CHƯƠNG 1: ĐỘ PHỨC TẠP CỦA GIẢI THUẬT

- Vòng lặp trong vòng lặp: $O(n^2)$

- Quicksort: O(n * log(n))

* Một số lưu ý:

- Độ phức tạp không có hằng số C, sẽ được bỏ lược đi

VD: $O(2n^2) = O(n^2)$

- Quy tắc cộng: Nếu T1(n) và T2(n) là thời gian thực hiện của hai đoạn chương trình P1 và P2, nối tiếp nhau, thì T(n)=O(max(f(n),g(n)))

CHƯƠNG 2: CÁC KIỂU DỮ LIỆU TRỪU TƯỢNG CƠ BẢN

I. Ngăn xếp:

1. Xóa phần tử:

```
void pop(Stack *pS){
    if(pS-> Top_idx != MaxLength){
        pS-> Top_idx ++;
    }
    else printf("Loi! Ngan xep rong!");
}
```

2. Chèn phần tử:

```
void push(int X, Stack *pS){
    if(pS->Top_idx == 0){
        printf("Loi! Ngan xep day!");
    }
    else{
        pS->Top_idx --;
        pS->Elements[pS->Top_idx] = X;
    }
}
```

II. Hàng đợi:

1. Mảng di chuyển tịnh tiến

- Kiểm tra hàng rỗng: có thể thử return Q.Front == -1; return Q.Rear == -1;

return Q.Rear == -1 && return Q.Rear == -1;

- Kiểm tra đầy: return Q.Rear - Q.Front + 1 == MaxLength

return Q.Front == 0 && Q.Rear == MaxLength - 1

*** Xóa phần tử:**

```
void deQueue(Queue *pQ){
    if (!emptyQueue(*pQ)){
        pQ->Front=pQ->Front+1;
        if (pQ->Front>pQ->Rear)
            makenullQueue(pQ); //Dat lai hang rong
    }else printf("Loi: Hang rong!");
}
```

- Bình thường: có thể không thay đổi, có thể tăng front lên 1, có thể cả 2 về -1
- Nếu rỗng: cả 2 giá trị không thay đổi
- Nếu không rỗng: front luôn thay đổi, rear có thể thay đổi
    + front = rear: cả 2 luôn thay đổi
    + front < rear: front luôn thay đổi, rear không thay đổi

* Lưu ý: Nếu chỉ ghi front = rear thì có thể cả 2 = -1 hoặc != -1 -> có thể thay đổi hoặc không

*** Chèn phần tử:**

```
void enQueue(ElementType X,Queue *pQ){
    if (!fullQueue(*pQ)){
        //Neu khong day
        if (emptyQueue(*pQ)) pQ->Front=0;
        if (pQ->Rear==MaxLength-1){
            //Di chuyen tinh tien ra truoc Front-1 vi tri
            for(int i=pQ->Front;i<=pQ->Rear;i++)
            pQ->Elements[i-pQ->Front]=pQ->Elements[i];
            //Xac dinh vi tri Rear moi
            pQ->Rear=MaxLength- pQ->Front-1;
            pQ->Front=0;
            }
        //Tang Rear de luu noi dung moi
        pQ->Rear=pQ->Rear+1;
        pQ->Elements[pQ->Rear]=X;
    }
    else printf("Loi: Hang day!");
}
```

- Bình thường: front có thể thay đổi, rear có thể thay đổi

- Chưa đầy: front có thể thay đổi, rear luôn luôn thay đổi

    + Rỗng: Front luôn thay đổi, rear luôn thay đổi

    + Tràn: front luôn thay đổi, rear luôn luôn thay đổi

    + Không phải 2 cái trên: rear luôn luôn thay đổi, front không thay đổi

- Đầy: do nothing

Ngoài ra có dạng như sau:

- Không đầy không rỗng: Front có thể thay đổi, Rear luôn luôn thay đổi

- Không đầy không rỗng không tràn: Front luôn luôn không đổi, Rear luôn luôn thay đổi

