Chương 4: CẤU TRÚC CÂY (TREES)

Data structures and Algorithms

Cây nhị phân

- Định nghĩa
- Biểu diễn máy của cây nhị phân
  - Lưu trữ tuần tự (kế tiếp)
  - Lưu trữ móc nối
- Duyệt cây nhị phân
- Áp dụng cấu trúc cây
  - Sắp xếp
  - Tìm kiếm

Định nghĩa

- Cây là một cấu trúc phi tuyến, thiết lập trên một tập hữu hạn các phần tử được gọi là “nút”
  - Nút đặc biệt gọi là gốc (root)
  - Liên kết phân cấp với các nút con gọi là quan hệ cha-con
  - Cây không có nút nào gọi là cây rỗng

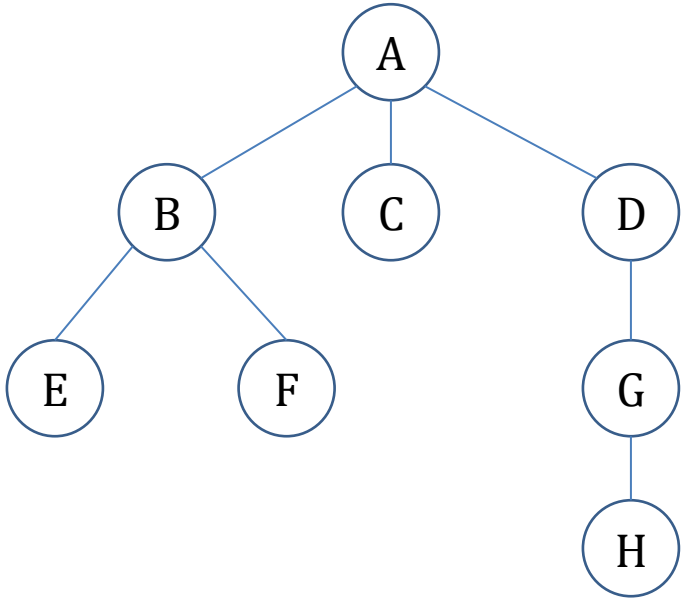
Định nghĩa

- Một nút là một cây, nút đó cũng là gốc của cây
- Nếu **$T_1, T_2, \ldots, T_k$** là các cây với $n_1, n_2,\ldots,n_k$ lần lượt là các gốc, n là một nút và có quan hệ cha con với $n_1, n_2,\ldots,n_k$
- Cây T được tạo lập khi n được gọi là cha của $n_1, n_2,\ldots,n_k$, các cây **$T_1, T_2, \ldots, T_k$** gọi là cây con của n

Định nghĩa

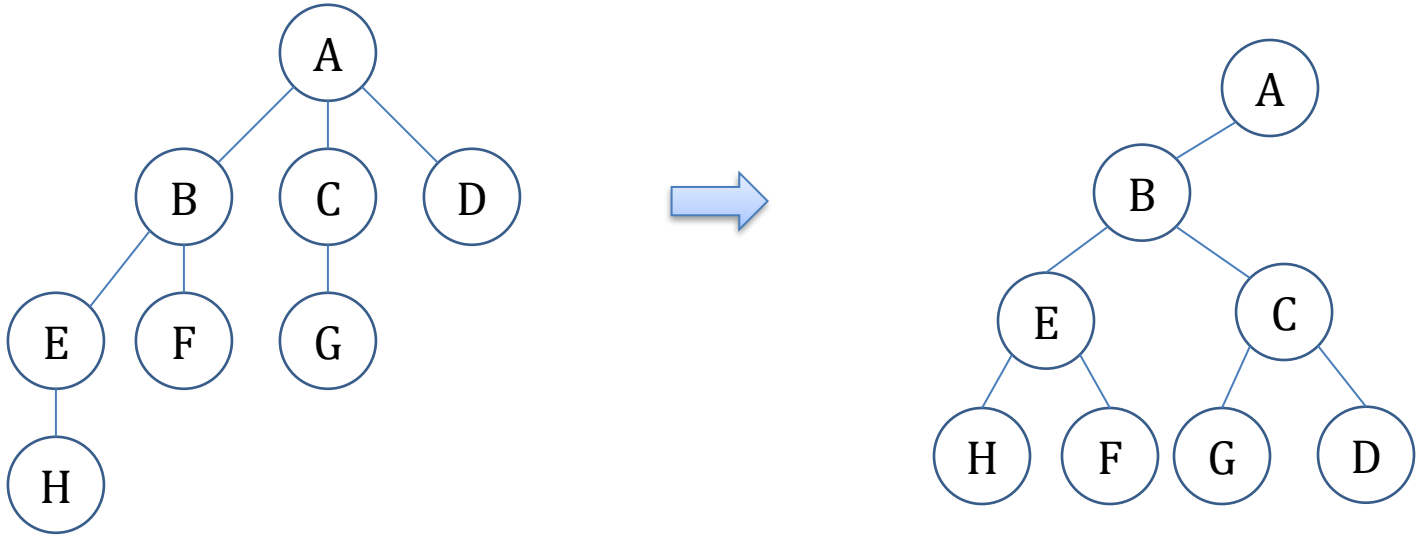
- Số các con của một nút được gọi là cấp
- Nút có cấp bằng 0 được gọi là lá hay nút tận cùng
- Nút không phải là lá gọi là nút nhánh (cành)
- Cấp cao nhất của nút trên cây gọi là cấp của cây đó
- Độ dài của đường đi: bằng số nút trên đường đi đó trừ 1

Gốc	A
Cành	B, D, G
Lá	C, E, F, H
Cấp	3
Chiều cao	4
Mức 1	A
Mức 2	B, C, D
Mức 3	E, F, G
Mức 4	H



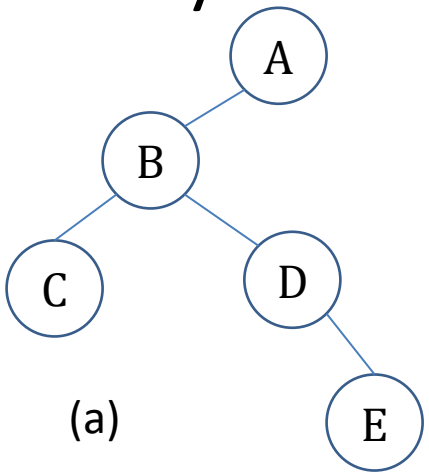
Cây nhị phân

- Cây có thứ tự và là cây cấp hai, tức là mỗi nút có tối đa 2 con.
- Hai con của một nút được phân biệt thứ tự
  - Nút trước gọi là nút con trái
  - Nút sau được gọi là nút con phải
- Khi biểu diễn cây nhị phân, ta cũng cần có sự phân biệt rõ ràng giữa con trái và con phải, nhất là khi nút chỉ có một con



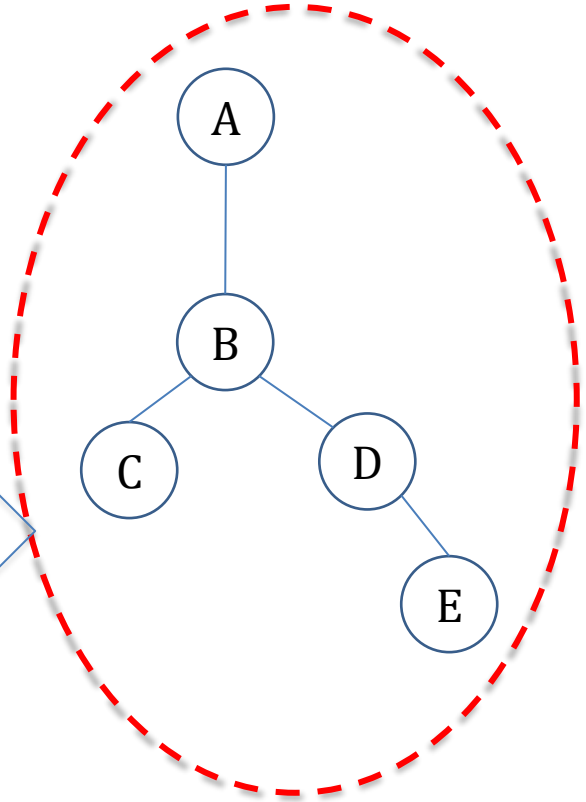
Cây nhị phân

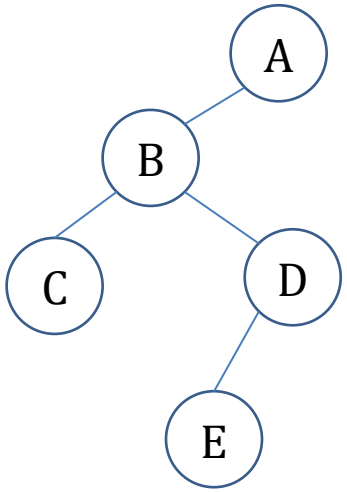
- Là cây có thứ tự



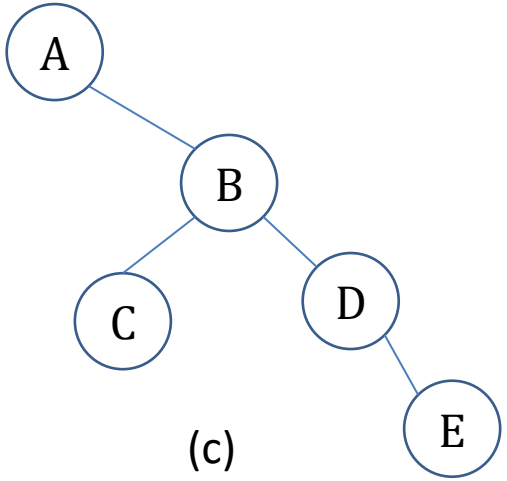
(a)

Cây không thứ tự thì (a, b, c, d) là một cây duy nhất

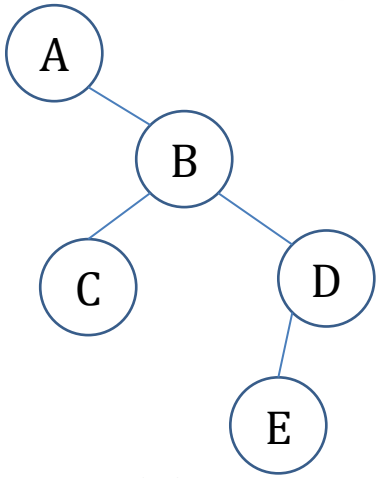




(b)



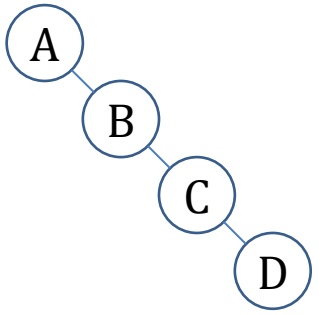
(c)



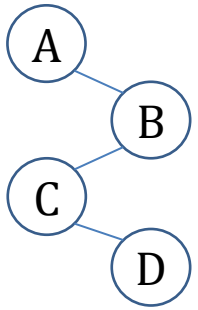
(d)

Cây nhị phân

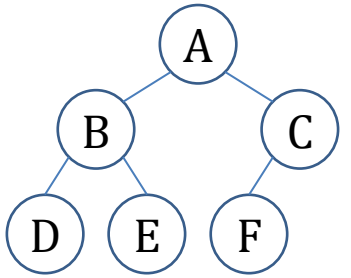
- Cây suy biến: có các nút nhánh chỉ có 1 nút con
- Cây gần đầy, cây đầy đủ, cây hoàn chỉnh.



