Chương 5
BÀI TẬP CẤU TRÚC DỮ LIỆU CÂY NHỊ PHÂN

TS. Nguyễn Tấn Trần Minh Khang

ThS. Cáp Phạm đình Thăng

BÀI 001

- Cho một cây nhị phân có nút gốc là Root, mỗi nút trong cây chứa một số nguyên:
  a) Viết hàm tính trung bình cộng các nút trong cây.
  b) Viết hàm tính trung bình cộng các số dương trong cây.
  c) Viết hàm tính trung bình cộng các số âm trong cây.
  d) Viết chương trình tính tính tỉ số R=a/b. Với a là tổng các nút có giá trị dương, b là tổng các nút có giá trị âm.

BÀI 001

- Cấu trúc dữ liệu

```
1. struct node
2. {
3.     int info;
4.     struct node*pLeft;
5.     struct node*pRight;
6. };
7. typedef struct node NODE;
8. typedef NODE*TREE;
```

BÀI 001

- Viết chương trình tính trung bình cộng các nút trong cây.

BÀI 001

- Viết chương trình tính trung bình cộng các nút trong cây.

```
1. int DemNode(TREE Root)
2. {
3.     if(Root==NULL)
4.         return 0;
5.     int a=DemNode(Root->pLeft);
6.     int b=DemNode(Root->pRight);
7.     return (a+b+1);
8. }
```

BÀI 001

- Viết chương trình tính trung bình cộng các nút trong cây.

```
1. int TongNode(TREE Root)
2. {
3.     if(Root==NULL)
4.         return 0;
5.     int a=TongNode(Root->pLeft);
6.     int b=TongNode(Root->pRight);
7.     return (a+b+Root->info);
8. }
```

BÀI 001

- Viết chương trình tính trung bình cộng các nút trong cây.

```
1. float TrungBinhCong(TREE Root)
2. {
3.     int s = TongNode(Root);
4.     int dem = DemNode(Root);
5.     if(dem==0)
6.         return 0;
7.     return (float)s/dem;
8. }
```

BÀI 001

- Viết chương trình tính trung bình cộng các nút dương trong cây.

BÀI 001

- Viết chương trình tính trung bình cộng các nút dương trong cây.

```
11. int DemDuong(TREE Root)
12. {
13.     if(Root==NULL)
14.         return 0;
15.     int a=DemDuong(Root->pLeft);
16.     int b=DemDuong(Root->pRight);
17.     if(Root->info>0)
18.         return (a+b+1);
19.     return (a+b);
20. }
```

BÀI 001

- Viết chương trình tính trung bình cộng các nút dương trong cây.

```
11. int TongDuong(TREE Root)
12. {
13.   if(Root==NULL)
14.     return 0;
15.   int a=TongDuong(Root->pLeft);
16.   int b=TongDuong(Root->pRight);
17.   if(Root->info>0)
18.     return (a+b+Root->info);
19.   return (a+b);
20. }
```

BÀI 001

- Viết chương trình tính trung bình cộng các nút dương trong cây.

```
1. float TrungBinhDuong(TREE Root)
2. {
3.     int s = TongDuong(Root);
4.     int dem=DemDuong(Root);
5.     if(dem==0)
6.         return 0;
7.     return (float)s/dem;
8. }
```

BÀI 001

- Viết chương trình tính trung bình cộng các nút âm trong cây.

BÀI 001

- Viết chương trình tính trung bình cộng các nút âm trong cây.

```
11. int DemAm(TREE Root)
12. {
13.     if(Root==NULL)
14.         return 0;
15.     int a=DemAm(Root->pLeft);
16.     int b=DemAm(Root->pRight);
17.     if(Root->info<0)
18.         return (a+b+1);
19.     return (a+b);
20. }
```

BÀI 001

- Viết chương trình tính trung bình cộng các nút âm trong cây.

```
11. int TongAm(TREE Root)
12. {
13.     if(Root==NULL)
14.         return 0;
15.     int a=TongAm(Root->pLeft);
16.     int b=TongAm(Root->pRight);
17.     if(Root->info<0)
18.         return (a+b+Root->info);
19.     return (a+b);
20. }
```

BÀI 001

- Viết chương trình tính trung bình cộng các nút âm trong cây.

```
1. float TrungBinhCongAm(TREE Root)
2. {
3.     int s = TongAm(Root);
4.     int dem = DemAm(Root);
5.     if(dem==0)
6.         return 0;
7.     return (float)s/dem;
8. }
```

BÀI 001

- Viết chương trình tính tính tỉ số R=a/b. Với a là tổng các nút có giá trị dương, b là tổng các nút có giá trị âm.

BÀI 001

- Viết chương trình tính tính tỉ số R=a/b. Với a là tổng các nút có giá trị dương, b là tổng các nút có giá trị âm.

```
11. int TongAm(TREE Root)
12. {
13.     if(Root==NULL)
14.         return 0;
15.     int a=TongAm(Root->pLeft);
16.     int b=TongAm(Root->pRight);
17.     if(Root->info<0)
18.         return (a+b+Root->info);
19.     return (a+b+Root);
20. }
```

BÀI 001

- Viết chương trình tính tính tỉ số R=a/b. Với a là tổng các nút có giá trị dương, b là tổng các nút có giá trị âm.

```
11. int TongDuong(TREE Root)
12. {
13.     if(Root==NULL)
14.         return 0;
15.     int a=TongDuong(Root->pLeft);
16.     int b=TongDuong(Root->pRight);
17.     if(Root->info>0)
18.         return (a+b+Root->info);
19.     return (a+b+Root);
20. }
```

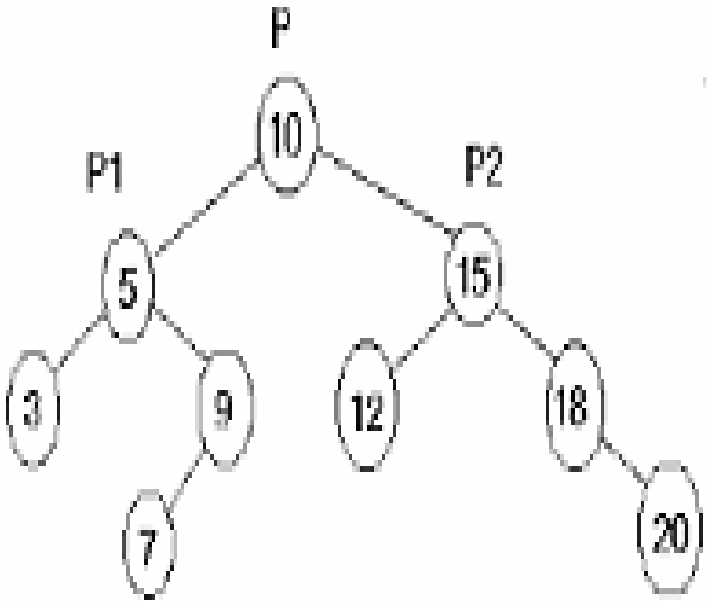
BÀI 001

- Viết chương trình tính tính tỉ số R=a/b. Với a là tổng các nút có giá trị dương, b là tổng các nút có giá trị âm.

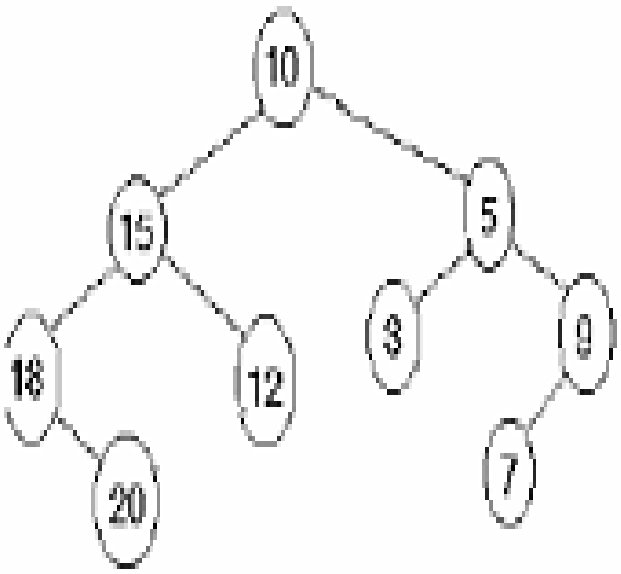
```
1. float TinhTySo(TREE Root)
2. {
3.     int a = TongDuong(Root);
4.     int b = TongAm(Root);
5.     if(b==0)
6.         return 0;
7.     return (float)a/b;
8. }
```

BÀI 002

- Cho cây như hình 1, cho trước nút p



Hình 1



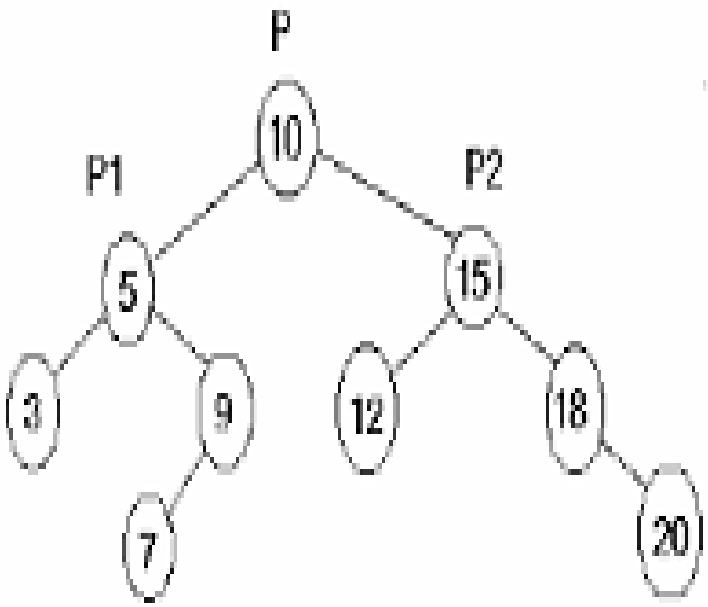
Hình 2

- Hãy viết các câu lệnh cần thiết để chuyển cây sang dạng biểu diễn của hình 2

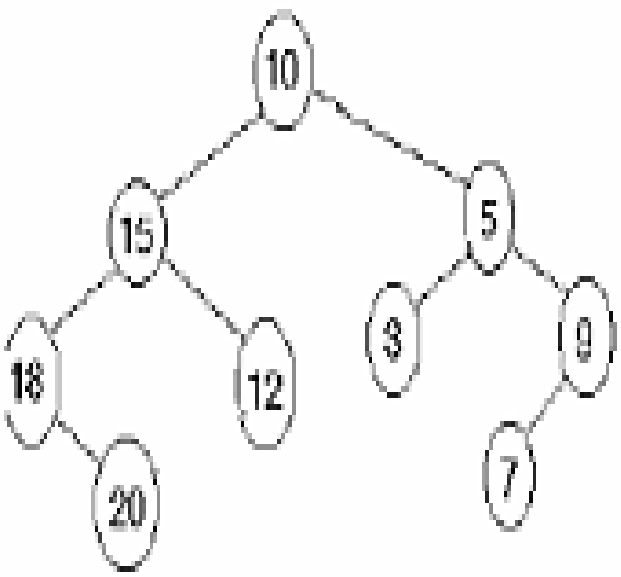
BÀI 002

- **Các câu lệnh cần thiết.**

```
1. NODE *temp=p->pLeft;
2. p->pLeft = p->pRight;
3. p->pRight=temp;
4. temp = p->pLeft->pLeft;
5. p->pLeft->pLeft=p->pLeft->pRight;
6. p->pLeft->pRight=temp;
```



