Cây nhị phân tìm kiếm

Cây nhị phân tìm kiếm

- Trong chương 3, chúng ta đã làm quen với một số cấu trúc dữ liệu động. Các cấu trúc này có sự mềm dẻo nhưng lại bị hạn chế trong việc tìm kiếm thông tin trên chúng (chỉ có thể tìm kiếm tuần tự).
- Nhu cầu tìm kiếm là rất quan trọng. Vì lý do này, người

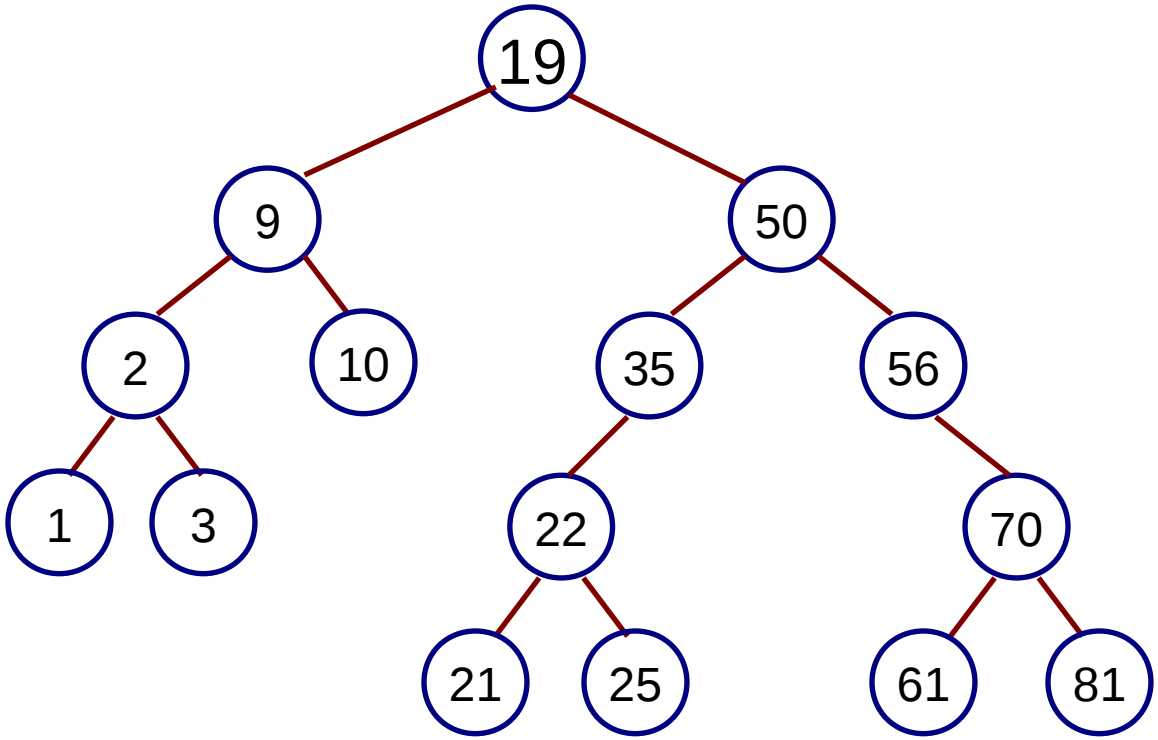
ta đã đưa ra cấu trúc cây để thỏa mãn nhu cầu trên.

- □□ Tuy nhiên, nếu chỉ với cấu trúc cây nhị phân đã định
- nghĩa ở trên, việc tìm kiếm còn rất mơ hồ.
- □□ Cần có thêm một số ràng buộc để cấu trúc cây trở nên chặt chẽ, dễ dùng hơn.
- □□ Một cấu trúc như vậy chính là **cây nhị phân tìm kiếm.**

