



## CHƯƠNG 2. DUYỆT VÀ ĐỆ QUI

### NỘI DUNG:

- 2.1. Thuật toán duyệt
- 2.2. Thuật toán sinh
- 2.3. Thuật toán đệ qui
- 2.4. Thuật toán quay lui
- 2.5. Khử đệ qui
- 2.6. CASE STUDY

## 2.1. Thuật toán duyệt

**Phương pháp giải quyết bài toán (Problem):**

**Sử dụng các công cụ toán học:**

- Sử dụng các định lý, mệnh đề, lập luận và suy logic của trong toán học để tìm ra nghiệm của bài toán.
- Ưu điểm: dễ dàng máy tính hóa các bài toán đã có thuật giải (MathLab).
- Nhược điểm: chỉ thực hiện được trên lớp các bài toán đã có thuật giải. Lớp bài toán này rất nhỏ so với lớp các bài toán thực tế.

**Sử dụng máy tính và các công cụ tính toán:**

- Giải được mọi bài toán đã có thuật giải bằng máy tính.
- Đối với một số bài toán chưa có thuật giải, ta có thể sử dụng máy tính để xem xét tất cả các khả năng có thể để từ đó đưa ra nghiệm của bài toán. Một thuật toán duyệt cần thỏa mãn hai điều kiện:
  - Không được lặp lại bất kỳ khả năng nào.
  - Không được bỏ sót bất kỳ cấu hình nào.

**Ví dụ 1.** Cho hình vuông gồm 25 hình vuông đơn vị. Hãy điền các số từ 0 đến 9 vào mỗi hình vuông đơn vị sao cho những điều kiện sau được thỏa mãn.

- Đọc từ trái sang phải theo hàng ta nhận được 5 số nguyên tố có 5 chữ số;
- Đọc từ trên xuống dưới theo cột ta nhận được 5 số nguyên tố có 5 chữ số;
- Đọc theo hai đường chéo chính ta nhận được 2 số nguyên tố có 5 chữ số;
- Tổng các chữ số trong mỗi số nguyên tố đều là S cho trước.

Ví dụ hình vuông dưới đây với  $S = 11$ .

|   |   |   |   |   |
|---|---|---|---|---|
| 3 | 5 | 1 | 1 | 1 |
| 5 | 0 | 0 | 3 | 3 |
| 1 | 0 | 3 | 4 | 3 |
| 1 | 3 | 4 | 2 | 1 |
| 1 | 3 | 3 | 1 | 3 |

**Thuật giải duyệt.** Chia bài toán thành hai bài toán con như sau:

- Tìm  $X = \{x \in [10001, \dots, 99999] \mid x \text{ là nguyên tố và tổng các chữ số là } S\}$ .
- Chiến lược vét cạn được thực hiện như sau:
  - Lấy  $x \in X$  đặt vào hàng 1(H1): ta điền được ô vuông 1, 2, 3, 4, 5.
  - Lấy  $x \in X$  có số đầu tiên trùng với ô số 1 đặt vào cột 1 (C1): ta điền được ô vuông 6, 7, 8, 9.
  - Lấy  $x \in X$  có số đầu tiên trùng với ô số 9, số cuối cùng trùng với ô số 5 đặt vào đường chéo chính 2 (D2): ta điền được ô vuông 10, 11, 12.
  - Lấy  $x \in X$  có số thứ nhất và số thứ 4 trùng với ô số 6 và 12 đặt vào hàng 2 (H2): ta điền được ô vuông 13, 14, 15.
  - Lấy  $x \in X$  có số thứ nhất, thứ hai, thứ 4 trùng với ô số 2, 13, 10 đặt vào cột 2 (C2): ta điền được ô vuông 16, 17.
  - Làm tương tự như vậy ta điền vào hàng 5 ô số 25.
  - Cuối cùng ta chỉ cần kiểm tra  $D1 \in X$  và  $C5 \in X$ ?

|   |   |   |   |   |
|---|---|---|---|---|
| 3 | 5 | 1 | 1 | 1 |
| 5 | 0 | 0 | 3 | 3 |
| 1 | 0 | 3 | 4 | 3 |
| 1 | 3 | 4 | 2 | 1 |
| 1 | 3 | 3 | 1 | 3 |

|   |    |    |    |    |
|---|----|----|----|----|
| 1 | 2  | 3  | 4  | 5  |
| 6 | 13 | 14 | 12 | 15 |
| 7 | 16 | 11 | 18 | 19 |
| 8 | 10 | 20 | 22 | 23 |
| 9 | 17 | 21 | 24 | 25 |

## 2.2. Thuật toán sinh (Generation)

Thuật toán sinh được dùng để giải lớp các bài toán thỏa mãn hai điều kiện:

- Xác định được một thứ tự trên tập các cấu hình cần liệt kê của bài toán. Biết được cấu hình đầu tiên, biết được cấu hình cuối cùng.
- Từ một cấu hình cuối cùng, ta xây dựng được thuật toán sinh ra cấu hình đứng ngay sau nó theo thứ tự.

**Thuật toán:**

**Thuật toán Generation:**

**Bước1 (Khởi tạo):**

<Thiết lập cấu hình đầu tiên>;

**Bước 2 (Bước lặp):**

**while** (<Lặp khi cấu hình chưa phải cuối cùng>) **do**

<Đưa ra cấu hình hiện tại>;

<Sinh ra cấu hình kế tiếp>;

**endwhile;**

**End.**

**Ví dụ 1.** Cho số tự nhiên  $N$  ( $N \leq 100$ ). Hãy liệt kê tất cả các cách chia số tự nhiên  $N$  thành tổng của các số tự nhiên nhỏ hơn  $N$ . Các cách chia là hoán vị của nhau chỉ được tính là một cách.

Ví dụ với  $N=5$  ta có 7 cách chia như sau:

|   |   |   |   |   |
|---|---|---|---|---|
| 5 |   |   |   |   |
| 4 | 1 |   |   |   |
| 3 | 2 |   |   |   |
| 3 | 1 | 1 |   |   |
| 2 | 2 | 1 |   |   |
| 2 | 1 | 1 | 1 |   |
| 1 | 1 | 1 | 1 | 1 |

**Thuật toán:**

**Input:**

- Số tự nhiên  $n$ .

**Output:**

- Các cách chia  $n$  thành tổng các số nhỏ hơn  $n$ .