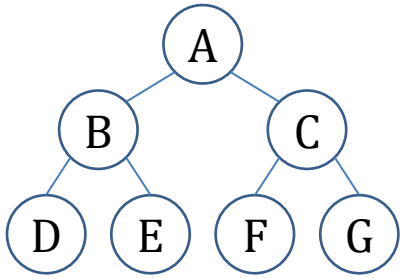
Cây lệch phải



Cây zíc-zắc



Cây hoàn chỉnh



Cây đầy đủ

Tính chất cây nhị phân

- Số lượng tối đa của các nút ở mức i trên cây nhị phân là 2^{i-1} (i>=1)
- Số lượng tối đa các nút trên một cây nhị phân có chiều cao h là $2^h - 1$ (h>=1)
- Nếu cây nhị phân đầy đủ có chiều cao h và số nút là n thì: $h = \log_2(n+1)$

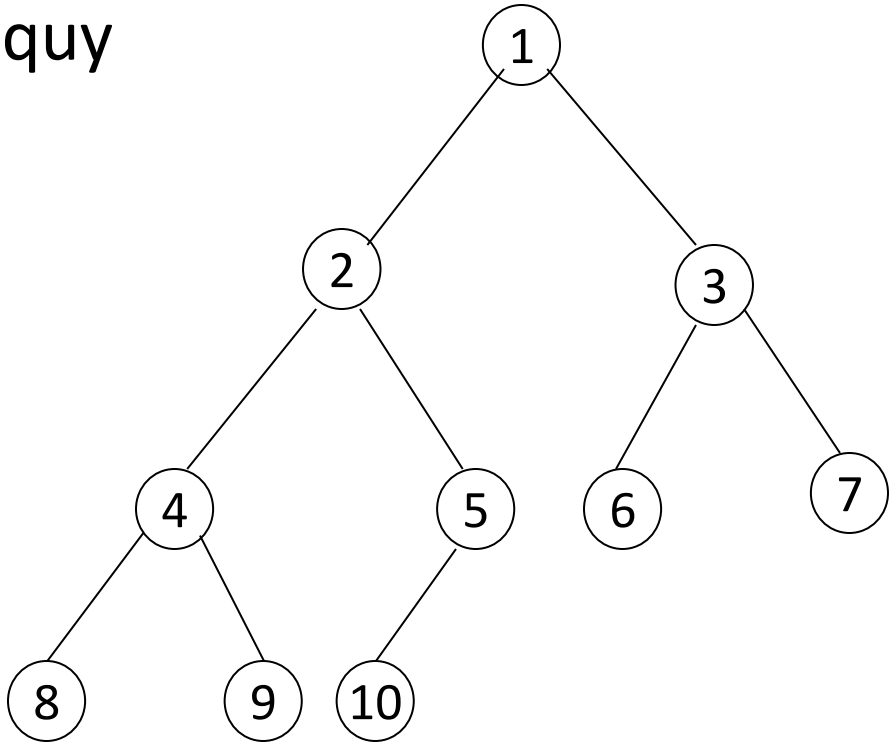
Phép duyệt cây nhị phân

- Phép thăm một nút (visit): là thao tác truy nhập vào một nút của cây để xử lý nút đó.
- Phép duyệt (traversal): là phép thăm một cách hệ thống tất cả các nút của cây, mỗi nút đúng một lần.
- Có 3 phép duyệt cây thông dụng:
  - Duyệt theo thứ tự trước (preorder traversal)
  - Duyệt theo thứ tự giữa (inorder traversal)
  - Duyệt theo thứ tự sau (postorder traversal)

Duyệt theo thứ tự trước (preorder traversal)

- Duyệt cây theo chiều sâu: đây là giải thuật đệ quy
- Nếu cây rỗng thì không làm gì
- Nếu cây không rỗng: trường hợp đệ quy
  - Thăm gốc
  - Duyệt cây con trái theo thứ tự trước
  - Duyệt cây con phải theo thứ tự trước

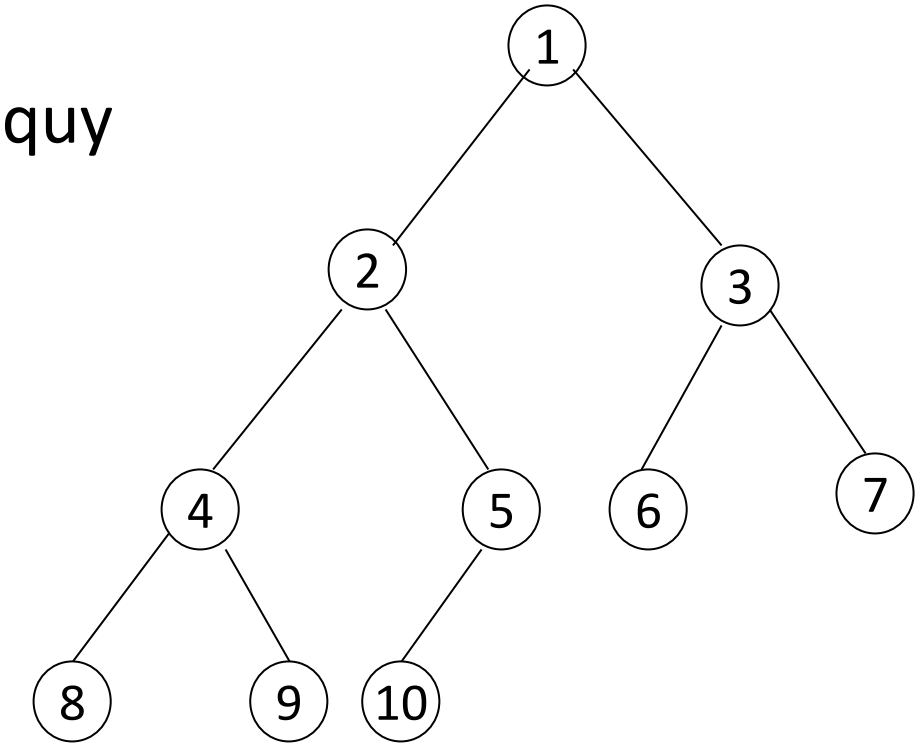
VD: 1, 2, 4, 8, 9, 5, 10, 3, 6, 7



Duyệt theo thứ tự giữa (inorder travelsal)

- Đây là giải thuật đệ quy
- Nếu cây rỗng thì không làm gì
- Nếu cây không rỗng: trường hợp đệ quy
  - Duyệt cây con trái theo thứ tự giữa
  - Thăm gốc
  - Duyệt cây con phải theo thứ tự giữa

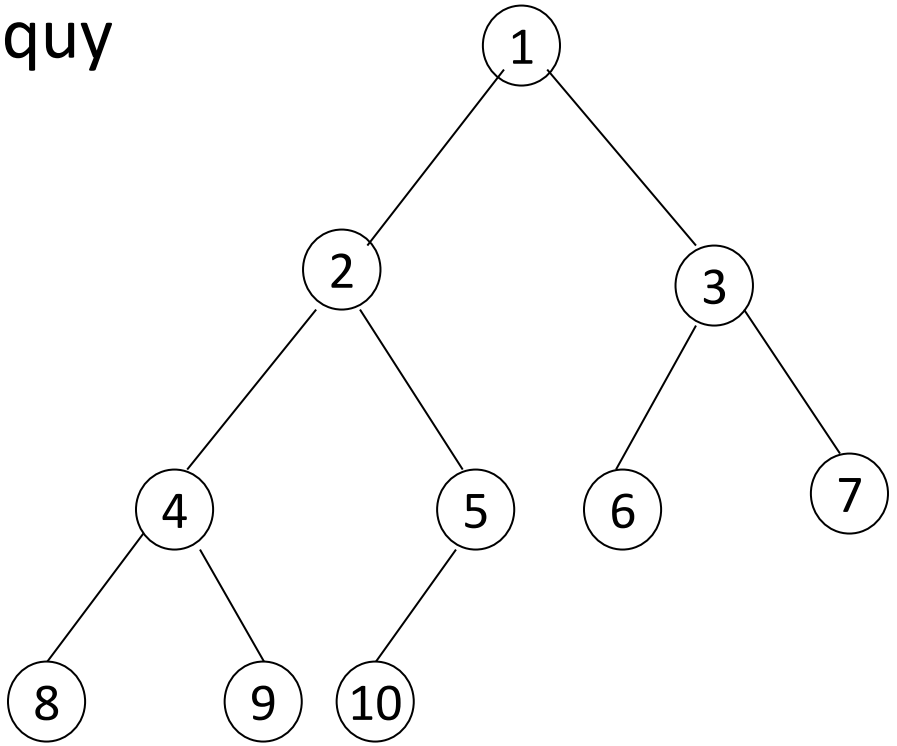
VD: 8, 4, 9, 2, 10, 5, 1, 6, 3, 7



Giải thuật duyệt theo thứ tự sau (postorder travelsal):

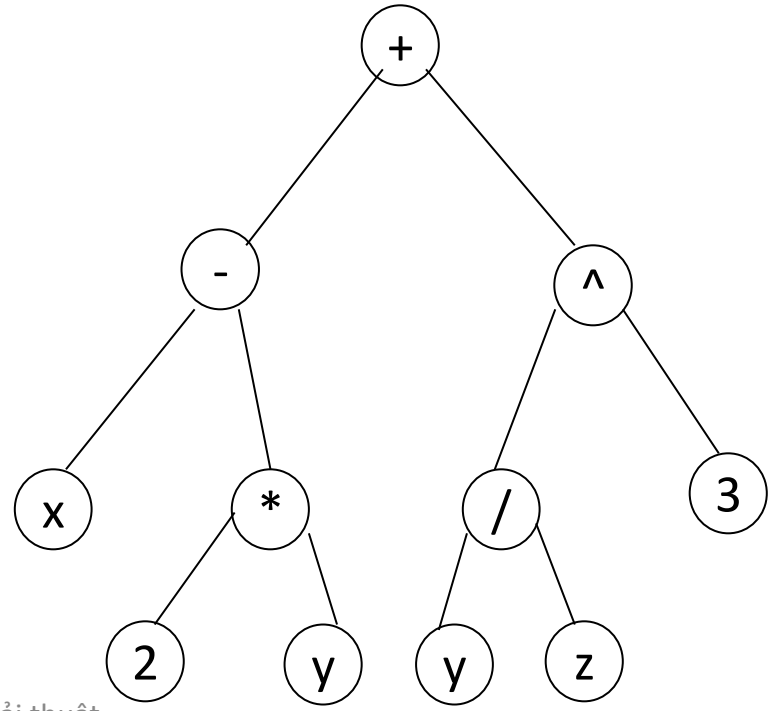
- Đây là giải thuật đệ quy
- Nếu cây rỗng thì không làm gì
- Nếu cây không rỗng: trường hợp đệ quy
  - Duyệt cây con trái theo thứ tự sau
  - Duyệt cây con phải theo thứ tự sau
  - Thăm gốc

VD: 8, 9, 4, 10, 5, 2, 6, 7, 3, 1



Ví dụ

- Biểu thức toán học biểu diễn như sau: (x-2*y) + (y/z)^3
  - Duyệt theo thứ tự trước (tiền tố): +-x*2y^/yz3
  - Duyệt theo thứ tự sau (hậu tố): x2y*-yz/3^+
  - Duyệt theo thứ tự giữa (trung tố): (x-2*y) + (y/z)^3

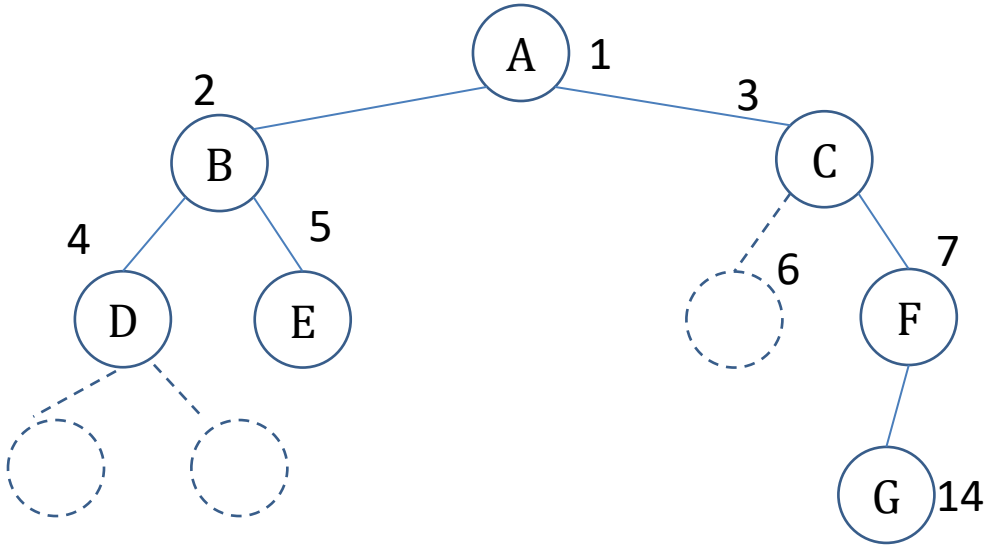


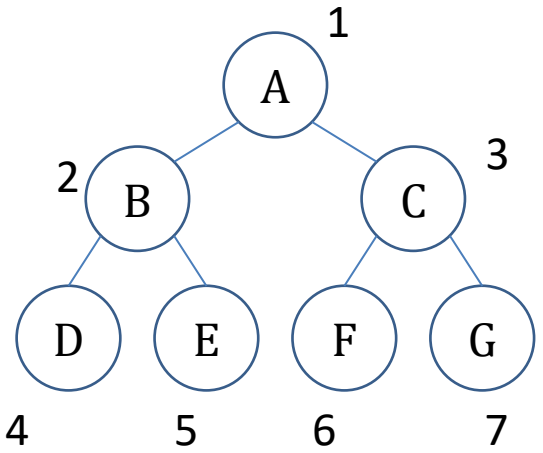
Cài đặt cây nhị phân

- Lưu trữ tuần tự (kế tiếp): Dùng vector lưu trữ V, nút thứ i trên cây thì lưu trữ ở V[i]
- Lưu trữ móc nối: Nút trên cây là một phần tử có quy cách.