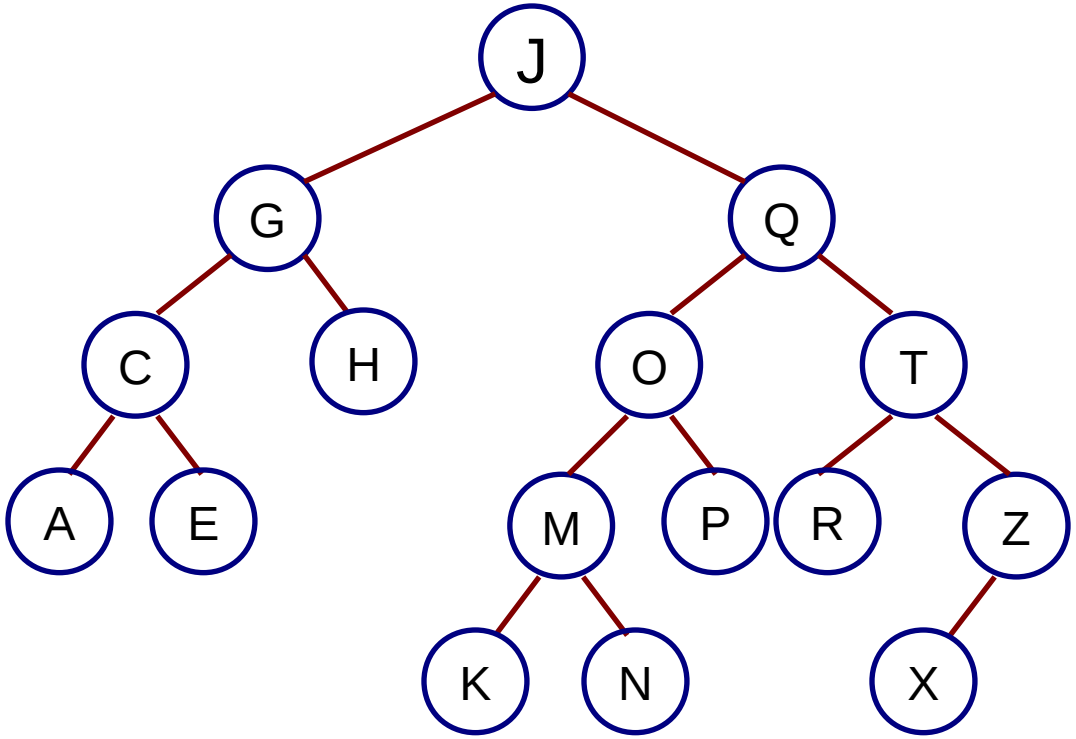
- Cây nhị phân tìm kiếm (CNPTK) là :
  - cây nhị phân
  - Tại mỗi nút, khóa của nút đang xét lớn hơn khóa của tất cả các nút thuộc cây con trái và nhỏ hơn khóa của tất cả các nút thuộc cây con phải.

    Data trái ≤ node ≤ data phải
  - Nếu số nút trên cây là N thì chi phí tìm kiếm trung bình chỉ khoảng $\log_2 N$.

Ví dụ cây nhị phân số



Ví dụ cây nhị phân ký tự



Định nghĩa cấu trúc cây

- Class Node{
  Kdl info; // khóa phục vụ tìm kiếm
  Node left;//cây con trái
  Node right;// cây con phải

  …
  }

  Class Tree {
  Int Spt;
  Node root ;

  …
  };