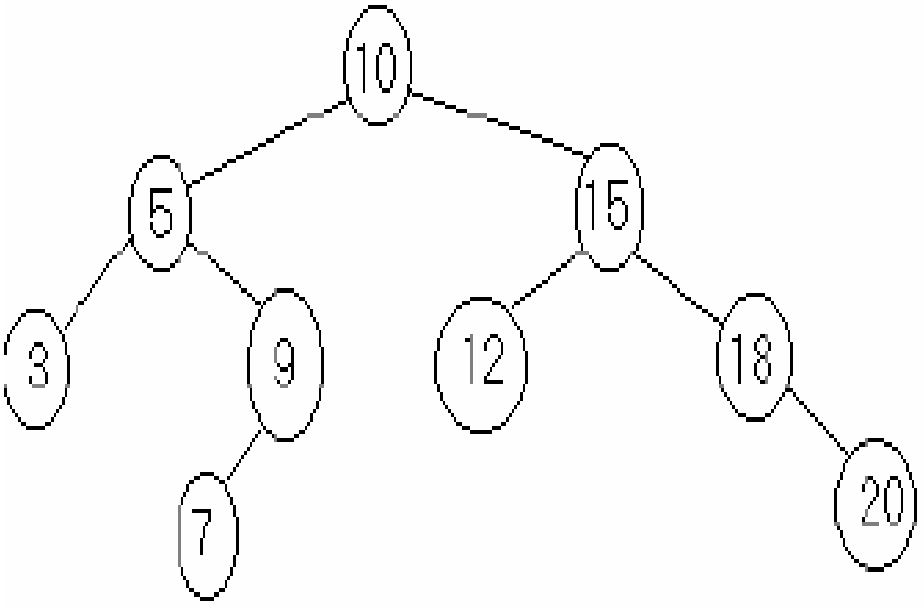
Hình 1



Hình 2

BÀI 003

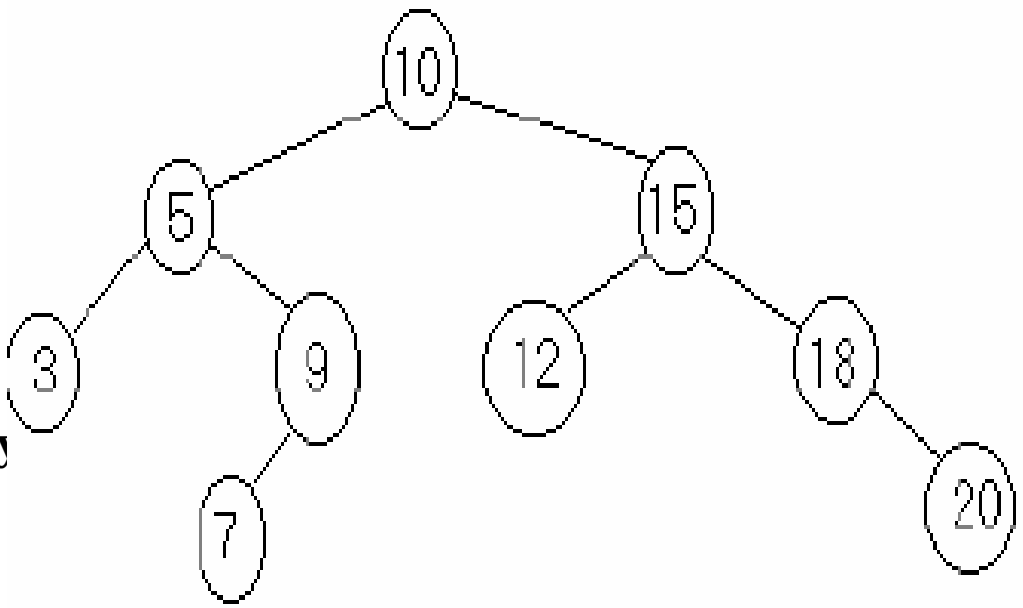
- Cho cây nhị phân tìm kiếm như hình vẽ. Hãy cho biết thứ tự các nút thêm vào cây sao cho để có được cấu trúc này? (Giả sử lúc đầu cây rỗng).



- Nếu kết quả của phép duyệt cây trên là: 3, 7, 9, 5, 12, 20, 18, 15, 10. Hãy cho biết người ta đã áp dụng phép duyệt nào?

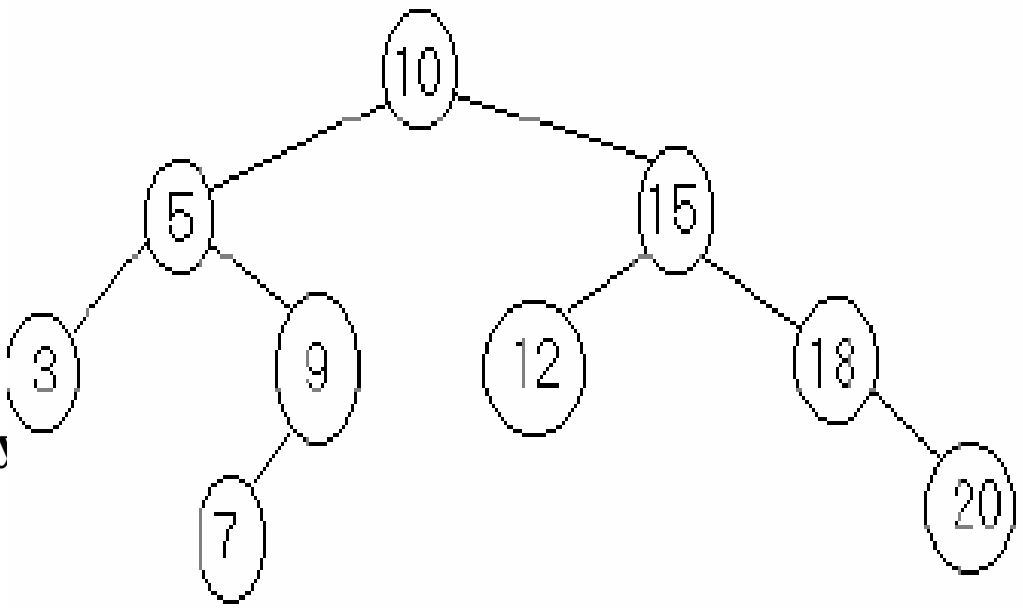
BÀI 003

- Thứ tự các nút thêm vào cây để có được cây cấu trúc như trên là:
- Cách 01: 10, 5, 15, 3, 9, 12, 18, 7, 20.
- Cách 02: 10, 5, 3, 9, 7, 15, 12, 18, 20.
- Cách 03: 10, 15, 18, 20, 12, 5, 9, 7, 3.



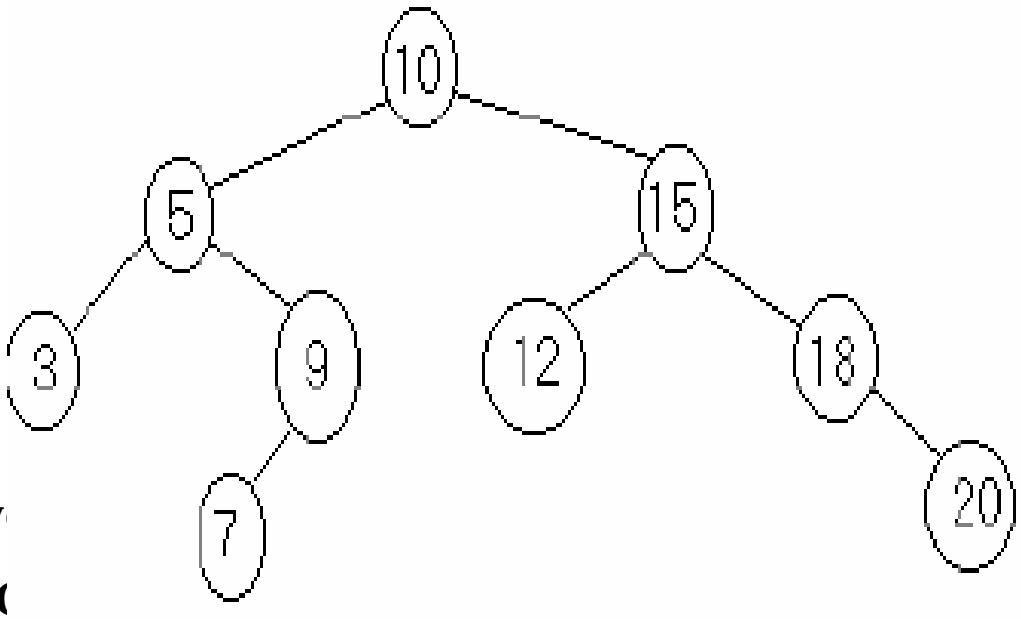
BÀI 003

- Nếu kết quả của phép duyệt cây trên là: 3, 7, 9, 5, 12, 20, 18, 15, 10. Hãy cho biết người ta đã áp dụng phép duyệt nào?
- Đó là phép duyệt cây theo phương pháp: LRN.



BÀI 004

- Cho một cây nhị phân tìm kiếm với nút gốc là Root, giá trị lưu trữ tại mỗi nút là một số nguyên (int). Hãy viết hàm tìm phần tử nhỏ nhất và lớn nhất trong cây.



