

skywang12345

导航

博客园  
首页  
新随笔  
联系  
订阅 [XML](#)  
管理

2014年11月						
日	一	二	三	四	五	六
26	27	28	29	30	31	1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	1	2	3	4	5	6

统计

随笔 - 278  
文章 - 0  
评论 - 337  
引用 - 0

搜索

找找看

常用链接

我的随笔  
我的评论  
我的参与  
最新评论  
我的标签

随笔分类 (275)

[Android\(7\)](#)  
[Android NDK编程\(9\)](#)  
[Android 系统层\(5\)](#)  
[Android 应用层\(46\)](#)  
[Computer Culture\(2\)](#)  
[Java\(111\)](#)  
[Linux/Ubuntu\(5\)](#)  
[UML\(5\)](#)  
[Windows\(1\)](#)  
[设计模式\(1\)](#)  
[数据结构\\_算法\(79\)](#)  
[索引\(4\)](#)

最新评论

1. [Re:Java多线程系列目录\(共43篇\)](#)  
感谢, 收藏了

--smh821025

2. [Re:Java多线程系列--"基础篇"09之 interrupt\(\)和线程终止方式](#)  
上面的代码还是有错误的地方, 终止处于阻塞状态的 (wait(),sleep()等) 线程, 捕获异常的代码在while循环内, 就算break了, 也不会终止当前线程, 他只是跳出了while循环, 而你的示例代码恰好又没有继续执行的代码, 所以是线程是正常终止, 而非调用interrupt()方法捕获异常终止的!

--John Hou

3. [Re:java io系列01之 "目录&目录"](#)  
排版好, 讲解详细清楚, 专门登录顶一个

--明镜湖水

4. [Re:Android JNI和NDK学习\(04\)-NDK调试方法](#)  
这个系列不错,

## 斐波那契堆(一)- 图文解析 和 C语言的实现

### 概要

本章介绍斐波那契堆。和以往一样, 本文会先对斐波那契堆的理论知识进行简单介绍, 然后给出C语言的实现。后续再分别给C++和Java版本的实现; 实现的语言虽不同, 但是原理如出一辙, 选择其中之一进行了解即可。若文章有错误或不足的地方请不吝指出!

#### 目录

1. 斐波那契堆的介绍
2. 斐波那契堆的基本操作
3. 斐波那契堆的C实现(完整源码)
4. 斐波那契堆的C测试程序

转载请注明出处: <http://www.cnblogs.com/skywang12345/p/3659060.html>

更多内容: [数据结构与算法系列 目录](#)

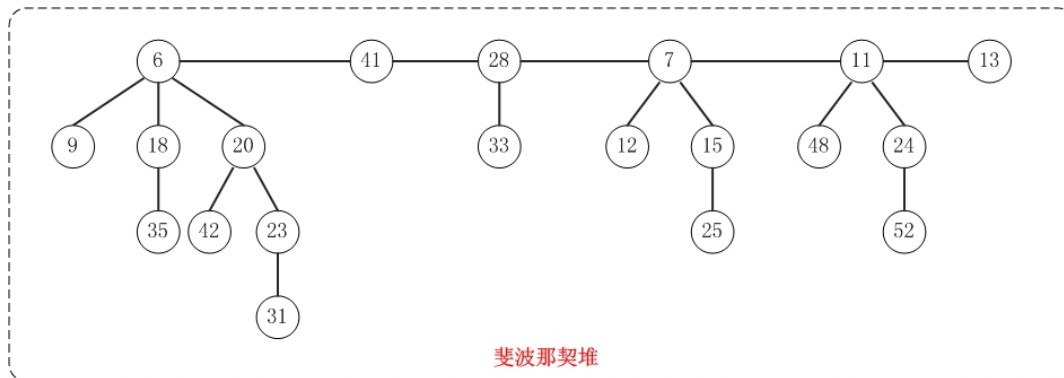
- (01) [斐波那契堆\(一\)- 图文解析 和 C语言的实现](#)  
(02) [斐波那契堆\(二\)- C++的实现](#)  
(03) [斐波那契堆\(三\)- Java的实现](#)

### 斐波那契堆的介绍

斐波那契堆(Fibonacci heap)是堆中一种, 它和二项堆一样, 也是一种可合并堆; 可用于实现合并优先队列。斐波那契堆比项堆具有更好的平摊分析性能, 它的合并操作的时间复杂度是 $O(1)$ 。