Lưu trữ tuần tự (kế tiếp)

- Đánh số liên tiếp các nút trên cây theo thứ tự từ mức 1, hết mức này đến mức khác từ trái sang phải đối với các nút trên cùng một mức





A	B	C	D	E	Φ	F	Φ			....	Φ	G

Phần tử rỗng

Lưu trữ móc nối

- Mỗi nút trên cây được lưu trữ bởi phần tử có quy cách sau:
  - INFO: Chứa thông tin của nút
  - RP: Chứa địa chỉ nút gốc cây con phải (trỏ tới cây con phải)
  - LP: Chứa địa chỉ nút gốc cây con trái (trỏ tới cây con trái)

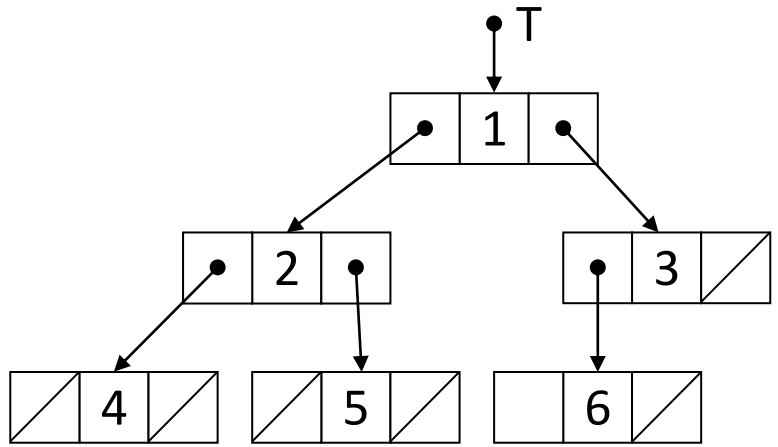

```
struct Node {
    type Info;
    Node * LP, *RP;
};

typedef Node* PNode;
```

Lưu trữ móc nối

- Nếu không có con thì giá trị con trỏ tương ứng sẽ rỗng (NIL/NULL)
- Tổ chức cấu trúc cây như sau (xem hình):
  - Ít nhất một con trỏ T trỏ vào nút gốc, vừa đại diện cho cây, vừa là điểm truy nhập vào các nút khác trong cây.
  - Các thao tác muốn truy nhập vào các nút trong cây phải thông qua con trỏ này.

Khai báo cấu trúc cây

```
Cách 1:     typedef  PNode  BinaryTree;
Cách 2:     struct  BinaryTree {
            PNode  T;  //con trỏ trỏ vào nút gốc
            int  n;        // kích thước cây
        }
```



Lưu trữ móc nối

- Thao tác khởi tạo: tạo một cây rỗng

```
void InitBT (BinaryTree & T) {
    T = NULL;
}
```

Lưu trữ móc nối

- **Thao tác duyệt cây theo thứ tự trước**

```
{Duyệt cây theo thứ tự trước}
void  PreOrderTraversal (BinaryTree  T) {
      if  (T==NULL) return;
      Visit(T);
      PreOrderTraversal(T->LP);
      PreOrderTraversal(T->RP);
}
```

- **Thao tác duyệt cây theo thứ tự giữa**

```
{Duyệt cây theo thứ tự giữa}
void  InOrderTraversal (BinaryTree  T) {
      if  (T==NULL) return;
      InOrderTraversal(T->LP);
      Visit(T);
      InOrderTraversal(T->RP);
}
```

- **Thao tác duyệt cây theo thứ tự sau**

```
{Duyệt cây theo thứ tự sau}
void   PostOrderTraversal (BinaryTree  T) {
      if  (T==NULL) return;
      PostOrderTraversal(T->LP);
      PostOrderTraversal(T->RP);
      Visit(T);
}
```

Cây nhị phân tìm kiếm

- Giới thiệu
  - Mục đích: ứng dụng vào lưu trữ và tìm kiếm thông tin một các hiệu quả nhất
  - Nguyên tắc: lưu trữ và tìm kiếm thông qua **"khoá"**
    - Nhóm thuộc tính khoá: bao gồm các thuộc tính, tính chất giúp chúng ta hoàn toàn xác định được đối tượng

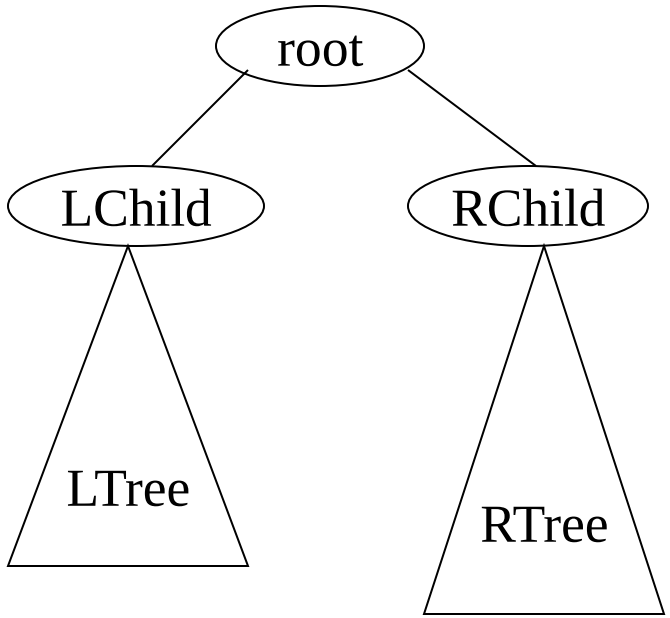
Cây nhị phân tìm kiếm

Là cây nhị phân thỏa mãn các điều kiện sau

- Khoá của các đỉnh thuộc cây con trái nhỏ hơn khoá của gốc
- Khóa của gốc nhỏ hơn khoá của các đỉnh thuộc cây con phải
- Cây con trái và cây con phải cũng là cây nhị phân tìm kiếm

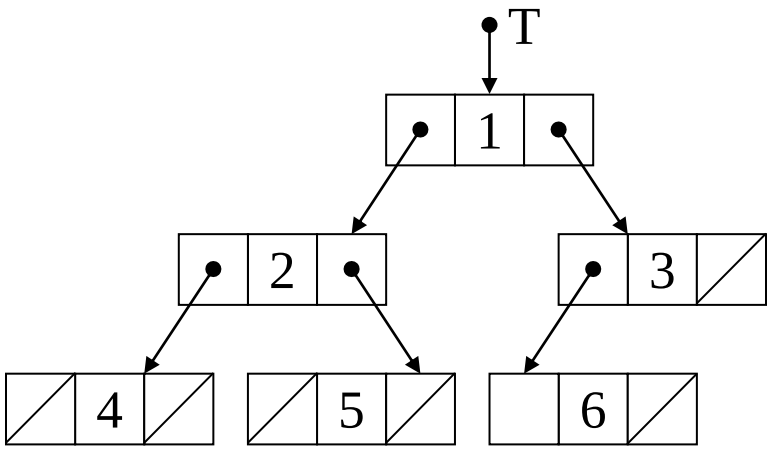
BSearchTree = BinaryTree; and
Key(LChild) < **Key**(root)**;** and
Key(root) < **Key**(RChild)**;**

AND
LTree = **BSearchTree;** and
RTree = **BSearchTree;**

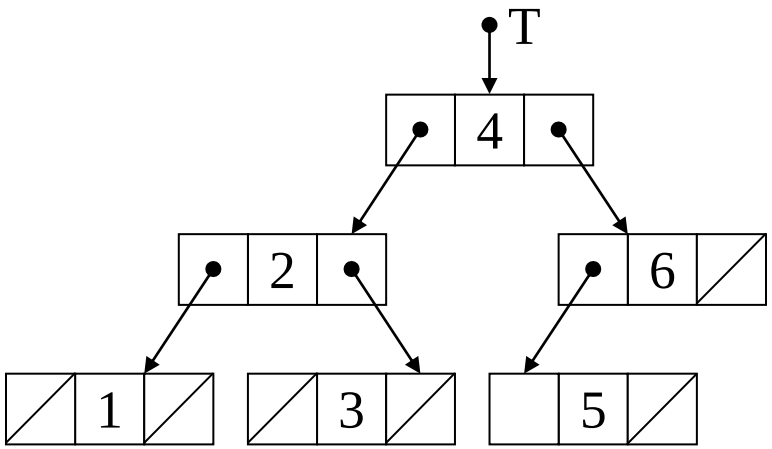


Cây nhị phân tìm kiếm

Ví dụ 1:



Cây nhị phân

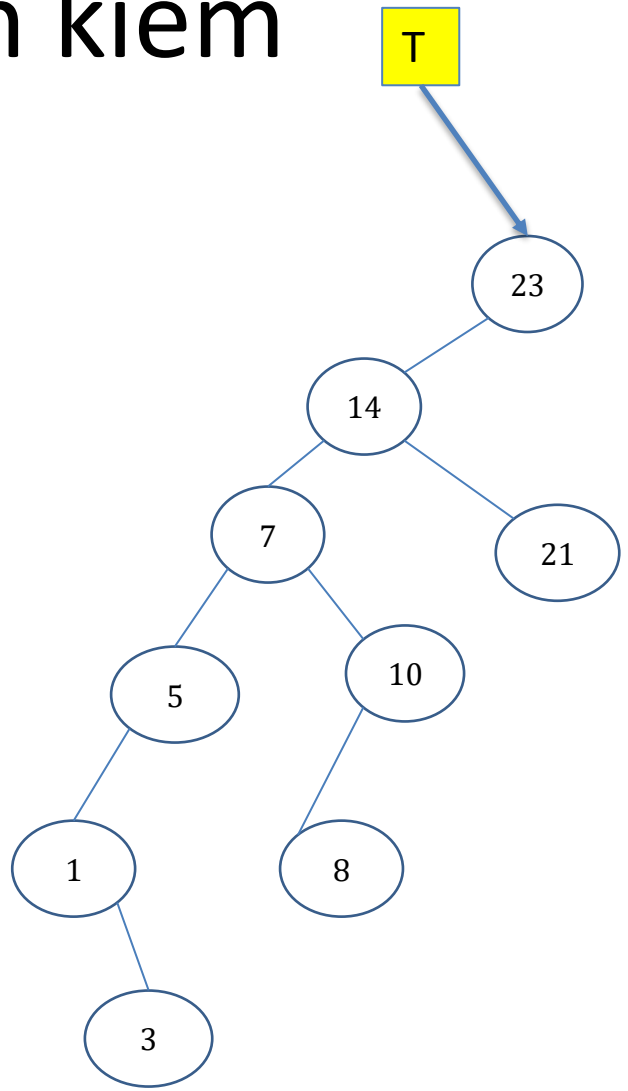


Cây nhị phân tìm kiếm

Cây nhị phân tìm kiếm

Ví dụ 2:

- Dựng một cây nhị phân tìm kiếm với một dãy số đã cho từ một cây rỗng
- Cho dãy số: 23 , 14, 7, 10, 8, 5,1 21, 3.
- Áp dụng giải thuật BTS ta có dạng cây sau
- Thứ tự của các phần tử mảng thay đổi cấu trúc cây nhị phân tìm kiếm
- Ví dụ: 8,3,14,1,10,23,7,21,5



Cây nhị phân tìm kiếm

- Khai báo

```
struct  Node  {
        keytype  Key;       //Thường là kiểu có thứ tự, có thể so sánh được
        Node * LP, *RP;
};

typedef  Node*  PNode;
typedef  PNode BinaryTree;
typedef  BinaryTree  BSearchTree;
```

Cây nhị phân tìm kiếm

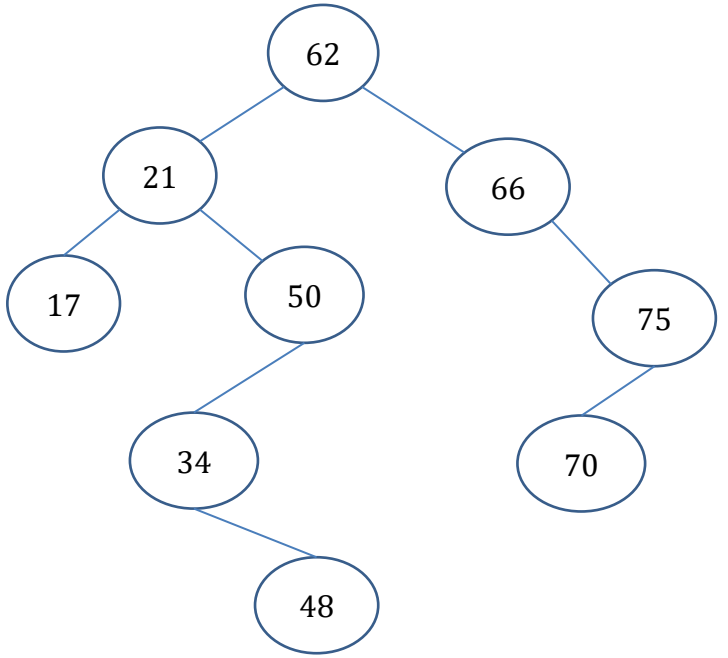
- Cài đặt các thao tác cơ bản: Tìm một nút có giá trị x cho trước. Hàm sẽ trả về con trỏ trỏ vào nút tìm được nếu có, trái lại trả về con trỏ NULL

```
PNode SearchT (BSearchTree Root, keytype  x){
      if  (Root==NULL)  return  NULL;
      if  (x == Root->Key) return  Root;
       else   if  (x < Root->Key)  return  SearchT (Root->LP, x);
      else  return  SearchT (Root->RP, x);
}
```

