

Chương 5. CÂY NHỊ PHÂN TÌM KIẾM

ThS. Nguyễn Chí Hiếu

2017

Giới thiệu cây nhị phân tìm kiếm

Các thao tác trong cây NPTK

Giới thiệu cây nhị phân tìm kiếm

Cây nhị phân tìm kiếm - NPTK (*Binary Search Tree - BST*)

Là một *cây nhị phân* thỏa các điều kiện sau:

- ▶ Khóa của các nút thuộc cây con trái *nhỏ* hơn khóa nút gốc.
- ▶ Khóa của các nút thuộc cây con phải *lớn* hơn khóa nút gốc.
- ▶ Hai cây con trái và phải của nút gốc cũng là một cây NPTK.

Giới thiệu cây nhị phân tìm kiếm

Cây nhị phân tìm kiếm - NPTK (*Binary Search Tree - BST*)

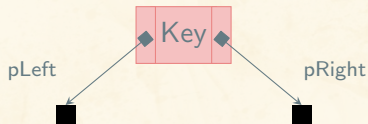
Đặc điểm cây NPTK

- ▶ Dữ liệu lưu trữ có thứ tự, hỗ trợ tìm kiếm tốt hơn danh sách liên kết, ngăn xếp, hàng đợi, ...
- ▶ Nếu tổng số nút trong cây NPTK là n thì chi phí tìm kiếm trung bình $\log_2 n$.

Cách lưu trữ cây NPTK

Cấu trúc dữ liệu của một nút

- ▶ Thành phần dữ liệu: khóa (*key*) của một nút.
- ▶ Thành phần liên kết: con trỏ pLeft liên kết với *cây con trái* và con trỏ pRight liên kết với *cây con phải*.



Cấu trúc dữ liệu của một cây

- ▶ Chỉ cần một con trỏ trỏ đến nút gốc của cây.

Thao tác thêm nút vào cây NPTK

Thuật toán 1: InsertNode(t, k)

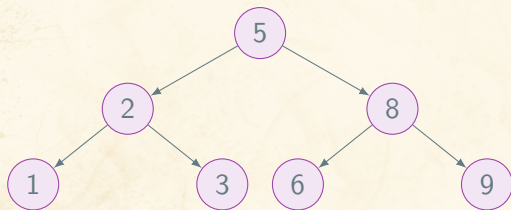
- Đầu vào: cây T và khóa k cần thêm.
- Đầu ra: cây T sau khi thêm k.

```
1  if cây rỗng // TH1. cây rỗng
2      Khởi tạo nút p có khóa k và cập nhật pLeft, pRight
3      pRoot trở đến p
4  else // TH2. cây khác rỗng
5      if pRoot->Key > k
6          Gọi đệ quy hàm InsertNode() với cây con bên trái
7      else if pRoot->Key < k
8          Gọi đệ quy hàm InsertNode() với cây con bên phải
```

Thao tác thêm nút vào cây NPTK

Ví dụ 1

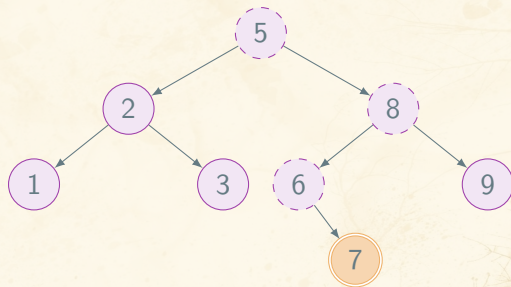
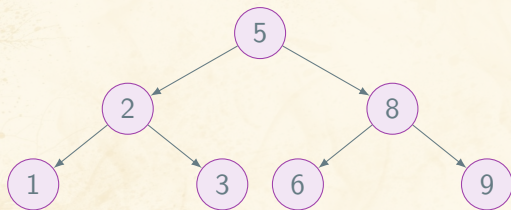
Cho cây NPTK, thêm nút có khóa là 7.



