ĐỀ CƯƠNG BÀI GIẢNG

BÀI 5: CÂY VÀ CÂY NHỊ PHÂN

5.3. Cây nhị phân tìm kiếm

Cây nhị phân được sử dụng vào nhiều mục đích khác nhau. Tuy nhiên việc sử dụng cây nhị phân để lưu giữ và tìm kiếm thông tin vẫn là một trong những áp dụng quan trọng nhất của cây nhị phân. Trong phần này chúng ta sẽ nghiên cứu một lớp cây nhị phân đặc biệt phục vụ cho việc tìm kiếm thông tin, đó là cây nhị phân tìm kiếm.

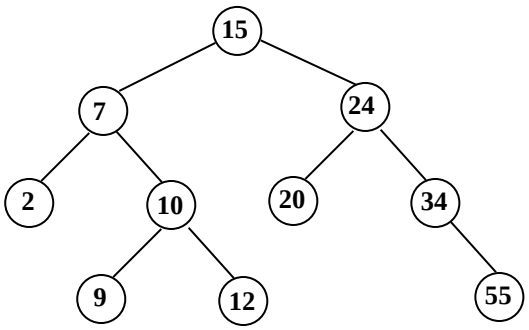
Trong thực tế, một lớp đối tượng nào đó có thể được mô tả bởi một kiểu bản ghi, các trường của bản ghi biểu diễn các thuộc tính của đối tượng. Trong bài toán tìm kiếm thông tin, ta thường quan tâm đến một nhóm các thuộc tính nào đó của đối tượng mà thuộc tính này hoàn toàn xác định được đối tượng. Chúng ta gọi các thuộc tính này là khoá. Như vậy, khoá là một nhóm các thuộc tính của một lớp đối tượng sao cho hai đối tượng khác nhau phải có giá trị khác nhau trên nhóm thuộc tính đó.

2.1. Định nghĩa

Cây nhị phân tìm kiếm (CNPTK) là cây nhị phân hoặc rỗng hoặc không rỗng thì phải thoả mãn đồng thời các điều kiện sau:

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

Hình 5.19 biểu diễn một cây nhị phân tìm kiếm, trong đó khoá của các đỉnh là các số nguyên.



Hình 2.17. Cây nhị phân tìm kiếm

2.2. Cài đặt cây nhị phân tìm kiếm

Mỗi nút trên cây nhị phân tìm kiếm có dạng

Left	infor	Right

Trong đó trường Left: con trỏ chỉ tới cây con trái

Right: con trỏ chỉ tới cây con phải

infor: chứa thông tin của nút

Giả sử dữ liệu trên mỗi nút của cây có kiểu dữ liệu là ***Item***, khi đó cấu trúc dữ liệu của cây TKNP được định nghĩa như sau:

```
struct  Node
{
    Item infor;
    struct  Node *Left, *Right;
};
struct  Node *Root;
```

Tiếp theo ta nghiên cứu các phép toán trên cây nhị phân tìm kiếm.

2.3. Các thao tác cơ bản trên cây nhị phân tìm kiếm

2.3.1. Tìm kiếm

Tìm kiếm trên cây là một trong các phép toán quan trọng nhất đối với cây nhị phân tìm kiếm. Ta xét bài toán sau

Bài toán: Giả sử mỗi nút trên cây nhị phân tìm kiếm là một bản ghi, biến con trỏ Root chỉ tới gốc của cây và X là khoá cho trước. Vấn đề đặt ra là, tìm xem trên cây có chứa nút với khoá X hay không.