与二项堆一样, 它也是由一组堆最小有序树组成, 并且是一种可合并堆。

与二项堆不同的是, 斐波那契堆中的树不一定是二项树; 而且二项堆中的树是有序排列的, 但是斐波那契堆中的树都是有根无序的。



### 斐波那契堆的基本操作

#### 1. 基本定义

已转载并附链接

by [www.elesos.com](http://www.elesos.com) 站长

赠免费vpn【[1](#)】访问youtube不卡，亲测可用，不限速。

--[www.elesos.com](http://www.elesos.com)站长

5. [Re:Java多线程系列目录\(共43篇\)](#) 很好很全面。赞一个！

--jNerd

阅读排行榜

1. [Android 布局之 GridLayout\(8894\)](#)
2. [Android 之窗口小部件详解--App Widget\(6692\)](#)
3. [Android layout属性详细说明\(5988\)](#)
4. [Android App组件之 ListFragment -- 说明和示例\(5750\)](#)
5. [Java 集合系列目录\(Category\)\(5699\)](#)

```
typedef int Type;

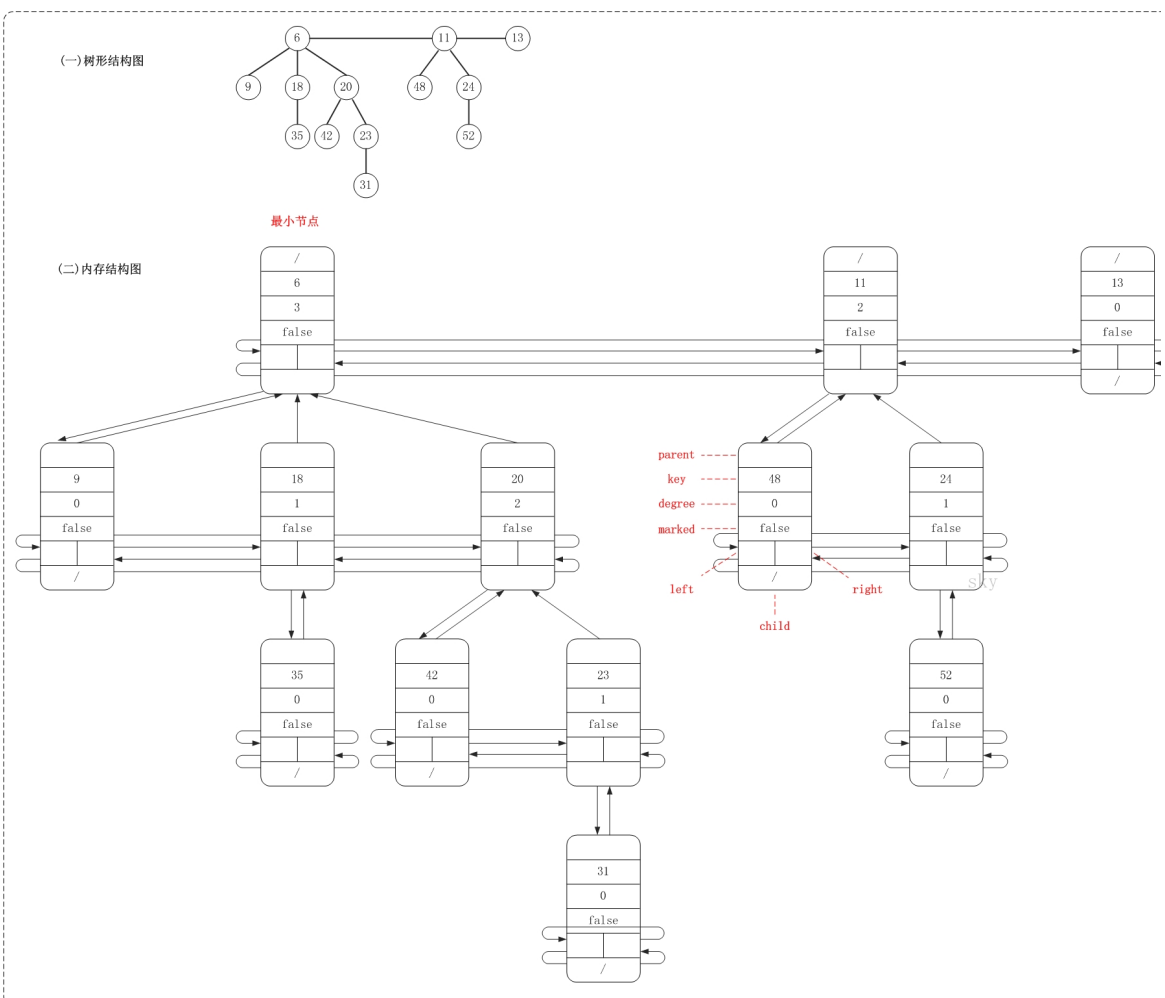
typedef struct _FibonacciNode
{
    Type    key;           // 关键字 (键值)
    int     degree;        // 度数
    struct _FibonacciNode *left; // 左兄弟
    struct _FibonacciNode *right; // 右兄弟
    struct _FibonacciNode *child; // 第一个孩子节点
    struct _FibonacciNode *parent; // 父节点
    int     marked;        // 是否被删除第1个孩子 (1表示删除, 0表示未删除)
}FibonacciNode, FibNode;
```