**Actions:**

*Bước 1(Khởi tạo):*

$k = 1; X[k] = n; OK = TRUE; //Đưa ra cách chia đầu tiên.$

*Bước 2 (Lặp):*

$\text{while } (OK) \{ //Lặp đến khi nào cách chia không phải là n số 1$

$\text{Result(); //Đưa ra phương án hiện tại.}$

$\text{Next_Division(); //Sinh ra phép chia kế tiếp}$

$\}$

**EndAction.**

Procedure Next\_Division() { int i = k, j, R, S,D;

$\text{while } (i > 0 \&\& X[i]==1 ) i--; //Lặp từ bên phải khi nào } X[i] = 1$

$\text{if } (i>0 ) \{ X[i] = X[i] - 1; D = k - i + 1; R = D / X[i]; S = D \% X[i]; k= i;$

$\text{if } (R>0) \{$

$\text{for } ( j = i + 1; j<=i + R; j++) X[j] = X[i];$

$\text{k = k + R;}$

$\}$

$\text{if } (S>0 )\{ k = k +1; X[k] = S; \}$

$\}$

$\text{else } OK =0;$

$\}$

## //Hãy cho biết kết quả thực hiện ?

```
#include <iostream.h>
#include <stdlib.h>
#define MAX 100
#define TRUE 1
#define FALSE 0
int n, k, X[MAX], dem =0, OK =TRUE;
void Init(void ){
    cout<<"\n Nhập n="; cin>>n;
    k = 1; X[k] = n;
}
void Result(void) {
    cout<<"\n Cách chia "<<++dem<<" :";
    for (int i=1; i<=k; i++)
        cout<<X[i]<<" ";
}
```

```
void Next_Division(void ){
    int i = k, j, R, S,D;
    while (i > 0 && X[i]==1 ) i--;
    if (i>0 ) {
        X[i] = X[i] - 1; D = k - i + 1;
        R = D / X[i]; S = D % X[i];
        k= i;
        if (R>0) {
            for ( j = i +1; j<=i + R; j++)
                X[j] = X[i];
            k = k + R;
        }
        if (S>0 ){
            k = k +1; X[k] = S;
        }
    }
    else OK =0;
}
int main() { Init(); //Nhập n = 5.
    while (OK ) {
        Result(); Next_Division();
    }
    return 0;
}
```

## 2.3. Thuật toán đệ qui (Recursion)

**Phương pháp định nghĩa bằng đệ qui:** Một đối tượng được định nghĩa trực tiếp hoặc gián tiếp thông qua chính nó được gọi là phép định nghĩa bằng đệ qui.

**Thuật toán đệ qui:** Thuật toán giải bài toán P trực tiếp hoặc gián tiếp thông qua bài toán P' giống như P được gọi là thuật toán đệ qui. Một hàm được gọi là đệ qui nếu nó được gọi trực tiếp hoặc gián tiếp đến chính nó. Một bài toán giải được bằng đệ qui nếu nó thỏa mãn hai điều kiện:

- **Phân tích được:** Có thể giải được bài toán P bằng bài toán P' giống như P và chỉ khác P ở dữ liệu đầu vào. Việc giải bài toán P' cũng được thực hiện theo cách phân tích giống như P.
- **Điều kiện dừng:** Dãy các bài toán P' giống như P là hữu hạn và sẽ dừng tại một bài toán xác định nào đó.

Thuật toán đệ qui tổng quát có thể được mô tả như sau:

*Thuật toán Recursion (P) {*

    1. Nếu P thỏa mãn điều kiện dừng:

*<Giải P với điều kiện dừng>;*

    2. Nếu P không thỏa mãn điều kiện dừng:

*Recursion(P').*

}

**Ví dụ:** Tìm tổng của n số tự nhiên bằng phương pháp đệ qui.

**Lời giải.** Gọi  $S_n$  là tổng của n số tự nhiên. Khi đó:

- **Bước phân tích:**  $S_n = n + S_{n-1}$ ,  $n > 1$ .
- **Điều kiện dừng:**  $s_1 = 1$  nếu  $n=1$ ;

Từ đó ta có lời giải của bài toán như sau:

```
int      Tong (int i ) {  
    if (i ==1 ) return(1); //Điều kiện dừng  
    else return(i + Tong(i-1)); //Điều kiện phân tích được  
}
```

**Ví dụ.** Tìm  $n!$ .

**Lời giải.** Gọi  $S_n$  là  $n!$ . Khi đó:

- **Bước phân tích:**  $S_n = n * (n-1)!$  nếu  $n > 0$ ;
- **Điều kiện dừng:**  $s_0=1$  nếu  $n=0$ .

Từ đó ta có lời giải của bài toán như sau:

```
long     Giaithua (int i ) {  
    if (i ==0 ) return(1); //Điều kiện dừng  
    else return(i *Giaithua(i-1)); //Điều kiện phân tích được  
}
```

**Ví dụ:** Tìm ước số chung lớn nhất của a và b bằng phương pháp đệ qui.

**Lời giải.** Gọi d =USCLN(a,b). Khi đó:

- **Bước phân tích:**

- $d = \text{USCLN}(a-b, b)$  nếu  $a>b$ .

- $d = \text{USCLN}(a, b-a)$  nếu  $a<b$ .

- **Điều kiện dừng:**  $d =a$  hoặc  $d=b$  nếu  $a=b$ ;

Từ đó ta có lời giải của bài toán như sau:

```
int    USCLN (int a, int b ) {  
    if (a ==b ) return(a); //Điều kiện dừng  
    else { //Điều kiện phân tích được  
        if (a> b) return(USCLN(a-b, b));  
        else return(USCLN(a, b-a));  
    }  
}
```

**Ví dụ . Hãy cho biết kết quả thực hiện chương trình dưới đây?**

```
#include <stdio.h>
#include <conio.h>
int convert(int X, int b){
    if (X == 0) return 0;
    else return (X % b + 10 * convert(X / b));
}
int main(){
    int X, b, bin;
    printf("Nhập số X: "); scanf("%d", &X);
    printf("Nhập số b: "); scanf("%d", &b);
    bin = convert(X, b);
    printf("Kết quả %d là %d.\n", X, bin);
    getch(); return 0;
}
```

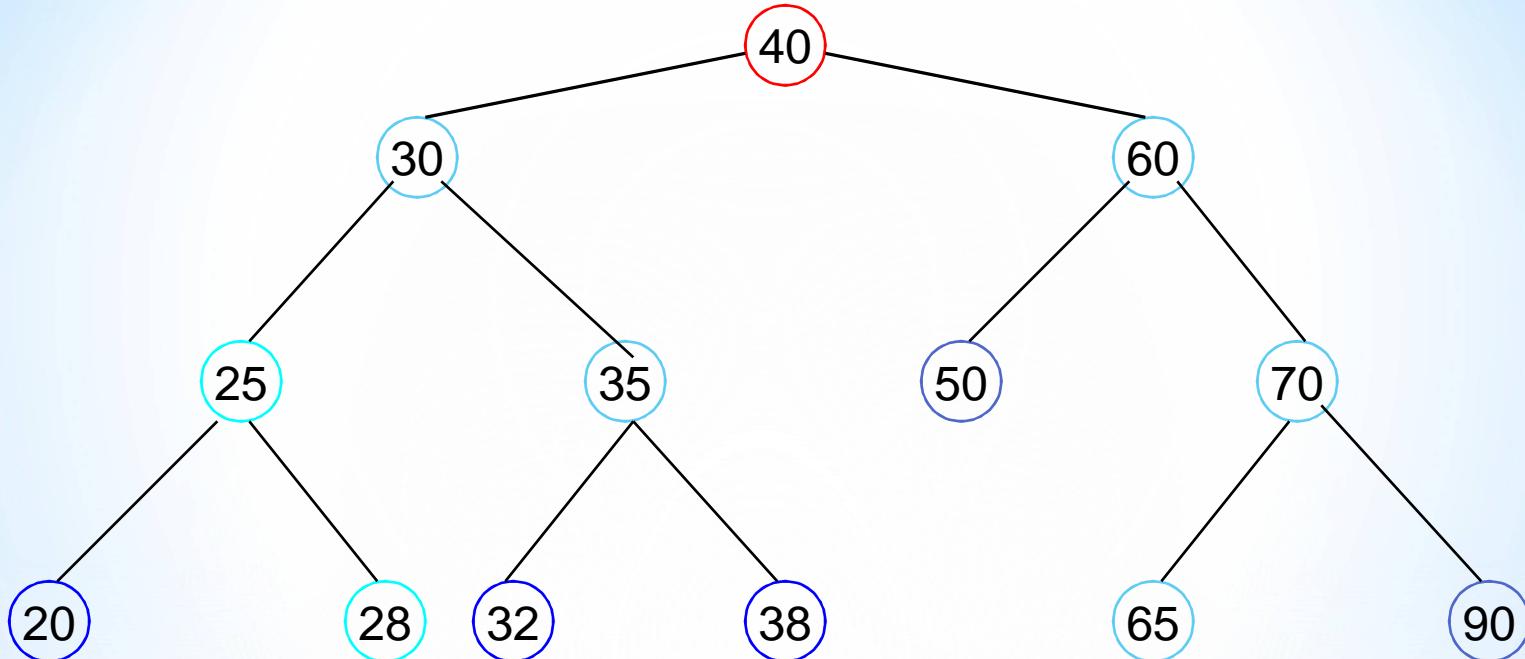
**Bài tập. Hãy cho biết kết quả thực hiện chương trình dưới đây?**

```
#include <stdio.h>
#include <string.h>
#include <conio.h>

void reverse(char str[], int index, int size){
    char temp = str[index];
    str[index] = str[size];
    str[index] = temp;
    if (index>= size)  return;
    reverse(str, index + 1, size-1);
}

int main(){
    char X[] ="ABCDEF";int size = strlen(X);
    reverse(X, 0, size - 1);
    printf("Ket qua is: %s\n", X);
    getch(); return 0;
}
```