BÀI 004

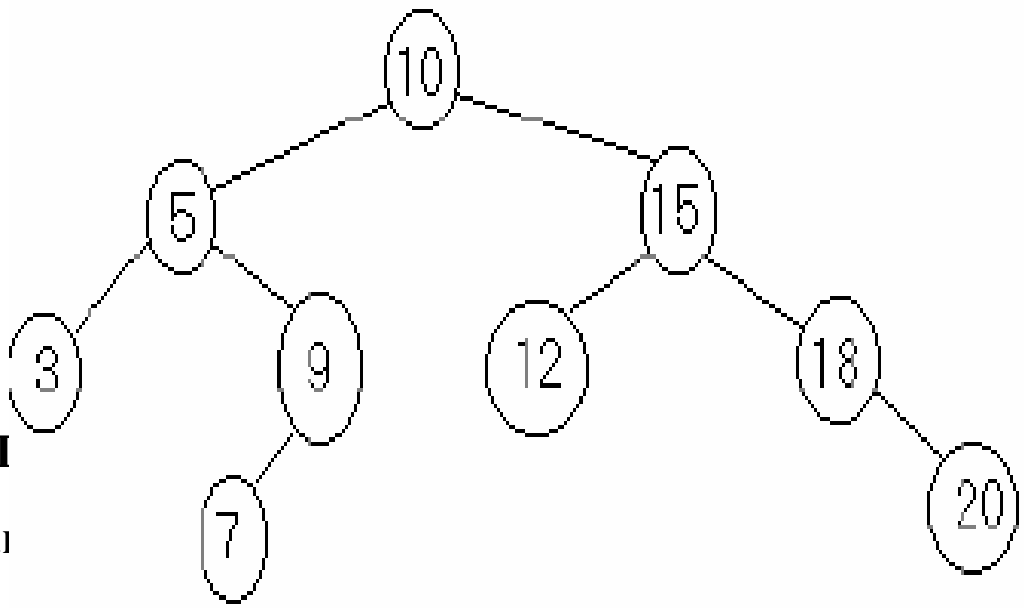
- Cấu trúc dữ liệu

```
1. struct node
2. {
3.     int info;
4.     struct node*pLeft;
5.     struct node*pRight;
6. };
7. typedef struct node NODE;
8. typedef NODE*TREE;
```

BÀI 004

- Định nghĩa hàm

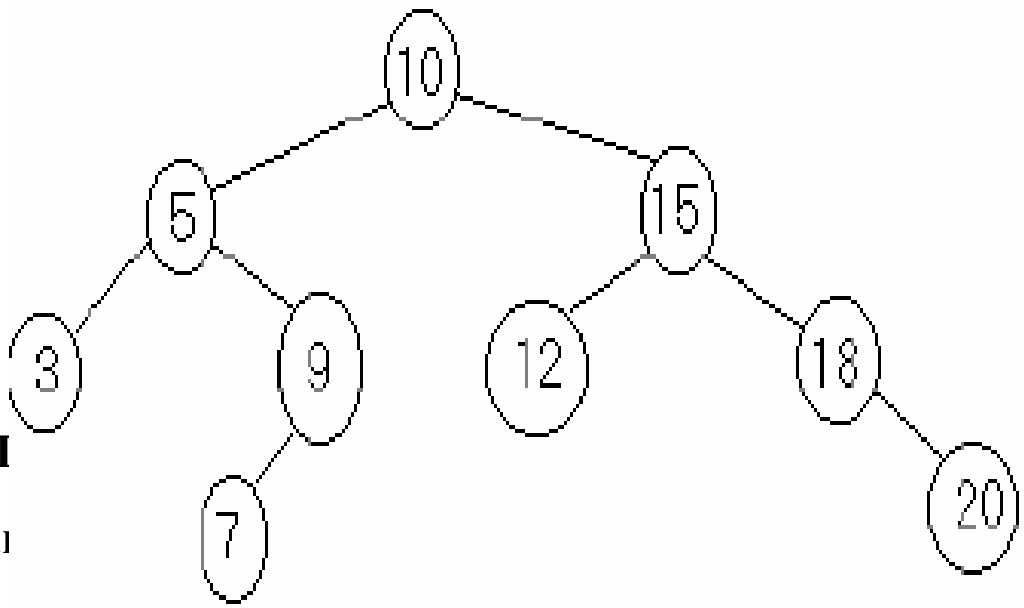
```
1. NODE* NhoNhat(TREE Root)
2. {
3.     if(Root==NULL)
4.         return NULL;
5.     NODE*lc = Root;
6.     while(lc->pLeft)
7.         lc = lc->pLeft;
8.     return lc;
9. }
```



BÀI 004

- Định nghĩa hàm

```
1. NODE* LonNhat(TREE Root)
2. {
3.     if(Root==NULL)
4.         return NULL;
5.     NODE*lc = Root;
6.     while(lc->pRight)
7.         lc = lc->pRight;
8.     return lc;
9. }
```



BÀI 005

- Cho một cây nhị phân có cấu trúc nút là NODE hãy:
  - Viết hàm để tính tổng số nút có một nhánh con (con trái HAY con phải) bằng cách dùng thuật toán duyệt nút gốc giữa NLR;
  - Thiết lập một công thức đệ quy để thực hiện yêu cầu của câu trên. Cài đặt công thức này thành hàm.

BÀI 005

- Câu b: Số lượng nút một con trong cây nhị phân bằng 1 cộng số lượng nút một con trong cây nhị phân con trái cộng số lượng nút một con trong cây nhị phân con phải nếu nút gốc có một con. Ngược lại số lượng nút một con trong cây nhị phân bằng số lượng nút một con trong cây nhị phân con trái cộng số lượng nút một con trong cây nhị phân con phải.

BÀI 005

```
1. int DemMotCon(TREE t)
2. {
3.     if(t==NULL)
4.         return 0;
5.     if((t->pLeft&&!t->pRight)||
           (!t->pLeft&&t->pRight))
6.         return 1+
               DemMotCon(t->pLeft)+
               DemMotCon(t->pRight);
7.     return DemMotCon(t->pLeft)+
               DemMotCon(t->pRight);
8. }
```

BÀI 006

- Hãy phát biểu công thức đệ quy để tính các giá trị sau đây:
  - Số nút trong cây nhị phân tìm kiếm.
  - Tổng giá trị các nút trong cây (giả sử mỗi phần tử là một số nguyên).
- Cài đặt thành hàm các công thức đã nêu ở trên.

BÀI 006

- Số nút trong cây nhị phân tìm kiếm bằng số nút trong cây nhị phân tìm kiếm trong cây con trái cộng số nút trong cây nhị phân tìm kiếm trong cây con phải cộng 1.

BÀI 006

- Tổng giá trị các nút trong cây tìm kiếm bằng tổng giá trị các nút trong cây nhị phân tìm kiếm con trái cộng tổng giá trị các nút trong cây nhị phân tìm kiếm con phải cộng giá trị tại nút gốc.

BÀI 006

- Cấu trúc dữ liệu

```
1. struct node
2. {
3.     int info;
4.     struct node*pLeft;
5.     struct node*pRight;
6. };
7. typedef struct node NODE;
8. typedef NODE*TREE;
```

BÀI 006

```
1. int DemNode(TREE t)
2. {
3.     if(t==NULL)
4.         return 0;
5.     int a=DemNode(t->pLeft);
6.     int b=DemNode(t->pRight);
7.     return (a+b+1);
8. }
```

BÀI 006

```
1. int TongNode(TREE t)
2. {
3.     if(t==NULL)
4.         return 0;
5.     int a=TongNode(t->pLeft);
6.     int b=TongNode(t->pRight);
7.     return (a + b + t->info);
8. }
```

BÀI 007

- Hãy mô tả ngắn gọn sự giống và khác nhau giữa hai cấu trúc Cây Nhị Phân tìm kiếm và Danh Sách Liên Kết Đơn.

BÀI 007

- Giống nhau:
  - CTDL động.
  - Các thao tác cơ bản Thêm, Xóa, Cập Nhật được thực hiện một cách linh hoạt.
- Khác nhau
  - Dữ liệu trên cây NPTK được tổ chức và dslk đơn thì không.
  - Chi phí tìm kiếm, thêm trên cây nhanh hơn trên dslk đơn.

BÀI 008

- Định nghĩa hàm duyệt và xuất cây nhị phân các số thực ra tập tin nhị phân data.out theo phương pháp LNR.

BÀI 008

- Cấu trúc dữ liệu

```
1. struct node
2. {
3.     float info;
4.     struct node*pLeft;
5.     struct node*pRight;
6. };
7. typedef struct node NODE;
8. typedef NODE*TREE;
```

BÀI 008

- Định nghĩa hàm

```
1. int Xuat(char *filename,
            TREE t)
2. {
3.     FILE*fp=fopen(filename,
                     "wb");
4.     if(fp==NULL)
5.         return 0;
6.     LNR(t,fp);
7.     fclose(fp);
8.     return 1;
9. }
```

BÀI 008

```
1. void LNR(TREE t,FILE*fp)
2. {
3.     if(t==NULL)
4.         return;
5.     LNR(t->pLeft,fp);
6.     fwrite(&t->info,
           sizeof(float),1,fp);
7.     LNR(t->pRight,fp);
8. }
```

BÀI 009

- Định nghĩa hàm duyệt và xuất cây nhị phân các số thực ra tập tin nhị phân data.out theo phương pháp NLR.

BÀI 009

- Cấu trúc dữ liệu

```
1. struct node
2. {
3.     float info;
4.     struct node*pLeft;
5.     struct node*pRight;
6. };
7. typedef struct node NODE;
8. typedef NODE*TREE;
```

BÀI 009

- Định nghĩa hàm

```
1. int Xuat(char *filename,
           TREE t)
2. {
3.     FILE*fp=fopen(filename,
                     "wb");
4.     if(fp==NULL)
5.         return 0;
6.     NLR(t,fp);
7.     fclose(fp);
8.     return 1;
9. }
```

BÀI 009

```
1. void NLR(TREE t,FILE*fp)
2. {
3.     if(t==NULL)
4.         return;
5.     fwrite(&t->info,
           sizeof(float),1,fp);
6.     NLR(t->pLeft,fp);
7.     NLR(t->pRight,fp);
8. }
```

BÀI 010

- Định nghĩa hàm duyệt và xuất cây nhị phân các số thực ra tập tin nhị phân data.out theo phương pháp LRN.

