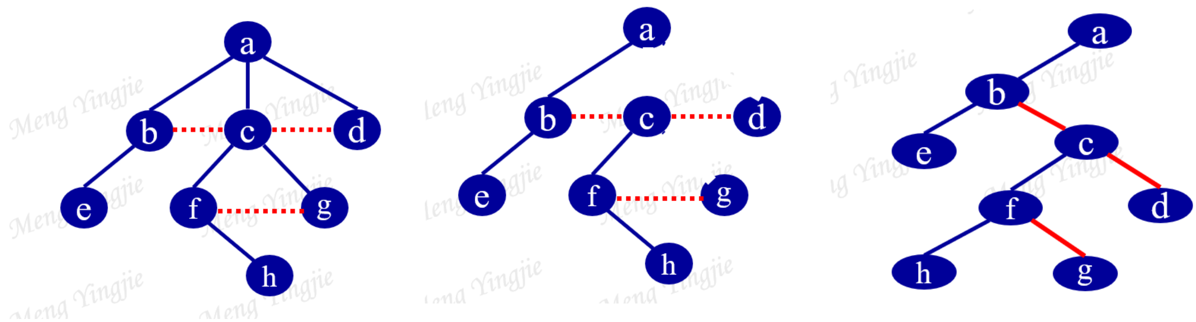
第二层面：

1.稀疏矩阵存储方法：三元组、十字链表 √

2.树、森林、二叉树相互转换

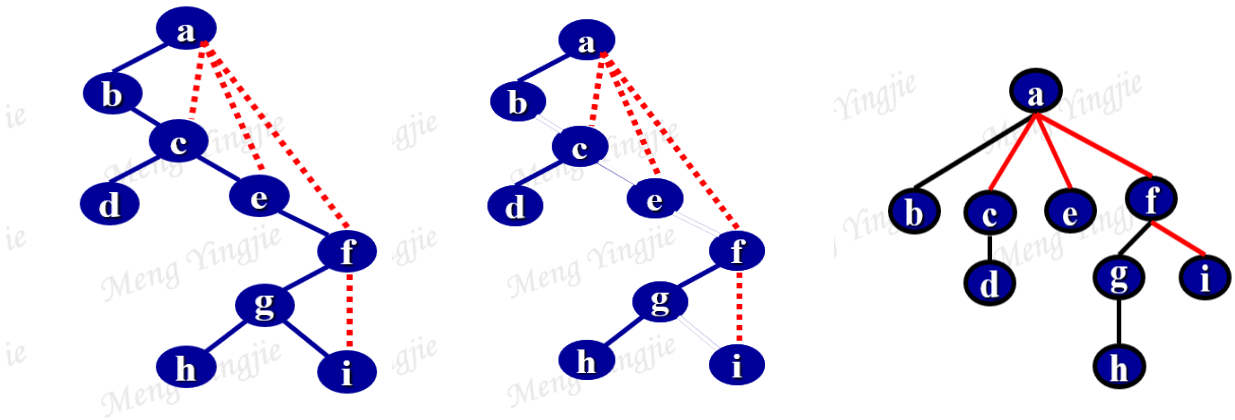
树➞二叉树转换：

1.加线 2.抹掉 3.调整



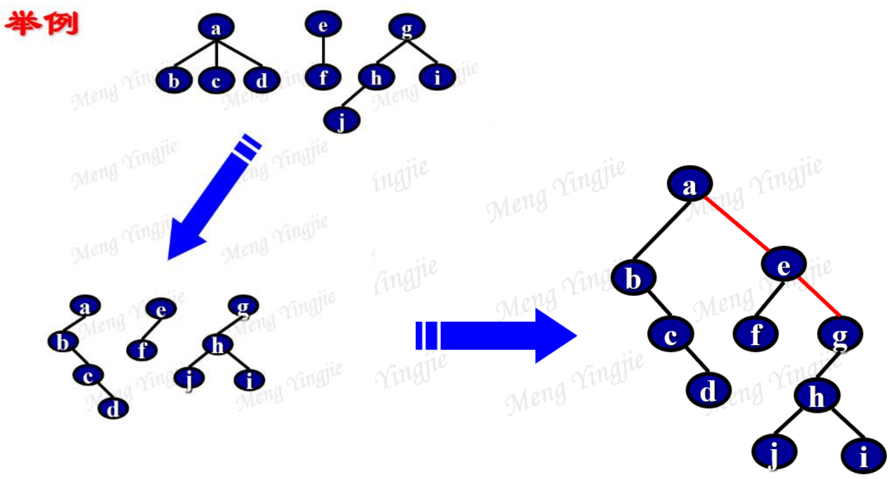
二叉树➞树转换：

1.加线 2.抹线 3.调整



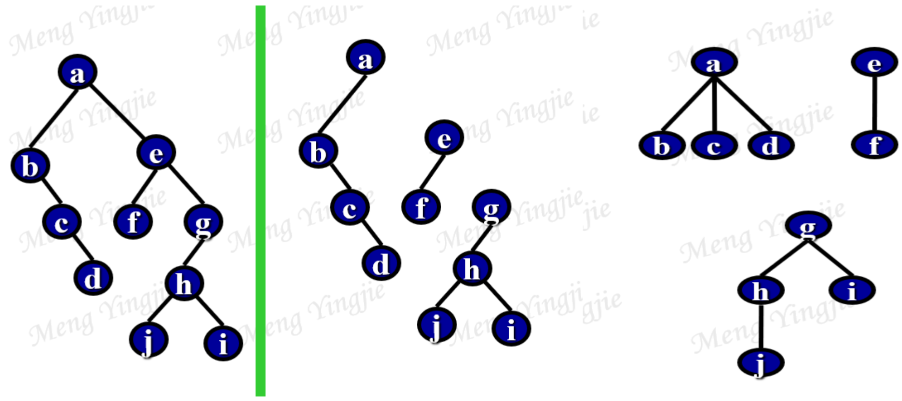
森林➞二叉树转换：

1. 树转二叉树 2.连接，后一棵作为前一棵右孩子



二叉树➞森林转换：

1. 抹线（沿根节点不断切断右孩子） 2.还原



3.二叉树的前中后三种遍历方法 √

4. 加线索 √

$$\text{tag}=\begin{cases}0,\ \text{表示该指针指向孩子。}\\ 1,\ \text{表示该指针为线索，指向前驱或后继。}\end{cases}$$