**** Giải thuật đệ qui***

```
struct Node  *search ( struct  Node *Root;  Item X)
/* Hàm search trả về con trỏ trỏ tới nút có khoá bằng X, ngược lại trả về NULL*/
{
        if  (Root = = NULL) return NULL;
        else
            if  (X < Root->infor) return search(Root->Left, X);
            else
                if  (X > Root->infor)  return search(Root->Right, X);
                else return Root;
}
```

**** Giải thuật lặp***

Trong thủ tục này, ta sẽ sử dụng biến địa phương *found* có kiểu int để điều khiển vòng lặp, nó có giá trị ban đầu là 0. Nếu tìm kiếm thành công thì *found* nhận giá trị 1, vòng lặp kết thúc và hàm *search*() trả về con trỏ trỏ đến nút có trường khoá bằng X. Còn nếu không tìm thấy thì giá trị của *found* vẫn là 0 và hàm *search*() trả về NULL.

```
struct Node  *search  (struct Node *Root, Item X)
{
        int found=0;
        struct Node *p;
        p = Root;
        while (p != NULL &&  found= =0)
        {
                if  (X > p->infor)
```

```
                p = p->Right;
            else
                if  (X < p->infor)
                    p = p->Left;
                else     found = 1;
        }
        return p;
}
```

2.3.2. Đi qua CNPTK

Như ta đã biết CNPTK cũng là cây nhị phân nên các phép duyệt trên cây nhị phân cũng vẫn đúng trên CNPTK. Chỉ có lưu ý nhỏ là khi duyệt theo thứ tự giữa thì được dãy khoá theo thứ tự tăng dần.

2.3.3. Chèn một nút vào CNPTK

Việc thêm một nút có trường khoá bằng X vào cây phải đảm bảo 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 hiện quá trình tương tự như thao tác tìm kiếm. Khi kết thúc việc tìm kiếm cũng chính là lúc tìm được chỗ cần chèn.

**** Giải thuật lặp***

Trong thủ tục này ta sử dụng biến con trỏ địa phương Q chạy trên các nút của cây bắt đầu từ gốc. Khi đang ở một nút nào đó, Q sẽ xuống nút con trái (hay phải) tuỳ theo khoá ở nút gốc lớn hơn (hay nhỏ hơn) khoá X.

Ở tại một nút nào đó khi P muốn xuống nút con trái (phải) thì phải kiểm tra xem nút này có nút con trái (phải) không. Nếu có thì tiếp tục xuống, ngược lại thì treo nút mới vào bên trái (phải) nút đó. Điều kiện Q = NULL sẽ kết thúc vòng lặp. Quá trình này được lại lặp khi có đỉnh mới được chèn vào.

/ Hàm insert() bổ sung thêm nút có khóa X vào cây, hàm trả về 1 nếu bổ sung thành công, ngược lại hàm trả về 0 – trường hợp nút có khóa X đã có trên cây*/*

```
int  Insert (struct Node **Root, Item X)
{
        struct Node *P, *Q;
        P = new Node;
        P->infor  = X;
        P->Left = NULL;
        P->Right = NULL;
        if  (*Root = = NULL)
        {
            *Root = P;
            return 1;
        }
        else
```

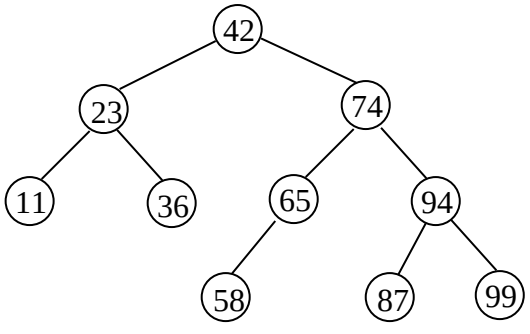
```
    {   Q = *Root;
        while (Q != NULL)
            if  (X < Q->infor)
            {
                    if  (Q->Left  !=  NULL)  Q = Q->Left;
                    else {   Q->Left = P;
                             return 1;
                     }
            }

            else
                    if  (X > Q->infor)
                    {
                                if (Q->Right  != NULL)  Q = Q->Right;
                                else  {
                                        Q->Right  = P;
                                        return 1;
                                }
                    }
                    else return 0;
    }
}
```

Nhận xét:

Để dựng được CNPTK ứng với một dãy khoá đưa vào bằng cách liên tục bổ các nút ứng với từng khoá, bắt đầu từ cây rỗng. Ban đầu phải dựng lên cây với nút gốc là khoá đầu tiên sau đó đối với các khoá tiếp theo, tìm trên cây không có thì bổ sung vào.