FibNode是斐波那契堆的节点类，它包含的信息较多。key是用于比较节点大小的，degree是记录节点的度，left和right是指向节点的左右兄弟，child是节点的第一个孩子，parent是节点的父节点，marked是记录该节点是否被删除第1个孩子(marked在删除节点时有用)。

```
typedef struct _FibonacciHeap{
    int     keyNum;        // 堆中节点的总数
    int     maxDegree;     // 最大度
    struct _FibonacciNode *min; // 最小节点 (某个最小堆的根节点)
    struct _FibonacciNode **cons; // 最大度的内存区域
}FibonacciHeap, FibHeap;
```

FibHeap是斐波那契堆对应的类。min是保存当前堆的最小节点，keyNum用于记录堆中节点的总数，maxDegree用于记录堆中最大度，而cons在删除节点时来暂时保存堆数据的临时空间。

下面看看斐波那契堆的内存结构图。

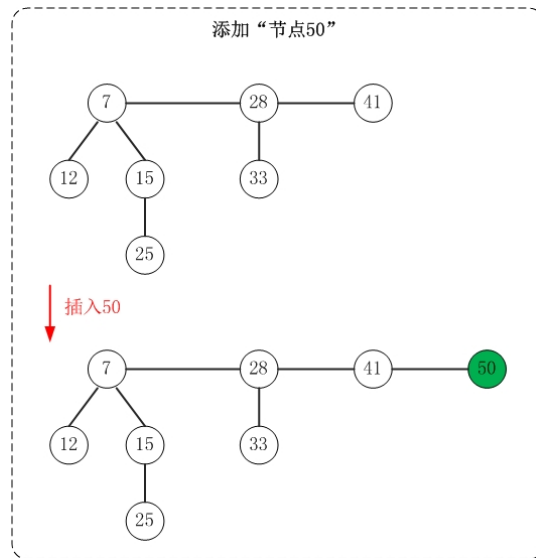


从图中可以看出，斐波那契堆是由一组最小堆组成，这些最小堆的根节点组成了双向链表(后文称为"根链表")；斐波那契堆的最小节点就是"根链表中的最小节点"！

PS. 上面这幅图的结构和测试代码中的"基本信息"测试函数的结果是一致的; 你可以通过测试程序来亲自验证!

## 2. 插入操作

插入操作非常简单: 插入一个节点到堆中, 直接将该节点插入到"根链表的min节点"之前即可; 若被插入节点比"min节点"小则更新"min节点"为被插入节点。



上面是插入操作的示意图。

斐波那契堆的根链表是"双向链表", 这里将min节点看作双向链表的表头(后文也是如此)。在插入节点时, 每次都是"将节点到min节点之前(即插入到双向链表末尾)". 此外, 对于根链表中最小堆都只有一个节点的情况, 插入操作就很演化成双向链表插入操作。

此外, 插入操作示意图与测试程序中的"插入操作"相对应, 感兴趣的可以亲自验证。

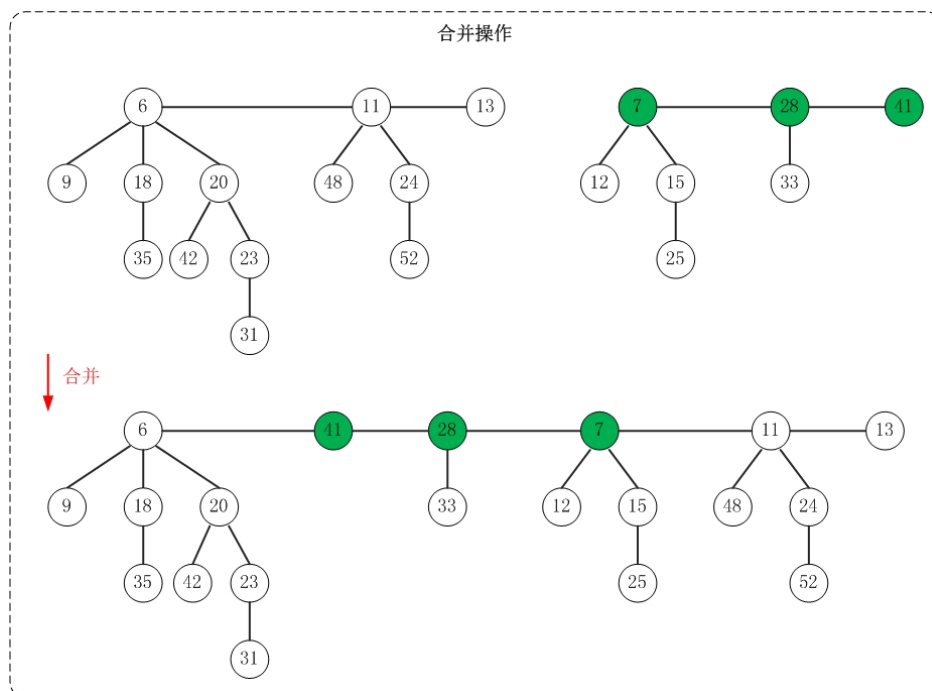
插入操作代码

```
/*
 * 将"单个节点node"加入"链表root"之前
 * a ..... root
 * a ..... node ..... root
 *
 * 注意: 此处node是单个节点, 而root是双向链表
 */
static void fib_node_add(FibNode *node, FibNode *root)
{
    node->left      = root->left;
    root->left->right = node;
    node->right      = root;
    root->left       = node;
}

/*
 * 将节点node插入到斐波那契堆heap中
 */
static void fib_heap_insert_node(FibHeap *heap, FibNode *node)
{
    if (heap->keyNum == 0)
        heap->min = node;
    else
    {
        fib_node_add(node, heap->min);
        if (node->key < heap->min->key)
            heap->min = node;
    }
    heap->keyNum++;
}
```

