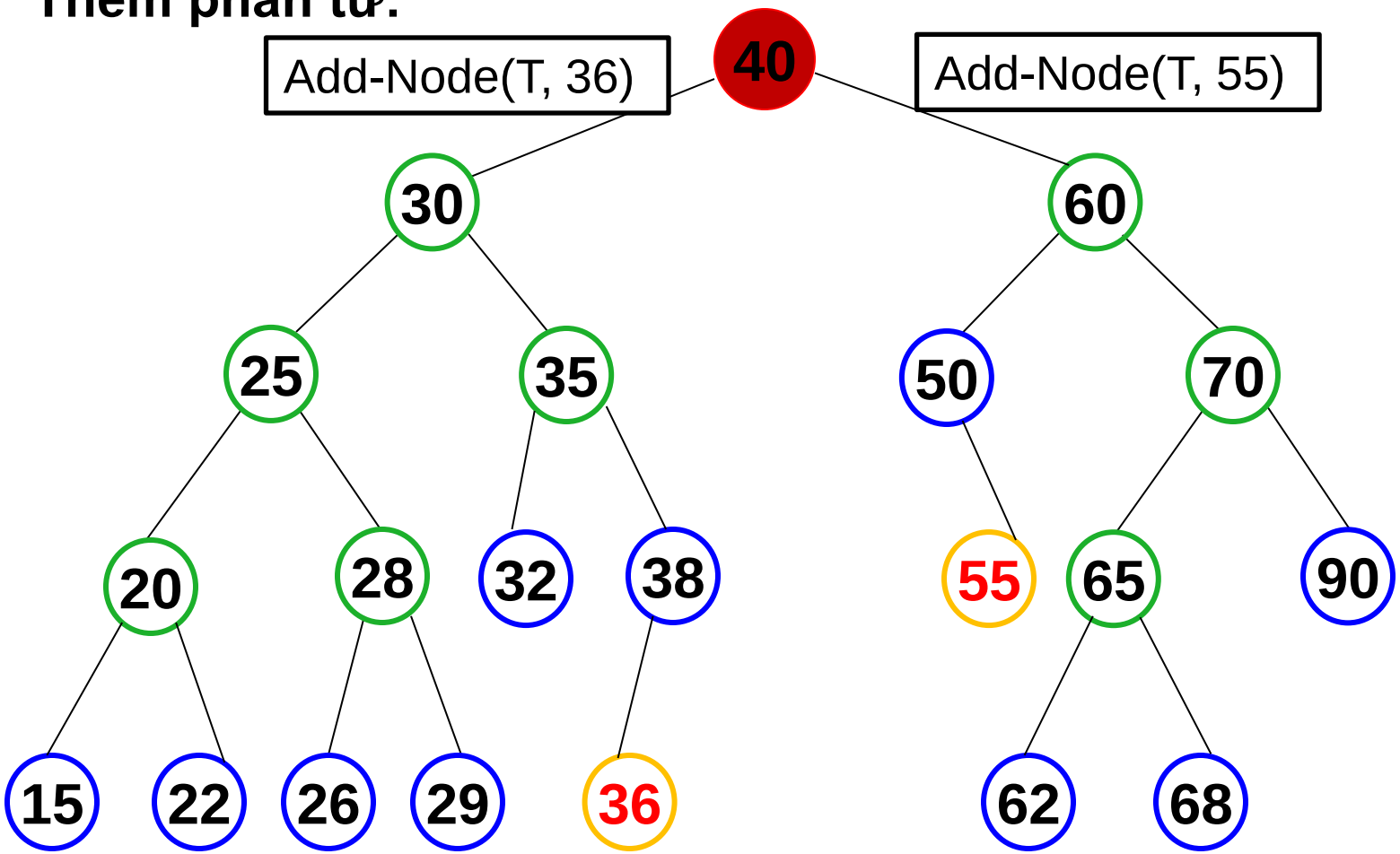
```
Node* insert(Node* node, int X) {
        if (node == NULL)
                return new Node(X);
        if (node->key == X)
                return node;

        if (node->key < X)
                node->light= insert(node->right, X);
        else
                node->left = insert(node->left, X);

        return node; // Return the (unchanged) node pointer
}
```

5. Cây nhị phân tìm kiếm

Thêm phần tử:



5. Cây nhị phân tìm kiếm

Tìm kiếm:

