



浙江财经学院

Zhejiang University Of Finance & Economics

浙江财经学院计算机系

程序设计实践与提高

堆及其应用



内容提要



- 1 堆的引入—例题**ZOJ2212**。
- 2 堆简介。
- 3 什么是堆？
- 4 最大堆的实现：插入和删除。
- 5 **STL**中的堆和优先队列。
- 6 堆的应用。



1 线段树引入—例题ZOJ2212



Argus

题目来源 : Asia 2004, Beijing (Mainland China)

题号 : ZOJ2212

题目描述 :

一个数据流是一个实时的、连续的、有序的数据序列。数据流的例子包括传感器数据、因特网数据等。同时，基于数据流的查询也要求不断地、定期地进行。例如，一个仓库的温度检测系统可能执行如下的一些查询：

Query-1: 每5分钟，返回过去5分钟之内的最高温度。

Query-2: (每10分钟)返回每块地板上过去10分钟之内的平均温度。

数据流管理系统Argus处理类似的数据流查询。用户可以向Argus注册一个查询。Argus则按预期的频率根据动态更新的数据执行查询并返回对应的结果。

在Argus中，查询指令的格式为：

Register Q_num Period

Q_num(0<Q_num<=3000)是查询号，**Period(0<Period<=3000)**是该查询两次连续查询之间的间隔时间。

每次Argus可以执行多个查询。每个查询有不同的Q_num。你的任务是返回前K次查询。如果两个或多个查询同时返回结果，则按Q_num的升序依次返回查询结果。





输入描述：

输入文件第一部分为向**Argus**注册的查询指令，每个指令占一行。假定查询指令数不超过**1000**，这些**查询同时开始执行**。第一部分数据以符号"**#**"作为结束的标志。

第二部分占一行，为正整数**K(≤ 10000)**。

输出描述：

输出前**K**次查询返回结果的**Q_num**，每个**Q_num**占一行。

样例输入：

```
Register 2004 200  
Register 2005 300  
#  
5
```

样例输出：

```
2004  
2005  
2004  
2004  
2005
```





理解：查询的**定时时间(即Period)**和**响应时间**。

对“Register 2004 200”，可以解释其定时时间为200，
响应时间为200、400、600、800、...。

样例输入：

Register 2004 200
Register 2005 300

5

样例输出：

2004
2005
2004
2004
2005





求解方法：

■ 二级排序：产生若干次查询，然后先按响应时间排序，响应时间相同再按Q_num排序。最后输出前K个查询。

■ 例如，假设两个查询分别响应3次：

2004 200 2005 300

2004 400 2005 600

2004 600 2005 900

样例输入：

Register 2004 200

Register 2005 300

#

5

样例输出：

2004

2005

2004

2004

2005





求解方法(continue) :

- ▣ 思考：每个查询需要**分别响应多少次**(才能保证有至少K次查询)？
- ▣ 方法1：设所有查询的最小定时时间为T(设对应的Q_num为num1)，对每个查询，产生**K×T(最坏情况下，前K个查询都是num1，因此时间为K×T)**时间内的所有查询。缺点：会“生成”很多多余的查询。最坏情况下可能导致内存超出限制。
- ▣ 思考：有没有更好的方法？

样例输入：

```
Register 2004 200  
Register 2005 300  
#  
5
```

样例输出：

```
2004  
2005  
2004  
2004  
2005
```





- 最小堆的引入：每次执行的是响应时间最小的查询—**最小堆**。
- 思考：如何实现，如果多个查询响应时间一样，优先执行**Q_num**值最小的查询？

样例输入：

```
Register 2004 200  
Register 2005 300  
#  
5
```

样例输出：

```
2004  
2005  
2004  
2004  
2005
```



2 堆简介



■ 堆在《数据结构》课程中的位置：

■ 二叉树：堆是一种特殊的二叉树。

■ 排序：堆排序是一种排序方法，但在实践中由于堆排序不如快速排序，所以很少用。

■ 堆的意义就在于：最快的找到最大/最小值，在堆结构中插入一个值重新构造堆结构，取走最大/最小值后重新构造堆结构，其时间复杂度为 $O(\log_2 N)$ ，而其他方法最少为 $O(N)$ 。