- Không rỗng: Front có thể thay đổi, Rear có thể thay đổi

- Chỉ ghi Rear = Maxlenght – 1: front có thể thay đổi, rear có thể thay đổi

2. Hàng đợi mảng vòng

- Kiểm tra hàng rỗng: tương tự
- Kiểm tra hàng đầy: return (Q.Rear - Q.Front + 1 % MaxLength == 0)

  return Q.Rear == Q.Front - 1

*** Xóa phần tử:**

```
void deQueue(Queue *pQ){
    if(!emptyQueue(*pQ)){
        if(pQ->Front==pQ->Rear)
            makenullQueue(pQ);
        else
        pQ->Front = (pQ->Front + 1) % MaxLength;

    }
    else    printf("Loi: Hang rong!");

}
```

- Bình thường: có thể không thay đổi, có thể tăng chỉ front thay đổi, có thể cả 2 về -1

- Nếu rỗng: cả 2 giá trị không thay đổi

- Nếu không rỗng: front luôn thay đổi, rear có thể thay đổi

    + front = rear: cả front và rear luôn thay đổi

+ front != rear: Front luôn thay đổi, rear không thay đổi

* Lưu ý: Nếu chỉ ghi front = rear thì có thể cả 2 = -1 hoặc != -1 -> cả 2 có thể thay đổi

*** Chèn phần tử:**

```
void enQueue(ElementType X,Queue *pQ){
    if (!fullQueue(*pQ)){
        if (emptyQueue(*pQ)) pQ->Front=0;
        //Tang Rear de luu noi dung moi
        pQ->Rear= (pQ->Rear+1)%MaxLength ;
        pQ->Elements[pQ->Rear]=X;
    }
    else printf("Loi: Hang day!");

}
```

- Chưa đầy: front có thể thay đổi, rear có thể thay đổi

    + Rỗng: Front luôn thay đổi, rear luôn thay đổi

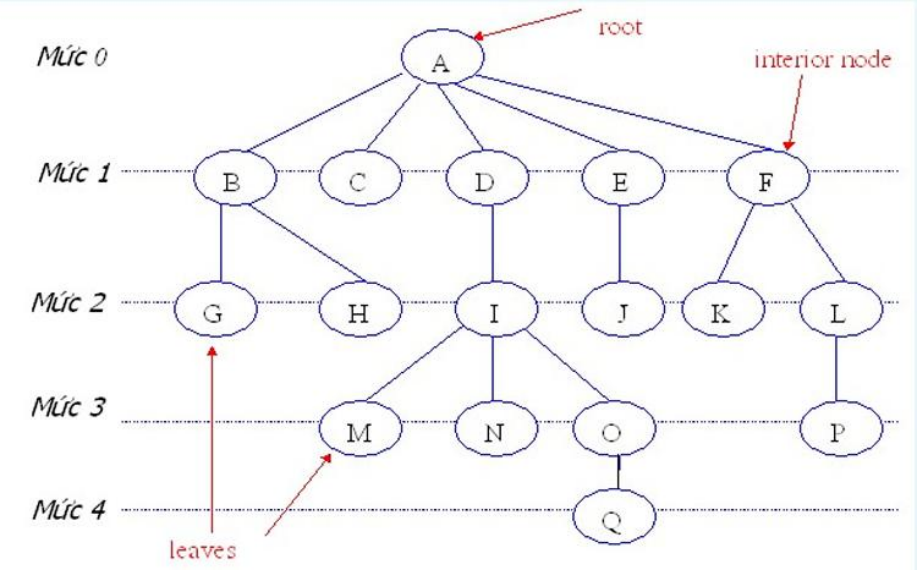
    + Không rỗng: front không thay đổi, rear luôn thay đổi,

- Đầy: do nothing

CHƯƠNG 3: CÂY TÌM KIẾM NHỊ PHÂN

I. Cây tổng quát

1. Lý thuyết



- Nút gốc là nút trên cùng (không có cha)

- Nút lá là nút không có con

- Nút trung gian không phải là hai cái trên

- Mức là số tầng

- Nút cha con: nút A là cha của nút B khi nút A ở mức i và nút B ở mức i+1, đồng thời giữa A và B có cạnh nối.

- Bậc của nút là số nút con của nút đó: A có bậc 5, C có bậc 0, I có bậc 3

- Bậc của cây là bậc lớn nhất của các nút trong cây

- Cây n phân là cây có bậc n.

- Đường đi = số nút – 1.