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

5. Cây nhị phân tìm kiếm

Tìm kiếm (không đệ qui):

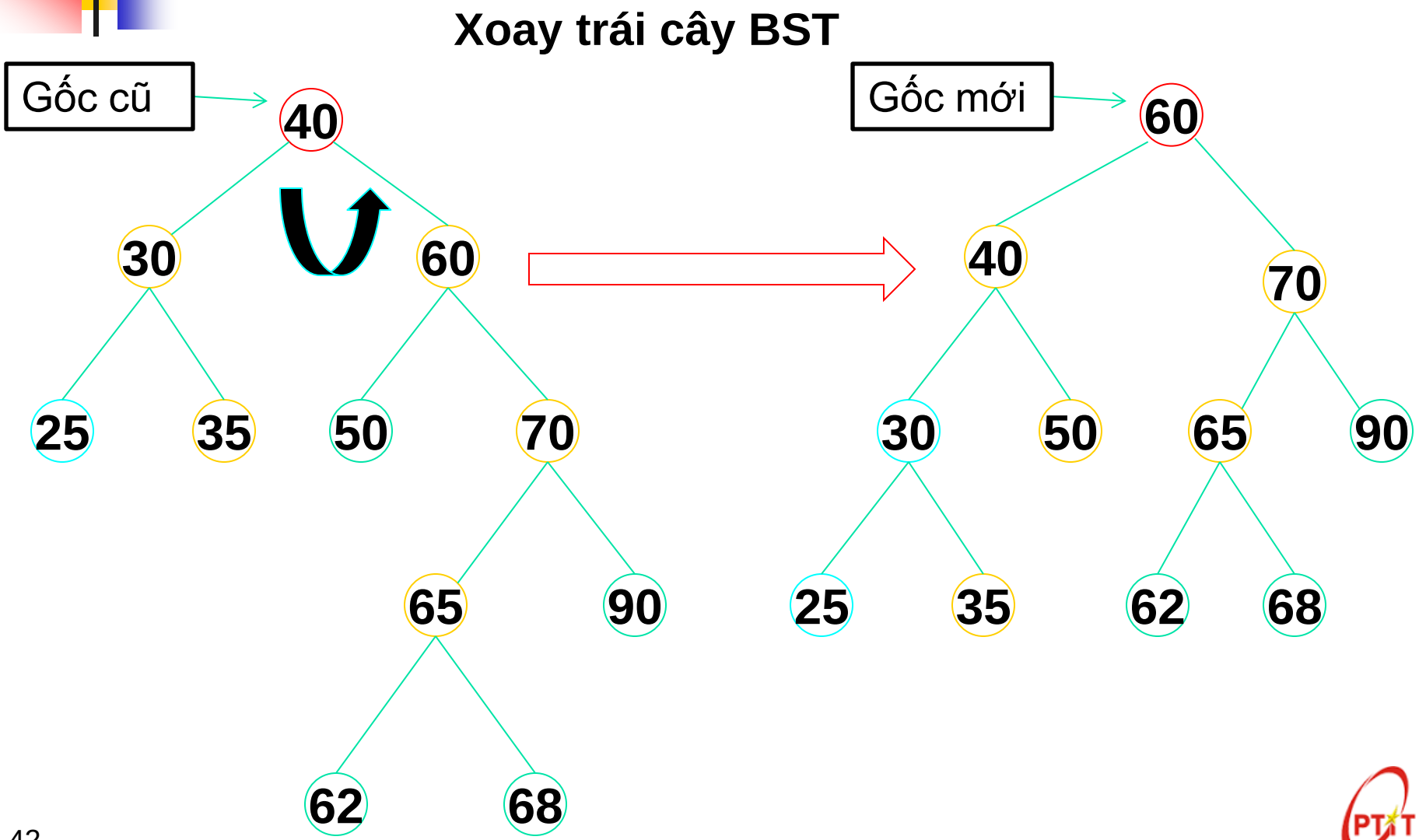
```
Node* Search(Node*  T, Item x){
            //
}
```

5. Cây nhị phân tìm kiếm

Xoay trái cây BST: chuyển node gốc sang trái, con phải → gốc mới

```
Node* RotateLeft (Node* T ) {
        Node* p = T;                        // p trỏ 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 trỏ đến cây con phải
                T -> right = p ->left;      //T trỏ đến node trái của p
                p -> left = T;              //Thiết lập liên kết trái cho p
        }
        return p;
}
```