Thao tác thêm nút vào cây NPTK

Ví dụ 1

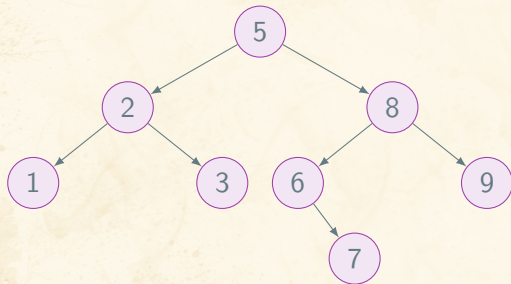
Cho cây NPTK, thêm nút có khóa là 7.



Thao tác thêm nút vào cây NPTK

Ví dụ 2

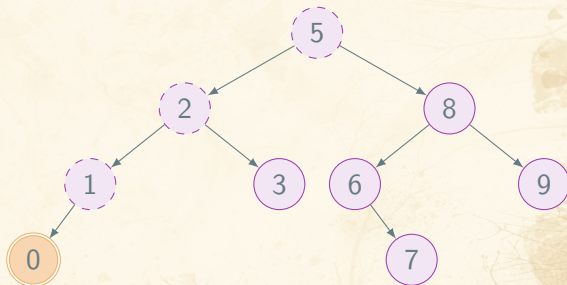
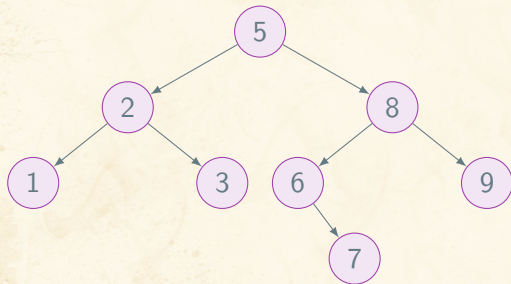
Cho cây NPTK, thêm nút có khóa là 0.



Thao tác thêm nút vào cây NPTK

Ví dụ 2

Cho cây NPTK, thêm nút có khóa là 0.



Thao tác thêm nút vào cây NPTK

```
1 void InsertNode(Node *&pRoot, Data k)
2 {
3     if (pRoot == NULL) // TH1. Cây rỗng
4     {
5         Node *p = new Node();
6         p->Key = k;
7         p->pLeft = NULL;
8         p->pRight = NULL;
9         pRoot = p;
10    }
```

Thao tác thêm nút vào cây NPTK

```
11  ____else // TH2. Cay khac rong
12  ____{
13  _____if (pRoot->Key > k)
14  _____InsertNode(pRoot->pLeft, k);
15  _____else if (pRoot->Key < k)
16  _____InsertNode(pRoot->pRight, k);
17  ____}
18  }
```

Thao tác tìm nút có khóa k trong cây NPTK

Thuật toán 2: SearchNode(t, k)

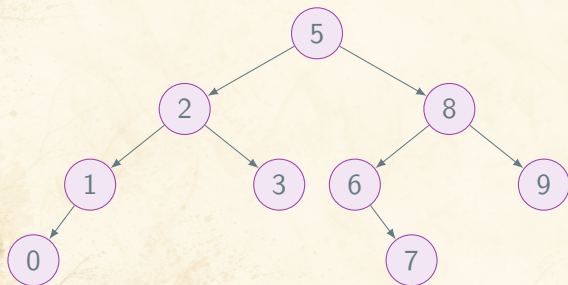
- Đầu vào: cây T và nút k cần tìm.
- Đầu ra: nút có khóa k hay NULL nếu không tìm thấy.

```
1  if cây khác rỗng
2      if pRoot->pKey = k
3          Trả về nút pRoot
4      else if pRoot->pKey > k
5          Gọi đệ quy hàm SearchNode() với cây con bên trái
6      else
7          Gọi đệ quy hàm SearchNode() với cây con bên phải
8      Trả về không tìm thấy
```