Minh họa thao tác tìm kiếm

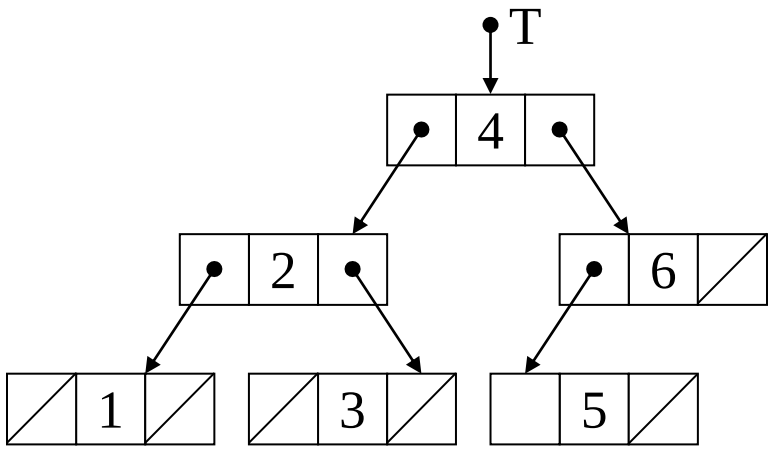
- Nếu X = 34, so sánh X với nút gốc là 62, thực hiện các bước sau
  - X<62: chuyển sang cây con trái
  - X>21: chuyển sang cây con phải
  - X<50: chuyển sang cây con trái
  - X = 34: tìm kiếm thoả

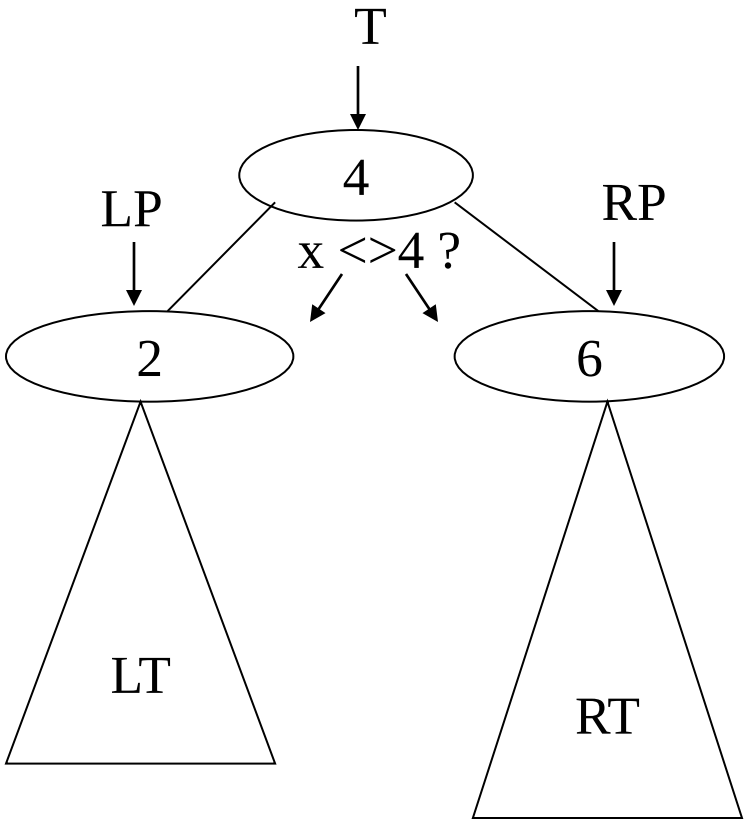
- Nếu X = 68, so sánh X với nút gốc là 62, thực hiện các bước sau
  - X>62: chuyển sang cây con phải
  - X>66: chuyển sang cây con phải
  - X<75: chuyển sang cây con trái
  - X<70: chuyển sang cây con trái , cây rỗng , tìm kiếm không thoả



Cây nhị phân tìm kiếm

- Cài đặt các thao tác cơ bản: Bổ sung một nút
  - Minh họa:





Cây nhị phân tìm kiếm

- Cài đặt các thao tác cơ bản: Bổ sung một nút

```
void InsertT (BSearchTree & Root, keytype x){
    PNode Q;
    if (Root==NULL) {          //khi cây rỗng
        Q = new Node;          //tạo ra đỉnh mới
        Q->Key = x;
        Q->LP =Q->RP = NULL;
        Root = Q;
    }
    else {
        if (x < Root->Key) InsertT (Root->LP, x);
        else if (x > Root->Key) InsertT (Root->RP, x);
    }
}
```

Cây nhị phân tìm kiếm

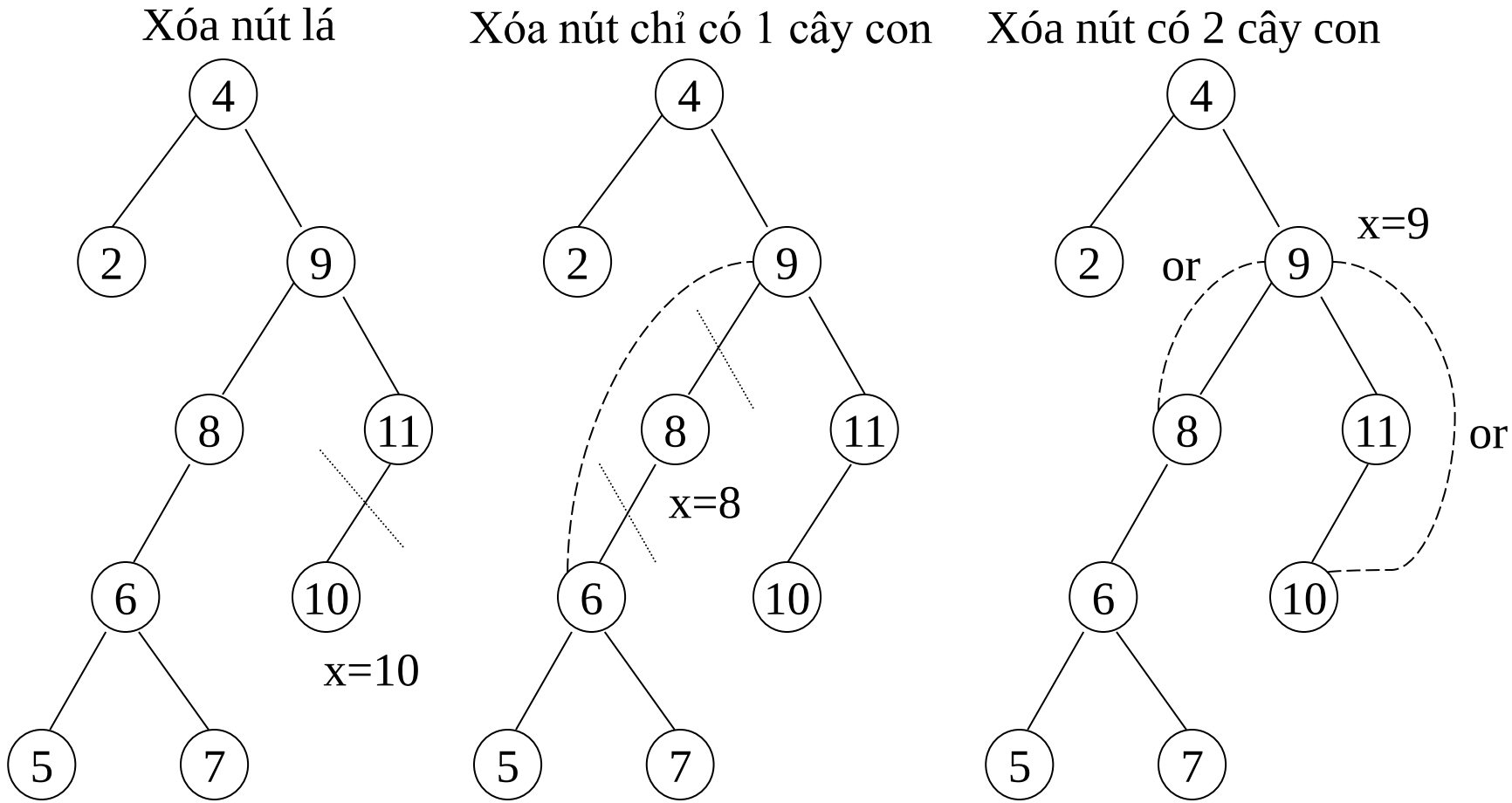
- Cài đặt các thao tác cơ bản: Lấy ra một nút

  Ý tưởng

  - Tìm đến nút có giá trị khóa cần loại bỏ
  - Thực hiện phép loại bỏ nút đó
  - Thực hiện xắp xếp lại để đảm bảo tính chất cây tìm kiếm NP
    - Nếu đỉnh cần loại bỏ là lá => không cần làm gì
    - Nếu đỉnh cần loại bỏ chỉ có 1 cây con => thực hiện phép nối
    - Nếu đỉnh cần loại bỏ có hai cây con LTree, RTree => cần thay khoá của đỉnh hai cây con này bằng giá trị Max(LTree) hoặc Min(RTree)
  - Lưu ý:
    - Giá trị Max(LTree) cần tìm ở phần tử bên phải ngoài cùng của cây con Ltree: *luôn đi theo bên phải của cây đến khi không được nữa Q->RP=NULL thì đó là nút ngoài cùng bên phải*
    - Giá trị Min(RTree) cần tìm ở phần tử bên trái ngoài cùng của cây con Rtree: *luôn đi theo bên trái của cây đến khi không được nữa Q->LP=NULL thì đó là nút ngoài cùng bên trái*