### 3. 合并操作

合并操作和插入操作的原理非常类似：将一个堆的根链表插入到另一个堆的根链表上即可。简单来说，就是将两个双向链表拼成一个双向链表。



上面是合并操作的示意图。该操作示意图与测试程序中的"合并操作"相对应！

合并操作代码

```

/*
 * 将双向链表b链接到双向链表a的后面
 *
 * 注意： 此处a和b都是双向链表
 */
static void fib_node_cat(FibNode *a, FibNode *b)
{
    FibNode *tmp;

    tmp = a->right;
    a->right = b->right;
    b->right->left = a;
    b->right = tmp;
    tmp->left = b;
}

/*
 * 将h1, h2合并成一个堆, 并返回合并后的堆
 */
FibHeap* fib_heap_union(FibHeap *h1, FibHeap *h2)
{
    FibHeap *tmp;

    if (h1==NULL)
        return h2;
    if (h2==NULL)
        return h1;

    // 以h1为"母", 将h2附加到h1上; 下面是保证h1的度数大, 尽可能的少操作。
    if (h2->maxDegree > h1->maxDegree)
    {
        tmp = h1;
        h1 = h2;
        h2 = tmp;
    }

    if ((h1->min) == NULL) // h1无"最小节点"
    {
        h1->min = h2->min;
        h1->keyNum = h2->keyNum;
        free(h2->cons);
        free(h2);
    }
    else if ((h2->min) == NULL) // h1有"最小节点" && h2无"最小节点"
    {
        free(h2->cons);
        free(h2);
    }
    // h1有"最小节点" && h2有"最小节点"
    else
    {
        // 将"h2中根链表"添加到"h1"中
        fib_node_cat(h1->min, h2->min);
        if (h1->min->key > h2->min->key)
            h1->min = h2->min;
        h1->keyNum += h2->keyNum;
        free(h2->cons);
        free(h2);
    }

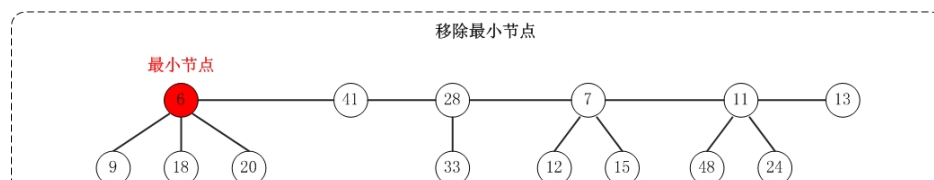
    return h1;
}

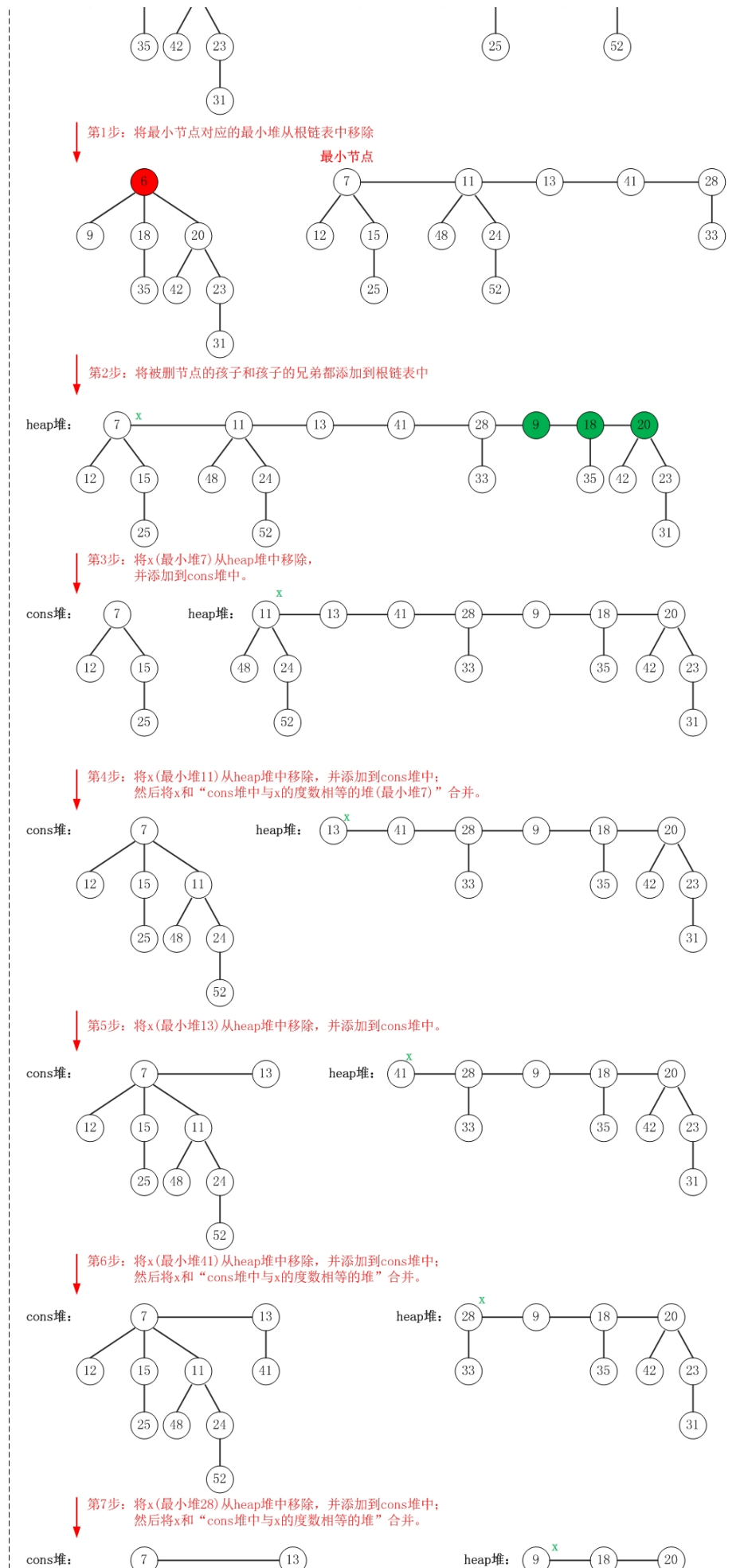
```