Thao tác tìm nút có khóa k trong cây NPTK

Ví dụ 3

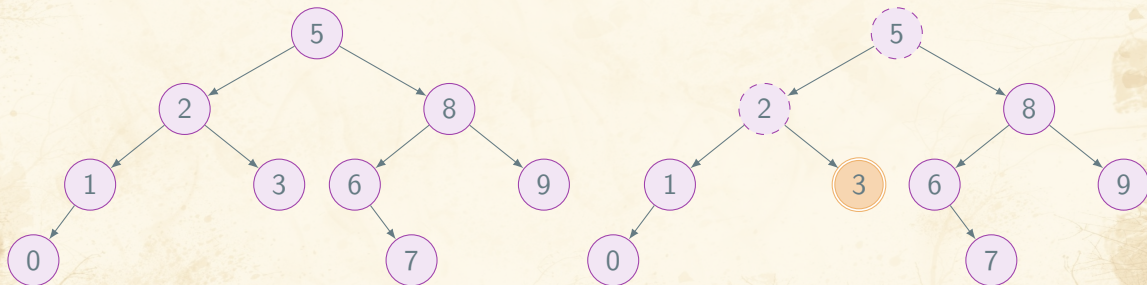
Cho cây NPTK, tìm nút có khóa là 3.



Thao tác tìm nút có khóa k trong cây NPTK

Ví dụ 3

Cho cây NPTK, tìm nút có khóa là 3.



Thao tác tìm nút có khóa k trong cây NPTK

```
1 Node *SearchNode(Node *pRoot, Data k)
2 {
3     if (pRoot != NULL)
4     {
5         if (pRoot->Key == k) // TH1. Tim thay nut co khoa k
6             return pRoot;
7         else if (pRoot->Key > k) // Tim cay con trai
8             SearchNode(pRoot->pLeft, k);
9         else // Tim cay con phai
10            SearchNode(pRoot->pRight, k);
11    }
12    // TH2. Khong tim thay
13    return NULL;
14 }
```

Thao tác xóa một nút trong cây NPTK

Chia 3 trường hợp

- ▶ Trường hợp 1: nút khóa k là *nút lá*.
- ▶ Trường hợp 2: nút khóa k chỉ có *1* cây con trái hay phải.
- ▶ Trường hợp 3: nút khóa k chứa đầy đủ *2* cây con, thực hiện thao tác tìm *nút phải nhất của cây con trái* hay *nút trái nhất của cây con phải*.

Thao tác xóa một nút trong cây NPTK

Tìm nút thay thế là nút phải nhất cây con trái

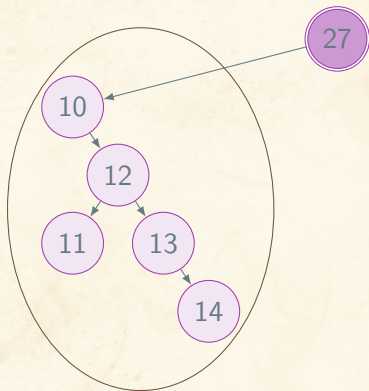
Thuật toán 3: SearchStandFor(t, p)

- Đầu vào: cây T.
- Đầu ra: nút p là nút thay thế (nút phải nhất/lớn nhất của cây T).

```
1 // Tìm theo nhánh bên phải của cây
2 if cây con phải nút đang xét khác rỗng
3     Gọi đệ quy hàm SearchStandFor() với cây con phải
4 else // Tìm phần tử thay thế
5     Chép dữ liệu của pRoot vào nút p ...
6     Lưu lại nhánh con trái (trường hợp nút p có cây con trái)
```