**Kích cỡ cây nhị phân (size of a tree).** Ta định nghĩa kích cỡ của một cây là số node có thực trên cây. Bài toán đặt ra là cho trước một cây hãy tìm kích cỡ của cây. Ví dụ: cây dưới đây có kích cỡ là 13



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

**Xác định hai cây nhị phân giống nhau.** Ta định nghĩa hai cây nhị phân giống nhau nếu chúng có cùng chung node và mỗi node được sắp đặt giống nhau trên cây. Cho hai cây nhị phân bất kỳ, hãy xác định xem hai cây có giống nhau hay không?

**Lời giải.** Thuật toán giải quyết bài toán được thực hiện như sau:

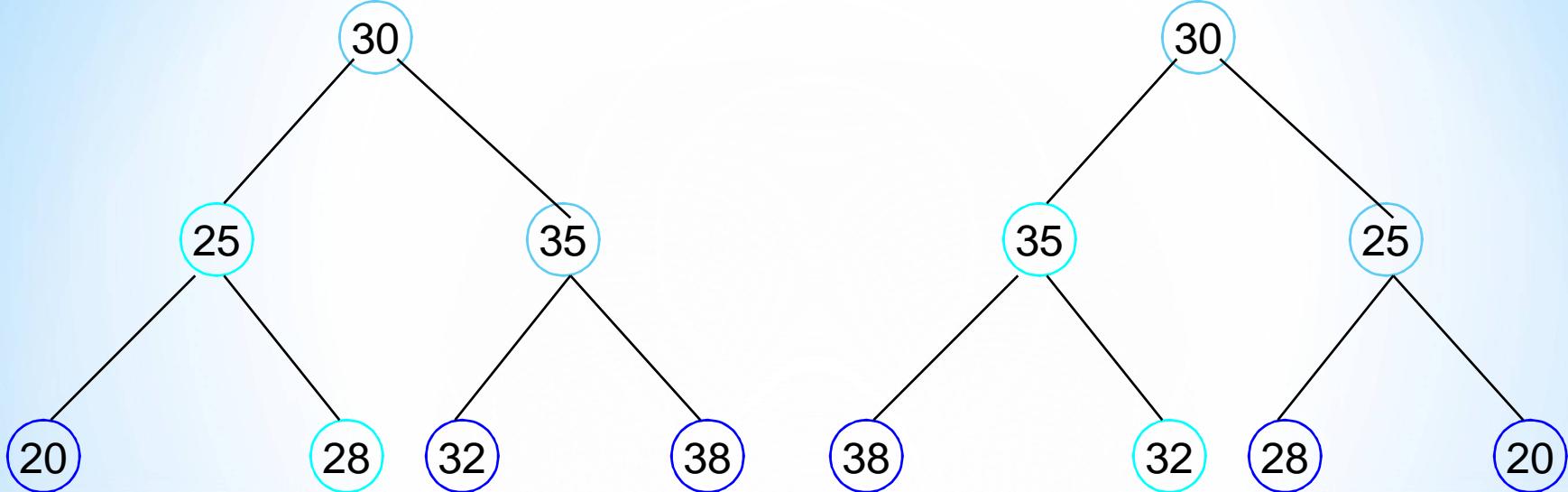
```
in identicalTrees(struct node* T1, struct node* T2) {  
    if (T1==NULL && T2==NULL) //nếu cả hai cây T1 và T2 đều rỗng  
        return 1; //rõ ràng chúng giống nhau  
    if (T1!=NULL && T2!=NULL){ //nếu cả hai cây khác rỗng  
        return (T1->data == T2->data &&  
                identicalTrees(T1->left,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

**Tìm độ cao của cây.** Độ cao của cây được định nghĩa là đường đi dài nhất từ node gốc đến node lá. Cho một cây nhị phân bất kỳ, hãy xác định độ cao của cây?

**Lời giải.** Thuật toán giải quyết bài toán được thực hiện như sau:

```
int maxDepth(struct 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  
        if (lDepth > rDepth)  
            return(lDepth);  
        else  
            return(rDepth);  
    }  
}
```

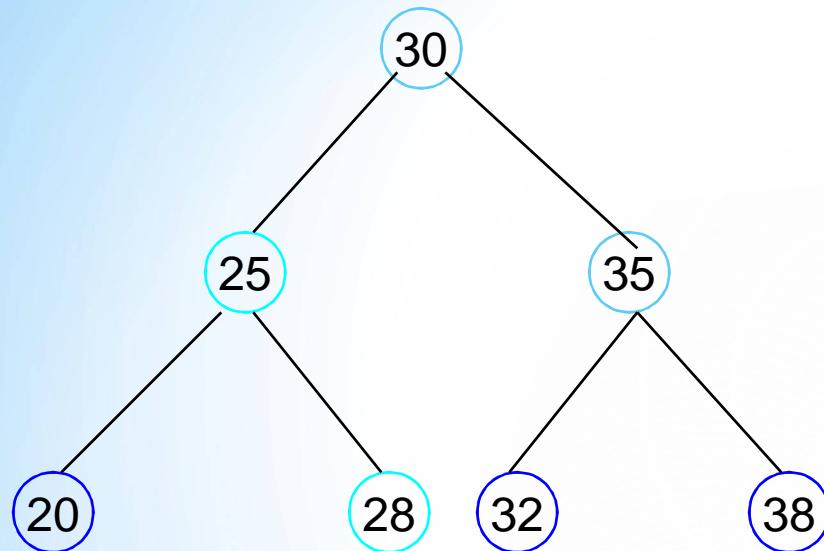
**Cây phản chiếu (Mirror Tree).** Cây phản chiếu của cây nhị phân T là cây nhị phân M(T), trong đó cây con bên trái của cây T trở thành cây con bên phải của M(T) và cây con bên phải của T trở thành cây con trái của M(T). Bài toán được đặt ra | hãy dịch chuyển cây nhị phân T cho trước thành cây nhị phân phản chiếu của T là cây M(T).