Ví dụ với dãy khoá: 42 23 74 11 65 58 94 36 99 87

thì cây nhị phân tìm kiếm dựng được sẽ có dạng ở hình 4.18



Hình 2.18. Một cây nhị phân tìm kiếm

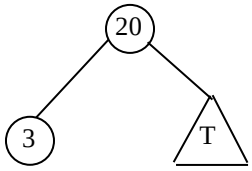
2.3.4. Loại bỏ nút trên cây nhị phân tìm kiếm

Đối lập với phép toán chèn vào là phép toán loại bỏ. Chúng ta cần phải loại bỏ khỏi CNPTK một đỉnh có khoá X (ta gọi tắt là nút X) cho trước, sao cho việc huỷ một nút ra khỏi cây cũng phải bảo đảm điều kiện ràng buộc của CNPTK.

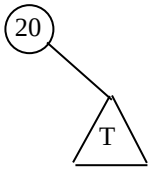
Có ba trường hợp khi huỷ một nút X có thể xảy ra:

- X là nút lá
- X là nút nửa lá (chỉ có một con trái hoặc con phải)
- X có đủ hai con (trường hợp tổng quát)

Trường hợp thứ nhất: chỉ đơn giản huỷ nút X vì nó không liên quan đến phần tử nào khác.



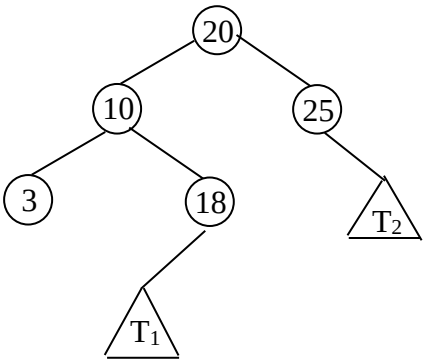
Cây trước khi xoá



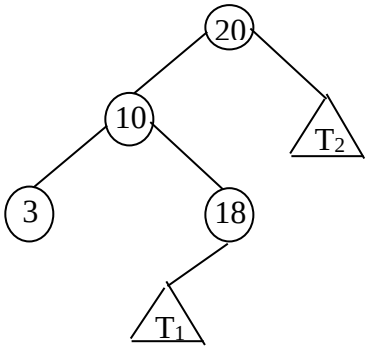
Cây sau khi xoá

Hình 2.19. Xóa nút lá trên cây

Trường hợp thứ hai: Trước khi xoá nút X cần móc nối cha của X với nút con (nút con trái hoặc nút con phải) của nó.



a. Cây trước khi xoá

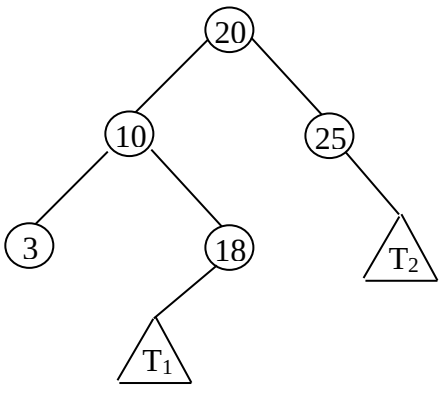


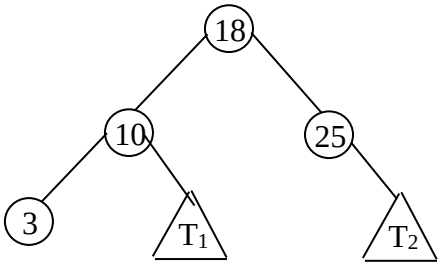
b. Cây sau khi xoá đỉnh (25)