- Tiền bối, hậu duệ: trên đường đi nếu a là tổ tiên nút b thì a gọi là tiền bối của b, b gọi là hậu duệ của D

- Chiều cao là độ dài đường đi từ nút đó đến nút xa nhất: D có chiều cao là 3.

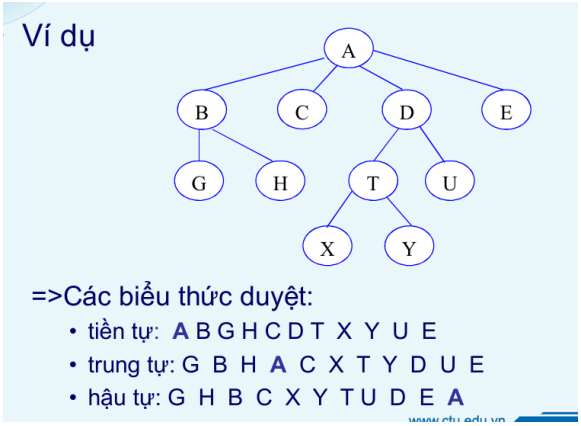
- Chiều cao của cây là chiều cao của nút gốc: Chiều cao của cây là 4.

- Độ sâu của một nút là độ dài đường đi từ nút gốc đến nút đó, hay còn gọi là mức (level) của nút đó: I có độ sâu 2

- Nhãn là giá trị của nút đó

- Những nút cùng cha gọi là nút anh em (siblings)

- Mở rộng: nếu n_i và n_k là hai nút anh em ruột và nút n_i ở bên trái nút n_k thì các hậu duệ của nút n_i là bên trái mọi hậu duệ của nút n_k. n_i được gọi là anh em ruột phải của các n_k



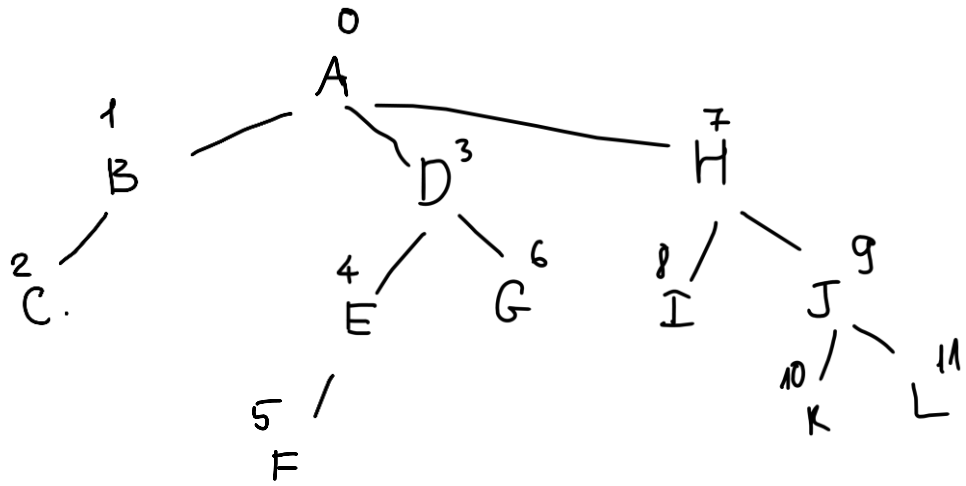
2. Cây tổng quát cài đặt bằng mảng

16. Cho cây tổng quát T2 được cài đặt bằng mảng như sau:
(cây T2 dùng chung cho 5 câu bên dưới)

MaxNode-1

0	1	2	3	4	5	6	7	8	9	10	11	…		Chỉ số mảng
A	B	C	D	E	F	G	H	I	J	K	L			Data
-1	0	1	0	3	4	3	0	7	7	9	9			Parent

* Ý của hàng parent trong bảng trên tức, ví dụ: B, D, H đều có chỉ số parent bằng 0 tức là cả ba label trên đều là con của chỉ số mảng 0, tức là A



II. Cây nhị phân

- Số bước cần duyệt tìm số không có trên cây tính cả khi nó tìm được NULL

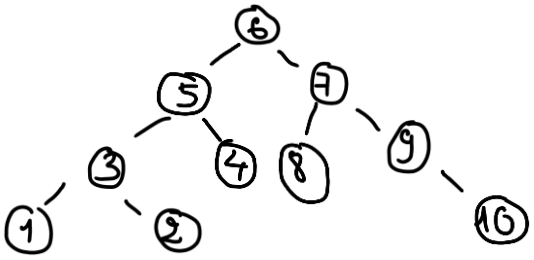
III. Dạng bài chuyển từ tiền tự (hoặc hậu tự) và trung tự tạo thành cây nhị phân:

LNR: 1,3,2,5,4,6,8,7,9,10 (trung tự)

LRN: 1,2,3,4,5,8,10,9,7,6 (hậu tự)

Nếu có tiền tự thì duyệt tiền tự: từ trái sang phải

Nếu có hậu tự thì duyệt hậu tự: từ phải sang trái



IV. Bài tập dạng biểu thức

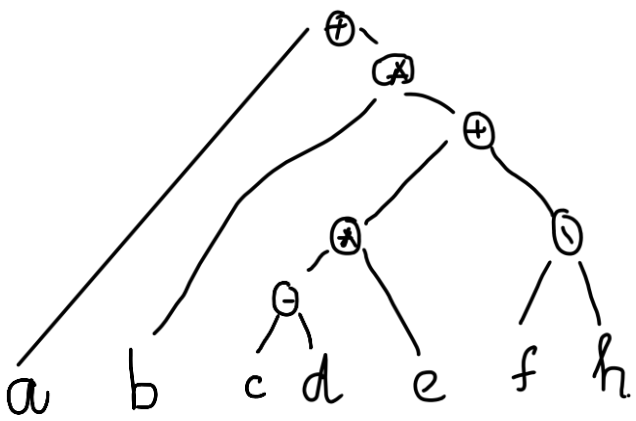
1. Dạng 1

VD: Cho biểu thức a + b*((c - d)*e + f/h). Danh sách duyệt tiền tự của biểu thức đã cho là

* Một số lưu ý:

- Ưu tiên trong ngoặc trước

- Nhân chia trước, cộng trừ sau



2. Dạng 2

VD: Giá trị biểu thức tiền tố /, *, +, 1, 2, -, 3, 4, -, 5, -, 8, 7 là

Chú ý: Ngược lại với duyệt tiền tự, tiền tố thì đọc từ phải sang trái, bỏ vào ngăn xếp, phép tính được thực hiện đúng với thứ tự rút nó ra, ví dụ:

-
8
7

=> 8 – 7 = 1

- Hậu tố thì ngược lại với tiền tố là duyệt từ trái sang phải, phép tính được thực hiện ngược lại với thứ tự rút ra, ví dụ:

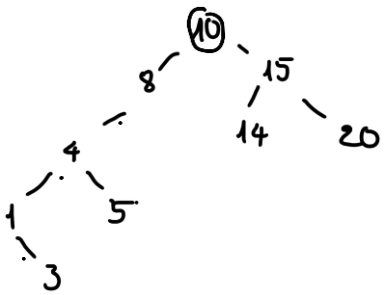
-
2
1

=> 1 - 2 = -1

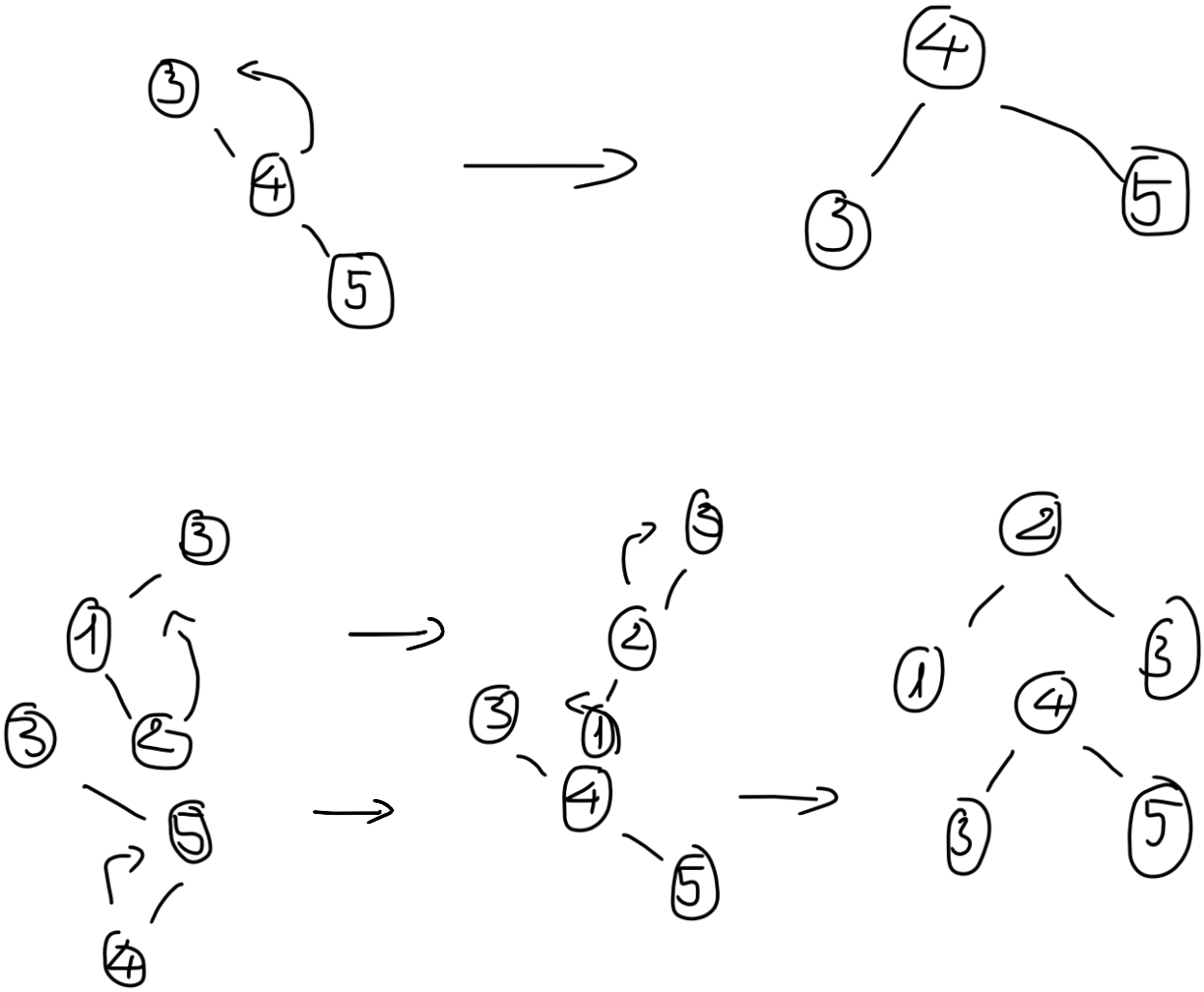
CHƯƠNG 4: CÂY AVL

*** Dạng bài:**

- Cho danh sách các phần tử đưa vào, biến đổi thành cây nhị phân: 10, 15, 8, 4, 20, 5, 1, 3, 14, 15 là



* Các kĩ thuật xoay cây:



- Cân bằng hoàn toàn là cân bằng theo số nút
- Cân bằng tương đối là cân bằng theo chiều cao

CHƯƠNG 5: TỰ ĐIỂN

* Dạng bài:

25. Cho bảng băm đóng với số bucket B=10 và hàm băm h(x)= x mod B và giải quyết đụng độ bằng phương pháp băm lại tuyến tính (h(x)= (x+i) mod B). Kết quả bảng băm sau khi thực hiện các thao tác thêm 3, 5, 9, 15, xoá 5, thêm 26, thêm 30, xoá 3 là:

0	30
1	
2	
3	3
4	
5	15
6	26
7	15
8	
9	9

a.

0	30
1	
2	
3	Deleted
4	
5	Deleted
6	15
7	26
8	
9	9

b.

0	30
1	
2	
3	Deleted
4	
5	Deleted
6	26
7	15
8	
9	9

c.

0	30
1	
2	
3	Deleted
4	
5	26
6	15
7	
8	
9	9

d.

* Một số lưu ý:

- Khi thêm:

+ Nếu chỉ số đó đã tồn tại giá trị rồi thì băm lại tuyến tính

- Khi xóa:

+ Xóa thì để chữ D để khi mà tìm không bị tìm sai (không tìm thấy). Sau đó thì có thể chèn thêm phần tử mới vào bình thường

- Khi tìm:

+ Số bước tìm = số chỉ số nhảy đến khi tìm được (hoặc thấy E)