5. 二叉排序树的构造 √

6. Huffman 树的构造

Huffman算法的基本思想：

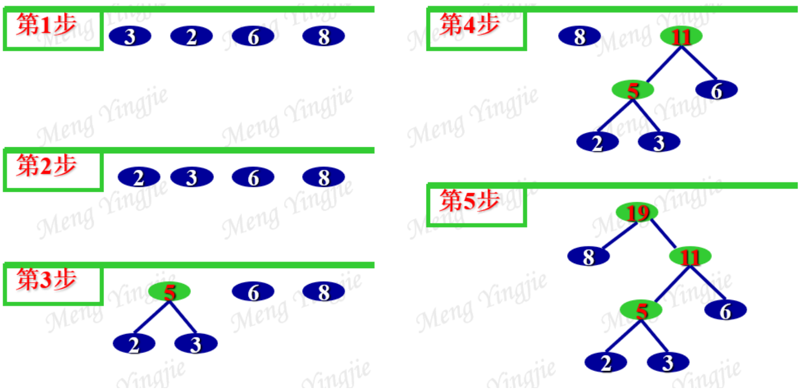
①给定一组权值集合$\{w_1, w_2, \ldots, w_n\}$.据此构成n棵二叉树组成的森林F。

注：F中的每棵二叉树只有一个带权值为$W_i(1\le i\le n)$的根结点。

②将$F=\{T_1, T_2, \ldots, T_n\}$按根结点的值由小到大进行排序。

③取出T_1和T_2组成一棵二叉树T；再将T插入到F中，并使F依据根结点的值有序。

④反复执行③直到F={T}为止。



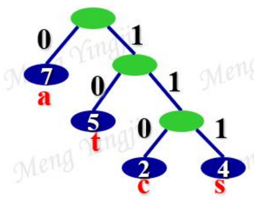
7.Huffman 编码

例,有下面正文，需对其编码，要求总编码长度最短。

cast cats sat at a tasa

符号集D=(c,a,s,t),对应的每个字符出现次数为WD=(2,7,4,5)