```
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  
    }  
}
```

Đếm tất cả các node lá trên cây:

Input:

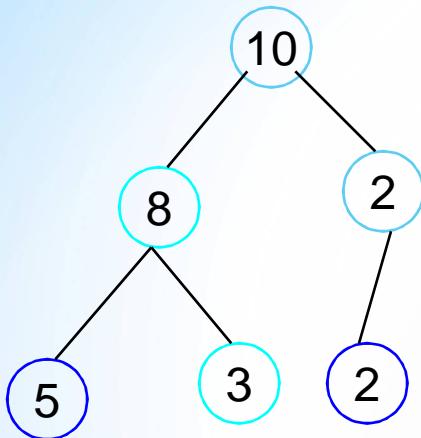


Output:

4

```
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);
}
```

Kiểm tra một cây thỏa mãn điều kiện node trung gian bằng tổng hai node con trái và phải hay không? Node không có node lá trái hoặc phải được xem có giá trị 0.



```
int isSumProperty(struct node* node) {
    int left_data = 0, right_data = 0;
    if(node == NULL || (node->left == NULL && node->right == NULL)) return 1;
    else {
        if(node->left != NULL)
            left_data = node->left->data;
        if(node->right != NULL)
            right_data = node->right->data;
        if((node->data == left_data + right_data)&& isSumProperty(node->left) &&
           isSumProperty(node->right))
            return 1;
        else return 0;
    }
}
```

## 2.4. Thuật toán quay lui (Back track)

Giả sử ta cần xác định bộ  $X = (x_1, x_2, \dots, x_n)$  thỏa mãn một số ràng buộc nào đó. Ứng với mỗi thành phần  $x_i$  ta có  $n_i$  khả năng cần lựa chọn. Ứng với mỗi khả năng  $j \in n_i$ , dành cho thành phần  $x_i$  ta cần thực hiện:

- *Kiểm tra xem khả năng j có được chấp thuận cho thành phần  $x_i$  hay không? Nếu khả năng j được chấp thuận thì nếu i là thành phần cuối cùng ( $i=n$ ) ta ghi nhận nghiệm của bài toán. Nếu i chưa phải cuối cùng ta xác định thành phần thứ  $i+1$ .*
- *Nếu không có khả năng j nào được chấp thuận cho thành phần  $x_i$  thì ta quay lại bước trước đó ( $i-1$ ) để thử lại các khả năng khác.*

*Thuật toán Back-Track ( int i ) {*

```
For ( j =<Khả năng 1>; j <=n; j++ ){  
    if (<chấp thuận khả năng j>) {  
        X[i] = <khả năng j>;  
        if ( i ==n) Result();  
        else Back-Track(i+1);  
    }  
}
```

**Ví dụ**. Bài toán N quân hậu. Trên bàn cờ kích cỡ  $N \times N$ , hãy đặt N quân hậu mỗi quân trên 1 hàng sao cho tất cả các quân hậu đều không ăn được lẫn nhau.

**Lời giải.** Gọi  $X = (x_1, x_2, \dots, x_n)$  là một nghiệm của bài toán. Khi đó,  $x_i = j$  được hiểu là quân hậu hàng thứ i đặt ở cột j. Để các quân hậu khác không thể ăn được, quân hậu thứ i cần không được lặp trùng với bất kỳ cột nào, không được cùng đường chéo xuôi, không được cùng trên đường chéo ngược. Ta có n cột  $A = (a_1, \dots, a_n)$ , có  $Xuoi[2^*n-1]$  đường chéo xuôi,  $Nguoc[2^*n-1]$  đường chéo ngược.

Nếu  $x_i = j$  thì  $A[j] = \text{True}$ ,  $Xuoi[i-j+n] = \text{True}$ ,  $Nguoc[i + j - 1] = \text{True}$ .

**Đường chéo xuôi: Xuoi [i – j + n]**

|    |    |    |    |    |    |   |   |   |
|----|----|----|----|----|----|---|---|---|
|    |    |    |    |    |    |   |   | 1 |
|    |    |    |    |    |    |   |   | 2 |
|    |    |    |    |    |    |   |   | 3 |
|    |    |    |    |    |    |   |   | 4 |
|    |    |    |    |    |    |   |   | 5 |
|    |    |    |    |    |    |   |   | 6 |
|    |    |    |    |    |    |   |   | 7 |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |   |

**Đường chéo ngược: Nguoc [i + j - 1]**

|   |   |    |    |    |    |    |    |  |
|---|---|----|----|----|----|----|----|--|
| 1 |   |    |    |    |    |    |    |  |
| 2 |   |    |    |    |    |    |    |  |
| 3 |   |    |    |    |    |    |    |  |
| 4 |   |    |    |    |    |    |    |  |
| 5 |   |    |    |    |    |    |    |  |
| 6 |   |    |    |    |    |    |    |  |
| 7 |   |    |    |    |    |    |    |  |
| 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |  |

## Thuật toán:

```
void Try (int i){  
    for(int j=1; j<=n; j++){  
        if( Cot[j] && DCXuoi[ i - j + n ] && DCNguoc[i + j -1]) {  
            X[i] =j; Cot[j]=FALSE;  
            DCXuoi[ i - j + n]=FALSE;  
            DCNguoc[ i + j - 1]=FALSE;  
            if(i==n) Result();  
            else Try(i+1);  
            Cot[j] = TRUE;  
            DCXuoi[ i - j + n] = TRUE;  
            DCNguoc[ i + j - 1] = TRUE;  
        }  
    }  
}
```

**2. Sudoku.** Cho một hình vuông gồm lưới lưới cấp  $9\times 9$  hình vuông đơn vị trong đó một số hình vuông đã điền các con số từ 1 đến 9. Hãy điền các số từ 1 đến 9 vào các hình vuông còn lại sao cho mỗi hàng, mỗi cột và mỗi lưới con kích cỡ  $3\times 3$  chứa đựng chỉ một trong các số từ 1 đến 9.

Ví dụ với lưới dưới đây sẽ cho ta kết quả sau:

|   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|
| 3 |   | 6 | 5 |   | 8 | 4 |   |   |
| 5 | 2 |   |   |   |   |   |   |   |
|   | 8 | 7 |   |   |   | 3 | 1 |   |
|   |   | 3 | 1 |   |   | 8 |   |   |
| 9 |   |   | 8 | 6 | 3 |   |   | 5 |
|   | 5 |   |   | 9 |   | 6 |   |   |
| 1 | 3 |   |   |   |   | 2 | 5 |   |
|   |   |   |   |   |   | 7 | 4 |   |
|   |   | 5 | 2 |   | 6 | 3 |   |   |

|   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|
| 3 | 1 | 6 | 5 | 7 | 8 | 4 | 9 | 2 |
| 5 | 2 | 9 | 1 | 3 | 4 | 7 | 6 | 8 |
| 4 | 8 | 7 | 6 | 2 | 9 | 5 | 3 | 1 |
| 2 | 6 | 3 | 4 | 1 | 5 | 9 | 8 | 7 |
| 9 | 7 | 4 | 8 | 6 | 3 | 1 | 2 | 5 |
| 8 | 5 | 1 | 7 | 9 | 2 | 6 | 4 | 3 |
| 1 | 3 | 8 | 9 | 4 | 7 | 2 | 5 | 6 |
| 6 | 9 | 2 | 3 | 5 | 1 | 8 | 7 | 4 |
| 7 | 4 | 5 | 2 | 8 | 6 | 3 | 1 | 9 |

## Thuật toán:

```
bool Sudoku( int grid[N][N]){ // N = 9
    int row, col; //hàng, cột cần đặt
    if (!FindUnassignedLocation(grid, row, col)) //nếu đã điền được tất cả
        return true; // thành công!
    for (int num = 1; num <= 9; num++){ // duyệt các số từ 1 đến 9
        if (isSafe(grid, row, col, num)){ // nếu đặt vào hàng cột là an toàn
            grid[row][col] = num; //thử đặt vào hàng cột
            if (Sudoku(grid))//Gọi đệ qui đến khi thành công
                return true;
            grid[row][col] = UNASSIGNED; //hoàn trả lại giá trị
        }
    }
    return false; //trường hợp không thành công
}
```

**Bài tập.** Viết chương trình giải các bài toán dưới đây bằng thuật toán Back-Track.

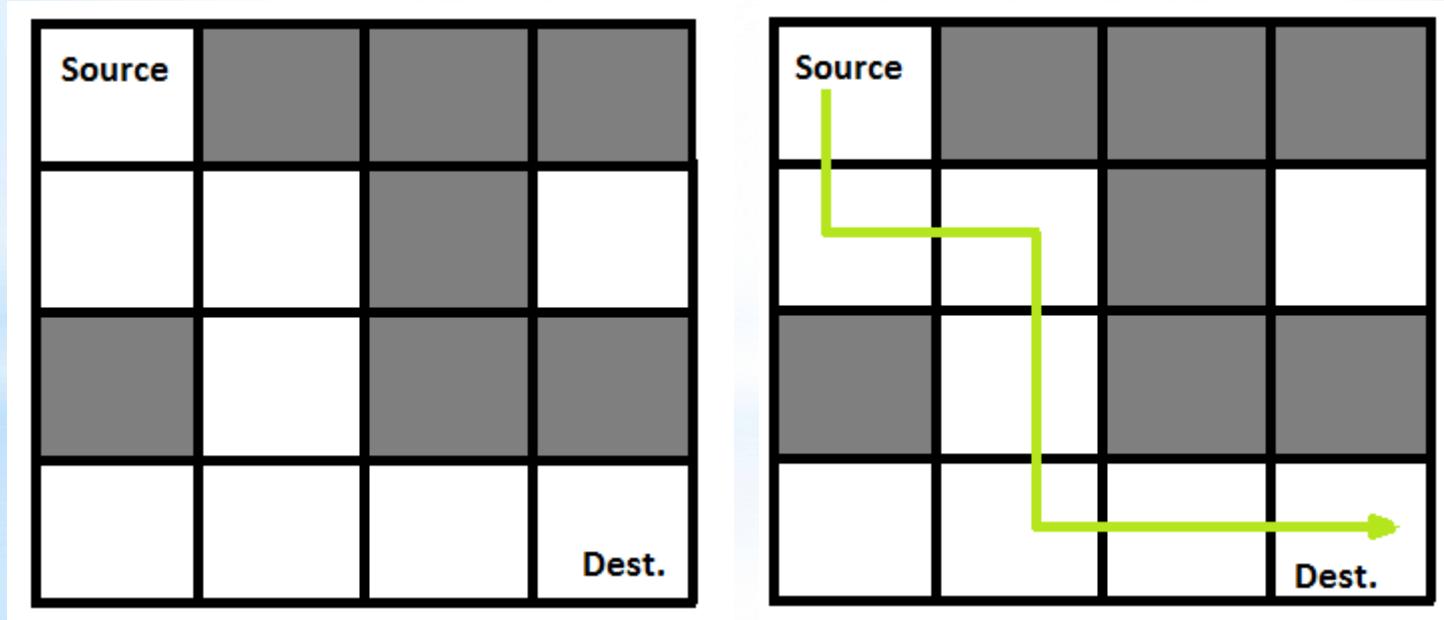
**1. Tug of War.** Cho tập gồm  $n$  số nguyên. Chia tập số nguyên thành hai tập con có kích cỡ  $n/2$  sao cho sự khác biệt của tổng hai tập con là nhỏ nhất có thể được. Nếu  $n$  là số chẵn thì kích cỡ của hai tập hợp là bằng nhau. Nếu  $n$  là số lẻ, thì một tập có kích cỡ là  $(n-1)/2$  và tập còn lại phải là  $(n+1)/2$ .

**Ví dụ.** Tập các số nguyên là  $\{3, 4, 5, -3, 100, 1, 89, 54, 23, 20\}$ ,  $n = 10$ . Kết quả cho tập số nguyên này có thể là  $\{4, 100, 1, 23, 20\}$  và  $\{3, 5, -3, 89, 54\}$ . Các tập con đều có kích cỡ là 5 và tổng các phần tử trong cả hai tập con bằng nhau và là 48.

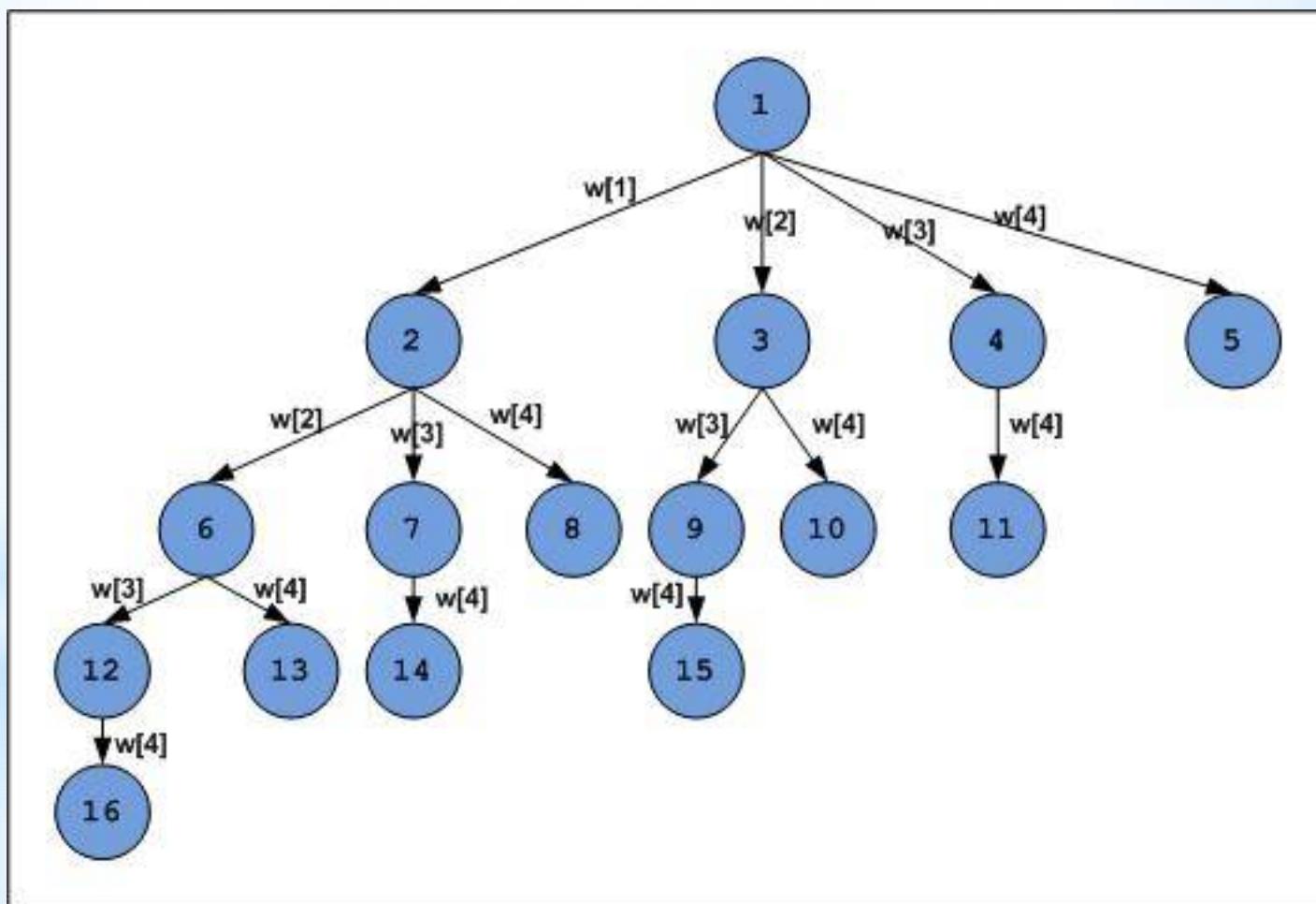