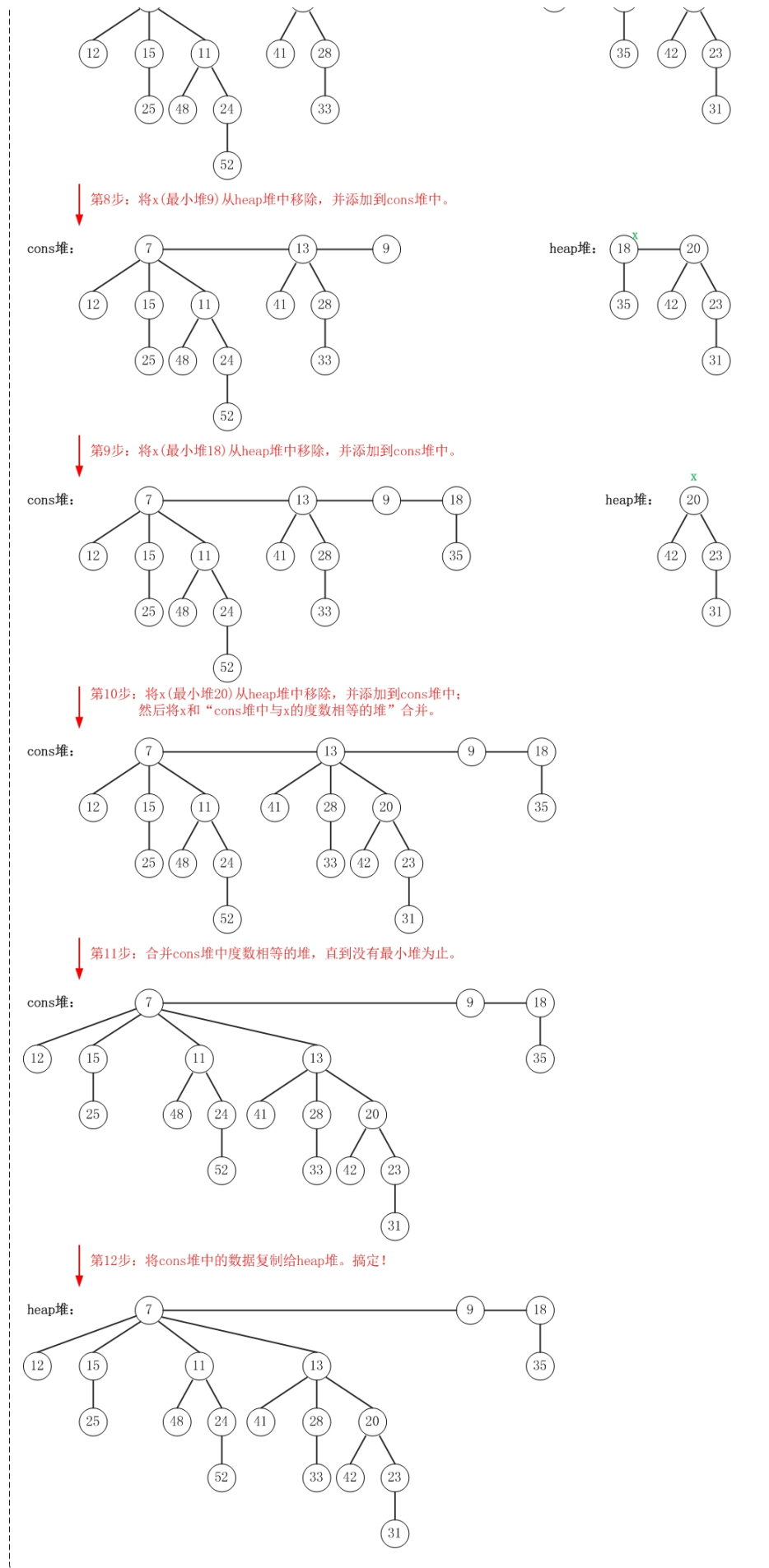
## 4. 取出最小节点

抽取最小节点的操作是斐波那契堆中较复杂的操作。

- (1) 将要抽取最小节点的子树都直接串联在根表中；
- (2) 合并所有degree相等的树，直到没有相等的degree的树。







上面是取出最小节点的示意图。图中应该写的非常明白了, 若有疑问, 看代码。

此外, 该操作示意图与测试程序中的"删除最小节点"相对应! 有兴趣的可以亲自验证。

取出最小节点代码

```
/*
 * 移除最小节点, 并返回移除节点后的斐波那契堆
 */
FibNode* _fib_heap_extract_min(FibHeap *heap)
{
    if (heap==NULL || heap->min==NULL)
        return NULL;

    FibNode *child = NULL;
    FibNode *min = heap->min;
    // 将min每一个儿子(儿子和儿子的兄弟)都添加到"斐波那契堆的根链表"中
    while (min->child != NULL)
    {
        child = min->child;
        fib_node_remove(child);
        if (child->right == child)
            min->child = NULL;
        else
            min->child = child->right;

        fib_node_add(child, heap->min);
        child->parent = NULL;
    }

    // 将min从根链表中移除
    fib_node_remove(min);
    // 若min是堆中唯一节点, 则设置堆的最小节点为NULL;
    // 否则, 设置堆的最小节点为一个非空节点(min->right), 然后再进行调节。
    if (min->right == min)
        heap->min = NULL;
    else
    {
        heap->min = min->right;
        fib_heap_consolidate(heap);
    }
    heap->keyNum--;

    return min;
}
```

其中fib\_heap\_consolidate(heap)的作用是合并斐波那契堆的根链表中相同度数的树, 它的相关代码如下:

[View Code](#)

## 5. 减小节点值

减少斐波那契堆中的节点的键值, 这个操作的难点是: 如果减少节点后破坏了"最小堆"性质, 如何去维护呢? 下面对一般性进行分析。

(1) 首先, 将"被减小节点"从"它所在的最小堆"剥离出来; 然后将"该节点"关联到"根链表"中。 倘若被减小的节点不是单独节点, 而是包含子树的树根。则是将以"被减小节点"为根的子树从"最小堆"中剥离出来, 然后将该树关联到根链表中。

(2) 接着, 对"被减少节点"的原父节点进行"级联剪切"。所谓"级联剪切", 就是在被减小节点破坏了最小堆性质, 并被切下后; 再从"它的父节点"进行递归级联剪切操作。