依据WD=(2,7,4,5)构造Huffman树，结果如下：



对该二叉树编码，约定左0，右1，则：

字符编码 a: 0

t: 10

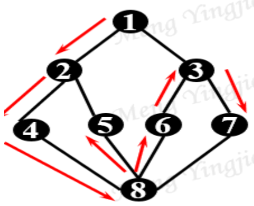
s: 111

c: 110

8. 邻接矩阵、邻接表 √

9. 深度和广度的遍历原理

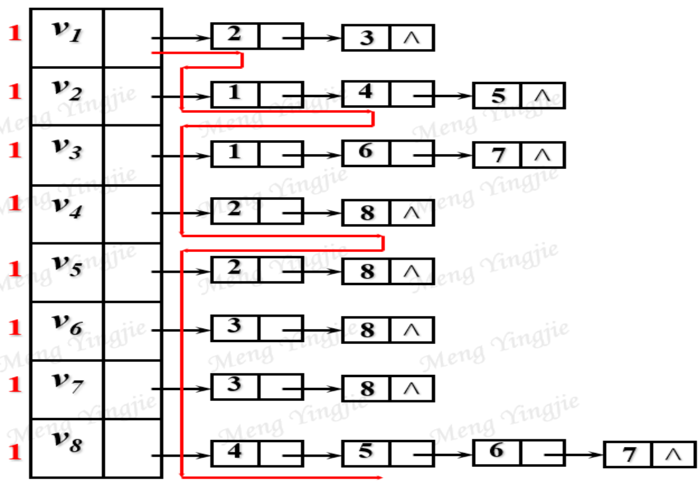
例：

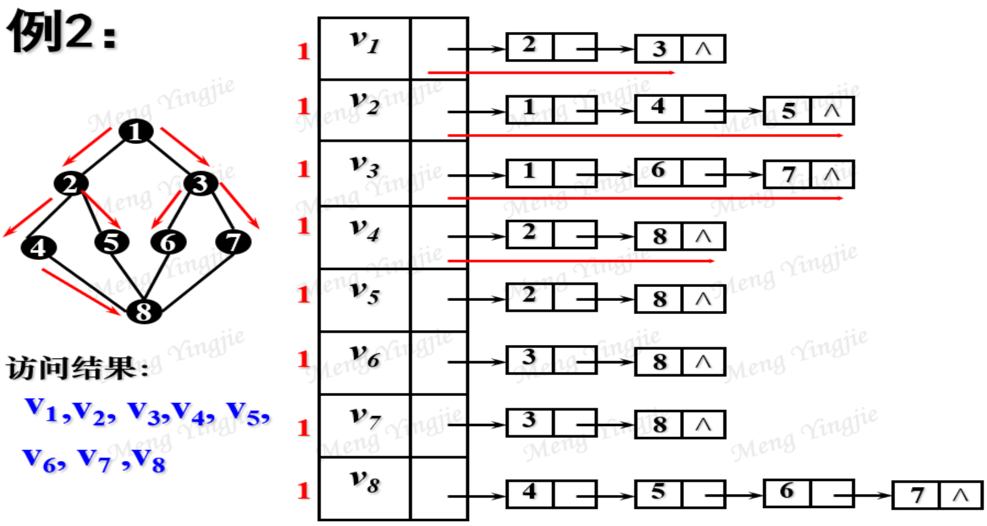


访问结果：

$v_1, v_2, v_4, v_8,$

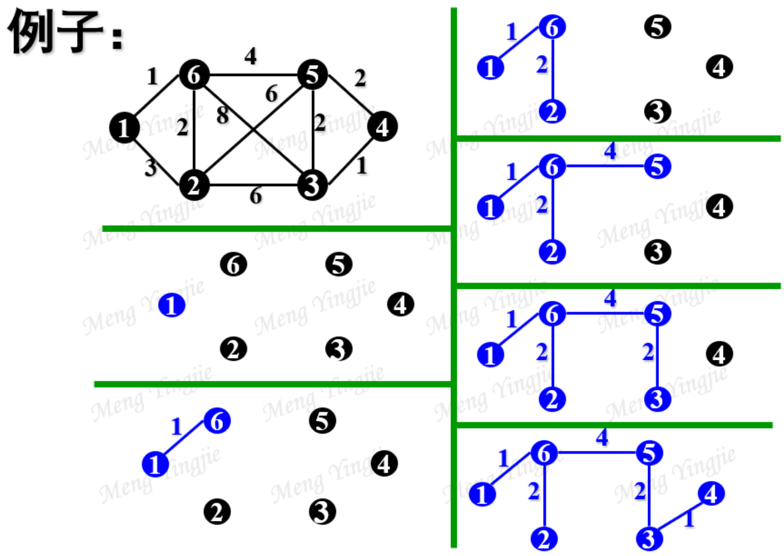
v_5, v_6, v_3, v_7





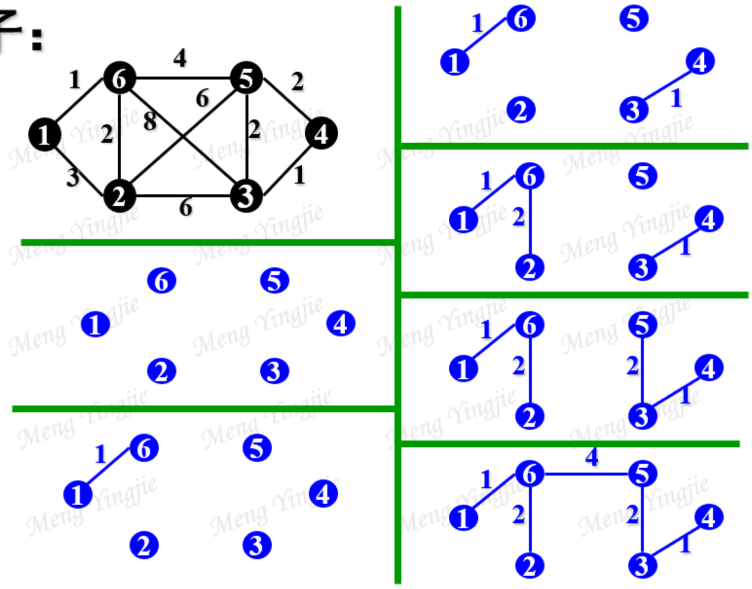
10. 用 Prim 算法和 Kruskal 算法构造最小生成树

Prim



Kruskal

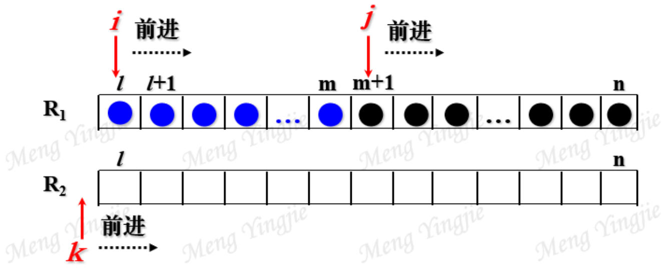
例子：



11. 直接插入、2-路归并、快速排序、基数排序

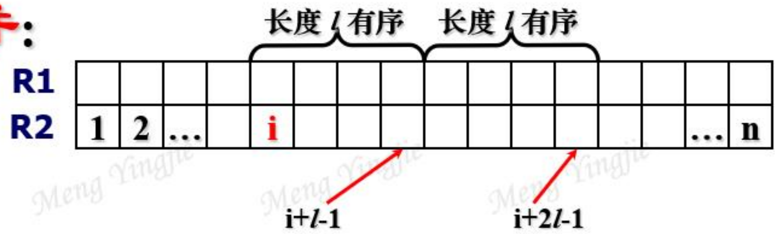
直接插入：√

2-路归并：