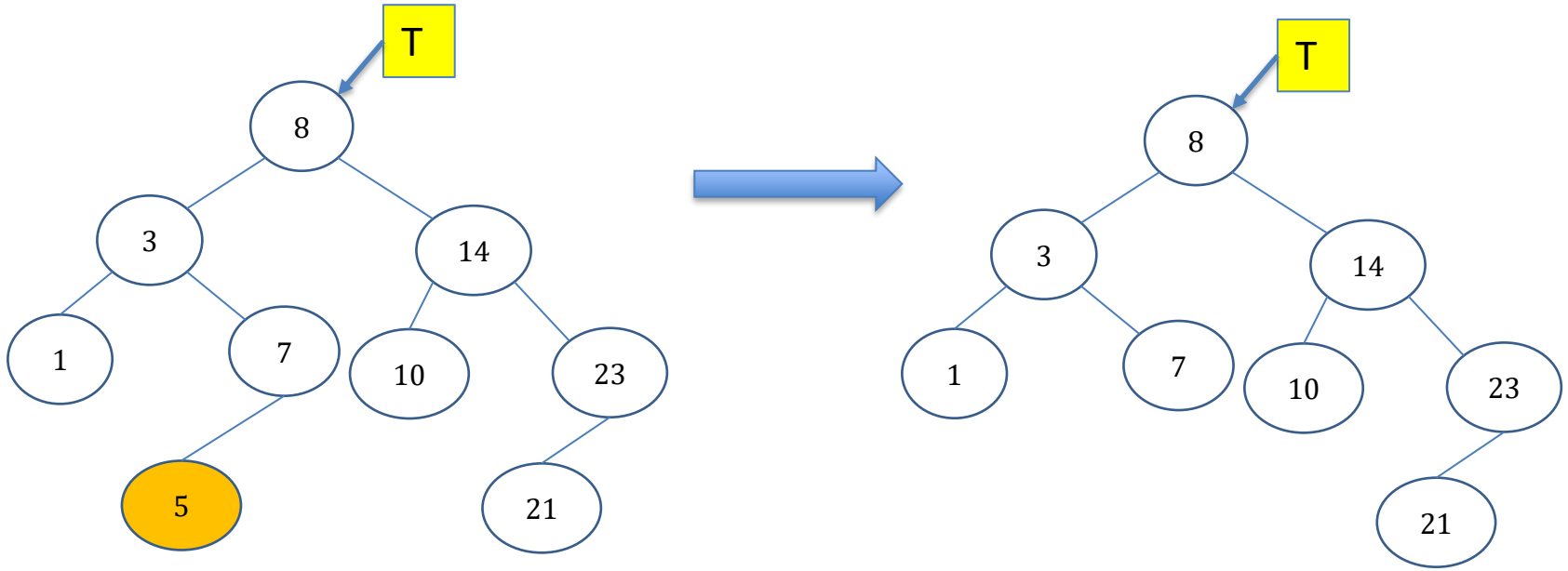
Cây nhị phân tìm kiếm

- Cài đặt các thao tác cơ bản: Lấy ra một nút



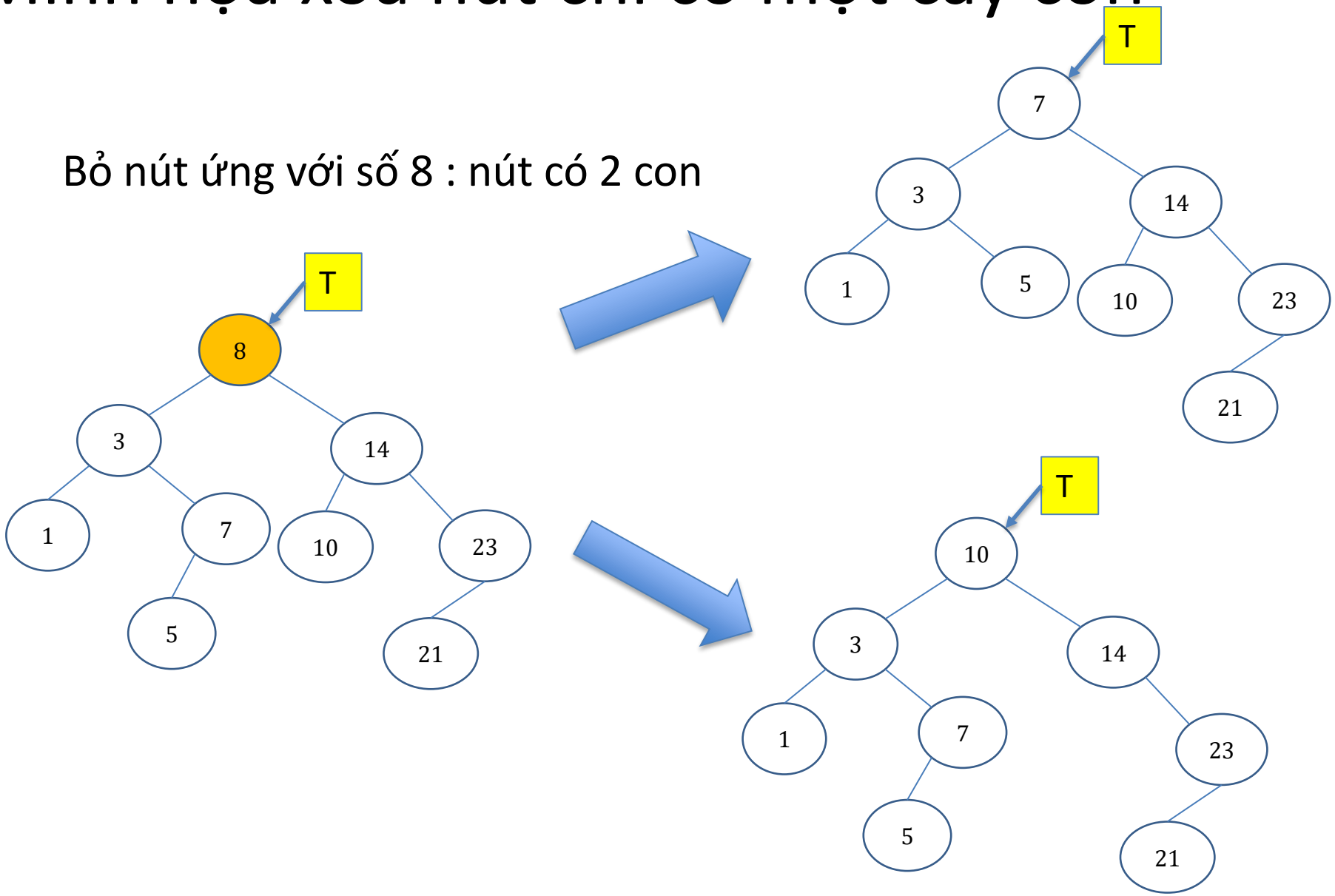
Minh họa thao tác xóa nút lá

Bỏ nút ứng với số 5 : nút lá



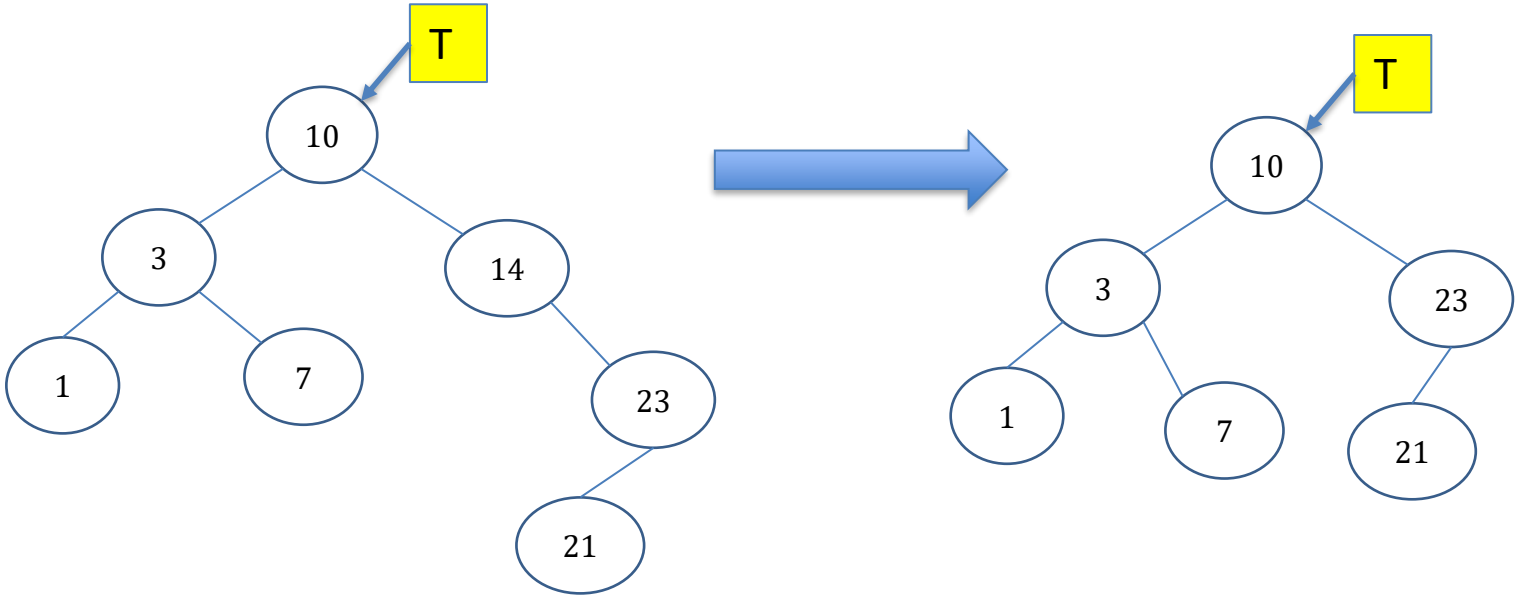
Minh họa xóa nút chỉ có một cây con

Bỏ nút ứng với số 8 : nút có 2 con



Minh họa xóa nút có hai cây con

Bỏ nút ứng với số 14 : nút có 1 con



Cây nhị phân tìm kiếm

- Cài đặt các thao tác cơ bản: Lấy ra một nút

```
void DeleteT (BSearchTree & Root, keytype x){
    if (Root != NULL) {
        if (x < Root->Key) DeleteT (Root->LP, x);
        else if (x > Root->key) DeleteT (Root->RP, x);
        else DelNode (Root);        //Xoá gốc của cây
    }
}
```

```
void DelNode (PNode & P) {  //Xóa giá trị ở nút P & sắp lại cây
    PNode Q, R;
    if (P->LP == NULL) {        //Xóa nút chỉ có cây con phải
        Q = P;
        P = P->RP;
    } else     if (P->RP = NULL)    //Xóa nút chỉ có cây con trái
    {
```

Cây nhị phân tìm kiếm

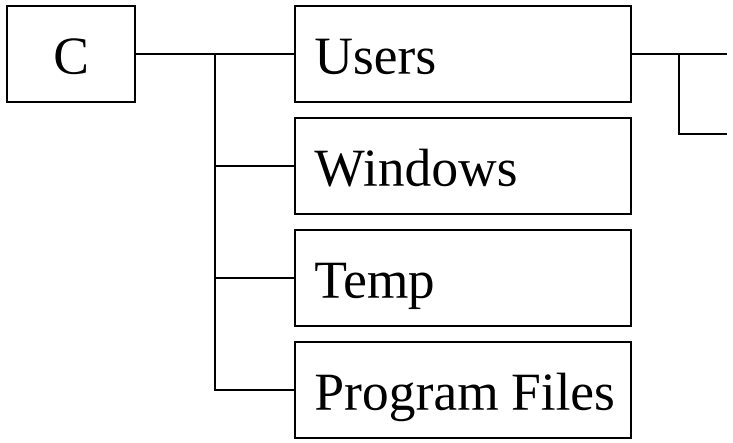
- Cài đặt các thao tác cơ bản: xoá một nút (tiếp...)

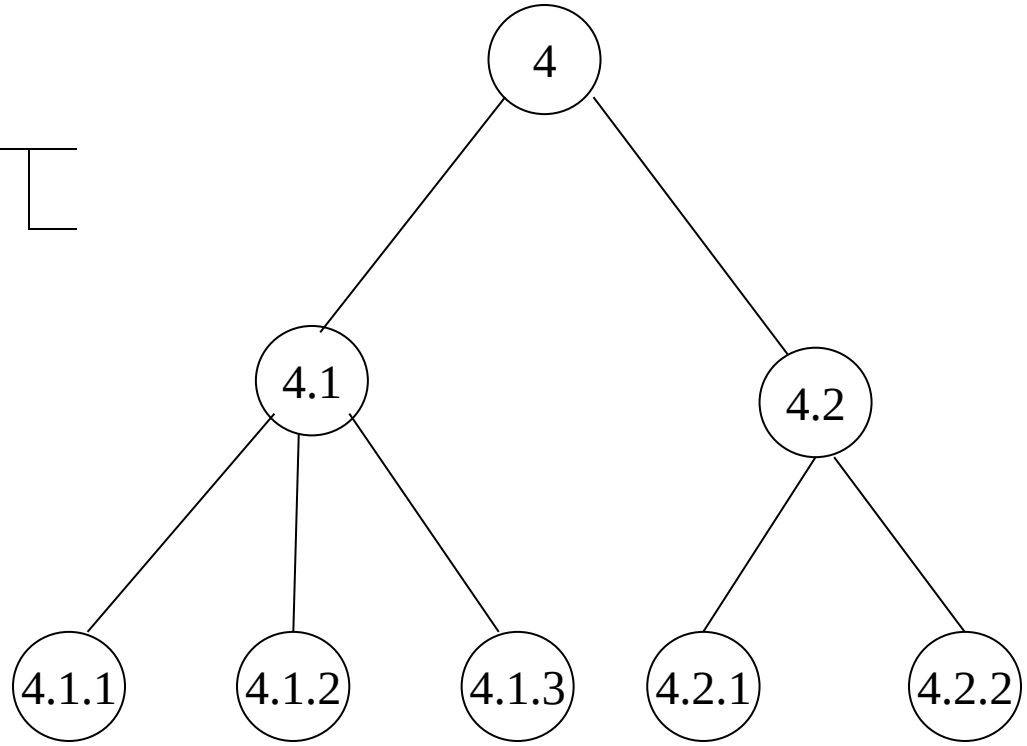
[Tìm giá trị Max ở cây con trái. Nó luôn là giá trị ở nút ngoài cùng bên phải cây con => giải thuật: Luôn đi theo bên phải của Q, khi nào không đi đươc nữa Q->RP = NULL thì đó là nút ngoài cùng bên phải]

```
                Q = P;
                P = P->LP;
            } else    {              //Xóa nút có 2 cây con
                Q = P->LP;
                if  (Q->RP == NULL) {
                    P->Key = Q->Key;
                    P->LP = Q->LP;
                } else {
                    do {        //Dùng R để lưu parent của Q
                      R = Q;
                      Q = Q->RP;
                    } while  (Q->RP != NULL);
                    P->Key = Q->Key;       //Lấy giá trị ở Q đưa lên
                    R->RP = Q->LP;    //Chuyển con của Q lên vị trí Q
                }
            }
        delete  Q;                 //Xoá Q
    }
```

Cây tổng quát

- Giới thiệu
  - Cây tổng quát là cây mà mỗi nút có thể có nhiều hơn hai con. Nói chung, cây càng có nhiều con thì việc cài đặt càng phức tạp, nhất là những cây mà số con tối đa thường không cố định (VD: cây gia phả, cây thư mục).





Cây tổng quát

- Các phương pháp cài đặt
  - Cài đặt trực tiếp sử dụng CTLT móc nối
  - Cài đặt gián tiếp qua cây nhị phân

Cây tổng quát

- Cài đặt trực tiếp sử dụng CTLT móc nối

  Ý tưởng:

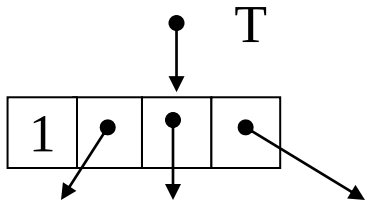
  – Tương tự khi cài đặt cây NP

  – Chỉ thích hợp khi ta biết trước cấp của cây (số con tối đa của một nút) và thường là số cấp không lớn (4-5).