而级联操作的具体动作则是: 若父节点(被减小节点的父节点)的marked标记为false, 则将其设为true, 然后退出。

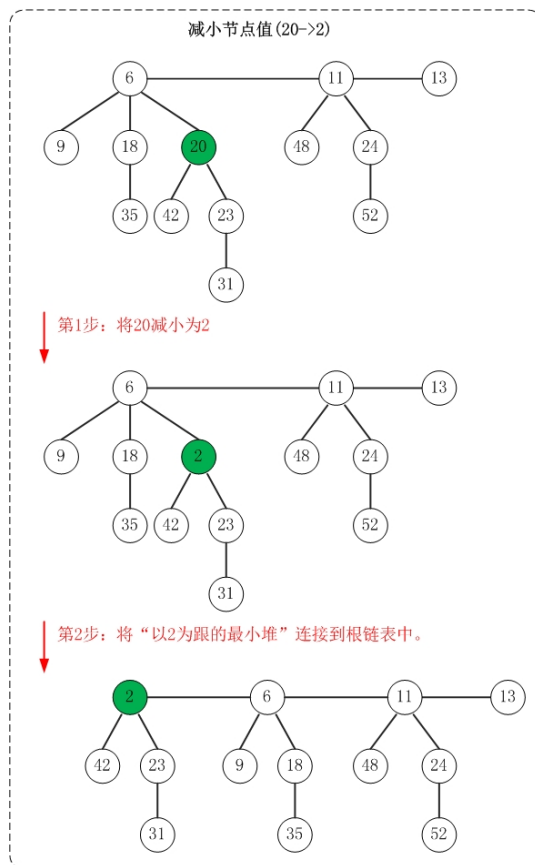
否则, 将父节点从最小堆中切下来(方式和"切被减小节点的方式"一样); 然后

对祖父节点进行"级联剪切"。

marked标记的作用就是用来标记"该节点的子节点是否有被删除过", 它的作用是实现级联剪切。而级联剪切的真正目的是为了"防止"最小堆"由二叉树演化成链表。

(3) 最后, 别忘了对根链表的最小节点进行更新。





上面是减小节点值的示意图。该操作示意图与测试程序中的“减小节点”相对应！

减小节点值的代码

```
/*
 * 将斐波那契堆heap中节点node的值减少为key
 */
static void fib_heap_decrease(FibHeap *heap, FibNode *node, Type key)
{
    FibNode *parent;

    if (heap==NULL || heap->min==NULL || node==NULL)
        return ;

    if ( key>=node->key)
    {
        printf("decrease failed: the new key(%d) is no smaller than current key(%d)\n", key, node->key);
        return ;
    }

    node->key = key;
    parent = node->parent;
    if (parent!=NULL && node->key < parent->key)
    {
        // 将node从父节点parent中剥离出来，并将node添加到根链表中
        fib_heap_cut(heap, node, parent);
        fib_heap_cascading_cut(heap, parent);
    }

    // 更新最小节点
    if (node->key < heap->min->key)
        heap->min = node;
}
```

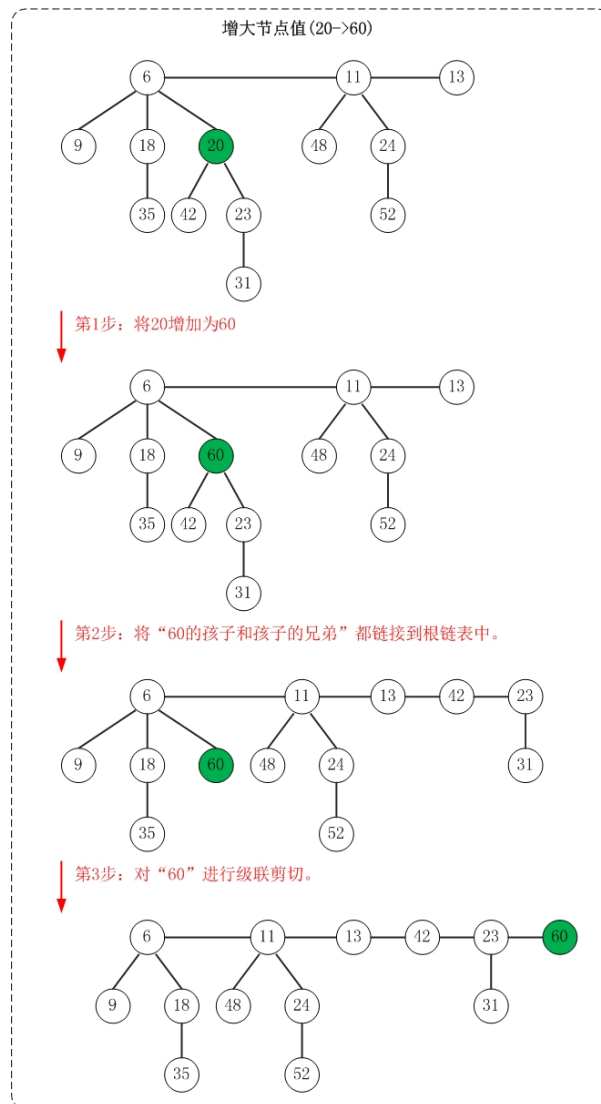
其中，fib\_heap\_cut()和fib\_heap\_cascading\_cut()的相关代码如下：

[View Code](#)

## 6. 增加节点值

增加节点值和减少节点值类似，这个操作的难点也是如何维护"最小堆"性质。思路如下：

- (1) 将"被增加节点"的"左孩子和左孩子的所有兄弟"都链接到根链表中。
- (2) 接下来，把"被增加节点"添加到根链表；但是别忘了对其进行级联剪切。



上面是增加节点值的示意图。该操作示意图与测试程序中的"增大节点"相对应！

增加节点值的代码

```


/*