**Ví dụ** với tập các số  $\{23, 45, -34, 12, 0, 98, -99, 4, 189, -1, 4\}$ ,  $n=11$ . Kết quả các tập con tìm được là  $\{45, -34, 12, 98, -1\}$  và  $\{23, 0, -99, 4, 189, 4\}$ . Tổng các phần tử của hai tập con khác biệt ít nhất lần lượt là 120 và 121 theo thứ tự.

**2. Mê cung (Rat in a Maze).** Một mê cung là một hình vuông gồm  $N^*N$ . hình vuông đơn vị. Một số ô vuông đã được đặt một vật cản (ký hiệu là 0). Một con chuột bắt đầu tìm đường đi tại một ô vuông nào đó (ô source) đến một ô vuông khác (ô destination). Biết con chuột chỉ được phép dịch chuyển theo hai hướng lùi lại hoặc đi xuống theo các ô vuông không có vật cản.

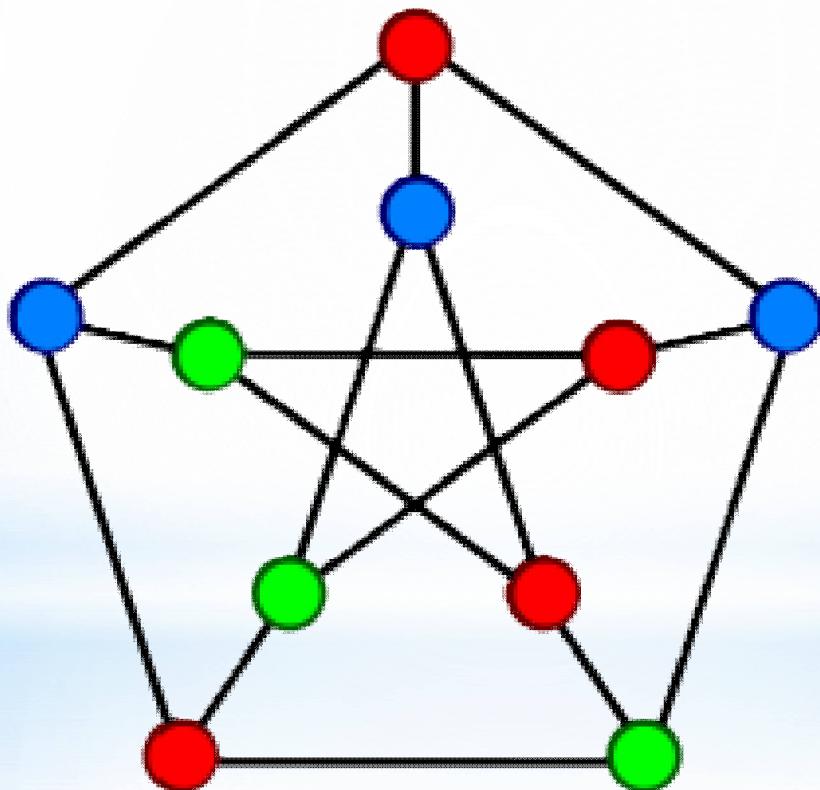
Ví dụ với lưới dưới đây sẽ cho ta kết quả sau:



**3. Dãy con có tổng bằng K (Set Sum).** Cho tập các số gồm n số khác nhau. Hãy tìm tất cả các tập con của tập các số đã cho sao cho tổng các phần tử đúng bằng K. Ví dụ dưới đây đưa ra cây quay lui của bài toán với  $n=4$ .



**4. Bài toán tô màu đồ thị (Graph Coloring).** Cho đồ thị vô hướng  $G = \langle V, E \rangle$  và số tự nhiên  $m$ . Hãy tìm cách tô màu đồ thị với nhiều nhất  $m$  màu sao cho không có hai đỉnh kề nhau nào được tô bởi cùng một màu. Ví dụ dưới đây chỉ ra một cách tô với 3 màu.



## 2.5.Khử đệ qui

**Nguyên tắc lập trình:** Không nên lạm dụng giải thuật đệ qui trong khi xây dựng ứng dụng. Mọi bài toán giải được bằng đệ qui đều có thể dịch chuyển về các cấu trúc lệnh lặp kết hợp với các cấu trúc dữ liệu phù hợp.

**Nguyên tắc khử đệ qui:**

- Dựa trên bộ xếp chồng của chương trình dịch (stack).
- Dựa trên các cấu trúc lệnh lặp biểu diễn lại thuật toán.

**Bài tập 1.** Khử đệ qui giải thuật dưới đây?