PROC Merge(**VAR** R_1,R_2:**ARRAY**[1..max] **OF** datatype;*l*,m,n:integer);

2.一趟合并:




```
PROC  MergePass(VAR R1,R2:ARRAY[1..max] OF datatype;l,n:integer);
{R1[1..n],由长度 l 的有序段构成，进行一趟合并使其变为长度为2l 有序}
BEGIN  i←1;
    WHILE (i≤n-2l+1) DO {对完整2l 段进行合并}
       【 CALL  Merge(R1,R2, i , i+l-1 , i+2l-1);
          i ←i +2l 】;
    IF  i+l-1＜n  {两两合并后，最后所剩段的段长是否超过l }
    THEN  Merge(R1,R2, i , i+l-1 , n) {合并最后不足2l 部分 }
    ELSE  CALL  COPY(R1,i,n,R2) {抄写不足l 长度的部分 }
END;
```

快速排序:

在待排序的n个记录中任取一个记录r(例如就取第1个)，作为**轴心元素**；

以r为标准将所有记录分为两组;

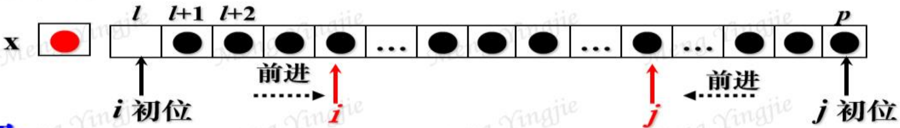
第1组中各记录的关键字都小于r的关键字;