 * 将斐波那契堆heap中节点node的值增加为key
 */
static void fib_heap_increase(FibHeap *heap, FibNode *node, Type key)
{
    FibNode *child, *parent, *right;

    if (heap==NULL || heap->min==NULL || node==NULL)
        return ;

    if (key <= node->key)
    {
        printf("increase failed: the new key(%d) is no greater than current key(%d)\n", key, node->key);
        return ;
    }

    // 将node每一个儿子(不包括孙子,重孙,...)都添加到"斐波那契堆的根链表"中
    while (node->child != NULL)
    {
        child = node->child;
        fib_node_remove(child);           // 将child从node的子链表中删除
        if (child->right == child)
            node->child = NULL;
        else
            node->child = child->right;

        fib_node_add(child, heap->min);    // 将child添加到根链表中
        child->parent = NULL;
    }
    node->degree = 0;
    node->key = key;

    // 如果node不在根链表中,
    // 则将node从父节点parent的子链接中剥离出来,
    // 并使node成为"堆的根链表"中的一员,
    // 然后进行"级联剪切"
    // 否则, 则判断是否需要更新堆的最小节点
    parent = node->parent;
    if (parent != NULL)
    {
        fib_heap_cut(heap, node, parent);
        fib_heap_cascading_cut(heap, parent);
    }
    else if (heap->min == node)
    {
        right = node->right;
        while(right != node)
        {
            if (node->key > right->key)
                heap->min = right;
            right = right->right;
        }
    }
}