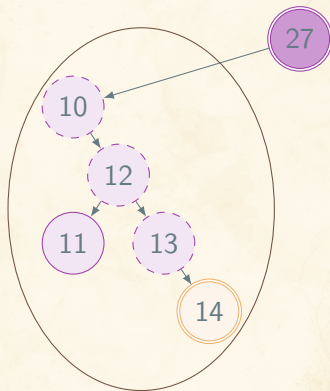
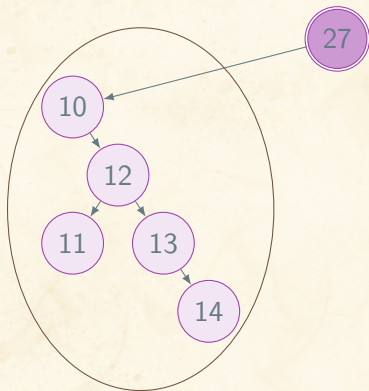

Thao tác xóa một nút trong cây NPTK

duyệt



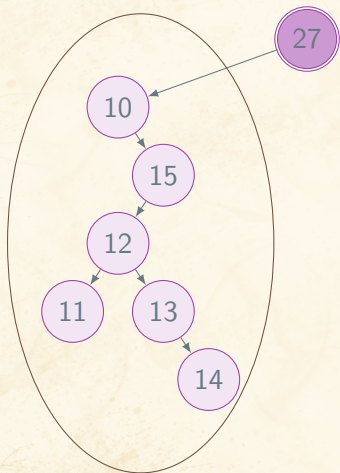
Thao tác xóa một nút trong cây NPTK

duyên



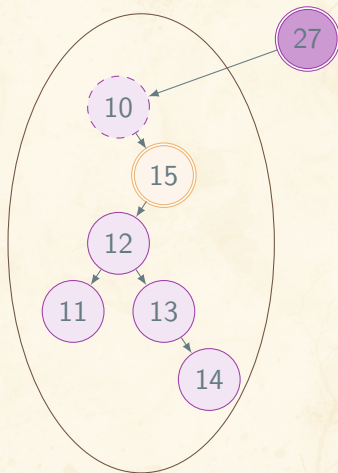
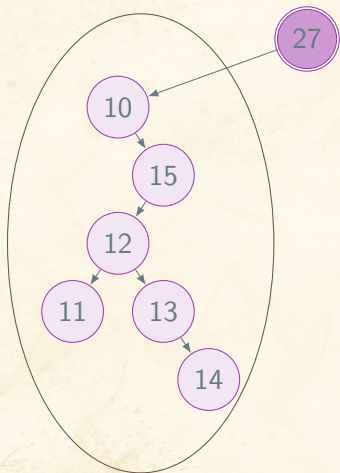
Thao tác xóa một nút trong cây NPTK

duyệt



Thao tác xóa một nút trong cây NPTK

duyên



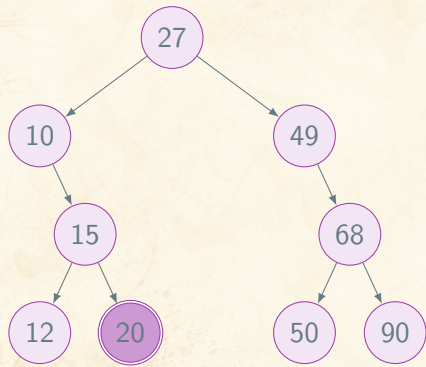
Thao tác xóa một nút trong cây NPTK

Tìm nút thay thế là nút phải nhất cây con trái

```
1 void SearchStandFor(Node *&pRoot, Node *&p)
2 {
3     if (pRoot->pRight != NULL)
4         SearchStandFor(pRoot->pRight, p);
5     else
6     {
7         p->Key = pRoot->Key;
8         p = pRoot;
9         pRoot = pRoot->pLeft;
10    }
11 }
```