  – Khi số cấp của cây lớn thì cách cài đặt này sẽ không hiệu quả do tốn khá nhiều bộ nhớ để lưu các con trỏ rỗng.

  – Nguyên tắc cài đặt cụ thể: nếu gọi cấp của cây là d, thì cấu tạo một nút gồm 2 phần:

    - Phần Info chứa nội dung thông tin của nút
    - Phần móc nối sẽ có d con trỏ p_1, p_2,…p_d để trỏ đến tối đa d con của nút.



Cây tổng quát

- Cài đặt trực tiếp sử dụng CTLT móc nối
  - Tổ chức cấu trúc cây
    - Vẫn có một con trỏ T trỏ vào nút gốc của cây.
    - N là kích thước của cây => số con trỏ rỗng trong cây T là: N(d-1) + 1.
    - Khi d càng lớn thì số con trỏ rỗng trong cây sẽ càng nhiều, lớn hơn số nút của cây nhiều lần => lãng phí bộ nhớ rất lớn.
    - Hơn nữa, cách cài đặt này đòi hỏi chúng ta phải biết trước số con lớn nhất có trong một nút của cây. Điều này không phải lúc nào cũng có được trong các cây trong thực tế.

Cây tổng quát

- Cài đặt gián tiếp qua cây nhị phân

  Ý tưởng:

  – Tận dụng các kết quả cài đặt từ cây NP, đồng thời khắc phục các hạn chế của cách cài đặt trực tiếp.

  – Chuyển cây tổng quát về cây NP.

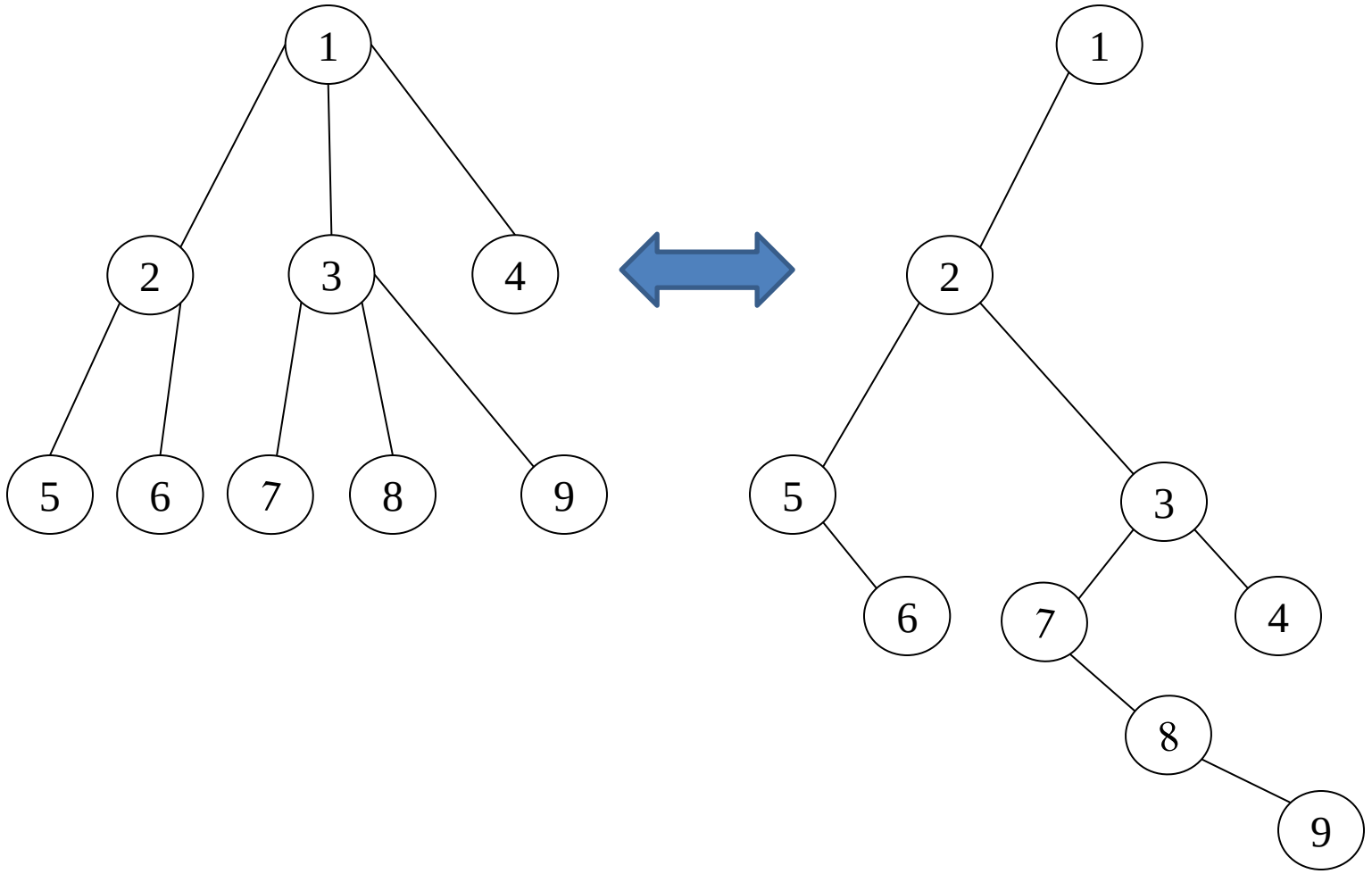
  – Cài đặt gián tiếp thông qua cài đặt cây NP. Muốn như vậy, ta phải xác định một quy tắc chuyển đổi hai chiều kiểu song ánh từ cây tổng quát sang cây NP và ngược lại.

  – Quy tắc:

    - Coi cây tổng quát là cây có thứ tự. Nếu cây tổng quát không có thứ tự thì ta đưa thêm vào một thứ tự trong cây đó.
    - Chuyển nút con trái nhất (con cả) của một nút thành nút con trái của nó
    - Chuyển nút em kế cận của một nút thành nút con phải của nút đó.

Cây tổng quát

– Ví dụ



Cây tổng quát

- Cài đặt gián tiếp qua cây nhị phân

  Nhận xét:

  – Cách cài đặt này cho phép ta cài đặt một cây bất kì, có thể có cấp rất lớn và rất hiệu quả về bộ nhớ.

  – Cần thêm quá trình chuyển đổi ngữ nghĩa thuận và nghịch giữa cây tổng quát và cây nhị phân. Ví như bổ sung một nút là con cả của một nút trong cây tổng quát sẽ thành bổ sung một nút thành nút con trái trong cây NP => giảm tốc độ thực hiện các thao tác cơ bản như bổ sung, tìm kiếm trên cây.

Ứng dụng của cấu trúc cây

- Một số ứng dụng cây NP
  - Cây nhị phân tìm kiếm: ứng dụng vào lưu trữ và tìm kiếm thông tin một các hiệu quả nhất (tốn ít bộ nhớ và thời gian chạy nhanh nhất)
  - Cây biểu thức: giúp ta thực hiện hàng loạt các thao tác tính toán cơ bản cũng như phức tạp (do người dùng định nghĩa)
  - Cây cân bằng
- Một số ứng dụng khác
  - Cây biểu diễn các tập hợp
  - Cây hỗ trợ ra quyết định: decision tree

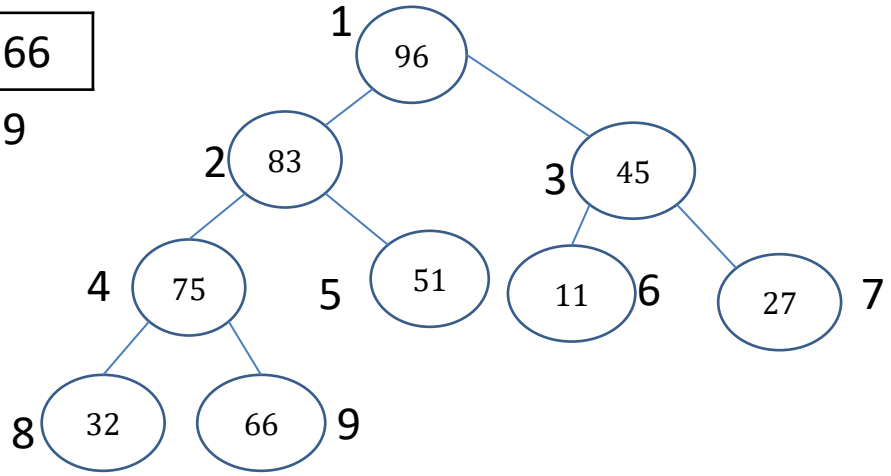
Cấu trúc cây nhị phân trong sắp xếp

- Cấu trúc đống
- Phép tạo đống
- Sắp xếp kiểu vun đống (Heap – sort)

Cấu trúc đống

- Đống là một cây nhị phân mà mỗi nút gắn với một số sao cho số ở nút cha bao giờ cũng lớn hơn số ở nút con
- Ví dụ: dùng cây nhị phân hoàn chỉnh
  - Số ứng với gốc của đống chính là số lớn nhất
  - Biểu diễn trong máy dưới dạng vector như sau

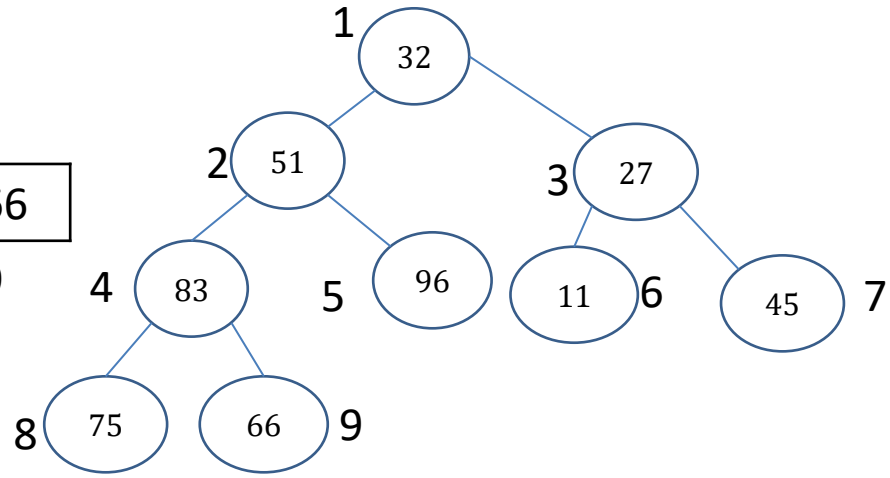
96	83	45	75	51	11	27	32	66
1	2	3	4	5	6	7	8	9



Phép tạo đống

- Đặt vấn đề: Một dãy số biểu diễn dạng vector trong máy có thể biểu diễn dưới dạng cây nhị phân hoàn chỉnh và ngược lại. Tuy nhiên cây này chưa phải là đống
- Tạo đống như thế nào ?

32	51	27	83	96	11	45	75	66
1	2	3	4	5	6	7	8	9



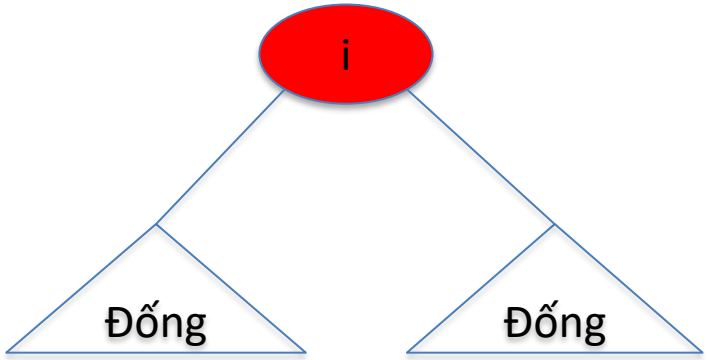
Phép tạo đống

- Nếu một cây nhị phân hoàn chỉnh là đống thì cây con cũng là đống
  - Cây nhị phân hoàn chỉnh có n nút thì chỉ có n/2 nút cha.
  - Nút lá bao giờ cũng coi là đống
- Bài toán: tạo đống cho cây nhị phân hoàn chỉnh, gốc cây này có thứ tự là i (theo cách lưu trữ kế tiếp) và hai nút con của nút gốc này đã là đống rồi

Giải thuật tạo đống

- Tiến hành theo kiểu từ dưới lên (bottom up)
- Lá là đống, nên tạo đống cho cây con mà gốc của nó có số thứ tự từ n/2 trở xuống
  - i: là thứ tự của nút gốc cây con cần xét
  - n: là số nút trên cây nhị phân hoàn chỉnh
- Có n nút biểu diễn bởi vector A, lệnh sau đây thực hiện tạo cây nhị phân hoàn chỉnh thành đống

  For i = n/2 down to 1 Call Adjust(i,n)
- Hàm tạo đống: **Adjust(i,n)**



Giải thuật tạo đống

- Procedure Adjust(i,n)

1. Key = A[i] ; j = 2*i; //bảo lưu số ở nút i và ghi nhận chỉ số nút con trái
2. While j <=n do {
3. If j < n and A[j] < A[j+1] then j = j+1
4. If Key > A[j] then { A[j/2] = Key ; return;}
5. A[j/2] = A[j]; j = 2*j; // đưa số lớn hơn lên vị trí cha, tiếp tục xuống con trái