■ 因此，堆在实践中的用途不在于排序，其主要用在调度算法中，比如优先级调度，每次取优先级最高的；或时间驱动，取时间最小/等待最长的等等。

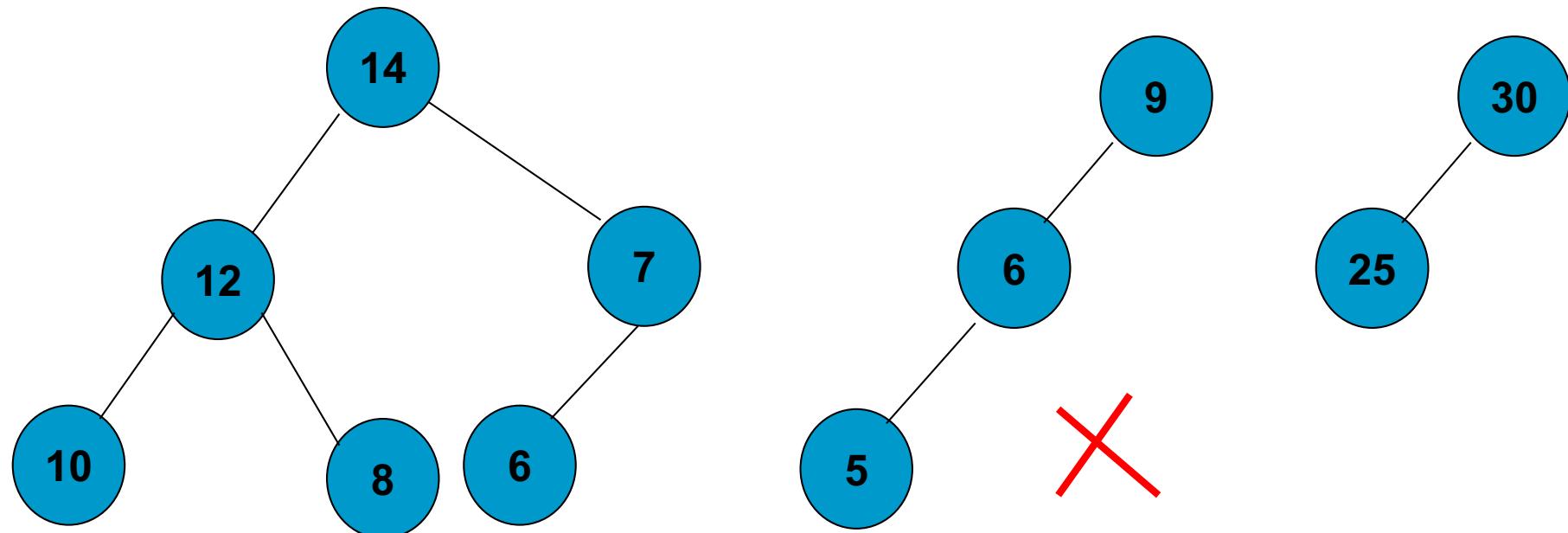
■ 堆分为最大堆/最小堆。



3 什么是堆？（最大堆）