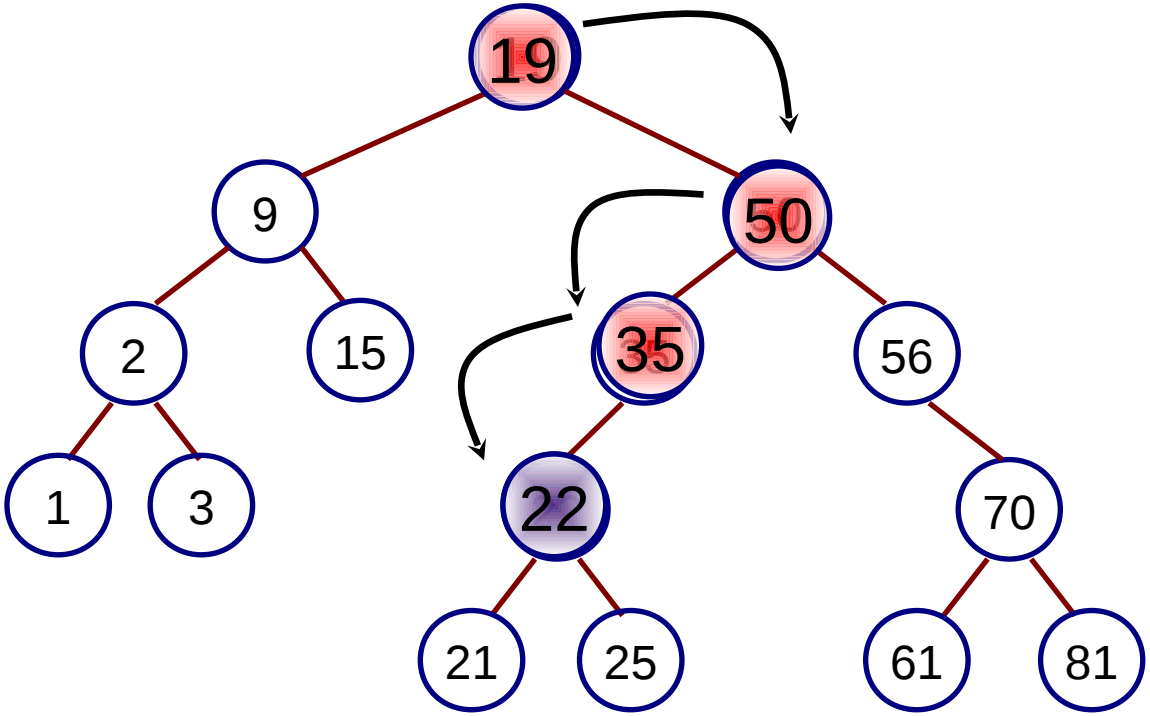
Các thao tác trên cây nhị phân

- Duyệt cây
- Thêm phần tử
- Xóa phần tử
- Sửa thông tin phần tử
- Tìm kiếm phần tử

- **Duyệt cây**
- Thao tác duyệt cây trên cây nhị phân tìm kiếm hoàn toàn giống như trên cây nhị phân.
- Khi duyệt theo thứ tự giữa, trình tự các nút duyệt qua sẽ cho ta một dãy các nút theo thứ tự tăng dần của khóa.

Tìm 1 phần tử

- Tìm phần tử x
- X=22



Code tìm kiếm

- Đệ qui
- NODE searchNode(NODE T, Data X)
  {
- if(T){
- if(T.Key == X) return T;
  if(T.Key > X)
  return searchNode(T.pLeft, X);
  else
  return searchNode(T.pRight, X);
- }
- return NULL;
- }

Code tìm kiếm

- Không đệ qui
- NODE searchNode(Data x)
- { NODE p = Root;
- while (p != NULL)
- {
- if(x == p.Key) return p;
  else
- if(x < p.Key) p = p.Left;
  else p = p.Right;
- }
- return NULL;
- }

- Dễ dàng thấy rằng số lần so sánh tối đa phải thực hiện để tìm phần tử X là h, với h là chiều cao của cây.
- Như vậy thao tác tìm kiếm trên CNPTK có n nút tốn chi phí trung bình khoảng O(log2n) .