   }
7. A[j/2] = Key ;
8. Return

Sắp xếp vun đống

- Từ dãy số đã cho, quan niệm như một vector biểu diễn cho một cây hoàn chỉnh, tạo nên đống đầu tiên -> giai đoạn tạo đống ban đầu
- Đổi chỗ giữa số ở đỉnh đống với số ở cuối đống sau đó vun đống mới
- Quá trình này sẽ được lặp lại cho đến khi đống chỉ còn 1 nút -> giai đoạn sắp xếp

Sắp xếp vun đống

- Procedure HEAP-SORT(A,n)

1. for i = [n/2] down to 1 Call Adjust(i,n);
2. for i = n-1 down to 1 {

A[1] ⟷ A[i+1]; call Adjust(1,i);}

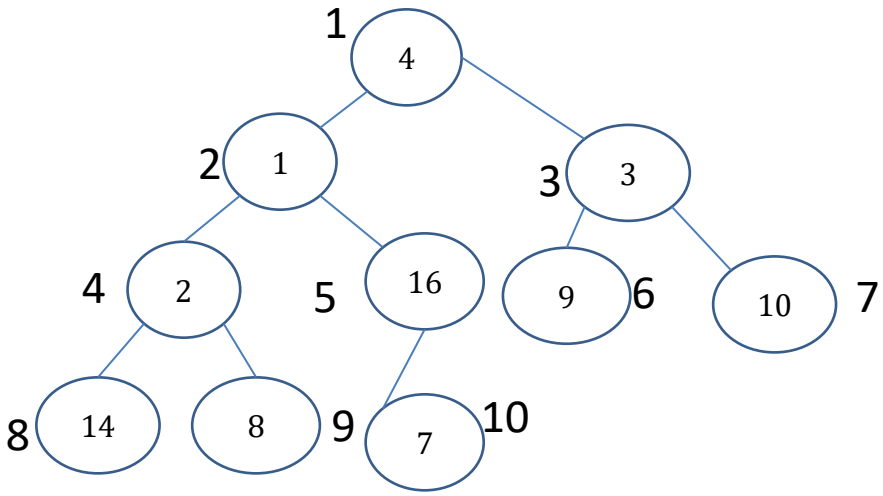
3. Return

- Sắp xếp dãy số sau

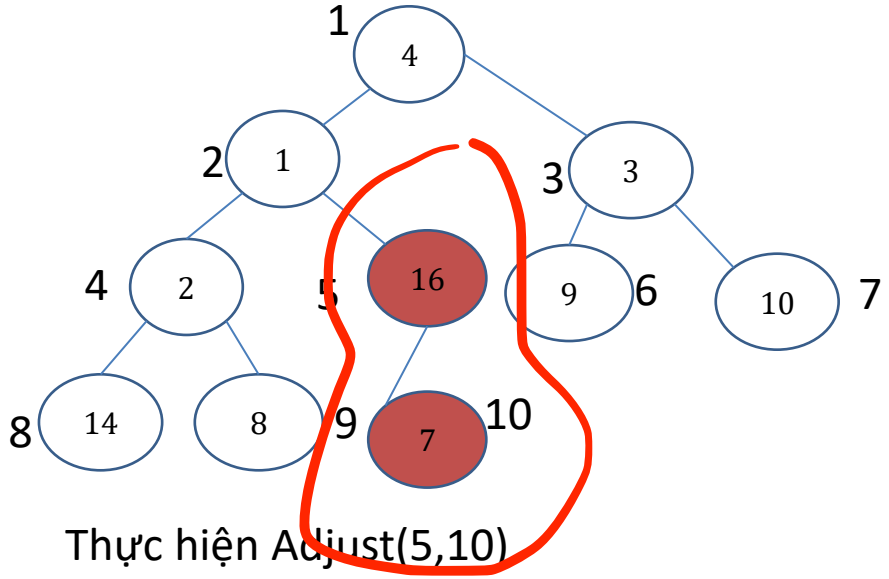
4	1	3	2	16	9	10	14	8	7

A[1] A[10]

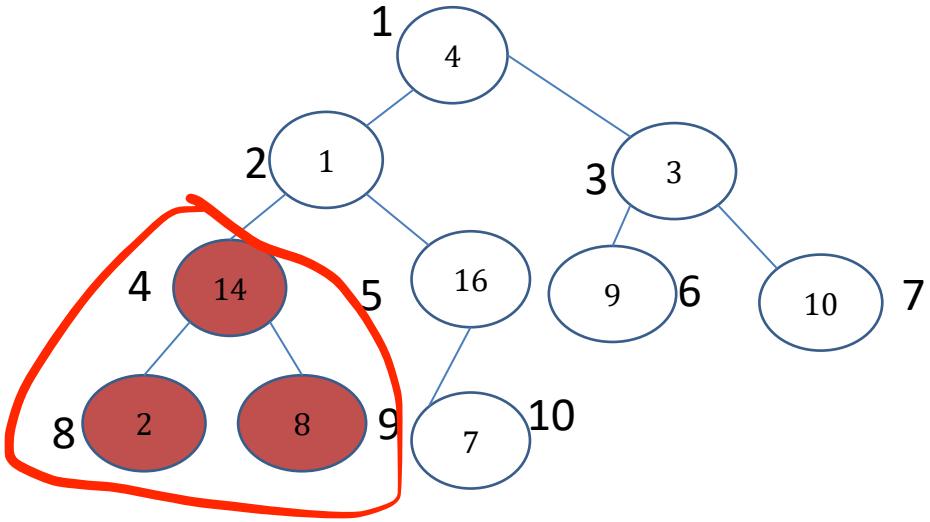
Mô tả các bước vun đống



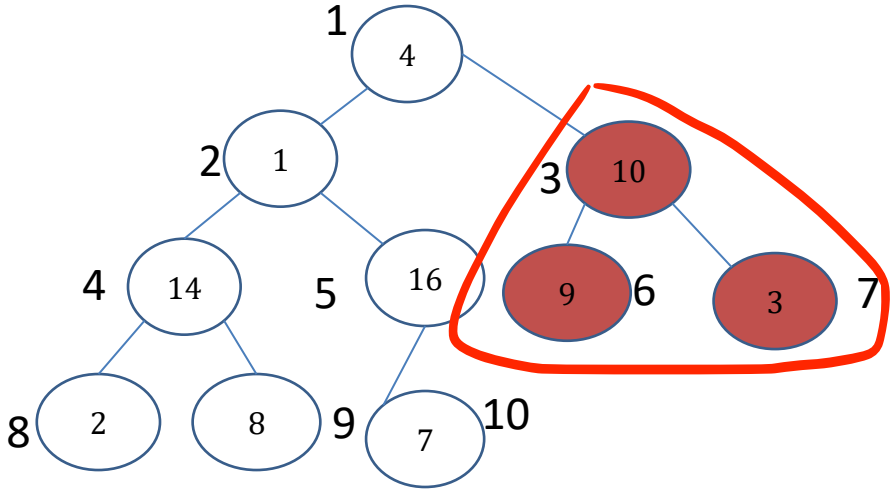
Tạo đống ban đầu



Thực hiện Adjust(5,10)

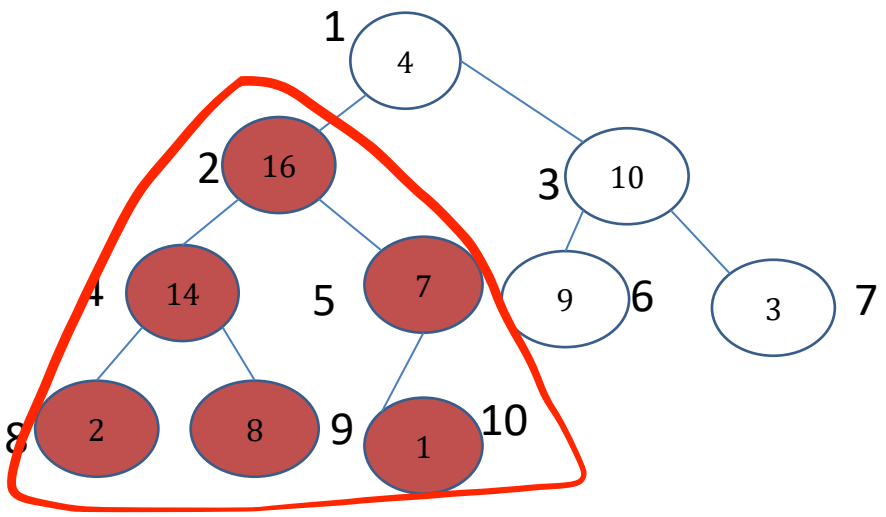


Thực hiện Adjust(4,10)

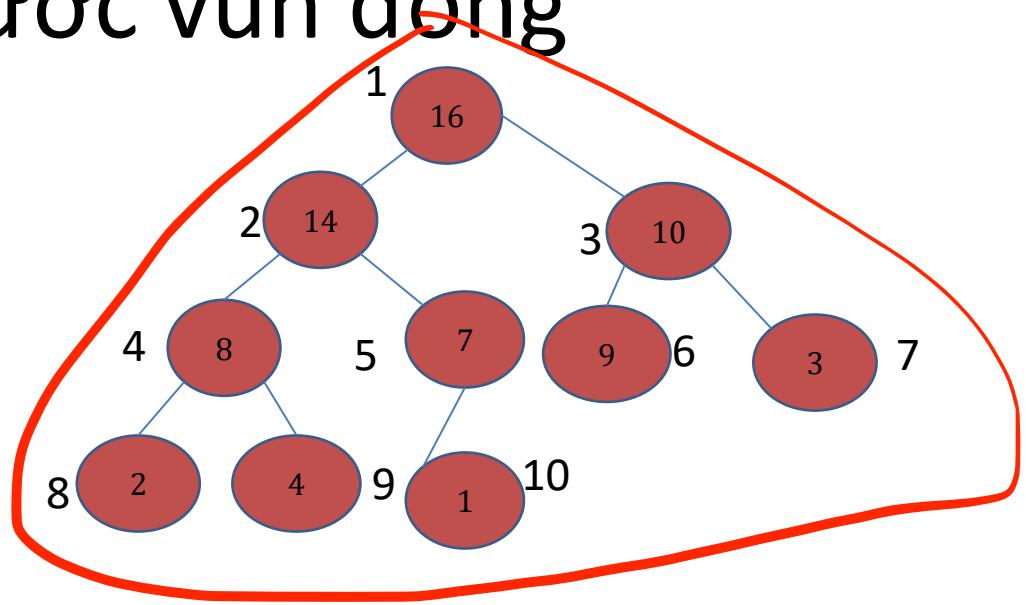


Thực hiện Adjust(3,10)

Mô tả các bước vun đống



Thực hiện Adjust(2,10)

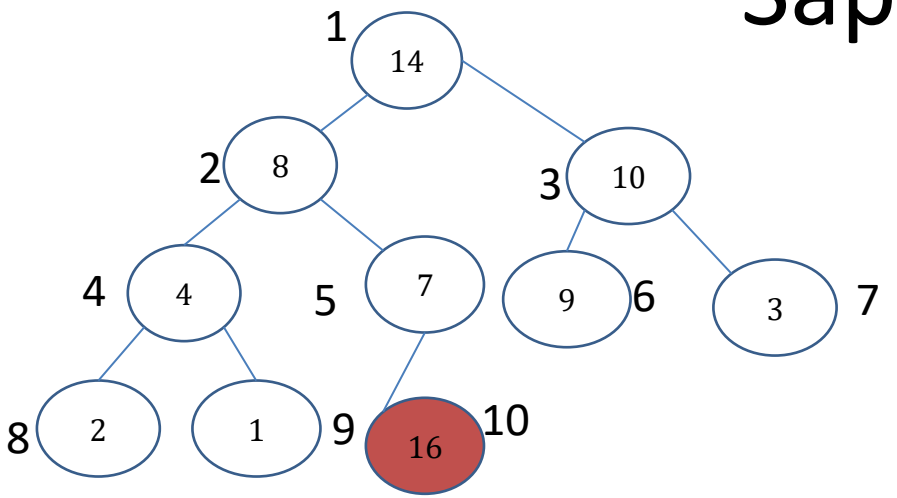


Thực hiện Adjust(1,10)