Hình 4.20. Xóa nút nửa lá

Trường hợp tổng quát: khi nút bị loại bỏ có cả cây con trái và cây con phải, thì nút thay thế nó hoặc là nút ứng với khoá nhỏ hơn ngay sát trước nó (nút cực phải của cây con trái nó) hoặc nút ứng với khoá lớn hơn ngay sát sau nó (nút cực trái của cây con phải nó). Như vậy ta sẽ phải thay đổi một số mối nối ở các nút:

- Nút cha của nút bị loại bỏ
- Nút được chọn làm nút thay thế
- Nút cha của nút được chọn làm nút thay thế



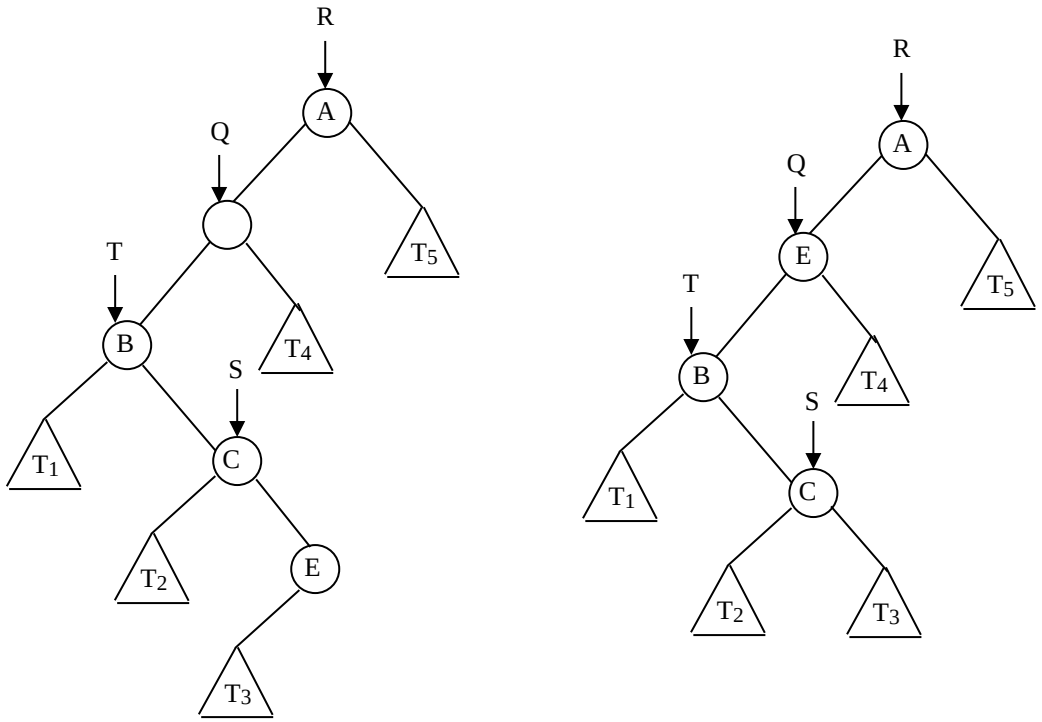


2. Cây sau khi xoá đỉnh 20

Hình 4.21. Xóa nút đầy đủ

Ở đây ta chọn nút thay thế nút bị xoá là nút cực phải của cây con trái (nút 18).

Sau đây là giải thuật thực hiện việc loại bỏ một nút trỏ bởi Q. Ban đầu Q chính là nối trái hoặc nối phải của một nút R trên cây nhị phân tìm kiếm, mà ta giả sử đã biết rồi.



