# ***一、前言***

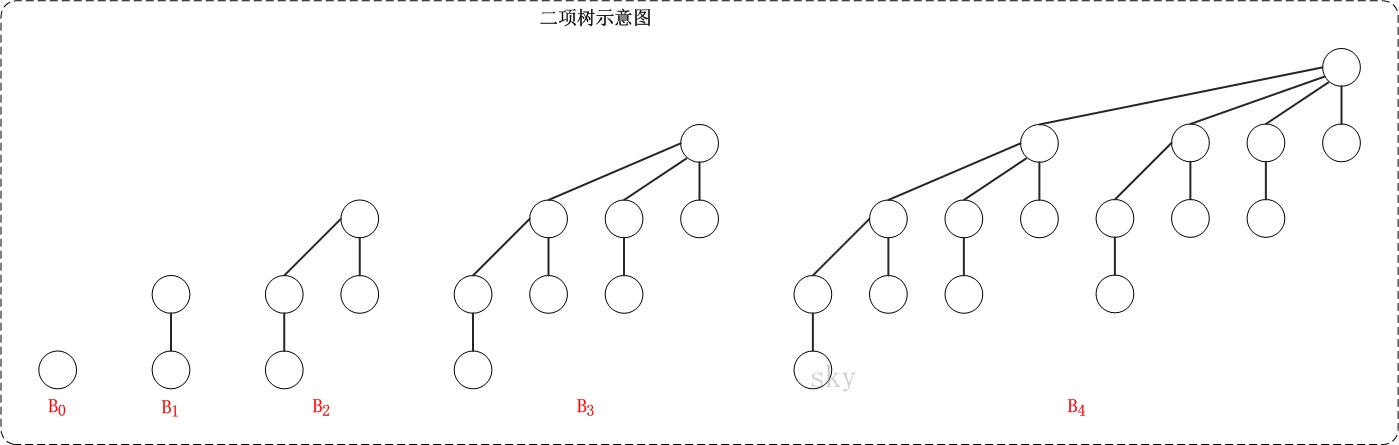
***二项堆与二叉堆的区别：***

1. ***存储结构不一样***
2. ***二项堆进行两个堆的合并操作比二叉堆的时间复杂度小***
3. ***堆其实就是优先队列***

# **二、二项树的定义**

二项堆是二项树的集合。在了解二项堆之前，先对二项树进行介绍。

二项树是一种递归定义的有序树。它的递归定义如下：  
(01) 二项树B0只有一个结点；  
(02) 二项树Bk由两棵二项树B(k-1)组成的，其中一棵树是另一棵树根的最左孩子。  
如下图所示：

[](http://images.cnitblog.com/i/497634/201404/101009066844094.jpg)

上图的B0、B1、B2、B3、B4都是二项树。对比前面提到的二项树的定义：B0只有一个节点，B1由两个B0所组成，B2由两个B1所组成，B3由两个B2所组成，B4由两个B3所组成；而且，当两颗相同的二项树组成另一棵树时，其中一棵树是另一棵树的最左孩子。

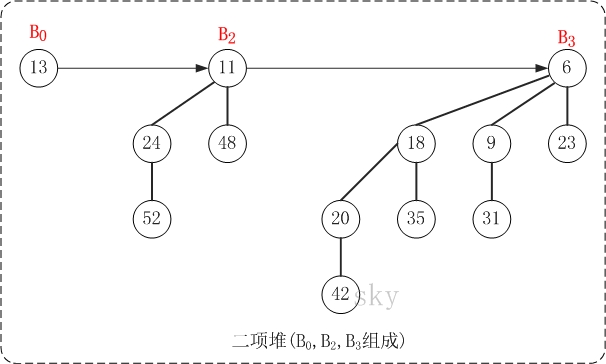
# ****三、二项树的性质****

二项树有以下性质：  
[性质一] Bk共有2k个节点。  
[性质二] Bk的高度为k。  
[性质三] Bk在深度i处恰好有C(k,i)个节点，其中i=0,1,2,...,k。  
[性质四] 根的度数为k，它大于任何其它节点的度数。  
注意：树的高度和深度是相同的。关于树的高度的概念，《算法导论》中只有一个节点的树的高度是0，而"维基百科"中只有一个节点的树的高度是1。本文使用了《算法导论中》"树的高度和深度"的概念。

