CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

Data Structures & Algorithms

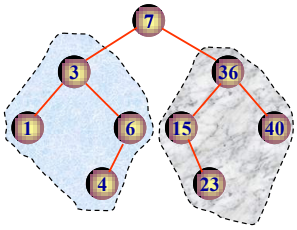
CÂY NHỊ PHÂN TÌM KIẾM – BINARY SEARCH TREE

Cây Nhị Phân Tìm Kiếm – Binary Search Tree

- Là cây **nhị phân**
- Giá trị của một node luôn **lớn hơn giá trị của các node nhánh trái** và **nhỏ hơn giá trị các node nhánh phải**

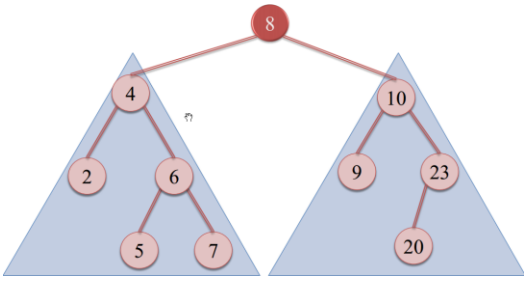
➔Nút có giá trị nhỏ nhất **nằm ở nút trái nhất của cây**

➔Nút có giá trị **lớn nhất nằm ở nút phải nhất của cây**



Nhờ **cấu trúc của cây** ➔Định hướng được khi **tìm kiếm**

Cây Nhị Phân Tìm Kiếm – Binary Search Tree



Đặc điểm Binary Search Tree

➢Dễ dàng tạo dữ liệu sắp xếp và tìm kiếm.

➢Có thứ tự.

➢Không có phần tử trùng.

Các thao tác trên Binary Search Tree

➢Tạo một cây rỗng

- Cây rỗng -> địa chỉ **nút gốc bằng NULL**

```
void CreateTree(TREE &T)
{
    T=NULL;
}
```


Các thao tác trên Binary Search Tree

➢Tạo một node có key x

```
TNode *CreateTNode(KDL x)
{
    TNode *p;
    p = new TNode; //cấp phát vùng nhớ động
    if(p==NULL)
        return NULL; // thoát
    else
    {
        p->key = x; //gán trường dữ liệu của nút = x
        p->pLeft = NULL;
        p->pRight = NULL;
    }
    return p;
}
```

Các thao tác trên Binary Search Tree

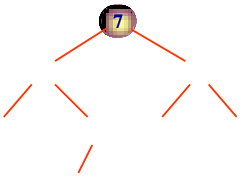
➢Thêm một node vào cây – **Tạo cây** - Sau khi thêm **cây đảm bảo là cây nhị phân tìm kiếm**.

Hãy **vẽ cây nhị phân tìm kiếm** lập được từ dãy số sau **theo chiều từ trái qua phải**

7	36	3	1	6	4	15	40

Tạo cây

7	36	3	1	6	4	15	40



- Nếu node cần thêm nhỏ hơn node đang xét thì thêm về bên trái
- Ngược lại thì thêm về bên phải

Thêm một node vào Binary Search Tree

```
int InsertNode (Tree & t, int x)
{
    if(t!=NULL)
    {     if(x==t->data)  return 0; //Có giá trị trùng
          else
          {       if(x<t->key)      InsertNode(t->pLeft, x);
                  else              InsertNode(t->pRight, x);
          }
    }
    else
    {     t= new TNode;
          if(t==NULL)               return -1; //Thiếu bộ nhớ
          t->key=x;
          t->pLeft=t->pRight=NULL;
          return 1; //Thêm thành công
    }
}
```

Tạo cây – Ví dụ

Bài toán 1: **Vẽ cây** nhị phân tìm kiếm **từ dãy số**

Vẽ cây nhị phân tìm kiếm (chỉ vẽ cây kết quả) từ dãy số nguyên khi **xây dựng cây theo thứ tự từ trái qua phải của dãy số**: 72; 67; 73; 58;5; 4; 27; 53; 61; 32.

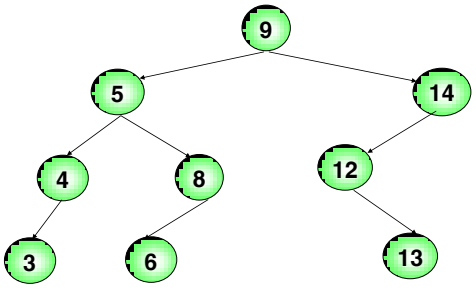
Làm theo nguyên tắc **thêm một node vào cây**:

* Luôn **bắt đầu so sánh từ node gốc**.