第2组中各记录的关键字都大于r的关键字;

并把r排在这两组中间(最终位置),这一过程称为**一趟快排**。

3.一趟快排描述

初始准备:



处理:

(1).当(i<j)且R[j].key＞x.key时，j指针前进;

(2).如果i＜j，则:

①R[j]⇒R[i]; i+1⇒i

②当(i<j)且R[i].key＜x.key时，i指针前进;

③如果 i＜j ，则: R[i]⇒R[j]; j-1⇒j

(3) 如果 i=j ，则本趟结束，否则转(1)

(4)以上结束后: x ⇒ R[i]

基数排序:

基数排序(radix sort)就是按照LSD法对单逻辑关键字进行排序的一种方法，也称桶排序。

把每个关键字看成d元组：

$K_i=(k^1{}_i, k^2{}_i, \ldots, k^d{}_i)$

其中：$C_0 \le k^j{}_i \le C_{r-1}$ $(1 \le i \le n, 1 \le j \le d)$

排序时先按$k^d{}_i$的值从小到大将记录分配到r个盒子中去，然后依次收集这些记录；

再按$k_i{}^{d-1}$的值从小到大将记录分配到r个盒子，如此反复，直到对$k^1{}_i$的分配和收集后就得到完全有序的文件。

其中r称为基数，执行基数排序时，为了实现分配和收集，需设立r个队列。

12.堆的构造:

（1）插入建堆:（往上爬）

插入时的调整过程：

```
PROC  InsertHeap(VAR A:ARRAY[1..max] OF datatype;n:integer);
BEGIN   i←n;
        WHILE  i＞1  DO
           IF  (A[i DIV 2].key＜A[i].key)
           THEN  exit;
           ELSE【 swop(A[i DIV 2], A[i])  {交换数组两个元素}
                   i← i DIV  2 】
END;
```

插入建堆算法过程：

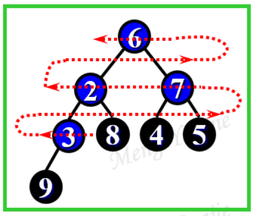
```
PROC  InsBuildHeap(VAR A:ARRAY[1..max] OF datatype;n:integer);
{建立有n个元素的堆}
BEGIN    read(A[1]);
       FOR  i←2  TO   n   DO
            【 read(A[i]);
               InsertHeap(A,i)】 ;
END;
```

时间复杂度:$O(nlog_2n)$

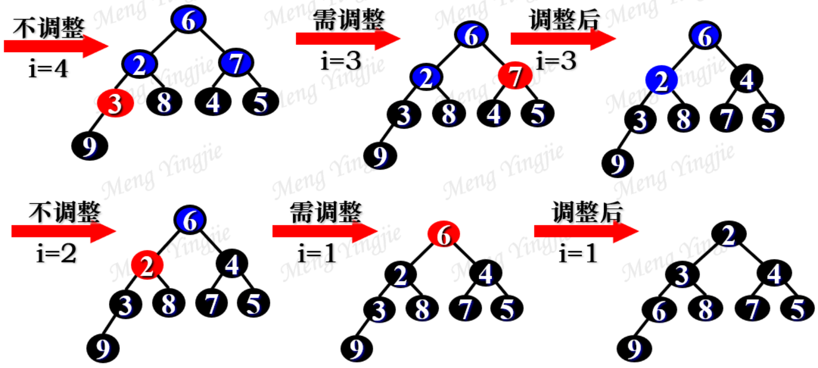
（2）筛选建堆：（向下退）

筛选的处理过程：

```
PROC  HeapRectify(VAR A:ARRAY[1..max] OF datatype;i,k:integer);
{i位堆顶，k为堆底}
BEGIN
      WHILE  2i+1≤k  DO
      【 min(A[2i], A[2i+1], m); {m为A(2i)A(2i+1)中较小者下标}
        IF (A[i].key＞A[m].key)  THEN  swap(A[i], A[m])
        ELSE  exit ;
        i←m;
      】
      IF (2i=k) AND (A[i].key＞A[k].key) THEN  swap(A[i],A[k])
END;
```



n=8，故： *i* 从⌊n/2⌋=4 **DOWNTO** 1 **DO** 筛选



筛选建堆算法过程：

```
PROC  LineBuildHeap(VAR A:ARRAY[1..max] OF datatype;n:integer);
{元素存储于A[1..max],建立一个有n个元素的堆}
BEGIN
      FOR  i←1  TO  n  DO
           read(A[i]);
      FOR  i←n DIV 2  DOWNTO  1  DO
           HeapRectify(A,i,n)
END;
```