BÀI 010

- Cấu trúc dữ liệu

```
1. struct node
2. {
3.     float info;
4.     struct node*pLeft;
5.     struct node*pRight;
6. };
7. typedef struct node NODE;
8. typedef NODE*TREE;
```

BÀI 010

- Định nghĩa hàm

```
1. int Xuat(char *filename,
           TREE t)
2. {
3.     FILE*fp=fopen(filename,
                     "wb");
4.     if(fp==NULL)
5.         return 0;
6.     LRN(t,fp);
7.     fclose(fp);
8.     return 1;
9. }
```

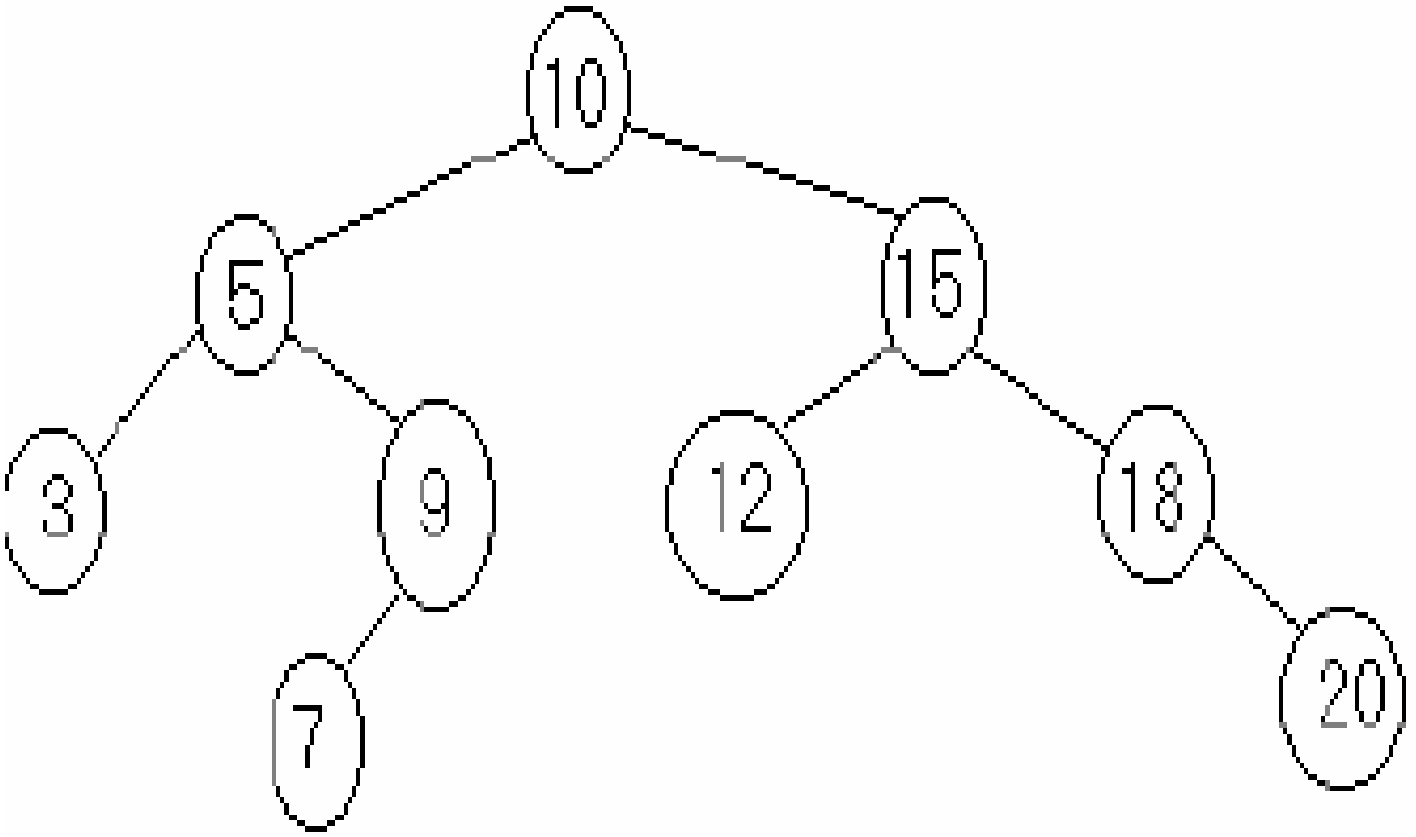
BÀI 010

```
1. void LRN(TREE t,FILE*fp)
2. {
3.     if(t==NULL)
4.         return;
5.     LRN(t->pLeft,fp);
6.     LRN(t->pRight,fp);
7.     fwrite(&t->info,
           sizeof(float),1,fp);
8. }
```

BÀI 011

- Cho cây nhị phân tìm kiếm các số nguyên t. Hãy cho biết cách thức duyệt cây như thế nào để ta được thứ tự các giá trị tăng dần.

BÀI 011



Duyệt cây theo phương pháp LNR ta sẽ được các giá trị tăng dần.

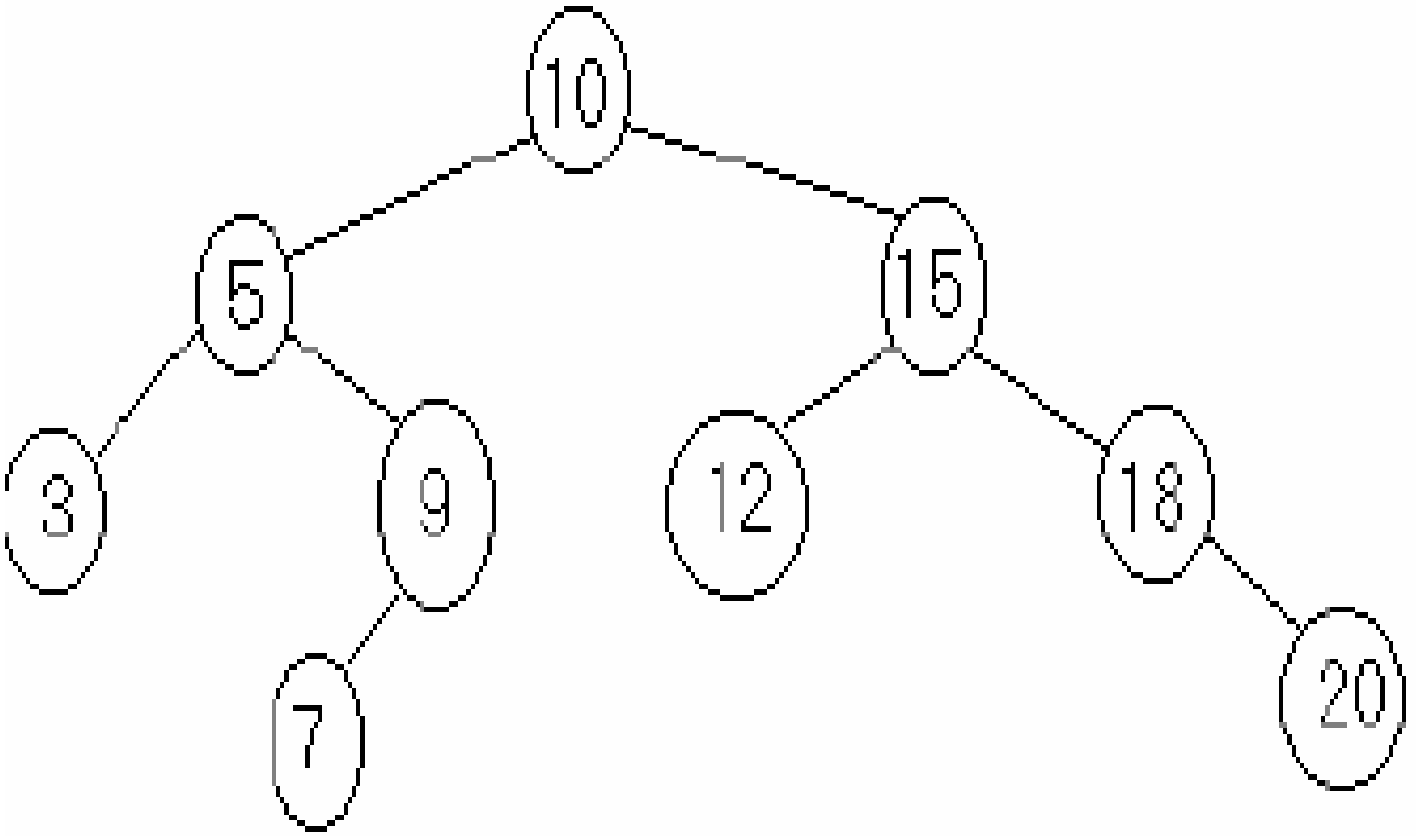
TS. Nguyễn Tấn Trần Minh Khang

ThS. Cáp Phạm đình Thăng

BÀI 012

- Cho cây nhị phân tìm kiếm các số nguyên t. Hãy cho biết cách thức duyệt cây như thế nào để ta được thứ tự các giá trị giảm dần.

BÀI 012



Duyệt cây theo phương pháp RNL ta sẽ được các giá trị giảm dần.

TS. Nguyễn Tấn Trần Minh Khang

ThS. Cáp Phạm đình Thăng

BÀI 013

- Định nghĩa hàm lưu cây nhị phân tìm kiếm các số thực xuống tập tin nhị phân sao cho khi đọc dữ liệu từ file ta có thể tạo lại được cây ban đầu.

BÀI 013

- Cấu trúc dữ liệu

```
1. struct node
2. {
3.     float info;
4.     struct node*pLeft;
5.     struct node*pRight;
6. };
7. typedef struct node NODE;
8. typedef NODE*TREE;
```

BÀI 013

- Định nghĩa hàm

```
1. int Xuat(char *filename,
            TREE t)
2. {
3.     FILE*fp=fopen(filename,
                     "wb");
4.     if(fp==NULL)
5.         return 0;
6.     NLR(t,fp);
7.     fclose(fp);
8.     return 1;
9. }
```

BÀI 013

```
1. void NLR(TREE t,FILE*fp)
2. {
3.     if(t==NULL)
4.         return;
5.     fwrite(&t->info,
           sizeof(float),1,fp);
6.     NLR(t->pLeft,fp);
7.     NLR(t->pRight,fp);
8. }
```

BÀI 014

- Hãy viết một hàm tạo danh sách liên kết đơn từ cây nhị phân tìm kiếm sao cho giá trị các phần tử trong danh sách có thứ tự giảm dần. Biết nút gốc của cây là Root.