下面对这几个性质进行简单说明：  
[性质一] Bk共有2k个节点。  
               如上图所示，B0有20=1节点，B1有21=2个节点，B2有22=4个节点，...  
[性质二] Bk的高度为k。  
               如上图所示，B0的高度为0，B1的高度为1，B2的高度为2，...  
[性质三] Bk在深度i处恰好有C(k,i)个节点，其中i=0,1,2,...,k。  
              C(k,i)是高中数学中阶乘元素，例如，C(10,3)=(10\*9\*8) / (3\*2\*1)=240  
              B4中深度为0的节点C(4,0)=1  
              B4中深度为1的节点C(4,1)= 4 / 1 = 4  
              B4中深度为2的节点C(4,2)= (4\*3) / (2\*1) = 6  
              B4中深度为3的节点C(4,3)= (4\*3\*2) / (3\*2\*1) = 4  
              B4中深度为4的节点C(4,4)= (4\*3\*2\*1) / (4\*3\*2\*1) = 1  
             合计得到B4的节点分布是(1,4,6,4,1)。  
[性质四] 根的度数为k，它大于任何其它节点的度数。  
              节点的度数是该结点拥有的子树的数目。

# 四、二项堆的介绍

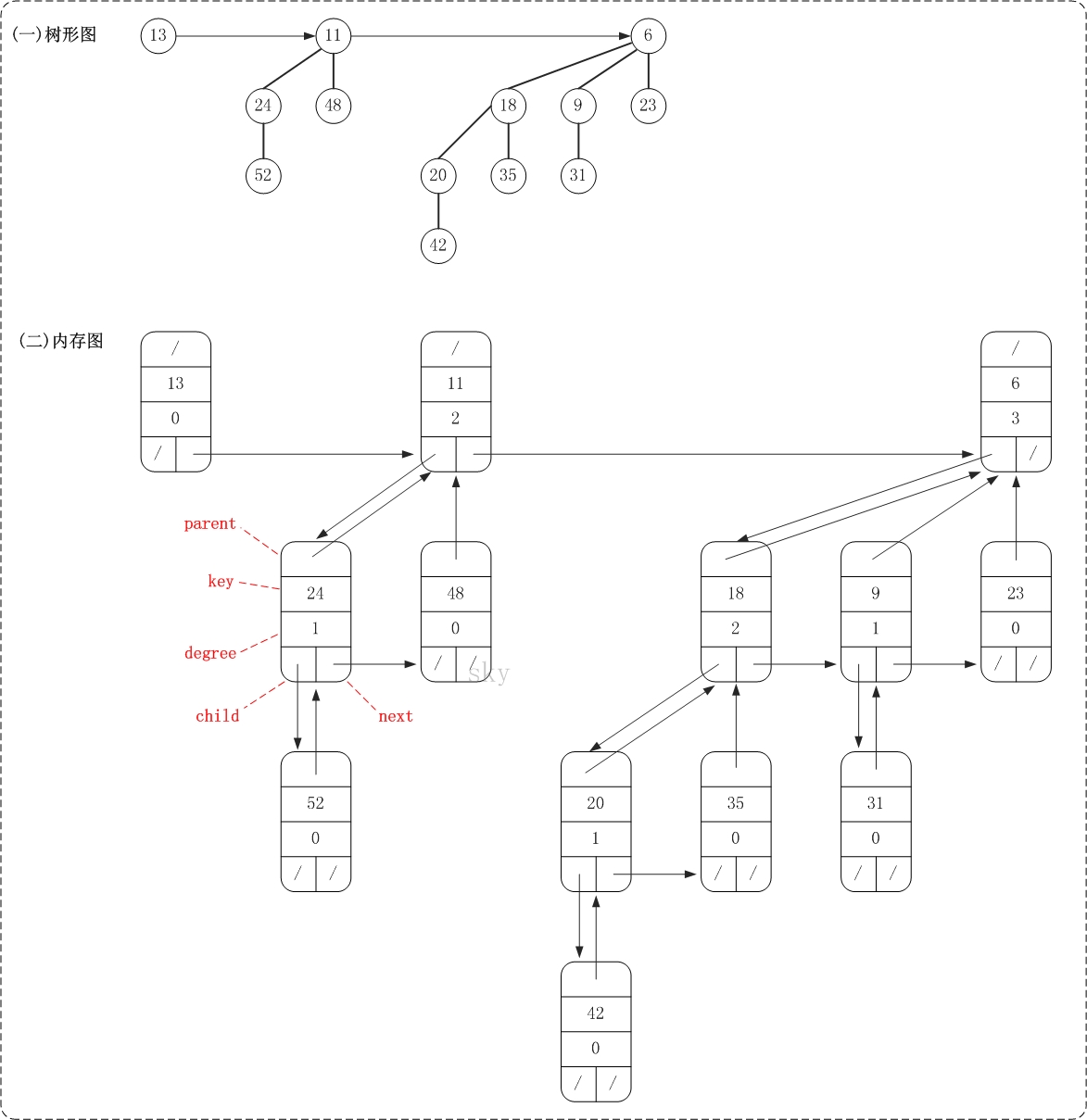
二项堆通常被用来实现优先队列，它堆是指满足以下性质的二项树的集合：  
(01) 每棵二项树都满足最小堆性质。即，父节点的关键字 <= 它的孩子的关键字。  
(02) 不能有两棵或以上的二项树具有相同的度数(包括度数为0)。换句话说，具有度数k的二项树有0个或1个。