1. Cây trước khi xoá nút trỏ bởi Q 2. Cây sau khi xoá nút trỏ bởi Q

Hình 4.22. Xóa nút trỏ bởi con trỏ Q

```
void Del (struct Node *Q) //xoá nút trỏ bởi Q
{
    struct Node *T,*S, *P;
    P = Q;   //Xử lý trường hợp nút lá và nút nửa lá
    if  (P->Left = = NULL)
    {
        Q=P->Right; //R^.Left = P^.Right
        delete P;
    }
    else
        if  (P->Right = = NULL)
        {       Q =P->Left; delete  P;
```

```
        }
        else    // Xử lý trường hợp tổng quát
        {
            T = P->Left;
            if  (T->Right = = NULL)
            {
                T->Right = P->Right;
                Q = T;
                delete P;
            }
            else
            {
                S = T->Right; //Tìm nút thay thế, là nút cực phải của cây
                while  (S->Right != NULL)
                {
                    T = S;
                    S = T->Right;
                }
                S->Right = P->Right;
                T->Right = S->Left;
                S->Left = P->Left;
                Q=S;
                delete P;
            }
        }
}
```

Thủ tục xoá trường dữ liệu bằng X

- Tìm đến nút có trường dữ liệu bằng X
- Áp dụng thủ tục Del để xoá

Sau đây chúng ta sẽ viết thủ tục loại khỏi cây gốc Root đỉnh có khoá X cho trước. Đó là thủ tục đệ qui, nó tìm ra đỉnh có khoá X, sau đó áp dụng thủ tục Del để loại đỉnh đó ra khỏi cây.

```
void Delete (Item X)
{
   if  (Root != NULL)
        if  (x < Root->infor)   Delete (Root->Left, X)
        else if  (X > Root->infor)  Delete (Root->Right, X)
        else  Del(Root);
```

```
}
```

Nhận xét

Việc huỷ toàn bộ cây có thể thực hiện thông qua thao tác duyệt cây theo thứ sau. Nghĩa là ta sẽ huỷ cây con trái, cây con phải rồi mới huỷ nút gốc.

```
void RemoveTree ()
{
    if (Root != NULL)
    {
        RemoveTree(Root->Left);
        RemoveTree(Root->Right);
        delete Root;
    }
}
```

2.4. Thời gian thực hiện các phép toán trên CNPTK

Trong mục này ta sẽ đánh giá thời gian để thực hiện các phép toán trên CNPTK. Ta có nhận xét rằng, thời gian thực các phép tìm kiếm là số phép so sánh giá trị khoá X cho trước với khoá của các nút nằm trên đường đi từ gốc tới nút nào đó trên cây. Do đó, thời gian thực hiện các phép tìm kiếm, bổ sung và loại bỏ là độ dài đường đi từ gốc tới một nút nào đó trên cây.

Với giải thuật tìm kiếm nêu trên, ta thấy dạng cây nhị phân tìm kiếm dựng được hoàn toàn phụ thuộc vào dãy khoá đưa vào. Như vậy nghĩa là trong quá trình xử lý động ta không thể biết trước được cây sẽ phát triển ra sao, hình dạng của nó sẽ như thế nào.

Trong trường hợp nó là một cây nhị phân hoàn chỉnh (ta goi là cân đối ngay cả khi nó chưa đầy đủ) thì chiều cao của nó là $\lceil \log_2(n+1) \rceil$, nên chi phí tìm kiếm có cấp độ là $O(\log_2 n)$.

Trong trường hợp nó là một cây nhị phân suy biến thành một danh sách tuyến tính (khi mà mỗi nút chỉ có một con trừ nút lá). Lúc đó các thao tác trên cây sẽ có độ phức tạp là O(n).

Người ta chứng minh được rằng số lượng trung bình các phép so sánh trong tìm kiếm trên cây nhị phân tìm kiếm là

$$C_{tb} \approx 1{,}386 \log_2 n$$

Do đó cấp độ lớn của thời gian thực hiện trung bình các phép toán cũng chỉ là $O(\log_2 n)$.