Thao tác xóa một nút trong cây NPTK

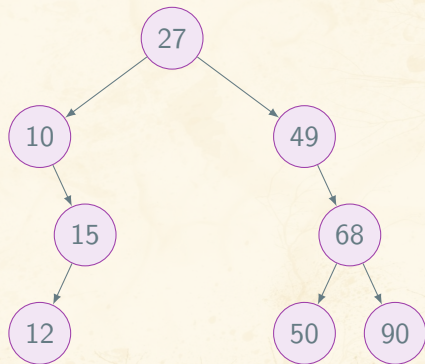
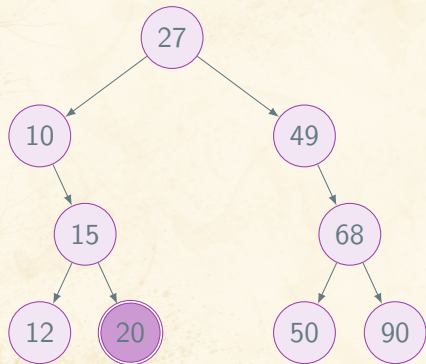
Trường hợp 1: nút k là *nút lá*.



Thao tác xóa một nút trong cây NPTK

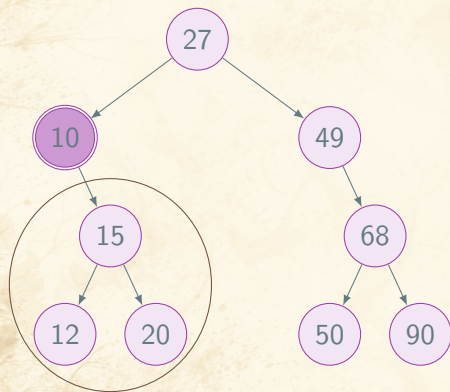
duyệt

Trường hợp 1: nút k là *nút lá*.



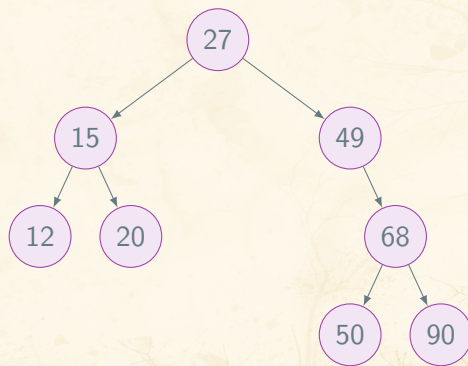
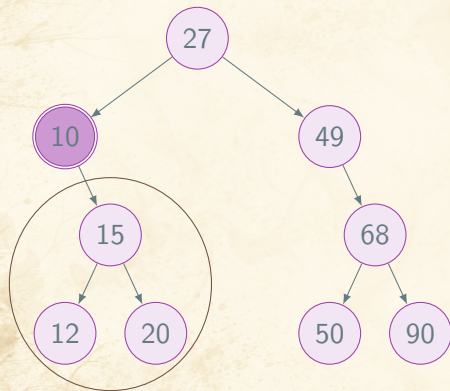
Thao tác xóa một nút trong cây NPTK

Trường hợp 2: nút k chỉ có 1 cây con trái hay phải



Thao tác xóa một nút trong cây NPTK

Trường hợp 2: nút k chỉ có 1 cây con trái hay phải



Thao tác xóa một nút trong cây NPTK

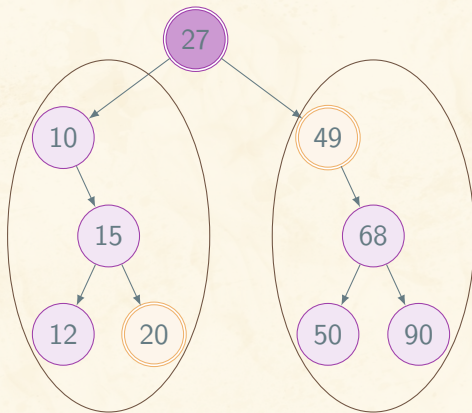
Trường hợp 3: nút k có đầy đủ 2 cây con

Thực hiện thao tác tìm *nút thay thế* trong 2 nút.

- ▶ Nút *phải nhất/lớn nhất* của *cây con trái*
- ▶ Nút *trái nhất/nhỏ nhất* của *cây con phải*.