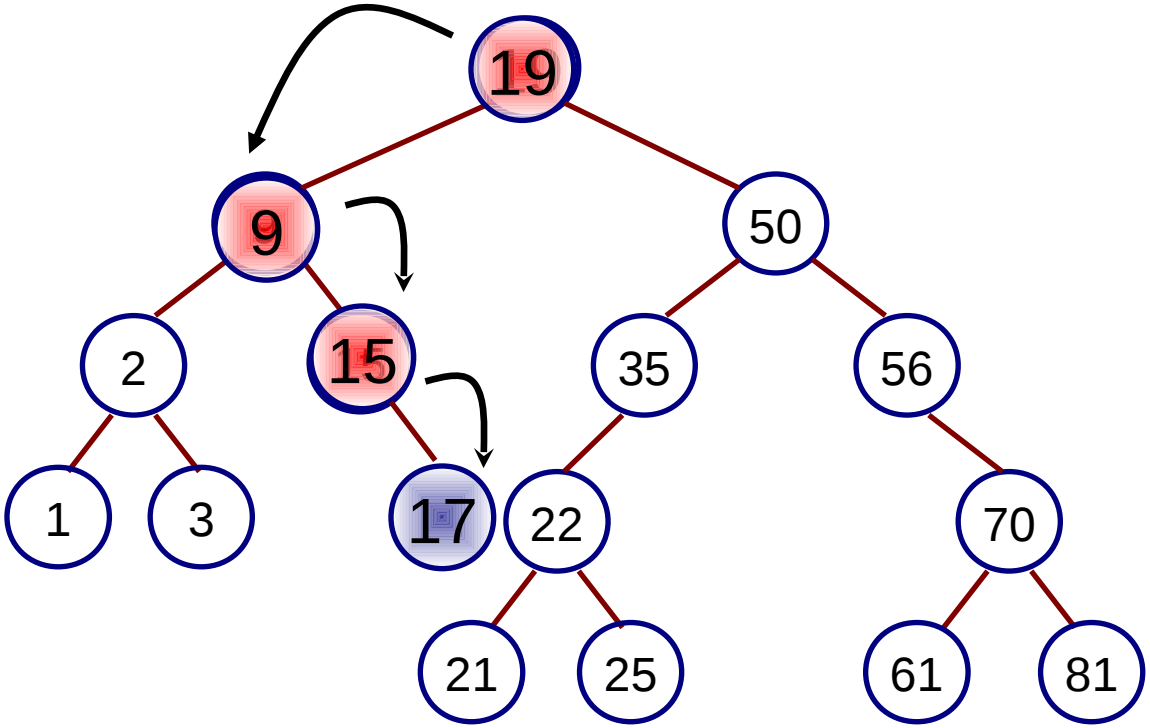
Thêm phần tử vào cây

- Việc thêm một phần tử X vào cây phải bảo đảm điều kiện ràng buộc của CNPTK. Ta có thể thêm vào nhiều chỗ khác nhau trên cây, nhưng nếu thêm vào một nút lá sẽ là tiện lợi nhất do ta có thể thực hiên quá trình tương tự thao tác tìm kiếm. Khi chấm dứt quá trình tìm kiếm cũng chính là lúc tìm được chỗ cần thêm.

✂→ thêm vào nút lá

- Hàm insert trả về giá trị –1, 0, 1 khi không đủ bộ nhớ, gặp nút cũ hay thành công:

- Thêm nút 17



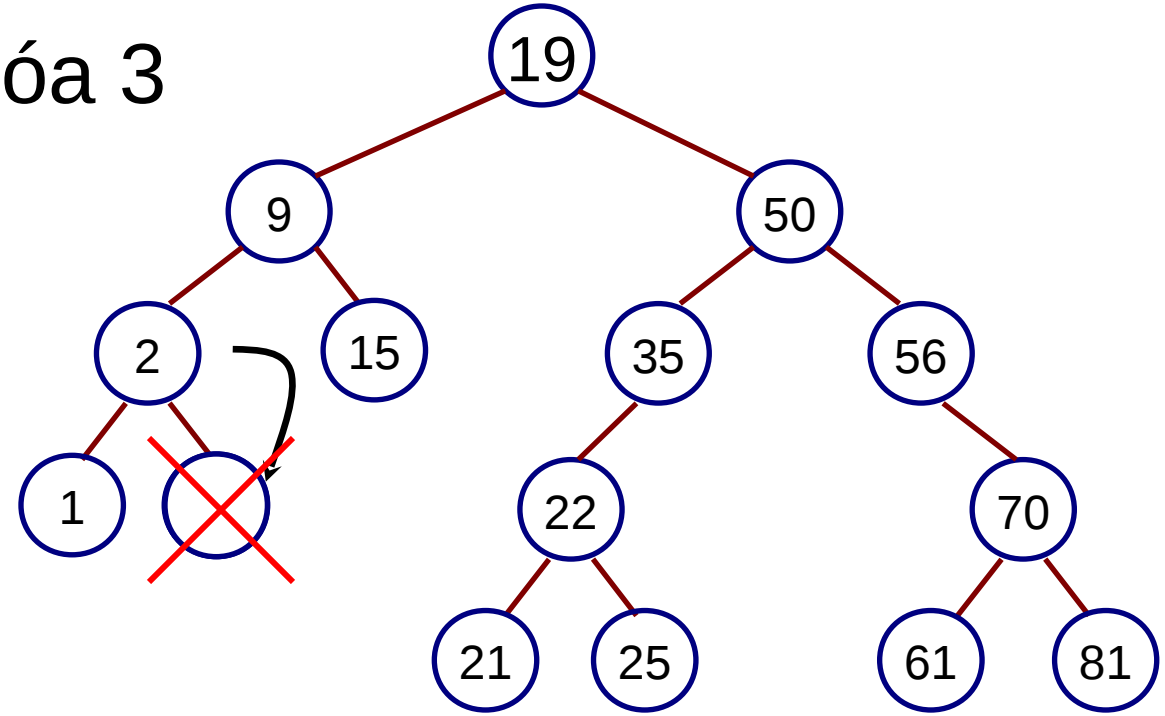
- Hàm insert trả về giá trị –1, 0, 1 khi không đủ bộ nhớ, gặp nút cũ hay thành công:
- int insertNode(Data X)
  {
- NODE T=root;
- if(T) {
- if(T.Key == X) return 0; //đã có
  if(T.Key > X)
  return insertNode(T.pLeft, X);
  else
  return insertNode(T.pRight, X);
- }
- T = new TNode;
  if(T == NULL) return -1; //thiếu bộ nhớ
  T.Key = X;
  T.pLeft =T.pRight = NULL;
  return 1; //thêm vào thành công
- }