5. Cây nhị phân tìm kiếm



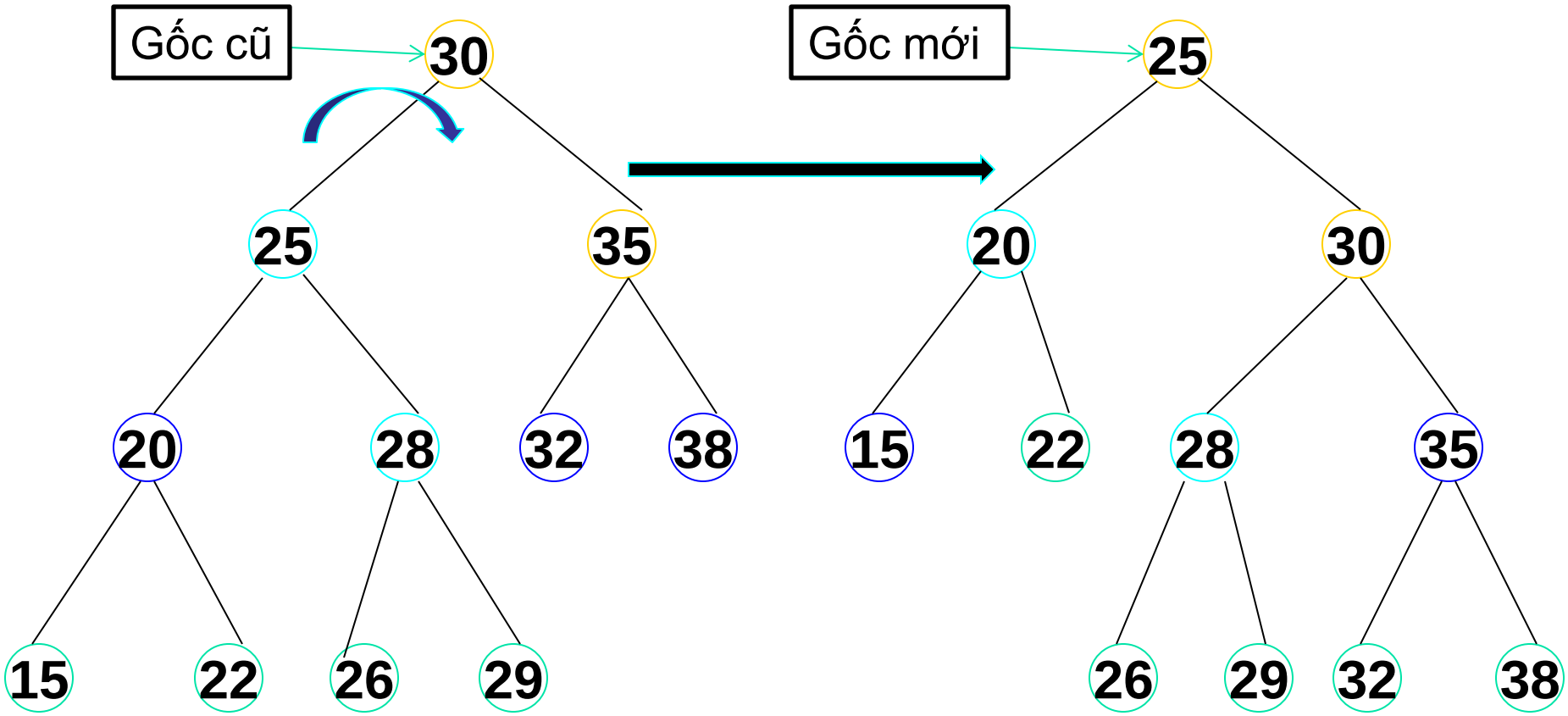
5. Cây nhị phân tìm kiếm

Xoay phải cây BST: chuyển node gốc sang phải, con phải → gốc mới

```
Node* RotateLeft (Node* T ) {
        Node* p = T;                        // p trỏ đến node gốc của cây
        if (T == NULL)
                <Cây rỗng>;
        else if (T -> right == 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;
}
```

5. Cây nhị phân tìm kiếm

Xoay phải cây BST



QUESTIONS & ANSWERS