Thao tác xóa một nút trong cây NPTK

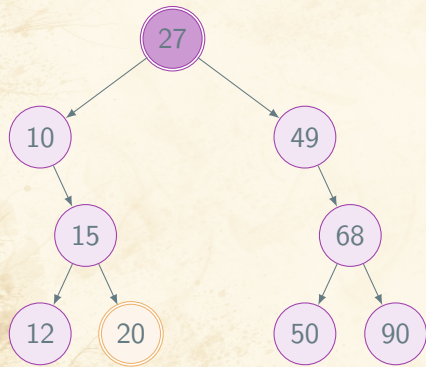
duyetho



Thao tác xóa một nút trong cây NPTK

Nút k có đầy đủ 2 cây con

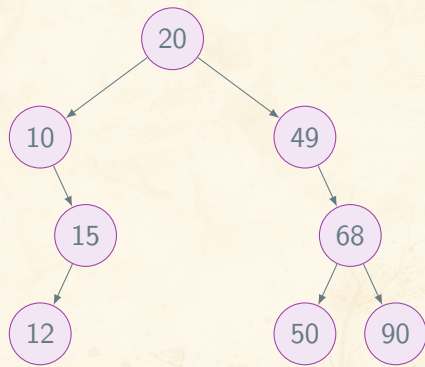
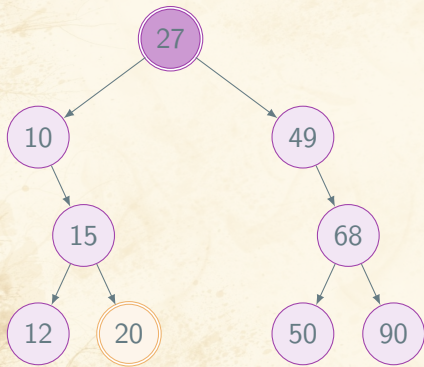
- Cách 1. Thực hiện thao tác tìm *nút phải nhất* của *cây con trái*.



Thao tác xóa một nút trong cây NPTK

Nút k có đầy đủ 2 cây con

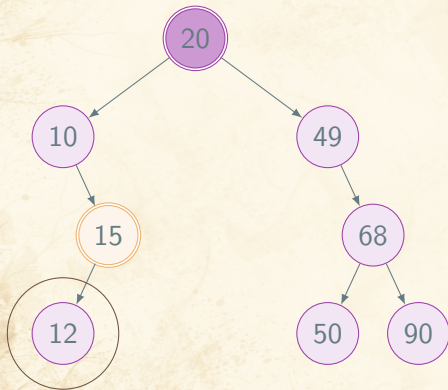
- Cách 1. Thực hiện thao tác tìm *nút phải nhất* của *cây con trái*.



Thao tác xóa một nút trong cây NPTK

Nút k có đầy đủ 2 cây con

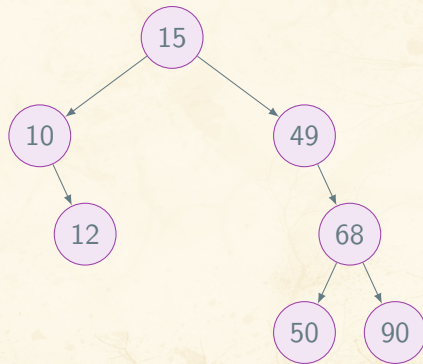
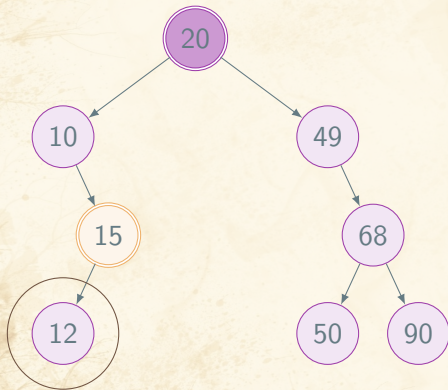
- Cách 1. Thực hiện thao tác tìm *nút phải nhất* của *cây con trái*.