* Đảm bảo đặc điểm **lớn bên phải, nhỏ bên trái.**

Tạo cây – Ví dụ

9, 5, 4, 8, 6, 3, 14,12,13



Tạo cây – Ví dụ

Bài toán 2: **Vẽ cây** khi biết kết quả duyệt **cây.**

Hãy **vẽ cây nhị phân tìm kiếm** T biết rằng khi duyệt cây **theo thứ tự Left –Right – Node** thì được dãy như sau: 5, 3, 7, 9, 8, 11, 6, 20, 19, 37, 25, 21, 15, 12.

Làm theo **nguyên tắc**:

(1) Tìm **node gốc**

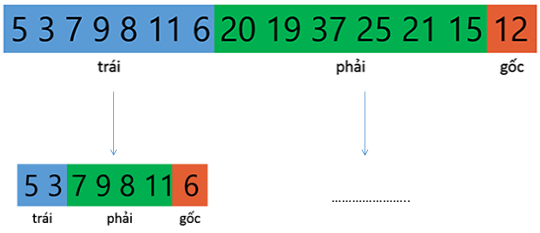
(2) Tìm đoạn **lớn hơn node gốc** sẽ là nhánh phải, **đoạn nhỏ hơn node gốc** sẽ là nhánh trái.

(3) Với mỗi đoạn vừa tìm được, **tìm node gốc của từng đoạn** và tiếp tục tìm đoạn lớn hơn và nhỏ hơn node gốc.

Tạo cây – Ví dụ

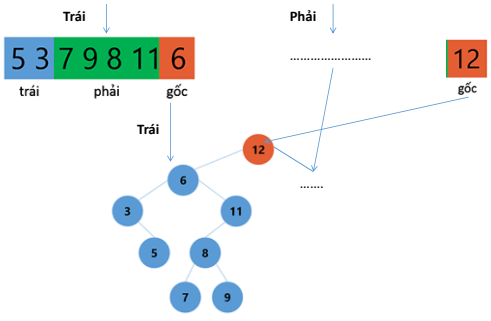
Cho kết quả duyệt LRN: **5 3 7 9 8 11 6 20 19 37 25 21 15 12**

* **Node gốc**: 12



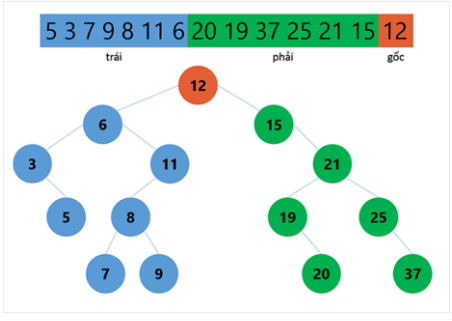
13

Tạo cây – Ví dụ



14

Tạo cây – Ví dụ



15

Tạo cây – Bài tập

Bài 1: **Vẽ cây nhị** phân tìm kiếm (chỉ vẽ cây kết quả) từ dãy số nguyên khi **xây dựng cây theo thứ tự từ trái qua phải của dãy số**: 72; 77; 64; 58;70; 75; 10; 53; 71; 59,82. Sau đó cho biết kết quả của phép duyệt cây theo thứ tự LRN.

Bài 2: Hãy **vẽ cây nhị phân tìm kiếm** T biết rằng khi duyệt cây **theo thứ tự Node - Left – Right** thì được dãy như sau: 67, 50, 44, 21, 47, 62, 60, 64, 80, 71, 69, 78, 96.

16

Các thao tác trên Binary Search Tree

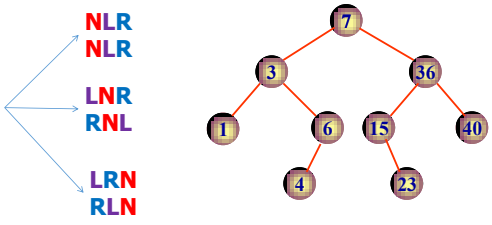
➢Duyệt cây

- ❑**Duyệt trước** (pre-oder)
- ❑**Duyệt giữa** (in-oder)
- ❑**Duyệt sau** (post-oder)

17

Các thao tác trên Binary Search Tree

➢ Cho cây nhị phân tìm kiếm duyệt cây theo thứ tự yêu cầu



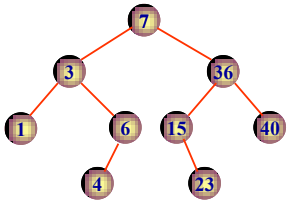
18

Các thao tác trên Binary Search Tree

➢ Cho cây nhị phân tìm kiếm duyệt cây theo thứ tự yêu cầu

NLR

```
void NLR (Tree t)
{
   if(t!=NULL)
   {
        cout<<t->key;
        NLR(t->pLeft);
        NLR(t->pRight);
   }
}
```