```
int convert(int X, int b){  
    if (X == 0) return 0;  
    else return (X % b + 10 * convert(X / b));  
}
```

```
int convert(int X, int b){  
    if (X == 0) return 0;  
    else return (X % b + 10 * convert(X / b));  
}  
  
int Convert(int X, int b){  
    int S=0, i=0;  
    while(X!=0) {  
        int du = X%b; //lấy phần dư  
        int k = (int) pow(10, i);  
        du = du* k;  
        S= S + du;  
        X =X / b; i++;  
    }  
    return(S);  
}
```

## Bài tập 2. Khử đệ qui phép duyệt cây theo thứ tự trước (NLR).

Algorithm NLR ( Tree \*T) {

    if (T!= NULL ) {

*<Thăm node>;*

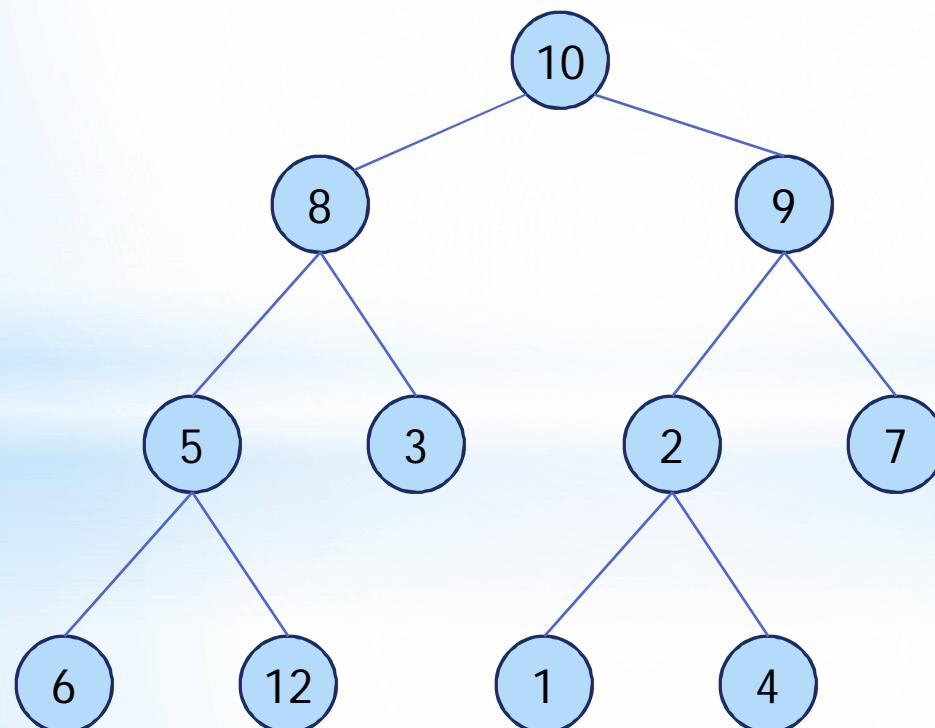
        NLR(T -> left); //duyệt thứ tự trước sang bên trái

        NLR( T -> right); //duyệt thứ tự trước sang bên phải

    }

}

Ví dụ : NLR(T) = 10, 8, 5, 6, 12, 3, 9, 2, 1, 4, 7.



## **Thuật toán NLR dựa vào stack:**

**Thuật toán NLR (Tree \*T) {**

**Bước 1 (Khởi tạo):**

```
if (T == NULL) //nếu cây rỗng ta không phải làm gì  
    return;
```

**Bước 2 ( Lắp):**