Thao tác xóa một nút trong cây NPTK

Nút k có đầy đủ 2 cây con

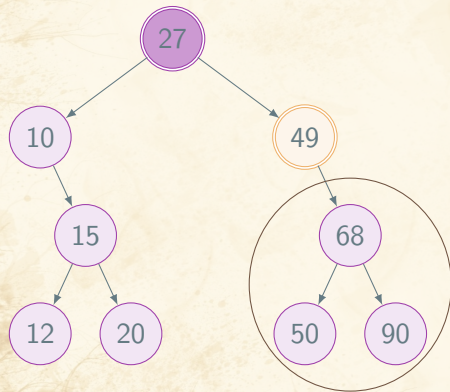
- Cách 1. Thực hiện thao tác tìm *nút phải nhất* của *cây con trái*.



Thao tác xóa một nút trong cây NPTK

Nút k có đầy đủ 2 cây con

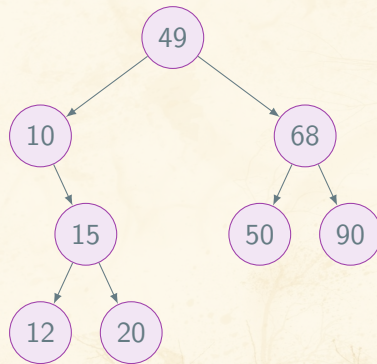
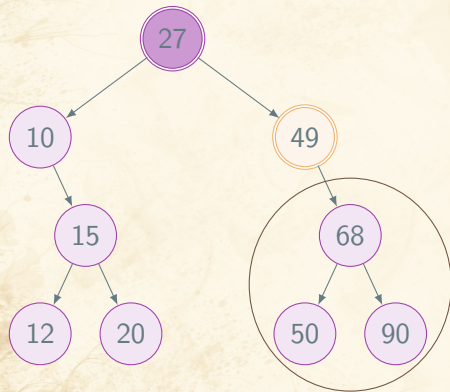
- Cách 2. Thực hiện thao tác tìm *nút trái nhất* của *cây con phải*.



Thao tác xóa một nút trong cây NPTK

Nút k có đầy đủ 2 cây con

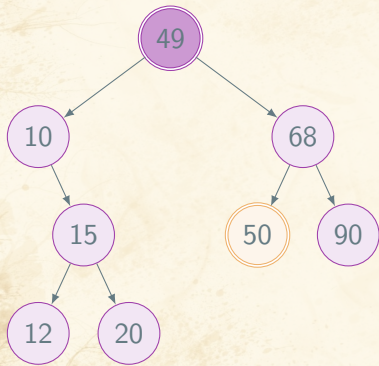
- Cách 2. Thực hiện thao tác tìm *nút trái nhất* của *cây con phải*.



Thao tác xóa một nút trong cây NPTK

Nút k có đầy đủ 2 cây con

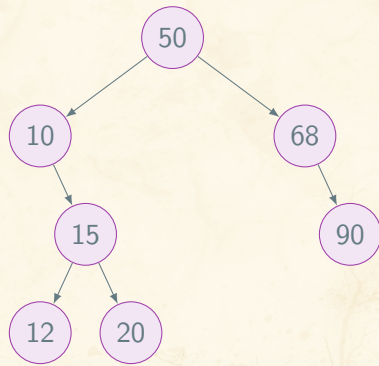
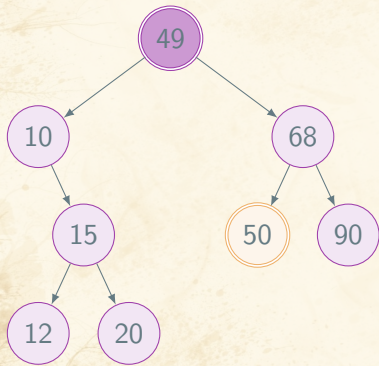
- Cách 2. Thực hiện thao tác tìm *nút trái nhất* của *cây con phải*.