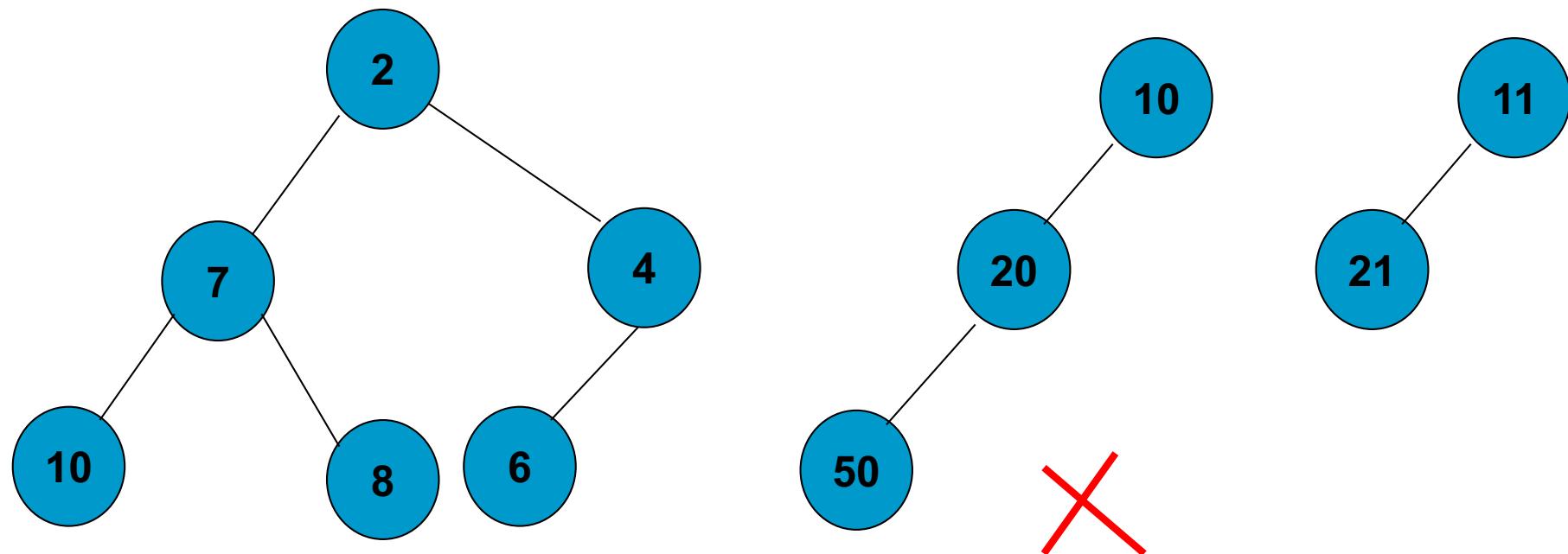
1. 完全二叉树。
2. 每个结点的值都大于或等于其子女结点的值（如果有的话）。



什么是堆？（最小堆）



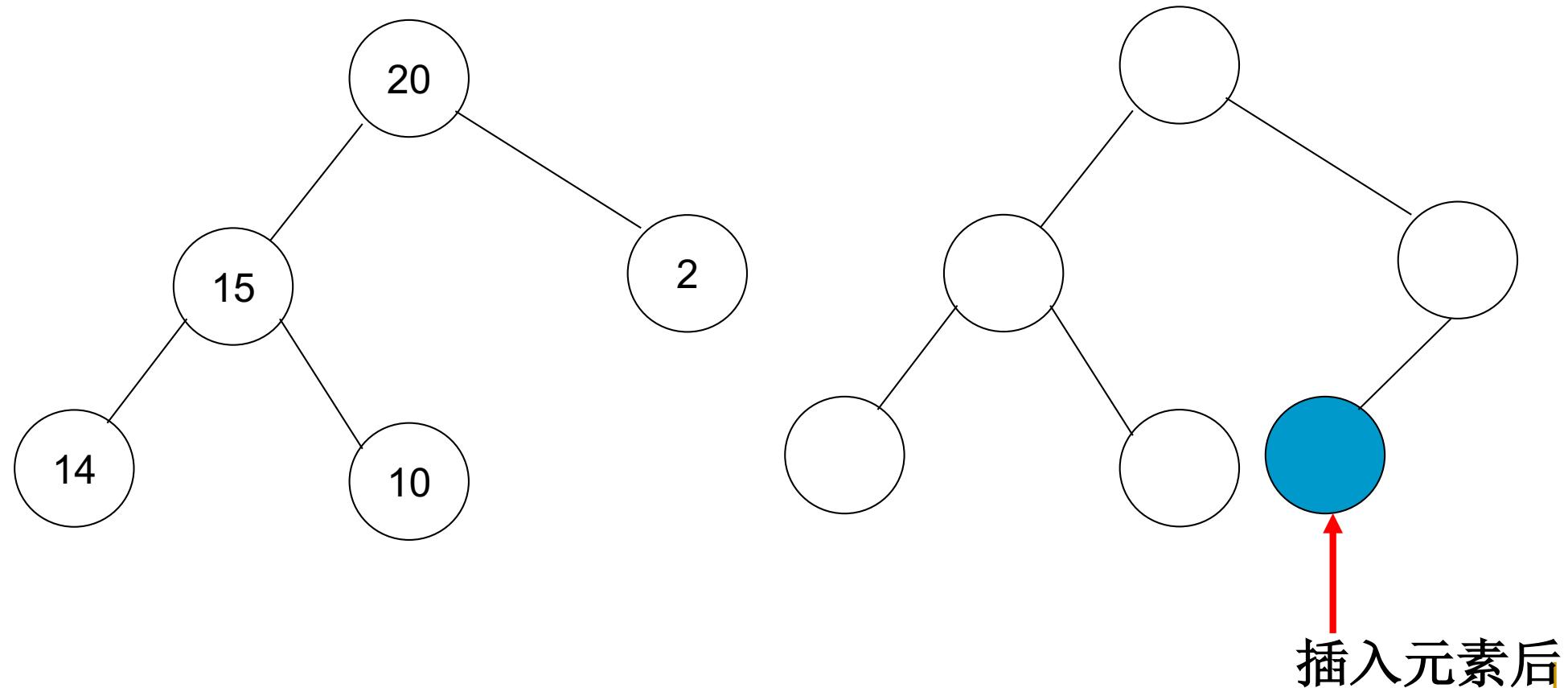
1. 完全二叉树。
2. 每个结点的值都小于或等于其子女结点的值（如果有的话）。