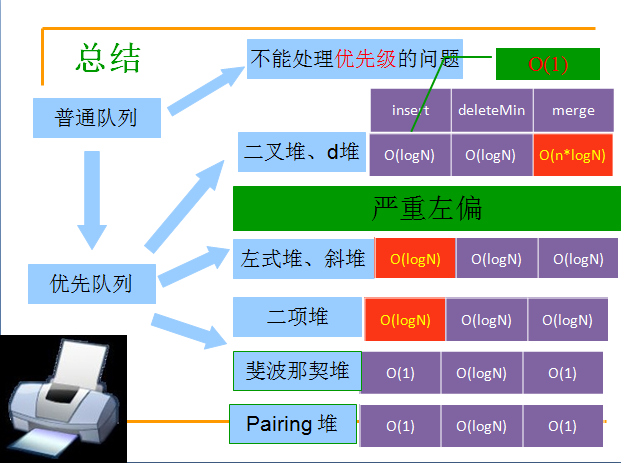
[](http://images.cnitblog.com/i/497634/201404/101016180284317.jpg)

上图就是一棵二项堆，它由二项树B0、B2和B3组成。对比二项堆的定义：(01)二项树B0、B2、B3都是最小堆；(02)二项堆不包含相同度数的二项树。

二项堆的第(01)个性质保证了二项堆的最小节点是某一棵二项树的根节点，第(02)个性质则说明结点数为n的二项堆最多只有log{n} + 1棵二项树。实际上，将包含n个节点的二项堆，表示成若干个2的指数和(或者转换成二进制)，则每一个2个指数都对应一棵二项树。例如，13(二进制是1101)的2个指数和为13=23 + 22 + 20, 因此具有13个节点的二项堆由度数为3, 2, 0的三棵二项树组成。

***二项堆的存储结构图***





Binary heap              Binomial heap          Fibonacci heap

二叉堆（最坏情况） 二项堆（最坏情况）（斐波那契堆（平摊））

(worst-case) (worst-case) (amortized)

MAKE-HEAP (1) (1) (1)

INSERT (lg *n*) *O*(lg *n*) (1)

MINIMUM (1) *O*(lg *n*) (l)

EXTRACT-MIN (lg *n*) (1g *n*) *O*(lg *n*)

UNION (*n*) *O*(lg *n*) (1)

DECREASE-KEY (lg *n*) (lg *n*) (1)

DELETE (1g *n*) (lg *n*) *O*(lg *n*)