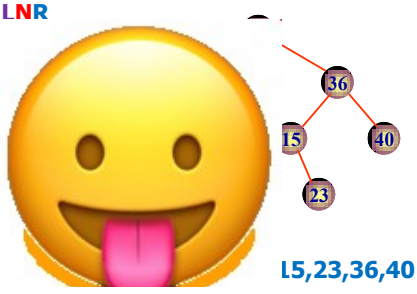
NLR:7,3,1,6,4,36,15,23,40

Các thao tác trên Binary Search Tree

➢ Cho cây nhị phân tìm kiếm duyệt cây theo thứ tự yêu cầu

LNR

```
void LNR (Tree t)
{
   if(t!=NULL)
   {
        LNR(t->pLef
        cout<<t->Ke
        LNR(t->pRig
   }
}
```



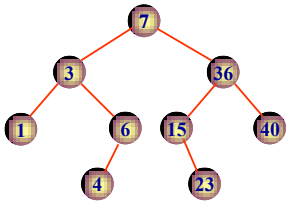
L5,23,36,40

Các thao tác trên Binary Search Tree

➢ Cho cây nhị phân tìm kiếm duyệt cây theo thứ tự yêu cầu

LRN

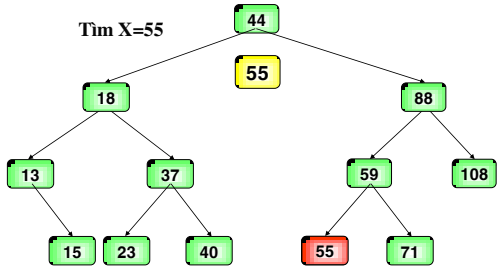
```
void LRN (Tree t)
{
   if(t!=NULL)
   {
        LRN(t->pLeft);
        LRN(t->pRight);
        cout<<t->Key<<" ";
   }
}
```



LRN:1,4,6,3,23,15,40,36,7

Các thao tác trên Binary Search Tree

➢Tìm node có key bằng x



Tìm thấy X=55

22

Các thao tác trên Binary Search Tree

➢Tìm node có **key bằng x** – Không dùng đệ quy

```
TNode * searchNode(TREE Root,  Data x)
{       TNode *p = Root;
        while (p != NULL)
        {       if(x == p->Key)
                        return p;
                else
                        if(x < p->Key)
                                p = p->pLeft;
                        else
                                p = p->pRight;
        }
        return NULL;
}
```

23

Các thao tác trên Binary Search Tree

➢Tìm node có **key bằng x** – dùng đệ quy

```
TNode *SearchTNode(TREE T, int x)
{
  if(T!=NULL)
  {
       if(T->key==x)
            return T;
       else
            if(x>T->key)
                 return SearchTNode(T->pRight,x);
            else
                 return SearchTNode(T->pLeft,x);
  }
  return NULL;
}
```

24

Binary Search Tree – Bài tập

Cho cây nhị phân tìm kiếm, mỗi node có giá trị nguyên, hãy định nghĩa các hàm sau:

1. In ra các node có giá trị chẵn
2. In ra các node có giá trị lớn hơn x
3. Đếm số node của cây
4. Tính độ cao của cây
5. Tìm node có giá trị x
6. Tìm node có giá trị lớn nhất
7. Tìm node có giá trị nhỏ nhất của cây con phải

25

Binary Search Tree – Bài tập

8. Đếm số node lá (node bậc 0)
9. Đếm số node có 1 cây con (node bậc 1)
10. Đếm số node chỉ có 1 cây con phải
11. Đếm số node có 1 cây con trái
12. Đếm số node 2 cây con (node bậc 2)
13. In các node trên từng mức của cây
14. Cho biết độ dài đường đi từ gốc đến node x

26

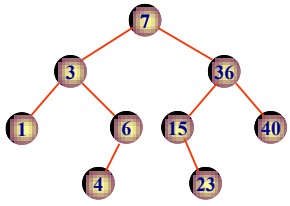
Các thao tác trên Binary Search Tree

➢Xóa một node trên cây

➢Hủy 1 phần tử trên cây **phải đảm bảo điều kiện ràng buộc của Cây nhị phân tìm kiếm**

➢Có 3 trường hợp khi hủy 1 nút trên cây

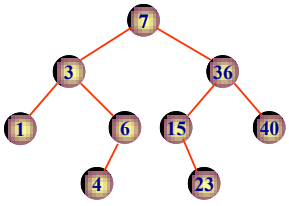
- TH1: X là nút **lá**
- TH2: X chỉ **có 1 cây con** (cây con trái hoặc cây con phải)
- TH3: **X có đầy đủ 2 cây con**