Xóa phần tử trên cây

- Có 3 trường hợp khi hủy nút X có thể xảy ra:
- X là nút lá.
- X chỉ có 1 con (trái hoặc phải).
- X có đủ cả 2 con

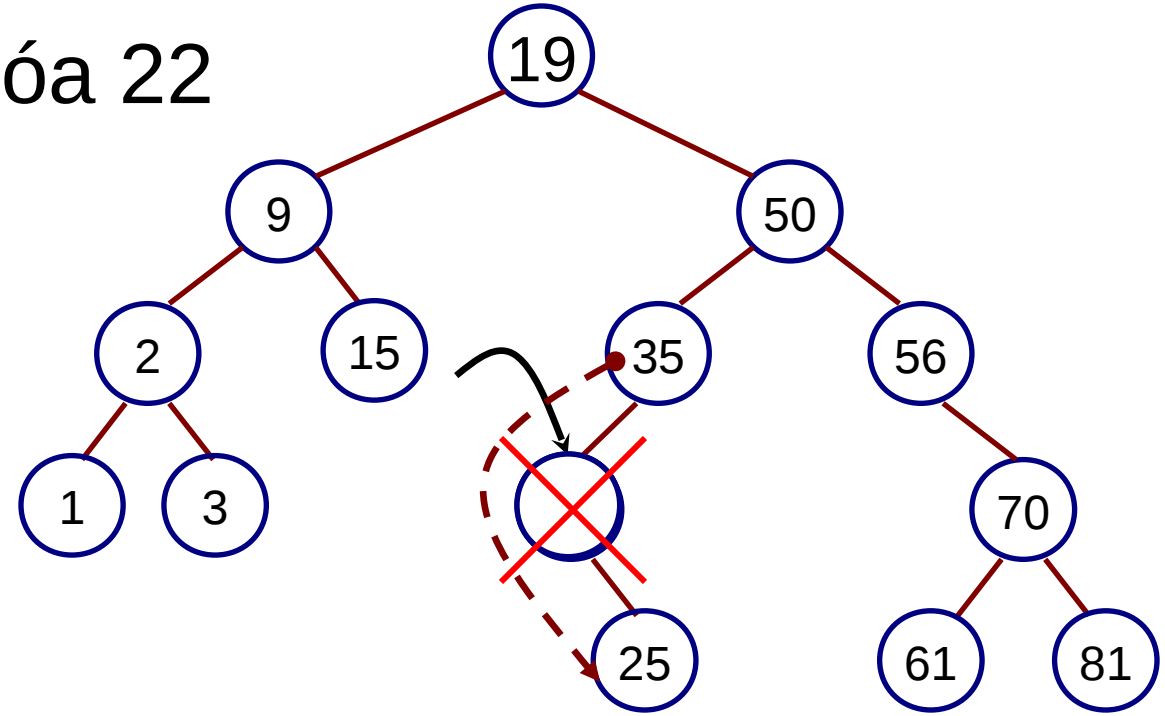
Trường hợp 1: X là nút lá.

- chỉ đơn giản hủy X vì nó không móc nối đến phần tử nào khác
- Ví dụ xóa 3



Trường hợp 2: X có một con.