时间复杂性O(n).(证明略)

13.AVL-树的构造：

AVL-树：①一棵空二叉树是AVL-树；②若T是一棵非空二叉树，其任何结点的左、右子树的高度相差不超过1，则T是AVL-树。

结点的平衡因子：结点的左子树高度减去右子树的高度的差值。

最优二叉排序树结点的平衡因子取值范围：{-1, 0 ,1}。

平衡规则：

选择离插入(或删除)结点最近的不平衡结点(其平衡因子为±2)开始调整。以下讨论以插入情况为例：

平衡类型：设结点A的平衡因子为±2

LL型：新结点插在 A 的 左子树的 左子树中导致不平衡；

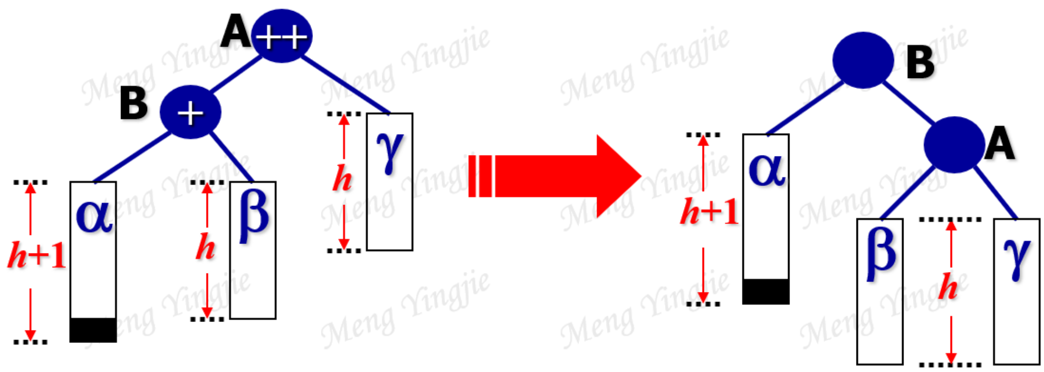
RR型：新结点插在 A 的 右子树的 右子树中导致不平衡；

LR型：新结点插在 A 的 左子树的 右子树中导致不平衡；

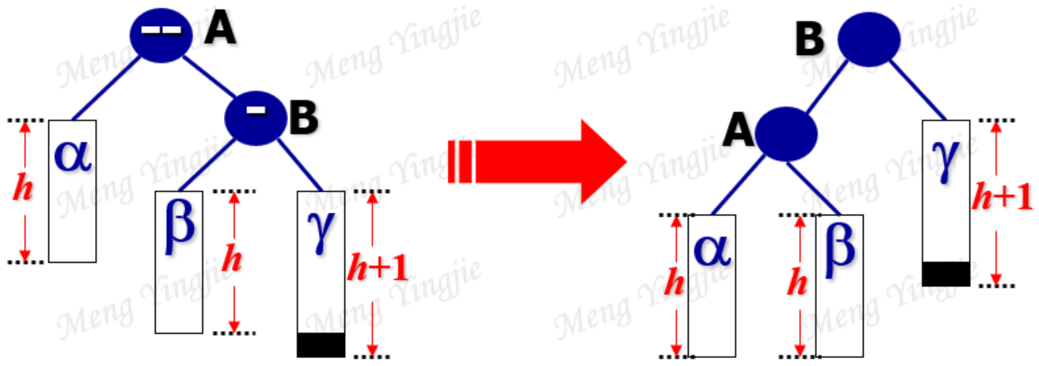
RL型：新结点插在 A 的 右子树的左 子树中导致不平衡.