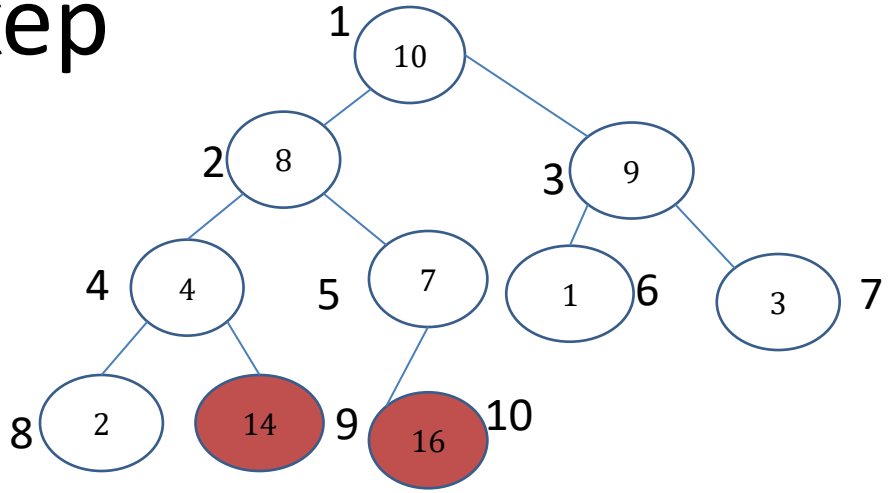
Lúc này cây đã là đống và biểu diễn trong máy là vector A như sau

16	14	10	8	7	9	3	2	4	1

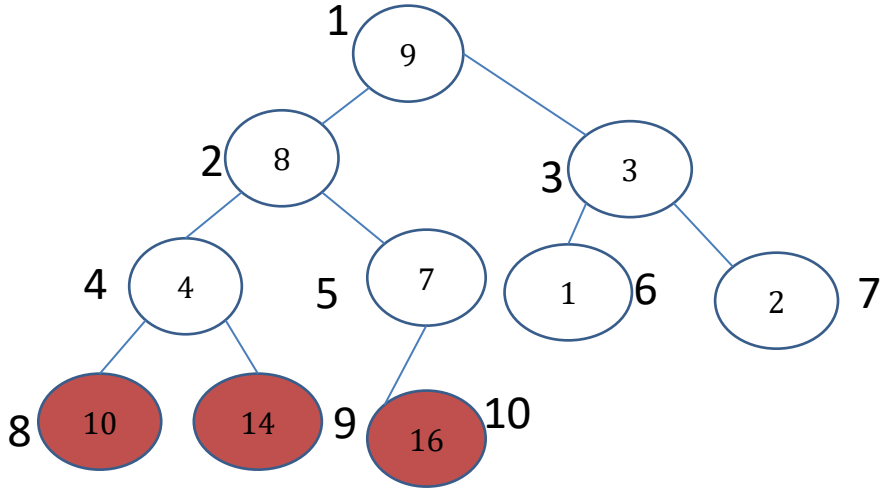
Sắp xếp



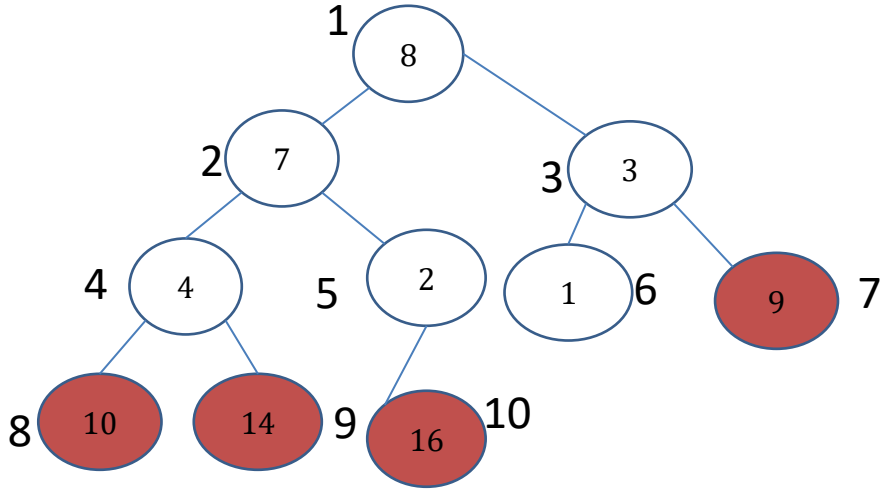
Đổi lần 1: giữa A[1] và A[10], vun đống cho cây với 9 nút còn lại, 16 đã vào đúng vị trí



Đổi lần 2: giữa A[1] và A[9], vun đống cho cây với 8 nút còn lại, 14, 16 vào đúng vị trí

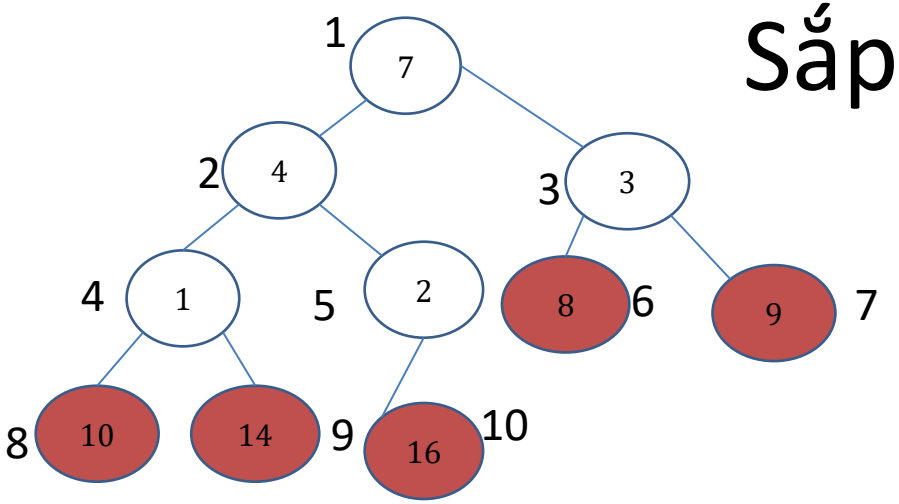


Đổi lần 3: giữa A[1] và A[8], vun đống cho cây với 7 nút còn lại, 10,14, 16 vào đúng vị trí

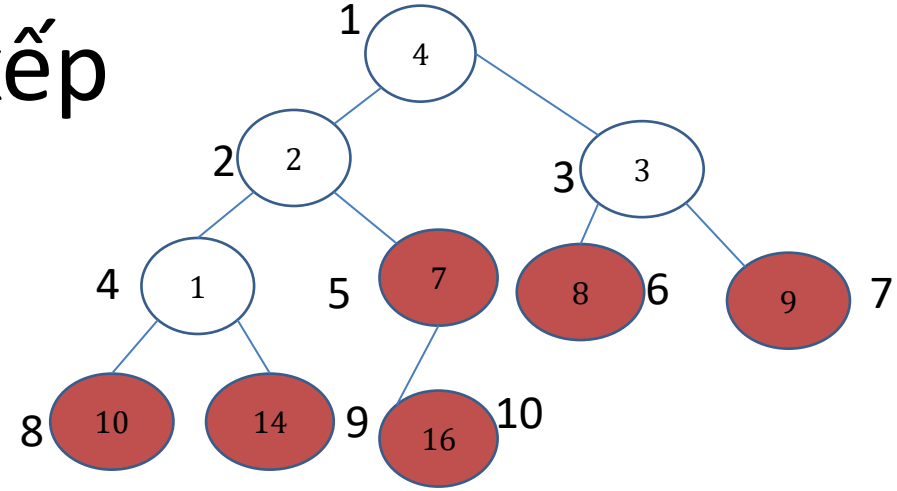


Đổi lần 4: giữa A[1] và A[7], vun đống cho cây với 6 nút còn lại

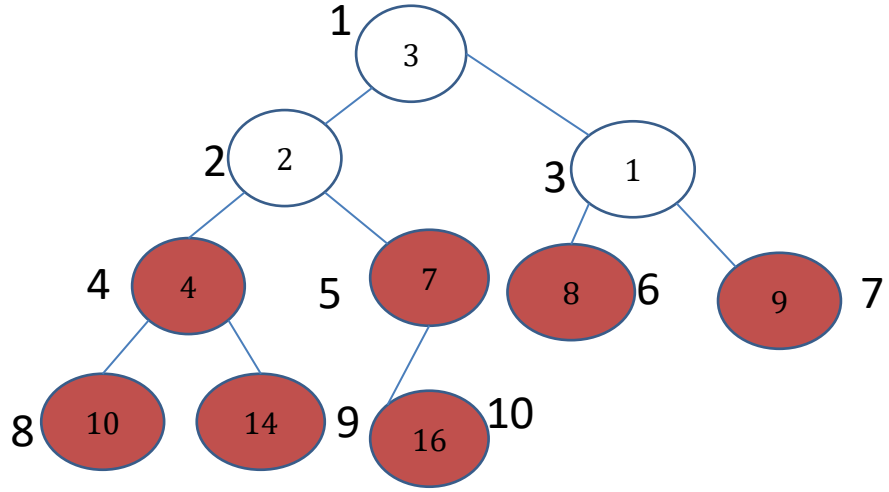
Sắp xếp



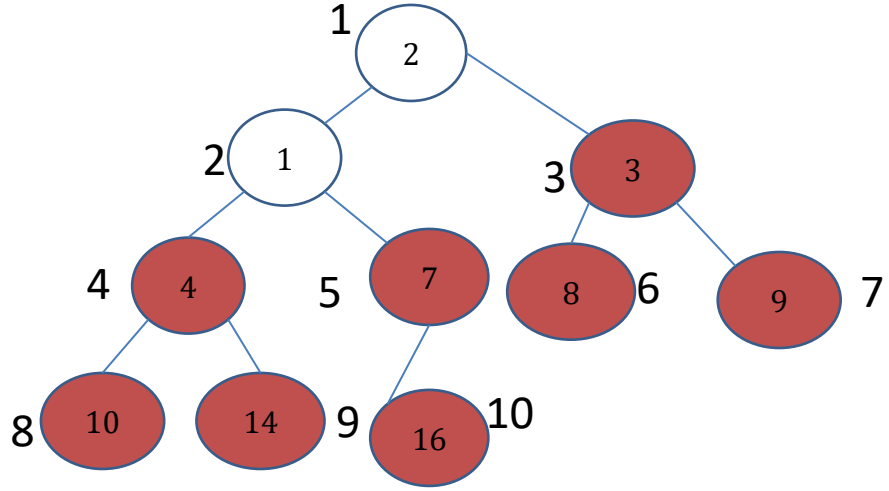
Đổi lần 5: giữa A[1] và A[6], vun đống cho cây với 5 nút còn lại



Đổi lần 6: giữa A[1] và A[5], vun đống cho cây với 4 nút còn lại

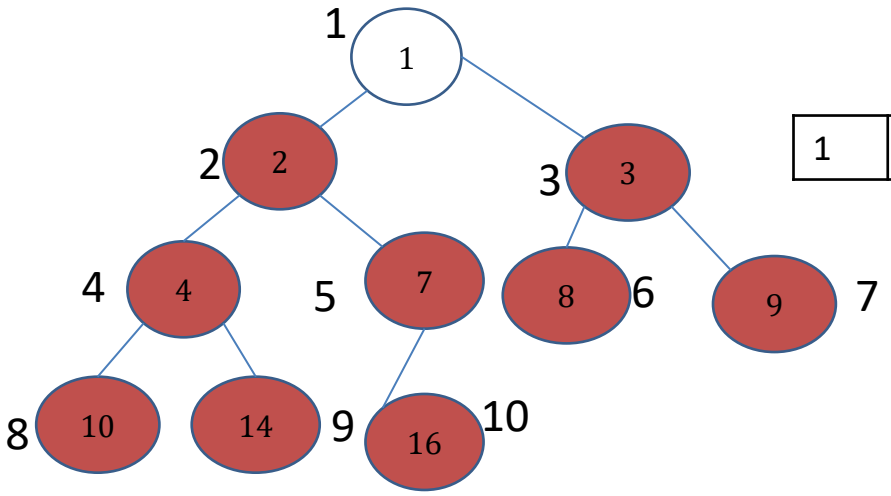


Đổi lần 7: giữa A[1] và A[4], vun đống cho cây với 3 nút còn lại



Đổi lần 8: giữa A[1] và A[3], vun đống cho cây với 2 nút còn lại

Sắp xếp



1	2	3	4	7	8	9	10	14	16

Đổi lần 9: giữa A[1] và A[2], đống chỉ còn 1 nút. Dãy A đã được sắp xếp

T(n) = O(nlogn)

Bài tập

Câu 1: cho biểu thức E =(2x+y)*(5a-b)^3

a. Hãy vẽ cây nhị phân T biểu diễn E
b. Muốn có biểu thức dạng tiền tố, hậu tố thì phải duyệt cây T theo thứ tự nào. Nêu kết quả của các phép duyệt đó

Câu 2: Cho cây nhị phân T, lưu trữ móc nối trong bộ nhớ. Hãy viết giải thuật đệ quy tính số nút của cây T

Câu 3: Dựng cây nhị phân tìm kiếm với dãy số sau đây : 33, 44,50,22,60,77,35,40,10

```
//Tìm SV có điểm nhỏ hơn m
PNode SearchT (BSearchTree T, int m){
    if  (T!=NULL) {
        if  (T->infor.mark<m) return  T;
        if  (T->infor.mark>=m)  {
            PNode node=SearchT(T->LP, m);
            If (node!=null) return node;
            Else return searchT(T->RP,m)
        }
    }
    Return null;
}
```