Thao tác xóa một nút trong cây NPTK

Nút k có đầy đủ 2 cây con

- Cách 2. Thực hiện thao tác tìm *nút trái nhất* của *cây con phải*.



Thao tác xóa một nút trong cây NPTK

Thuật toán 4: RemoveNode(t, k)

- Đầu vào: cây T và nút có khóa k cần xóa.
- Đầu ra: cây T sau khi xóa nút có khóa k.

```
1  if cây rỗng // TH1. Không tìm thấy khóa k
2  ____Dừng thuật toán
3  if pRoot->Key > k
4  ____Gọi đệ quy hàm RemoveNode() với cây con trái
5  else if pRoot->Key < k
6  ____Gọi đệ quy hàm RemoveNode() với cây con phải
7  // TH2. Tìm thấy nút pRoot có khóa k
```


Thao tác xóa một nút trong cây NPTK

```
8  else
9  ____// Xóa nút p tương ứng 3 trường hợp
10 ____// TH2.1. nút là
11 ____Khai báo nút p trở đến pRoot
12 ____// TH2.2. nút có 1 cây con trái/phải
13 ____if p chỉ có 1 cây con trái // nút có 1 cây con trái
14 ____pRoot trở đến cây con trái của p
15 ____else if p chỉ có 1 cây con phải // nút có 1 cây con phải
16 ____pRoot trở đến cây con phải của p
17 ____else // TH2.3. p có 2 cây con
18 ____Gọi hàm tìm nút thay thế trước khi xóa nút có khóa k ...
19 ____Xóa nút p
```

Thao tác xóa một nút trong cây NPTK

```
1 void RemoveNode(Node *&pRoot, Data k)
2 {
3     Node *p = new Node();
4     if (pRoot == NULL) // TH1. Không tìm thấy nút có khóa
5         k
6         return;
7     if (pRoot->Key > k)
8         RemoveNode(pRoot->pLeft, k);
9     else if (pRoot->Key < k)
10        RemoveNode(pRoot->pRight, k);
```

Thao tác xóa một nút trong cây NPTK

```
10  ____else // Tim thay nut co khoa k
11  ____{
12  ____    ____// TH2.1. nut la
13  ____    ____p = pRoot;
14  ____    ____// TH2.2 nut co 1 cay con trai/phai
15  ____    ____if (p->pRight == NULL) // nut co 1 cay con trai
16  ____    ____    ____pRoot = p->pLeft;
17  ____    ____else if (p->pLeft == NULL) // nut co 1 cay con phai
18  ____    ____    ____pRoot = p->pRight;
19  ____    ____// TH2.3. nut co 2 cay con
20  ____    ____else
21  ____    ____    ____SearchStandFor(pRoot->pLeft , p);
22  ____    ____delete p;
23  ____    ____}
24  ____}
```

1. Cho dãy gồm 9 phần tử: 5, 9, 3, 1, 8, 7, 4, 6, 2.
 - ▶ Lần lượt thêm các phần tử trên vào cây NPTK.
 - ▶ In cây nhị phân tìm kiếm theo 3 phương pháp duyệt cây: NLR, LNR, LRN.
 - ▶ Thêm vào phần tử 10.
 - ▶ Xóa phần tử 5.
2. Khử đệ quy các thuật toán duyệt cây NLR, LNR, LRN (*sử dụng phương pháp lặp*).
3. Viết thuật toán tìm kiếm phần tử nhỏ nhất/lớn nhất trong cây NPTK.

Tài liệu tham khảo



Dương Anh Đức, Trần Hạnh Nhi.

Nhập môn Cấu trúc dữ liệu và Thuật toán.

Đại học Khoa học tự nhiên TP Hồ Chí Minh, 2003.



Donald E. Knuth.

The Art of Computer Programming, Volume 3.

Addison-Wesley, 1998.



Niklaus Wirth.

Algorithms + Data Structures = Programs.

Prentice-Hall, 1976.



Robert Sedgewick.

Algorithms in C.

Addison-Wesley, 1990.