LL、RR 同一类型；LR、RL 同一类型

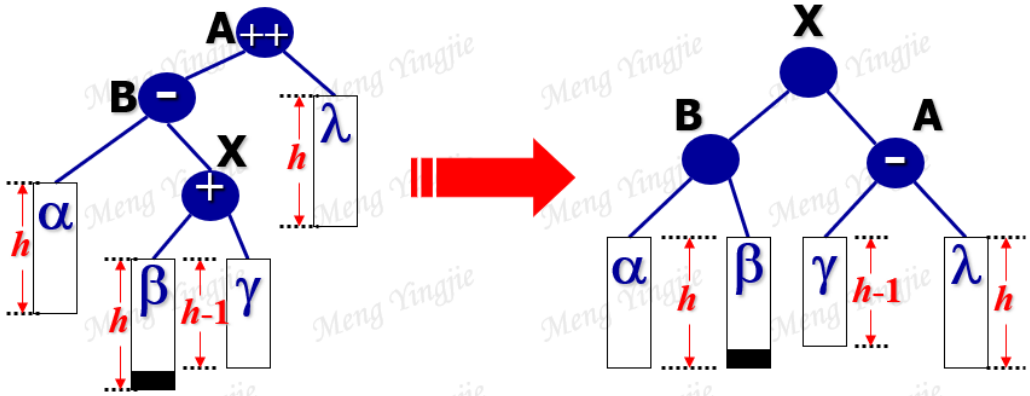
LL 型：A 是最近的不平衡结点，A 的左子树 B 成为根节点，B 的右子树成为 A 的左子树



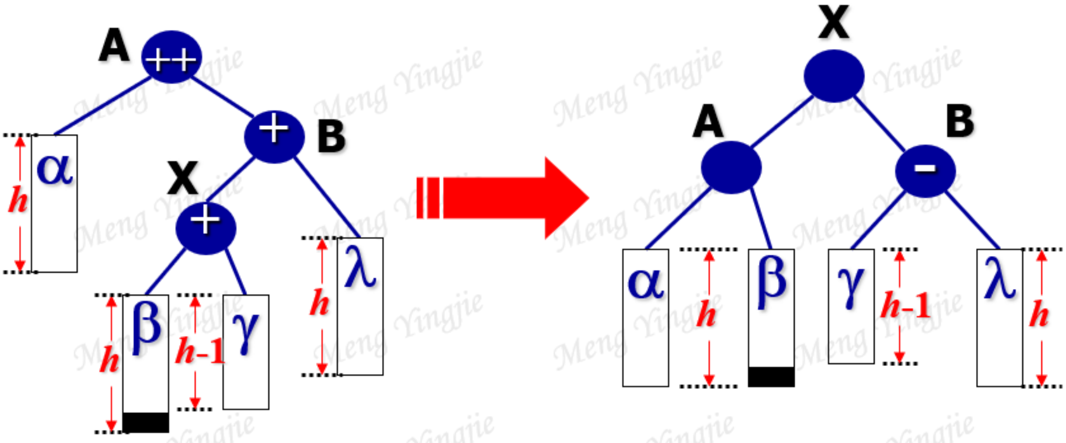
RR 型：A 是最近的不平衡结点，A 的右子树 B 成为根节点，B 的左子树成为 A 的右子树