- trước khi hủy X ta móc nối cha của X với con duy nhất của nó.
- Ví dụ xóa 22



Trường hợp 3: X là node trong

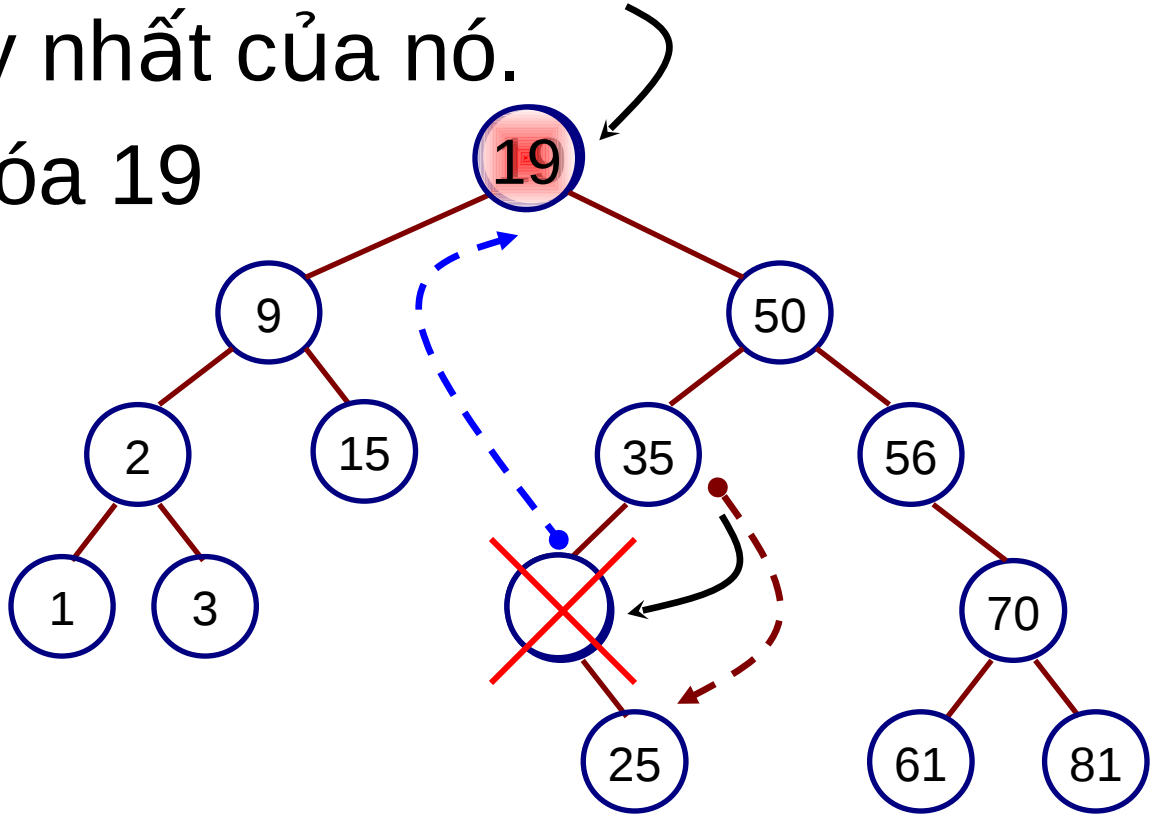
- ta không thể hủy trực tiếp do X có đủ 2 con $\Rightarrow$ Ta sẽ hủy gián tiếp. Thay vì hủy X, ta sẽ tìm một phần tử thế mạng Y. Phần tử này có tối đa một con. Thông tin lưu tại Y sẽ được chuyển lên lưu tại X. Sau đó, nút bị hủy thật sự sẽ là Y giống như 2 trường hợp đầu.

Phần tử thế mạng

- Có 2 phần tử thỏa mãn yêu cầu:
- Phần tử nhỏ nhất (trái nhất) trên cây con phải.
- Phần tử lớn nhất (phải nhất) trên cây con trái.
- Việc chọn lựa phần tử nào là phần tử thế mạng hoàn toàn phụ thuộc vào ý thích của người lập trình.

Trường hợp 3: X là node đầy

- trước khi hủy X ta móc nối cha của X với con duy nhất của nó.
- Ví dụ xóa 19



Thuật toán tìm phần tử thế mạng

- Void Timpttt(node p,node r,node t){
- R=p;
- T=r->right
- While(t->left!=NULL)
- R=t;
- T=t->left
- }
- Tra ve con trỏ t là vị trí phần tử thế mạng
- R là con trỏ cha của t