BÀI 014

- Cấu trúc dữ liệu

```
1. struct nodetree
2. {
3.     int info;
4.     struct nodetree*pLeft;
5.     struct nodetree*pRight;
6. };
7. typedef struct nodetree
                    NODETREE;
8. typdedef NODETREE*TREE;
```

BÀI 014

```
11. struct nodelist
12. {
13.     int info;
14.     struct nodelist*pNext;
15. };
16. typedef struct nodelist
                    NODELIST;
17. struct list
18. {
19.     NODELIST * pHead;
20.     NODELIST * pTail;
21. };
22. typedef struct list LIST;
```

BÀI 014

- Định nghĩa hàm:

```
11. void Init(LIST &ℓ)
12. {
13.     ℓ.pHead = ℓ.pTail = NULL;
14. }
15. NODELIST*GetNode(int x)
16. {
17.     NODELIST*p = new NODELIST;
18.     if(p==NULL)
19.         return NULL;
20.     p->info = x;
21.     p->pNext = NULL;
22.     return p;
23. }
```

BÀI 014

- Định nghĩa hàm:

```
11. void AddTail(LIST&ℓ,NODELIST*p)
12. {
13.     if(ℓ.pHead == NULL)
14.         ℓ.pHead = ℓ.pTail = p;
15.     else
16.     {
17.         ℓ.pTail->pNext = p;
18.         ℓ.pTail = p;
19.     }
20. }
```

BÀI 014

- Định nghĩa hàm:

```
1. void BuildList(TREE Root,
                  LIST&ℓ)
2. {
3.     Init(ℓ);
4.     RNL(Root,ℓ);
5. }
```

BÀI 014

- Định nghĩa hàm:

```
11. void RNL(TREE Root,LIST&ℓ)
12. {
13.     if(Root==NULL)
14.         return;
15.     RNL(Root->pRight,ℓ);
16.     NODELIST*p=GetNode
                    (Root->info);
17.     if(p!=NULL)
18.         AddTail(ℓ,p);
19.     RNL(Root->pLeft,ℓ);
20. }
```

BÀI 15

- Giữa cấu trúc cây nhị phân tìm kiếm và cấu trúc mảng các phần tử được sắp thứ tự tăng dần có những điểm giống và khác nhau như thế nào.

BÀI 15

- Giống nhau
  - Dữ liệu được tổ chức.
  - Chi phí tìm kiếm một phần tử trên cả hai ctdl là như nhau.
- Khác nhau
  - Chi phí thêm và xoá phần tử vào mảng lớn hơn chi phí cây nhị phân tìm kiếm.

BÀI 16

- Cho một cây nhị phân tìm kiếm t có cấu trúc nút là BST_NODE được khai báo như sau:

```
11. struct BST_NODE{
12.     int Key;      // Khóa của nút
13.     int So_lan; // Số lần xuất hiện.
14.     struct BST_NODE*Left,*Right;
15. };
16. struct BST_TREE
17. {
18.     struct BST_NODE *pRoot;
                  // Nút gốc của cây.
19. };
20. struct BST_TREE t; // Cây t
```

BÀI 16

- Hãy viết hàm thực hiện thao tác xoá phần tử có khoá X. Cách xoá như sau:
  - Nếu phần tử X có tồn tại giảm field So_lan của nó một đơn vị.
  - Nếu phần tử X không tồn tại. Thông báo.
- Hãy viết hàm in lên màn hình giá trị của các phần tử đang tồn tại trong cây theo thứ tự NLR.

BÀI 16

```
11. int DeleteNode(struct BST_TREE &t,
                   int x)
12. {
13.   if(t.pRoot==NULL)
14.     return 0;
15.   if(t.pRoot->Key==x)
16.   {
17.     if(t.pRoot->So_lan>0)
18.     {
19.       t.pRoot->So_lan--;
20.       return 1;
21.     }
22.     return 0;
23.   }
24.   if (x < t.pRoot->Key)
25.     return DeleteNode
                  (t.pRoot->Left,x);
26.   return DeleteNode
                  (t.pRoot->Right,x);
27. }
```

BÀI 16

```
1. void XoaGiaTri(struct BST_TREE t,
                  int X)
2. {
3.   int kq = DeleteNode(t,x);
4.   if(kq==0)
5.     printf("Khong ton tai X");
6.   else
7.     printf("Xoa thanh cong.");
8. }
```

BÀI 16

```
11. int DeleteNode(struct BST_TREE &t,
                   int x)
12. {
13.    if(t.pRoot==NULL)
14.       return 0;
15.    if(t.pRoot->Key==x)
16.    {
17.       if(t.pRoot->So_lan>0)
18.       {
19.          t.pRoot->So_lan--;
20.          return 1;
21.       }
22.       return 0;
23.    }
24.    if (x < t.pRoot->Key)
25.       return DeleteNode
                    (t.pRoot->Left,x);
26.    return DeleteNode
                    (t.pRoot->Right,x);
27. }
```

BÀI 16

```
1. void XoaGiaTri(struct BST_TREE t,
                  int X)
2. {
3.     int kq = DeleteNode(t,x);
4.     if(kq==0)
5.       printf("Khong ton tai X");
6.     else
7.       printf("Xoa thanh cong.");
8. }
```

BÀI 16

- Cho một cây nhị phân tìm kiếm t có cấu trúc nút là BST_NODE được khai báo như sau:

```
11. struct BST_NODE{
12.     int Key;      // Khóa của nút
13.     int So_lan; // Số lần xuất hiện.
14.     struct BST_NODE*Left,*Right;
15. };
16. struct BST_TREE
17. {
18.     struct BST_NODE *pRoot;
                  // Nút gốc của cây.
19. };
20. struct BST_TREE t; // Cây t
```

BÀI 16

```
11. int DeleteNode(struct BST_NODE*
                        Root,int x)
12. {
13.   if(Root==NULL)
14.     return 0;
15.   if(Root->Key==x)
16.   {
17.     if(Root->So_lan>0)
18.     {
19.       Root->So_lan--;
20.       return 1;
21.     }
22.     return 0;
23.   }
24.   if (x < Root->Key)
25.     return DeleteNode
                        (Root->Left,x);
26.   return DeleteNode
                        (Root->Right,x);
27. }
```

BÀI 16

```
1. void XoaGiaTri(struct BST_TREE t,
                  int X)
2. {
3.   int kq = DeleteNode(t.pRoot,
                         x);
4.   if(kq==0)
5.     printf("Khong ton tai X");
6.   else
7.     printf("Xoa thanh cong.");
8. }
```

BÀI 16

- Câu b: Hãy viết hàm in lên màn hình giá trị của các phần tử đang tồn tại trong cây theo thứ tự NLR.

BÀI 16

- Cho một cây nhị phân tìm kiếm t có cấu trúc nút là BST_NODE được khai báo như sau:

```
11. struct BST_NODE{
12.     int Key;      // Khóa của nút
13.     int So_lan; // Số lần xuất hiện.
14.     struct BST_NODE*Left,*Right;
15. };
16. struct BST_TREE
17. {
18.     struct BST_NODE *pRoot;
                  // Nút gốc của cây.
19. };
20. struct BST_TREE t; // Cây t
```

BÀI 16

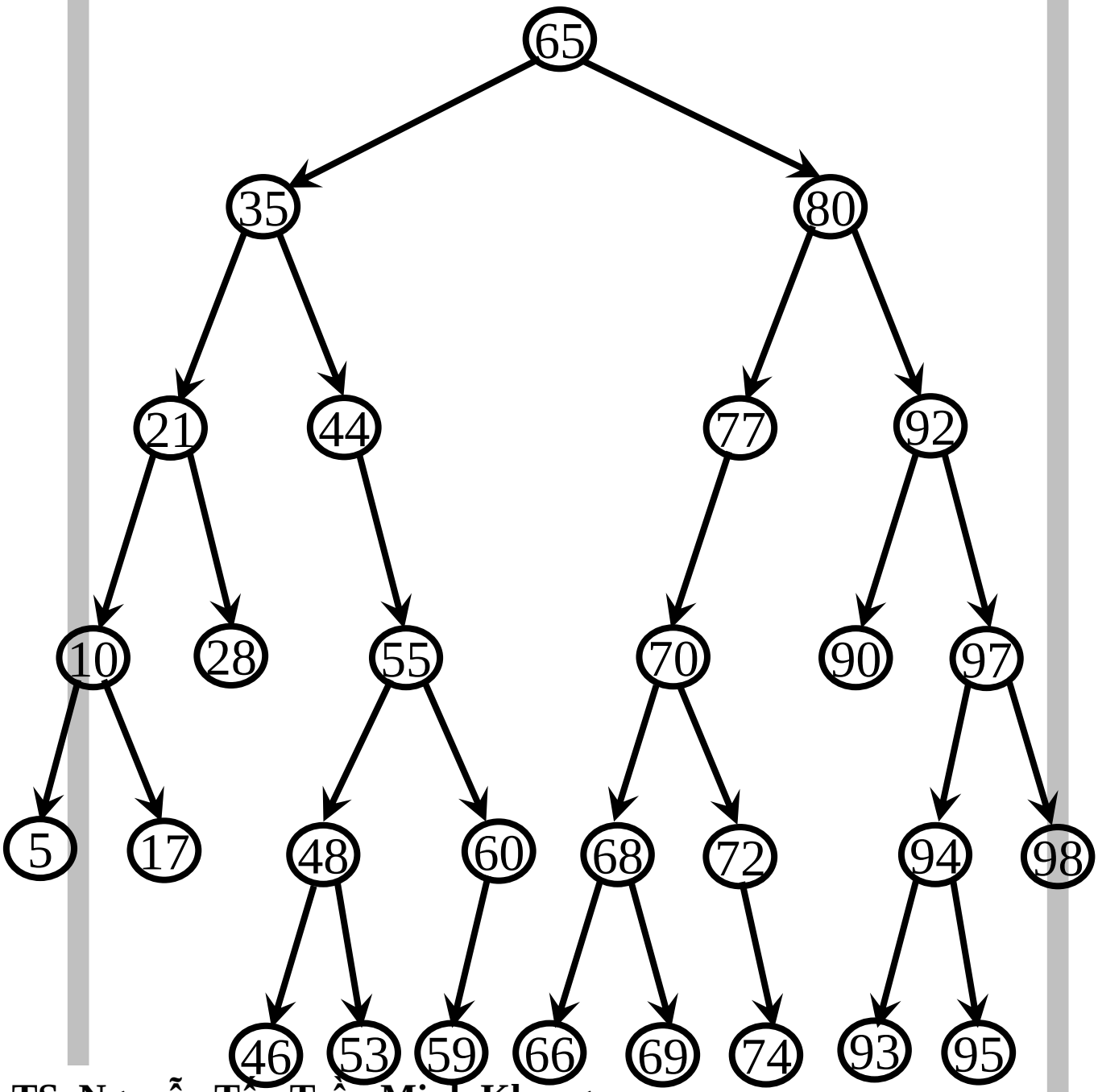
- **Câu b**

```
11. void LietKe(struct BST_TREE t)
12. {
13.     NLR(t.pRoot);
14. }
15. void NLR(struct BST_NODE*Root)
16. {
17.     if(Root==NULL)
18.         return;
19.     if(Root->So_lan>0)
20.         printf("%4d",Root->key);
21.     NLR(Root->Left);
22.     NLR(Root->Right);
23. }
```

BÀI 17

- Hãy viết hàm tìm phần tử thay thế trong thao tác "Xoá một phần tử P có 2 con trong cây BST", sử dụng nguyên tắc "tìm phần tử tận cùng bên trái của nhánh phải P".