```
stack = φ; Push(stack, T); //đưa node gốc vào stack  
while (stack≠φ){ //lắp đến khi stack rỗng  
    node = Pop(stack); //lấy node ra khỏi stack  
    <Thăm node>;  
    if (node->right ≠ φ) //nếu cây bên phải khác rỗng  
        Push(stack, node->right); //đưa node phải vào trước  
    if (node->left) //nếu cây bên trái khác rỗng  
        Push(stack, node->left); //đưa node trái vào sau  
    }  
}
```

## Kiểm nghiệm.

| Bước | Trạng thái stack | Các node đã thăm                  |
|------|------------------|-----------------------------------|
| 1    | 10               | $\emptyset$                       |
| 2    | 9, 8             | 10                                |
| 3    | 9, 3, 5          | 10, 8                             |
| 4    | 9, 3, 12, 6      | 10, 8, 5                          |
| 5    | 9, 3, 12         | 10, 8, 5, 6                       |
| 6    | 9, 3             | 10, 8, 5, 6, 12                   |
| 7    | 9                | 10, 8, 5, 6, 12, 3                |
| 8    | 7, 2             | 10, 8, 5, 6, 12, 3, 9             |
| 9    | 7, 4, 1          | 10, 8, 5, 6, 12, 3, 9, 2          |
| 10   | 7, 4             | 10, 8, 5, 6, 12, 3, 9, 2, 1       |
| 11   | 7                | 10, 8, 5, 6, 12, 3, 9, 2, 1, 4    |
| 12   | $\emptyset$      | 10, 8, 5, 6, 12, 3, 9, 2, 1, 4, 7 |

### Bài tập 3. Khử đệ qui phép duyệt cây theo thứ tự giữa (LNR).

Thuật toán LNR ( Tree \*T) {

    if (T!= $\phi$  ) { //nếu T khác rỗng

        LNR(T -> left); //Duyệt theo thứ tự giữa sang bên trái

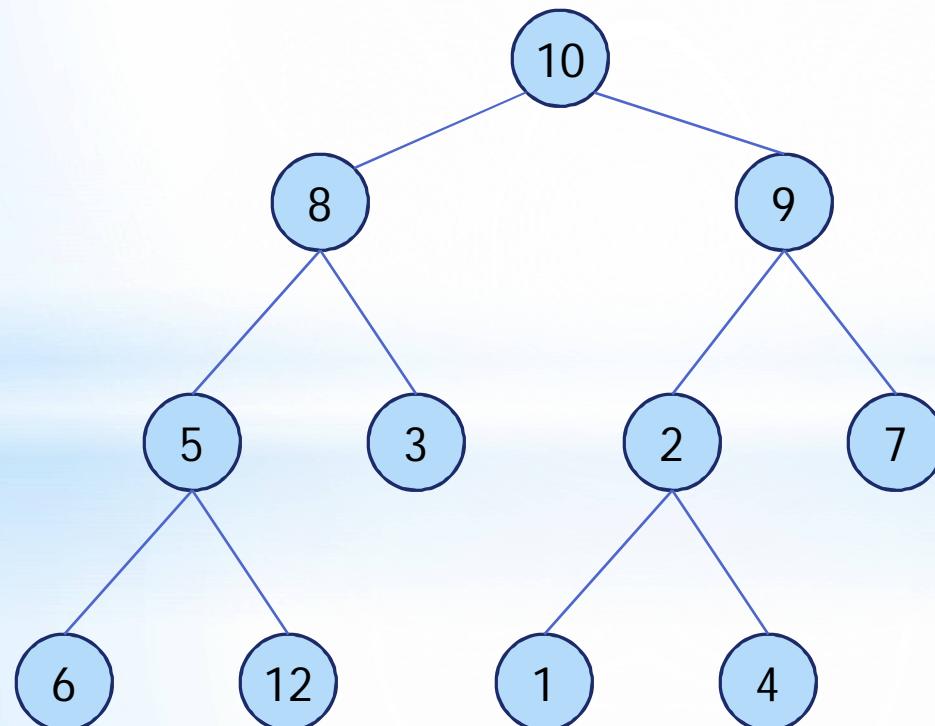
        <Thăm node>;

        LNR( T -> right); //Duyệt thứ tự giữa sang bên phải

    }

}

Ví dụ : LNR(T) =6, 5, 12, 8, 3, 10, 1, 2, 4, 9, 7.



## Thuật toán LNR dựa vào stack:

Thuật toán LNR (Tree \*T) {

**Bước 1** (Khởi tạo): done = false; //trạng thái duyệt ban đầu là false  
Stack =  $\emptyset$ ; // tạo Stack rỗng

Node = T; // Node hiện tại trả đến node gốc

**Bước 2** ( Lặp):

while(! done) { //lặp đến khi done là true

    if (Node  $\neq \emptyset$  ) { //nếu node khác rỗng

        Push(Stack, Node); //đưa node vào stack

        Node = Node -> left; //chuyển Node sang bên trái

    }

    else { //nếu Node rỗng

        if (Stack  $\neq \emptyset$ ) { //nếu Stack khác rỗng

            Node = Pop(Stack); //lấy node ra khỏi stack

            <Thăm Node>;

            Node = Node ->right; //Node trả sang bên phải

        }

        else //nếu Stack rỗng

            done = true; //kết thúc duyệt

    }

}

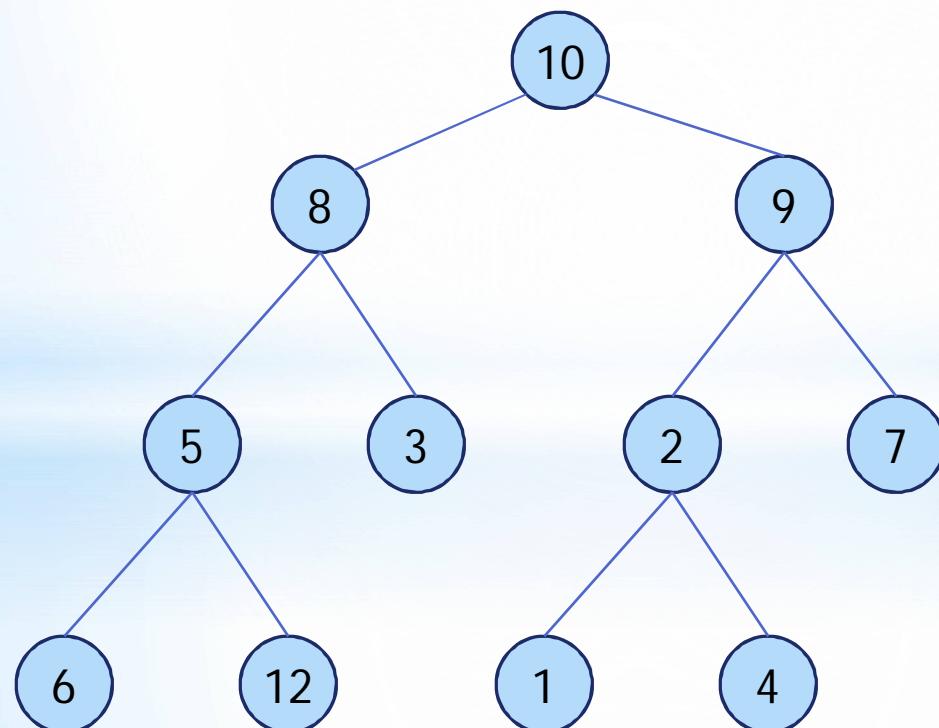
}

| Bước | Trạng thái stack | Các node đã thăm                  |
|------|------------------|-----------------------------------|
| 1    | 10               | $\emptyset$                       |
| 2    | 10, 8            | $\emptyset$                       |
| 3    | 10, 8, 5         | $\emptyset$                       |
| 4    | 10, 8, 5, 6      | $\emptyset$                       |
| 5    | 10, 8, 5         | 6                                 |
| 6    | 10, 8, 12        | 6, 5                              |
| 7    | 10, 8            | 6, 5, 12                          |
| 8    | 10, 3            | 6, 5, 12, 8                       |
| 9    | 10               | 6, 5, 12, 8, 3                    |
| 10   | 9                | 6, 5, 12, 8, 3, 10                |
| 11   | 9, 2             | 6, 5, 12, 8, 3, 10                |
| 12   | 9, 2, 1          | 6, 5, 12, 8, 3, 10                |
| 13   | 9, 2             | 6, 5, 12, 8, 3, 10, 1             |
| 14   | 9, 4             | 6, 5, 12, 8, 3, 10, 1, 2          |
| 15   | 9                | 6, 5, 12, 8, 3, 10, 1, 2, 4       |
| 16   | 7                | 6, 5, 12, 8, 3, 10, 1, 2, 4, 9    |
| 17   | $\emptyset$      | 6, 5, 12, 8, 3, 10, 1, 2, 4, 9, 7 |

## Bài tập 4. Khử đệ qui phép duyệt cây theo thứ tự sau (LRN).

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

Ví dụ : LNR(T) =6, 12, 5, 3, 8, 1, 4, 2, 7, 9, 10.



## Thuật toán LRN dựa vào stack:

Thuật toán LRN (Tree \*T) {

**Bước 1** (Khởi tạo): if (T =  $\emptyset$ ) return; //nếu T là rỗng

Stack =  $\emptyset$ ; Node = T; //tạo stack rỗng và Node trỏ đến gốc T

**Bước 2** ( Lặp):

do { //bắt đầu lặp

    while(Node  $\neq \emptyset$ ) { //lặp đến khi node là rỗng

        if (Node -> right  $\neq \emptyset$ ) //nếu node phải khác rỗng

            Push(Stack, Node->right); //đưa node phải vào stack

        Push(Stack, Node); //đưa node vào stack

        Node = Node ->left; //node trỏ sang bên trái

    }

    Node = Pop(Stack); //đưa node ra khỏi stack;

    if (Node->right && Top(Stack) == Node->right){

        Pop(stack); // loại node này khỏi stack

        Push(Stack, Node); // Đưa node vào stack

        Node = Node->right; // Trỏ sang node phải

    }

    else { <Thăm node>; Node = NULL; }

} while (Stack $\neq \emptyset$ ));

}

| Bước | Trạng thái stack      | Các node đã thăm                  |
|------|-----------------------|-----------------------------------|
| 1    | 9, 10, 3, 8, 12, 5, 6 | ∅                                 |
| 2    | 9, 10, 3, 8, 12, 5    | 6                                 |
| 3    | 9, 10, 3, 8, 12, 5    | 6                                 |
| 4    | 9, 10, 3, 8, 5, 12    | 6                                 |
| 5    | 9, 10, 3, 8, 5        | 6, 12                             |
| 6    | 9, 10, 3, 8           | 6, 12, 5                          |
| 7    | 9, 10, 8, 3           | 6, 12, 5                          |
| 8    | 9, 10, 8              | 6, 12, 5, 3                       |
| 9    | 9, 10                 | 6, 12, 5, 3, 8                    |
| 10   | 10, 9                 | 6, 12, 5, 3, 8                    |
| 11   | 10, 9, 4, 2, 1        | 6, 12, 5, 3, 8                    |
| 12   | 10, 9, 4, 2           | 6, 12, 5, 3, 8, 1                 |
| 13   | 10, 9, 2, 4           | 6, 12, 5, 3, 8, 1                 |
| 14   | 10, 9, 2              | 6, 12, 5, 3, 8, 1, 4              |
| 15   | 10, 9                 | 6, 12, 5, 3, 8, 1, 4, 2           |
| 16   | 10, 9, 7              | 6, 12, 5, 3, 8, 1, 4, 2           |
| 17   | 10, 9                 | 6, 12, 5, 3, 8, 1, 4, 2, 7        |
| 18   | 10                    | 6, 12, 5, 3, 8, 1, 4, 2, 7, 9     |
| 19   | ∅                     | 6, 12, 5, 3, 8, 1, 4, 2, 7, 9, 10 |