27

Các thao tác trên Binary Search Tree

➢TH1:Xóa **node lá** trên cây không ảnh hưởng đến các nút khác trên cây



➢ Xóa node giá trị 1

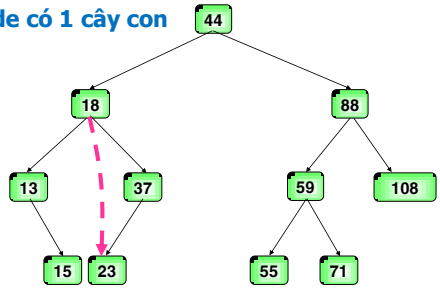
➢ Xóa node giá trị 40

28

Các thao tác trên Binary Search Tree

➢TH2 Xóa **node có 1 cây con**

Hủy X=37



Trước khi xoá x ta **móc nối cha của x** với **con duy nhất của x**

29

Các thao tác trên Binary Search Tree

➢TH3 Xóa **node có 2 cây con**

❖Tìm **phần tử thế mạng** cho phần tử cần xóa

❖Có 2 cách tìm nút thế mạng

- C1: Nút có khoá **nhỏ nhất (trái nhất) bên cây con phải** node cần xóa
- C2: Nút có khoá lớn nhất (phải nhất) bên cây con trái của node cần xóa

30

Các thao tác trên Binary Search Tree

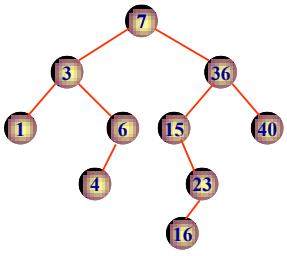
➢TH3 Xóa **node có 2 cây con**

Bước 1: Tìm node thế mạng

- ***Cách 1:*** *Tìm node trái nhất của cây con phải*
- ***Cách 2:*** *Tìm node phải nhất của cây con trái*

Bước 2: Thay giá trị của node thế mạng vào node cần xóa

Bước 3: Xóa node thế mạng

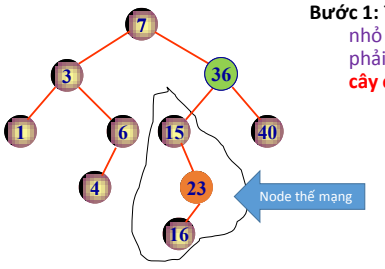


31

Các thao tác trên Binary Search Tree

➢Xóa **node có giá trị 36**

Bước 1: Tìm node thế mạng nhỏ nhất cây con bên phải hoặc **lớn nhất bên cây con trái** node 36

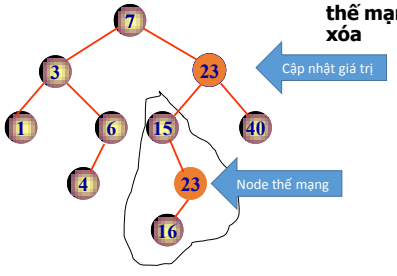


32

Các thao tác trên Binary Search Tree

➢Xóa **node có giá trị 36**

Bước 2: Thay **giá trị node thế mạng** cho **node cần xóa**

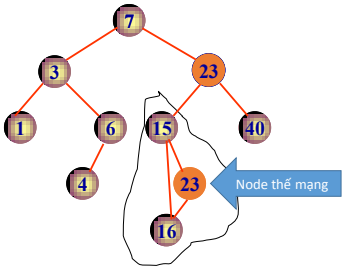


33

Các thao tác trên Binary Search Tree

➢Xóa **node có giá trị 36**

Bước 3: Xóa node thế mạng



34

```
void Remove(Tree & t, int x)
{
   if (t != NULL)
   {
      if (x < t->key)
            Remove(t->pLeft, x);
      else if (x > t->key)
            Remove(t->pRight, x);
      else
      {
         TNode * pHuy = t;
         if (t->pLeft == NULL)
            t = t->pRight;
         else if (t->pRight == NULL)
            t = t->pLeft;
         else SearchStandFor(pHuy, t->pRight);

         delete pHuy;
      }
   }
}
```

35

```
void SearchStandFor(Tree &pHuy, Tree &pTM)
{
   if(pTM->pLeft!=NULL)
      SearchStandFor(pHuy, pTM->pLeft);
   else
   {
      pHuy->key=pTM->key;
      pHuy=pTM;
      pTM=pTM->pRight;
   }
}
```

36

Binary Search Tree – Bài tập

Cho dãy số theo thứ tự nhập từ trái sang phải: **20, 15, 35, 30, 11, 13, 17, 36, 47, 16, 38, 28, 14**

- Vẽ cây nhị phân tìm kiếm cho dãy số trên
- Trình bày từng bước và vẽ lại cây sau khi lần lượt xoá các nút: **11** và **35**

37

39

Slide được tham khảo từ

- **Slide được tham khảo từ:**
  - Slide CTDL GT, Khoa Khoa Học Máy Tính, ĐHCNTT
  - Congdongcviet.com
  - Cplusplus.com

38