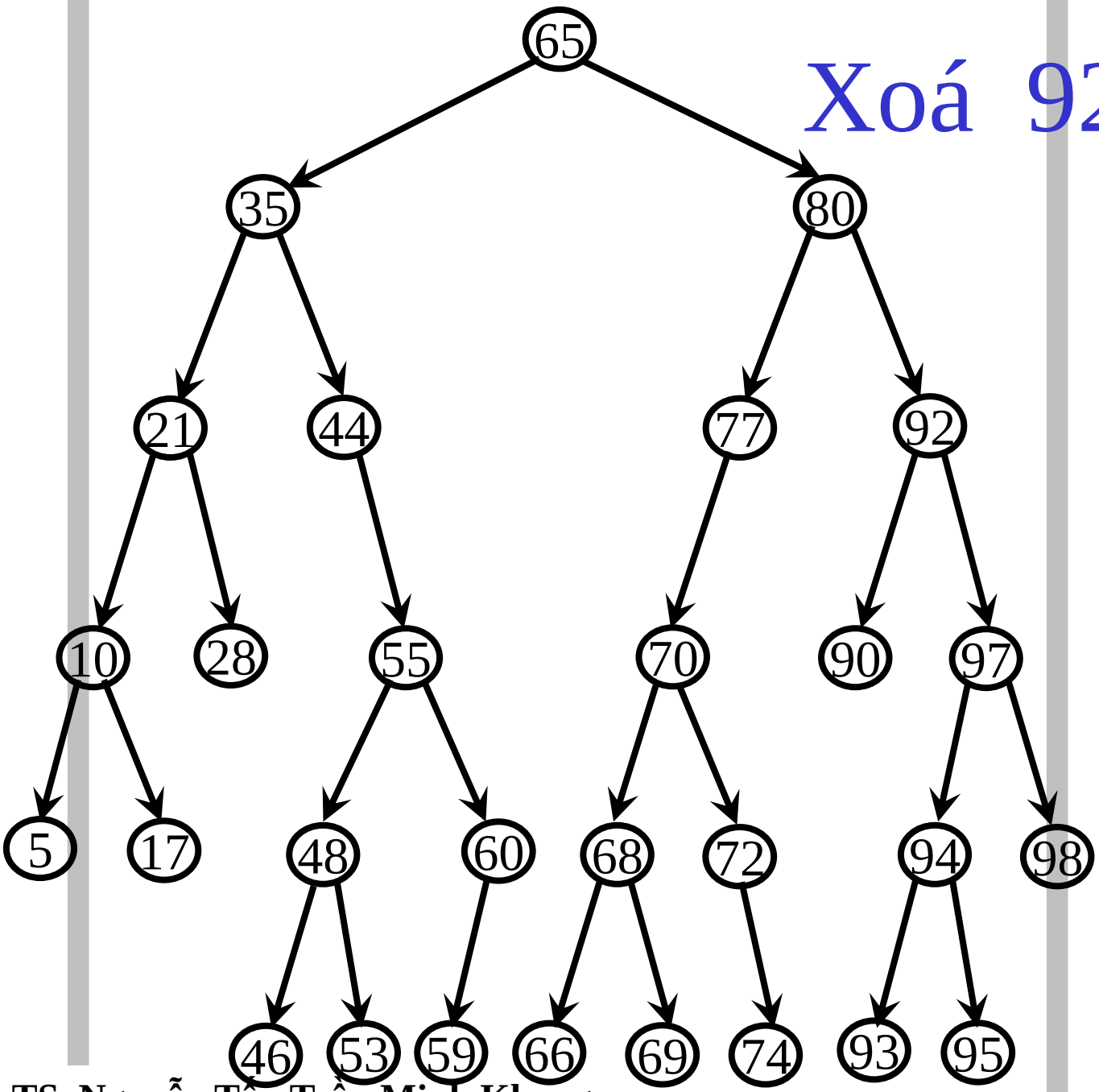
BÀI 17



TS. Nguyễn Tấn Trần Minh Khang

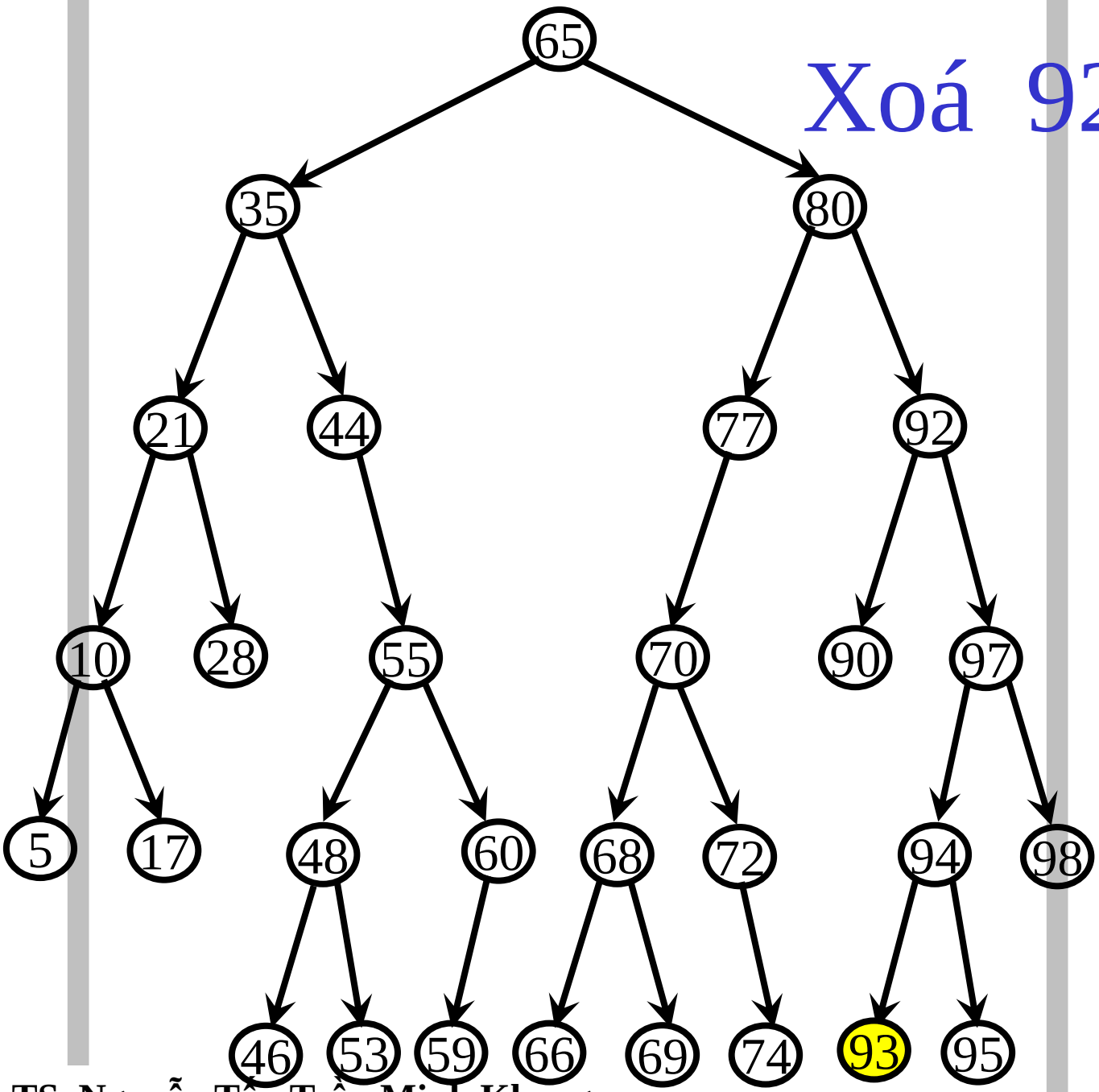
ThS. Cáp Phạm đình Thăng

“tìm phần tử tận cùng bên trái của nhánh phải P”



Xoá 92

“tìm phần tử tận cùng bên trái của nhánh phải P”