LR 型：X 作为根节点，B 成为 X 的左子树，A 成为 X 的右子树，X 的左子树成为 B 的右子树，X 的右子树成为 A 的左子树

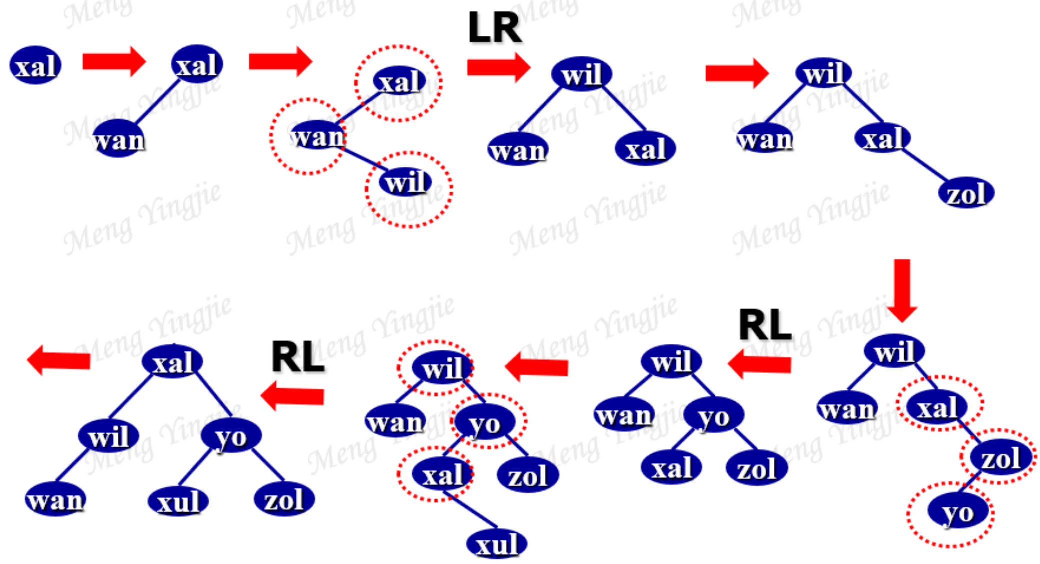


RL 型：X 作为根节点，A 成为 X 的左子树，B 成为 X 的右子树，X 的左子树成为 A 的右子树，X 的右子树成为 B 的左子树



实例：

4.举例： 有一个关键字集合K={xal,wan,wil,zol,yo, xul, yum, wen, wim}对其构造一棵最优二叉排序树。

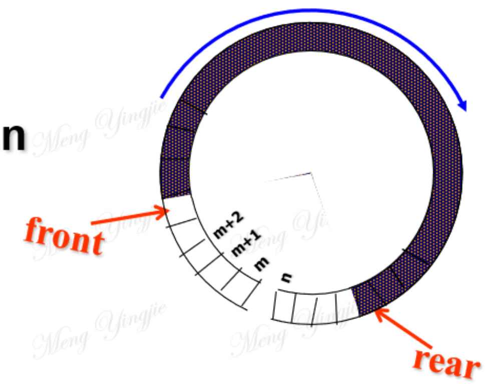


第三层面：

1. 链表的插入和删除问题
2. 循环队列的出入队[m..n]编址和[0..n-1]编址

4. 循环队列

若空间编址范围：m..n



存储结构定义实际上就是一个一维数组：

VAR CQ: **Array** [m..n] **OF** datatype;

front,rear: integer;

1.编址方式[m…n]

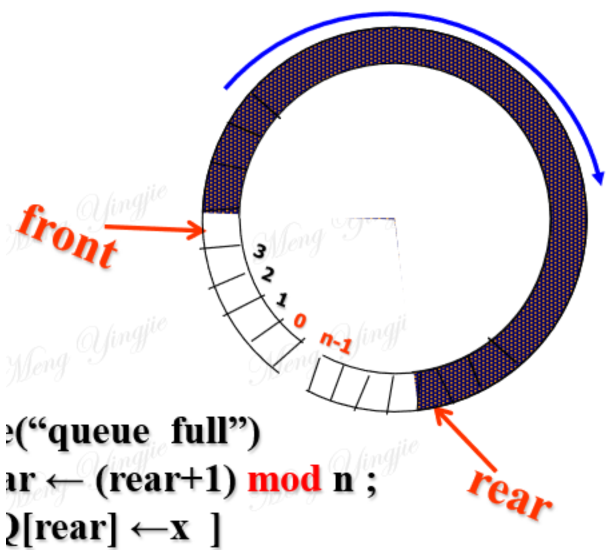
（1） 入队 AddQ

```
proc AddQ(CQ,x,front,rear)
begin
    rear ← rear+1;
    if rear = n+1 then rear ← m;
    if rear = front then write('full');
                    else CQ[rear] ← x;
end;
```

（2） 出队 DelQ

```
proc DelQ(CQ,x,front,rear)
begin
    if rear = front then write('empty');
    else
    [
        if front = n then front ← m;
                     else front ← front+1;
        x ← CQ[front];
    ]
end;
```

2.编址方式[0…n-1]



(1)　入队 AddQ

```
proc AddQ(CQ,x,front,rear)
begin
    if (rear+1)mod n = front then write('full');
    else
    [
        rear ← (rear+1)mod n;
        CQ[rear] ← x;
    ]
end;
```

(3)　出队 DelQ

```
proc DelQ(CQ,x,front,rear)
begin
    if rear = front then write('empty');
    else
    [
        front ← (front+1)mod n;
        x ← CQ[front];
    ]
end;
```

3.　二分插入

```
proc BinaryInsert(var R:array[1...n] of datatype)
begin
    for i←2 to n do
    [
        x ← R[i];
        low = 1;
        high = i-1;
        while low<=high do
        [
            mid = (low+high)div 2;
            if x.key < R[mid].key then
                high ← mid-1;
            else low ← mid+1;
        ]
        for j ← i-1 to l do
            R[j+1] ← R[j];
        R[l] ← x;
    ]
end;
```