```

## 7. 删除节点

删除节点, 本文采用了操作是: "取出最小节点"和"减小节点值"的组合。

- (1) 先将被删除节点的键值减少。减少后的值要比"原最小节点的值"即可。
- (2) 接着, 取出最小节点即可。

删除节点值的代码

```
/*
 * 删除结点node
 */
static void _fib_heap_delete(FibHeap *heap, FibNode *node)
{
    Type min = heap->min->key;
    fib_heap_decrease(heap, node, min-1);
    _fib_heap_extract_min(heap);
    free(node);
}
```

注意：关于斐波那契堆的"更新"、"打印"、"销毁"等接口就不再单独介绍了。后文的源码中有给出它们的实现代码，**Please RTFSC(Read The Fucking Source Code)!**

## 斐波那契堆的C实现(完整源码)

斐波那契堆的头文件(fibonacci\_heap.h)

[View Code](#)

斐波那契堆的实现文件(fibonacci\_heap.c)

[View Code](#)

斐波那契堆的测试程序(main.c)

[View Code](#)

## 斐波那契堆的C测试程序

斐波那契堆的测试程序包括了"插入"、"合并"、"增大"、"减小"、"删除"、"基本信息"等几种功能的测试代码。默认是运行"基本信息(验证斐波那契堆的结构)"测试代码，你可以根据自己的需要来对相应的功能进行验证！

注意：**C语言版的斐波那契堆的LOG2宏定义中使用了math.h**，记得引入**math库**。例如，若你是在Linux下通过gcc编译，**添加-lm参数(gcc \*.c -lm)**。

下面是基本信息测试代码的运行结果：

```
== 斐波那契堆(hb)中依次添加: 18 35 20 42 9 31 23 6 48 11 24 52 13 2
== 斐波那契堆(hb)删除最小节点
== 斐波那契堆的详细信息: ==
1.      6(3) is root
      9(0) is 6's child
     18(1) is 9's next
     35(0) is 18's child
     20(2) is 18's next
     42(0) is 20's child
     23(1) is 42's next
     31(0) is 23's child
2.     11(2) is root
     48(0) is 11's child
     24(1) is 48's next
     52(0) is 24's child
3.     13(0) is root
```



生活的悲欢离合永远在地平线以外，而眺望是一种青春的姿态...

**PS.**文章是笔者分享的学习笔记，若你觉得可以、还行、过得去、甚至不太差的话，可以“推荐下的哦。就此谢过！

分类: [数据结构\\_算法](#)

标签: [实现](#), [介绍](#), [数据结构](#), [c](#), [图文详解](#), [斐波那契堆](#), [fibonacci](#)



5 0

(请您对文章做出评价)

[« 上一篇: 二项堆\(三\)之 Java的实现](#)[» 下一篇: 斐波那契堆\(二\)之 C++的实现](#)

posted on 2014-04-17 09:30 如果天空不死 阅读(771) 评论(2) 编辑 收藏

## Comments

### #1楼

Anyleader

Posted @ 2014-04-17 09:48

Good job!

支持(0) 反

### #2楼

SeeMore

Posted @ 2014-04-17 23:31

楼主真的很用心，不顶不行啊

支持(0) 反

[刷新评论](#) [刷新页面](#) [返回](#)注册用户登录后才能发表评论，请 [登录](#) 或 [注册](#)，[访问](#)网站首页。[【免费课程】案例：前端开发工具技巧介绍—Sublime篇](#)[【推荐】50万行VC++源码：大型组态工控、电力仿真CAD与GIS源码库](#)[融云](#)，免费为你的App加入IM功能——让你的App“聊”起来！！[写“听云”原创博文，赢取iPhone 6超级大奖](#)

## 最新IT新闻：

- 海尔员工论坛最热帖：张瑞敏致创客的一封信
  - 商业路演设计中应避免的6大失误
  - 世界上第一台计算机被复活了！
  - 钛媒体专访陈小华：58到家不是手艺人的平台，是开放的“蓝翔”
  - 70%的代码跨平台重用，Google Inbox是如何做到的？
- » [更多新闻...](#)



## 最新知识库文章：

- [SOA与API的分裂和统一](#)
  - [可测性分析和实践](#)
  - [OWIN初探](#)
  - [也谈如何构建高性能服务端程序](#)
  - [什么是自组织团队？](#)
- » [更多知识库文章...](#)