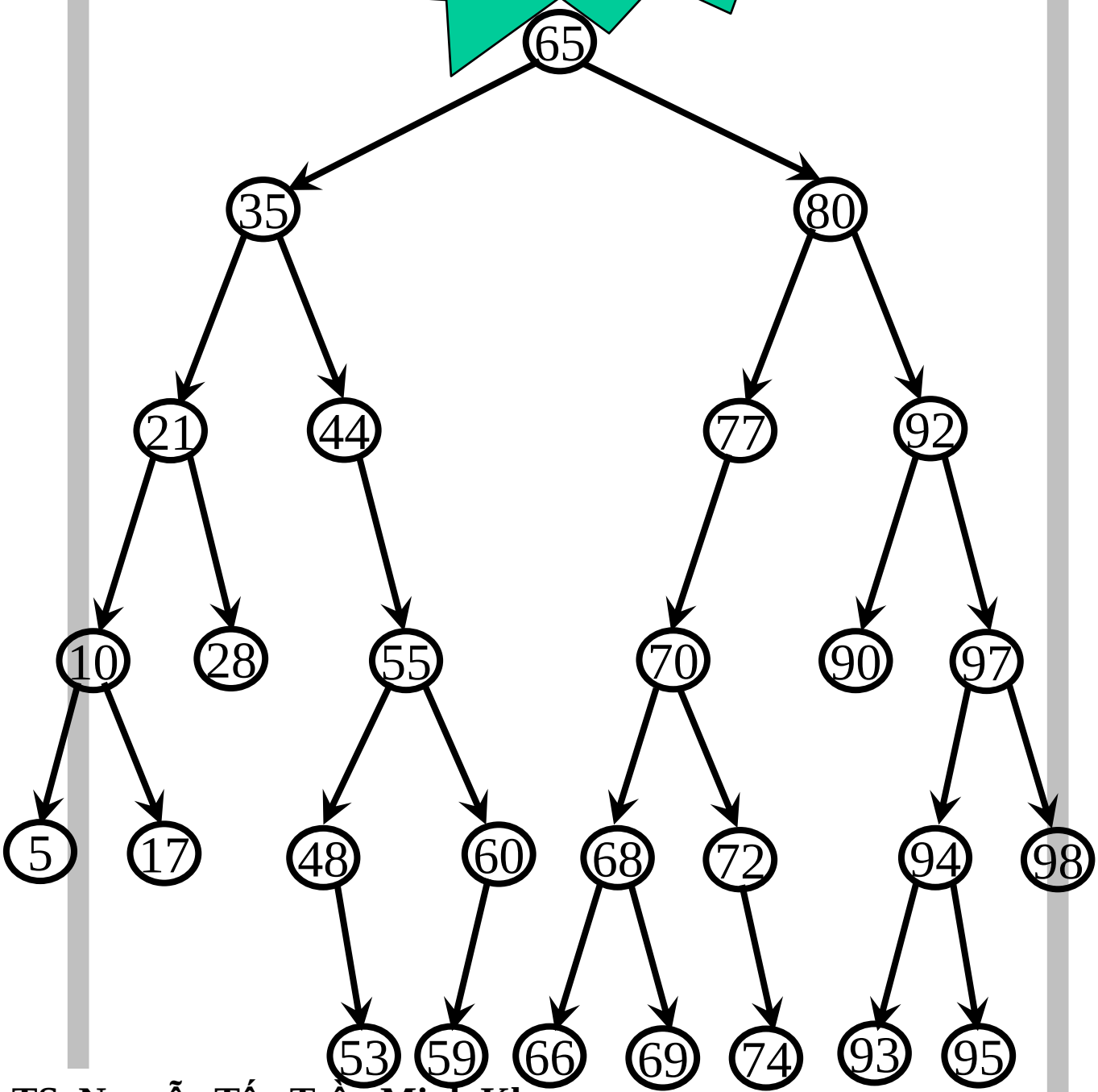
Xoá 92

BÀI T

Tận cùng bên trái của nhánh phải

```
11. void SearchStandFor(TREE&p,
                        TREE&q)
12. {
13.   if(q->pLeft)
14.     SearchStandFor(p,q->pLeft);
15.   else
16.   {
17.     p->info=q->info;
18.     p=q;
19.     q=q->pRight;
20.   }
21. }
```

Tận cùng
bên trái của
nhánh phải

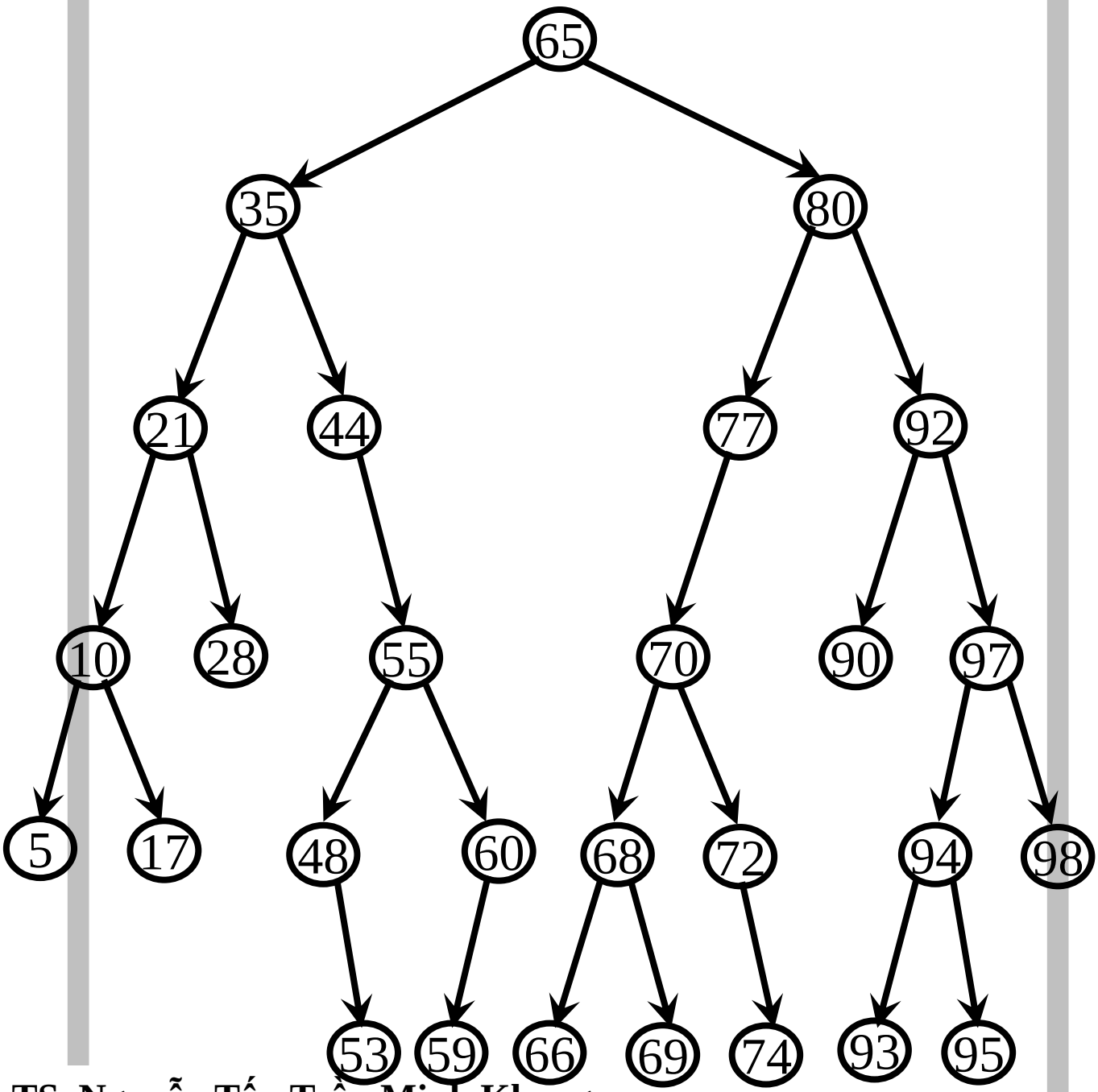


BÀI 17

Tận cùng bên trái của nhánh phải

```
11. void SearchStandFor(TREE&p,
                        TREE&q)
12. {
13.   if(q->pLeft)
14.     SearchStandFor(p,q->pLeft);
15.   else
16.   {
17.     p->info=q->info;
18.     p=q;
19.     q=q->pRight;
20.   }
21. }
```

BÀI 17



TS. Nguyễn Tấn Trần Minh Khang

ThS. Cáp Phạm đình Thăng

Tận cùng bên phải của nhánh trái

```
11. void SearchStandFor(TREE&p,
                        TREE&q)
12. {
13.   if(q->pRight)
14.     SearchStandFor(p,q->pRight);
15.   else
16.   {
17.     p->info = q->info;
18.     p = q;
19.     q = q->pLeft;
20.   }
21. }
```

BÀI 18

- Cho một mảng a gồm n phần tử kiểu số nguyên int. Ta có thể sắp xếp mảng a bằng cách:
  - Từ mảng a, tạo một cây nhị phân tìm kiếm T.
  - Duyệt cây T và đưa các nút trở lại mảng a.
- Yêu cầu

a) Cho biết phương pháp duyệt cây T để đưa các nút lên mảng sao cho mảng được sắp tăng dần.

b) Cho biết cấu trúc cây T.

c) Xây dựng hàm
  - Tạo cnp T từ mảng a.
  - Duyệt cây để đưa các phần tử trở lại mảng sao cho mảng được sắp thứ tự tăng dần.

BÀI 18

- Câu a: Phương pháp duyệt là LNR.
- Câu b: Cấu trúc dữ liệu:

```
1. struct node
2. {
3.     int info;
4.     struct node*pLeft;
5.     struct node*pRight;
6. };
7. typedef struct node NODE;
8. typedef NODE*TREE;
```

BÀI 18

- Cho một mảng a gồm n phần tử kiểu số nguyên int. Ta có thể sắp xếp mảng a bằng cách:
  - Từ mảng a, tạo một cây nhị phân tìm kiếm T.
  - Duyệt cây T và đưa các nút trở lại mảng a.
- Yêu cầu

a) Cho biết phương pháp duyệt cây T để đưa các nút lên mảng sao cho mảng được sắp tăng dần.

b) Cho biết cấu trúc cây T.

c) Xây dựng hàm
  - Tạo cnp T từ mảng a.
  - Duyệt cây để đưa các phần tử trở lại mảng sao cho mảng được sắp thứ tự tăng dần.

BÀI 18

- Câu c1: Tạo cây từ mảng a.