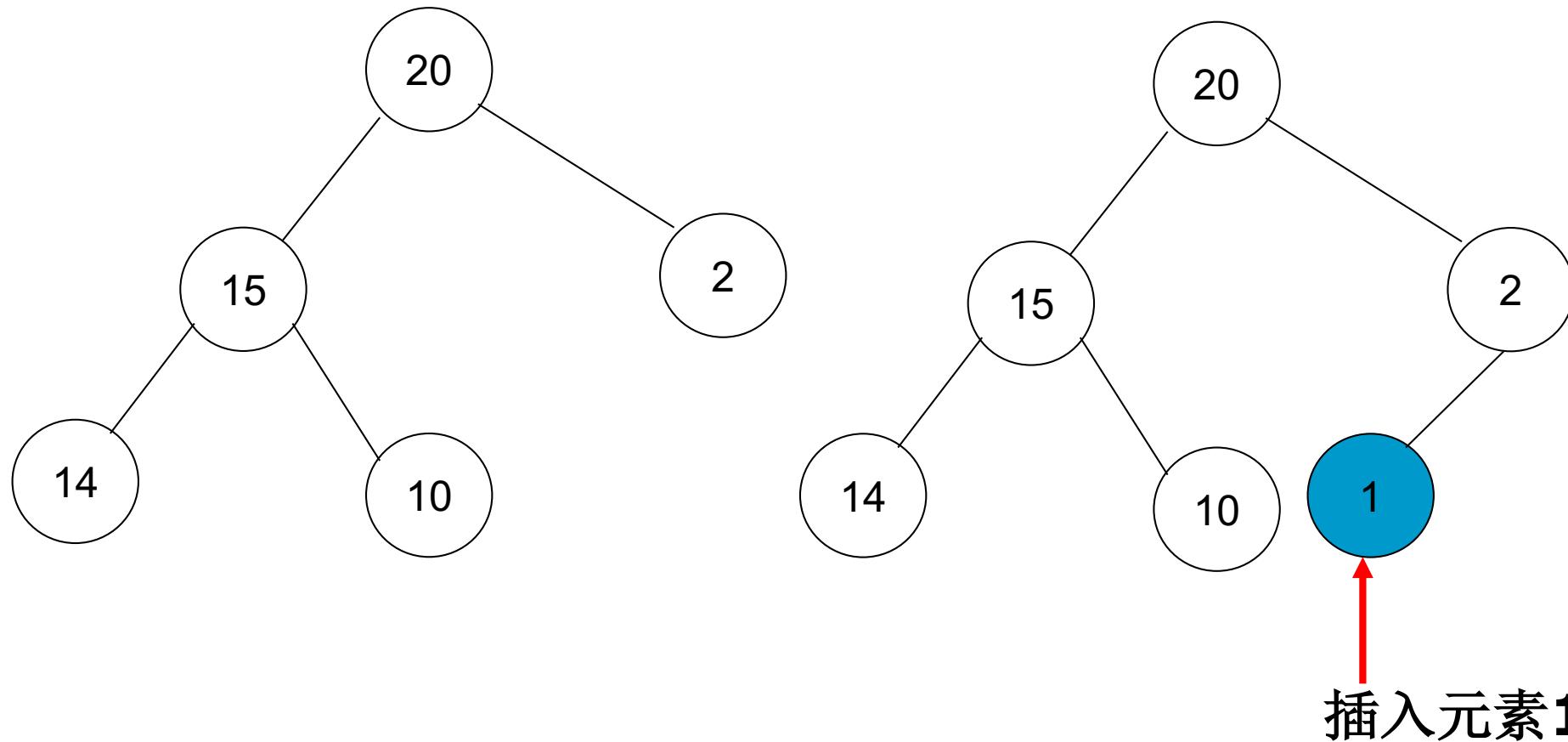
4 最大堆的实现：插入运算



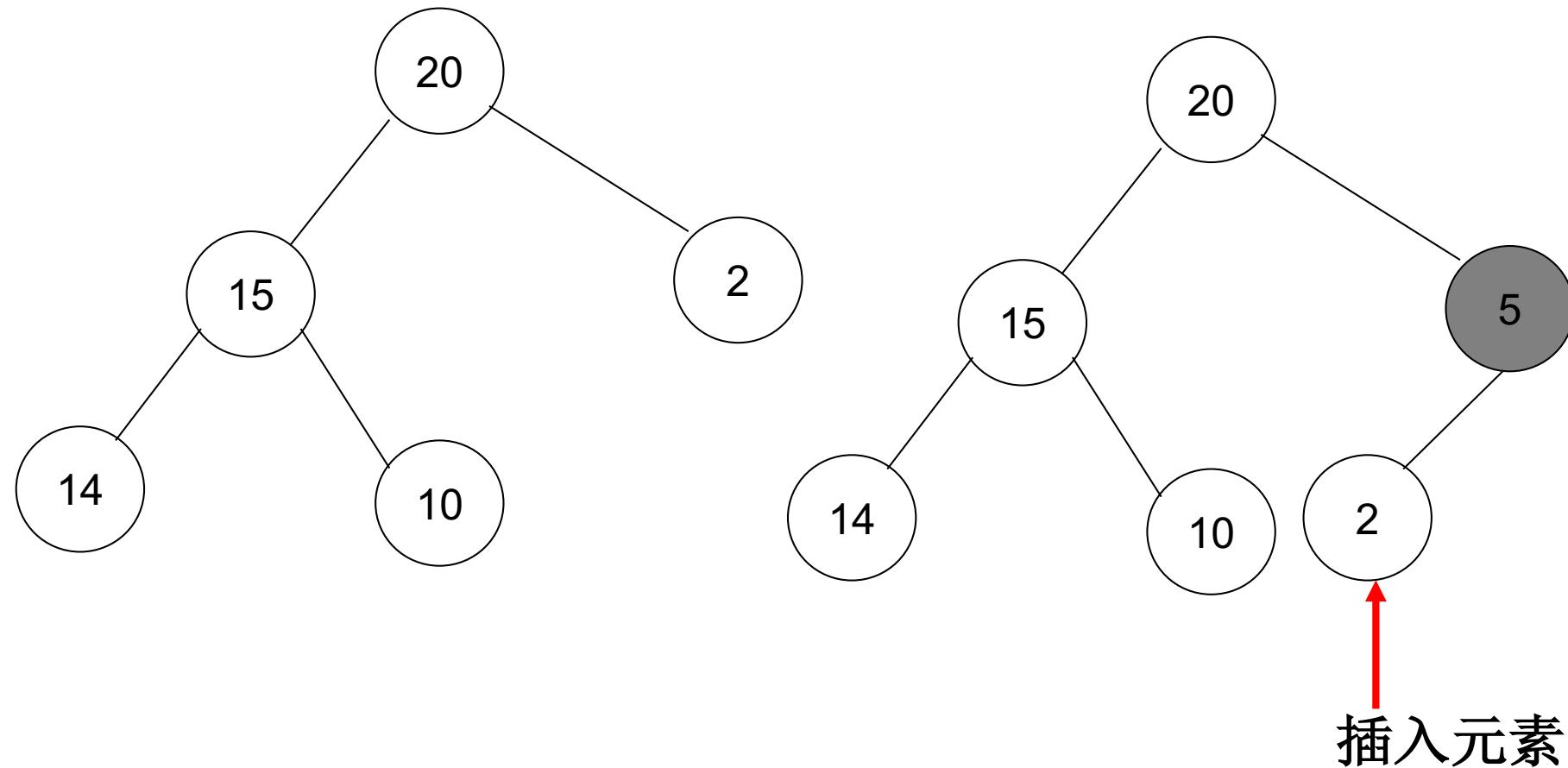
在STL中已经实现了堆(但使用方法很复杂),当然也可以自己编写代码实现堆(建议集训队队员收集堆的实现代码,但可以直接使用STL中的优先级队列)。