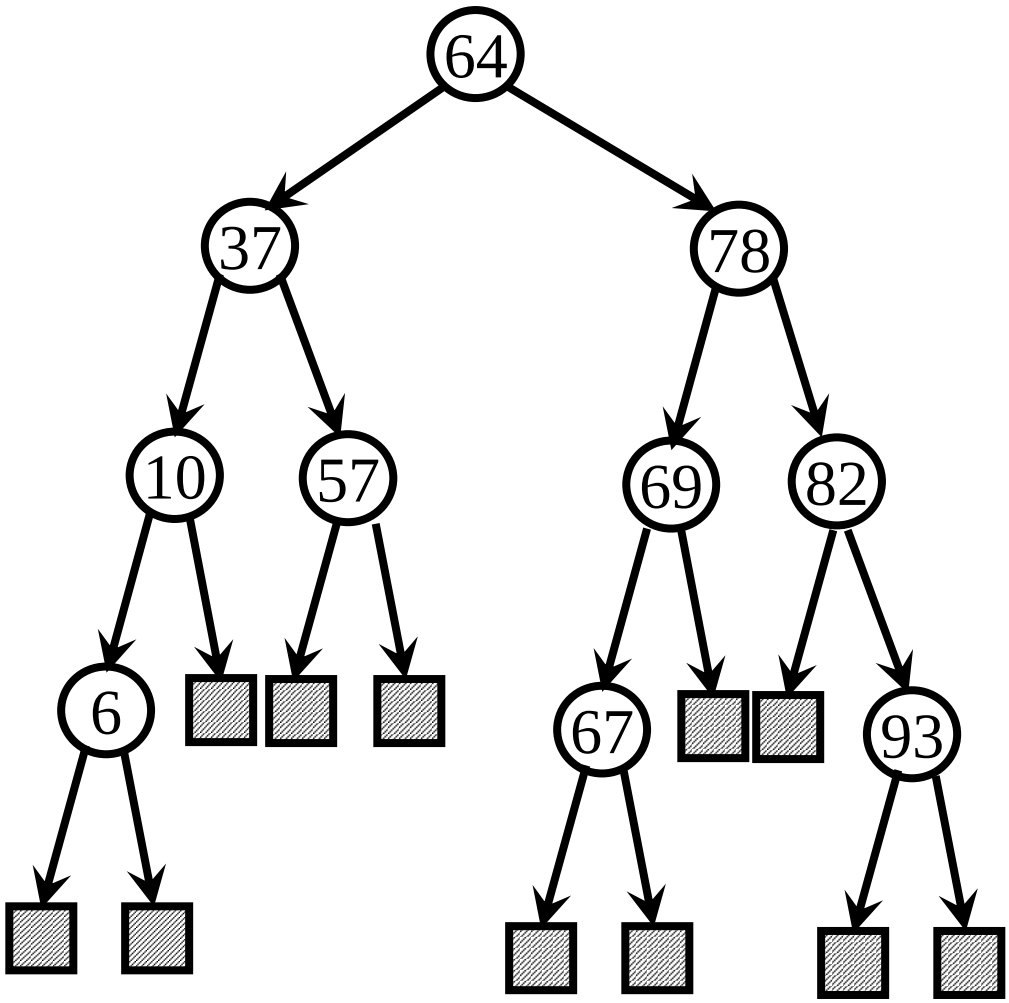
```
11. int TaoCay(TREE &t,
                int a[],int n)
12. {
13.     Init(t);
14.     for(int i=0;i<n;i++)
15.     {
16.         if(InsertNode(t,a[i])
                ==-1)
17.             return 0;
18.     }
19.     return 1;
20. }
```

BÀI 18

- Câu c1: Tạo cây từ mảng a.

```
11. void Init(TREE &t)
12. {
13.     t = NULL;
14. }
15. NODE* GetNode(int x)
16. {
17.     NODE*p = new NODE;
18.     if(p==NULL)
19.         return NULL;
20.     p->info=x;
21.     p->pLeft=p->pRight = NULL;
22.     return p;
23. }
```

BÀI 18



TS. Nguyễn Tấn Trần Minh Khang

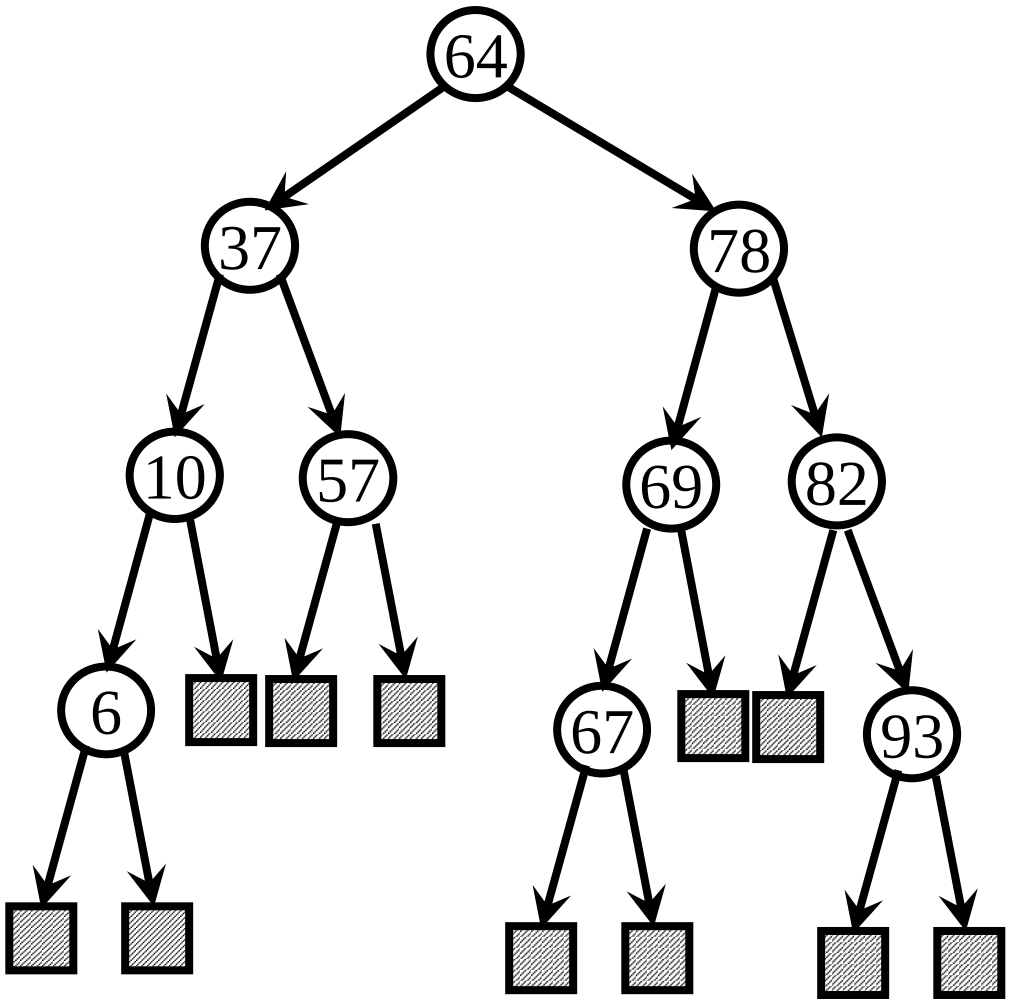
ThS. Cáp Phạm đình Thăng

BÀI 18

- Câu c1: Tạo cây từ mảng a.

```
11. int InsertNode(TREE &t,int x)
12. {
13.   if(t)
14.   {
15.     if(t->info==x)
16.       return 0;
17.     if(t->info<x)
18.       return InsertNode
                      (t->pRight,x);
19.     return InsertNode
                      (t->pLeft,x);
20.   }
21.   t = GetNode(x);
22.   if(t==NULL)
23.     return -1;
24.   return 1;
25. }
```

BÀI 18



TS. Nguyễn Tấn Trần Minh Khang

ThS. Cáp Phạm đình Thăng

BÀI 18

- Cho một mảng a gồm n phần tử kiểu số nguyên int. Ta có thể sắp xếp mảng a bằng cách:
  - Từ mảng a, tạo một cây nhị phân tìm kiếm T.
  - Duyệt cây T và đưa các nút trở lại mảng a.
- Yêu cầu

a) Cho biết phương pháp duyệt cây T để đưa các nút lên mảng sao cho mảng được sắp tăng dần.

b) Cho biết cấu trúc cây T.

c) Xây dựng hàm
  - Tạo cnp T từ mảng a.
  - Duyệt cây để đưa các phần tử trở lại mảng sao cho mảng được sắp thứ tự tăng dần.

BÀI 18

- Duyệt cây đưa trở lại vào mảng

```
1. void LNR(TREE t,int a[],int &n)
2. {
3.     if(t==NULL)
4.         return;
5.     LNR(t->pLeft,a,n);
6.     a[n] = t->info;
7.     n++;
8.     LNR(t->pRight,a,n);
9. }
```

BÀI TẬP CÂY NHỊ PHÂN

- Bài 33: Anh Tri Tuấn
- Hãy cài đặt thuật toán duyệt cây nhị phân (NLR) không dùng đệ quy (dùng stack).
- Hãy cài đặt thuật toán duyệt cây nhị phân theo mức không dùng đệ quy (dùng queue).

BÀI TẬP CÂY NHỊ PHÂN

- Bài 37: Anh Tri Tuấn
- Cho một cây nhị phân có nút gốc là Root. Hãy viết hàm kiểm tra xem cây này có phải là cây cân bằng không? (Giả sử đã có hàm tính chiều cao của nút p như sau:

```
int ChieuCao(NODE*p)
```

- và thông tin tại mỗi nút trong cây là số nguyên).

BÀI TẬP CÂY NHỊ PHÂN

```
1.  struct node
2.  {
3.      int info;
4.      struct node *pLeft;
5.      struct node *pRight;
6.  };
7.  typedef struct node NODE;
8.  typedef NODE*TREE;
```

BÀI TẬP CÂY NHỊ PHÂN

```
1.  NODE*LonNhat(TREE t)
2.  {
3.      if(t==NULL)
4.          return NULL;
5.      NODE*lc=t;
6.      NODE*a = LonNhat(t->pLeft);
7.      if(a && a->info>lc->info)
8.          lc = a;
9.      NODE*b = LonNhat(t->pRight);
10.     if(b && b->info>lc->info)
11.         lc = b;
12.     return lc;
13. }
```

BÀI TẬP CÂY NHỊ PHÂN

```
1.  NODE* NhoNhat(TREE t)
2.  {
3.      if(t==NULL)
4.          return NULL;
5.      NODE*lc=t;
6.      NODE*a = NhoNhat(t->pLeft);
7.      if(a && a->info<lc->info)
8.          lc = a;
9.      NODE*b = NhoNhat(t->pRight);
10.     if(b && b->info<lc->info)
11.         lc = b;
12.     return lc;
13. }
```

BÀI TẬP CÂY NHỊ PHÂN

```
1.  int ChieuCao(TREE t)
2.  {
3.      if(t==NULL)
4.          return 0;
5.      int a=ChieuCao(t->pLeft);
6.      int b=ChieuCao(t->pRight);
7.      if(a>b)
8.          return a+1;
9.      return b+1;
10. }
```

```
1.  int ktCanBang(TREE Root)
2.  {
3.      if(Root==NULL)
4.          return 1;
5.      if(ktCanBang(Root->pLeft)==0)
6.          return 0;
7.      if(ktCanBang(Root->pRight)==0)
8.          return 0;
9.      NODE*a=LonNhat(Root->pLeft);
10.     if(a && a->info>Root->info)
11.         return 0;
12.     a=NhoNhat(Root->pRight);
13.     if(a && a->info<Root->info)
14.         return 0;
15.     int x=ChieuCao(Root->pLeft);
16.     int y=ChieuCao(Root->pRight);
17.     if(abs(x-y)>1)
18.         return 0;
19.     return 1;
20. }
```