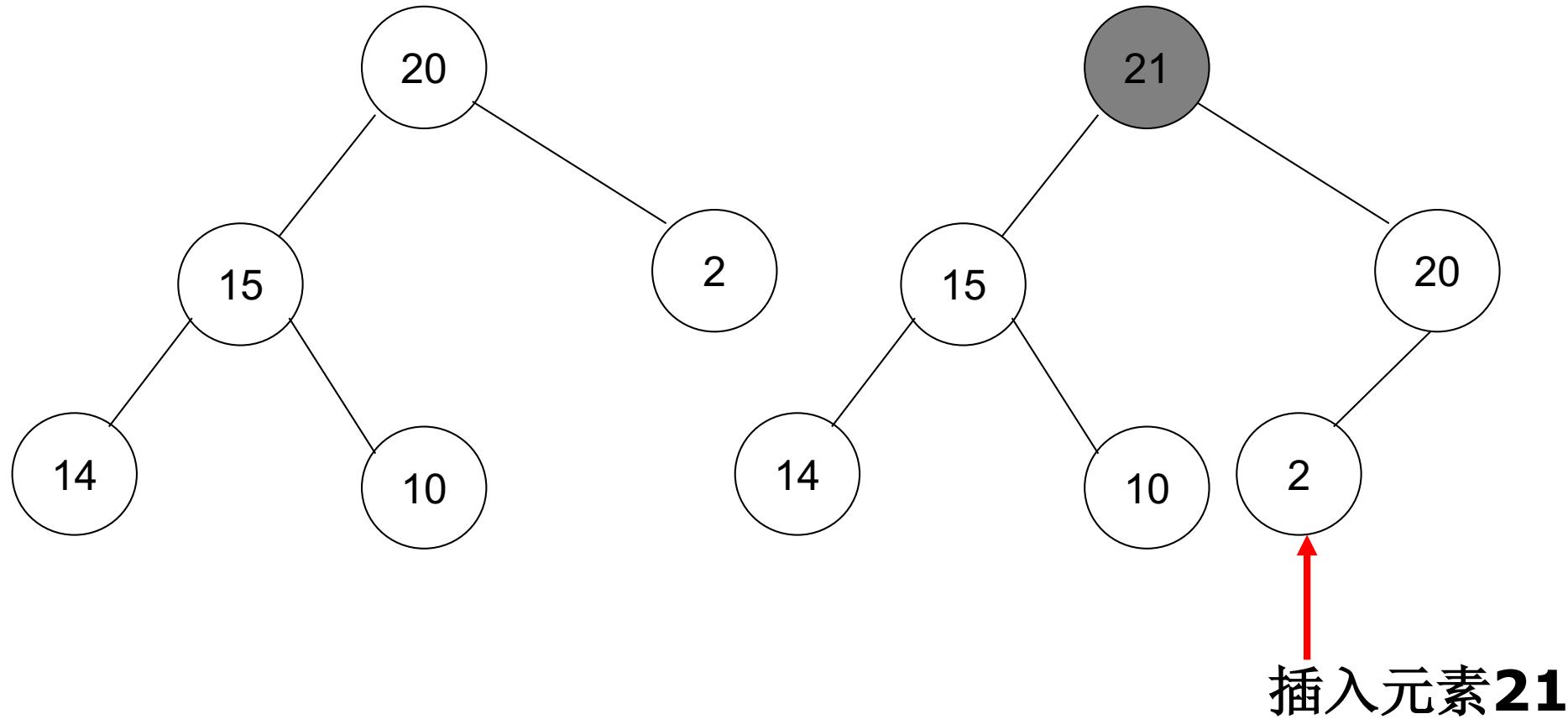
最大堆的插入



最大堆的插入



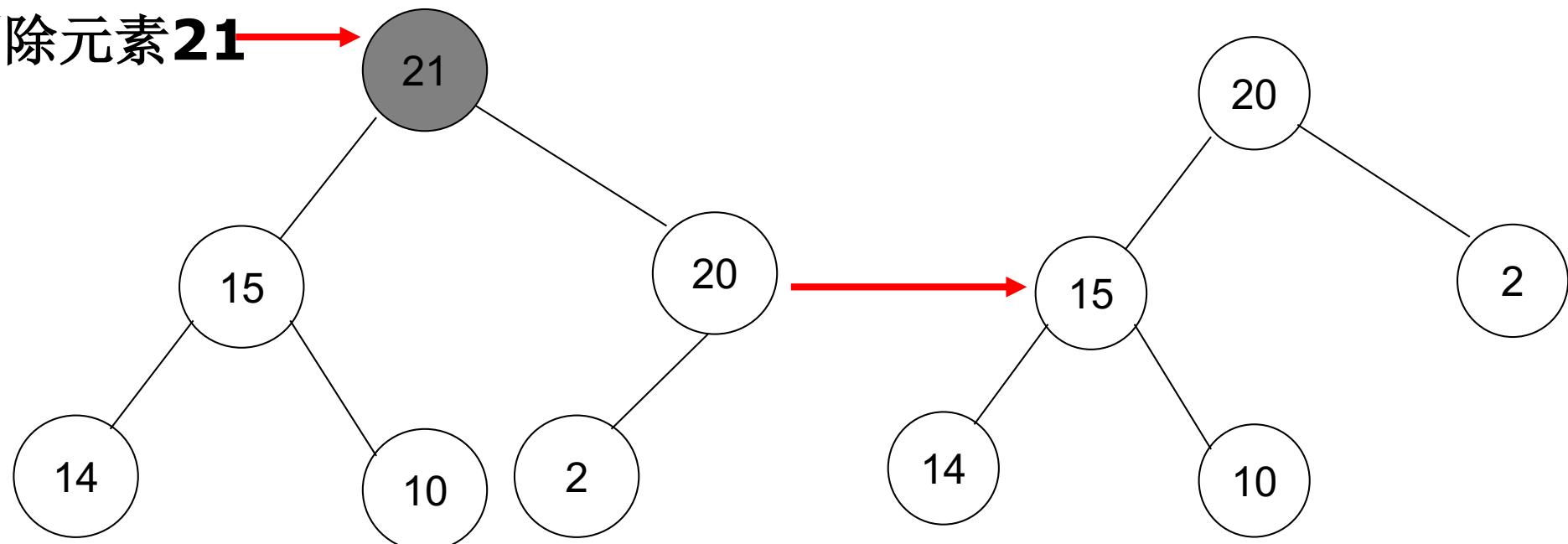
最大堆的插入



最大堆中删除最大元素



删除元素**21**



5 STL中的堆及优先队列



- ❑ 复习：什么是优先队列？
- ❑ 堆是优先队列的一种自然实现方法。
- ❑ STL提供了堆和优先队列的实现。
- ❑ STL中的堆：因为堆是完全二叉树，所以可以用数组(在STL中就是相邻)存储，然后调用以下函数实现堆的运算。
- ❑ 堆的函数：
 - ❑ **make_heap函数**：通常用于把一串数组建成堆，时间复杂度为线性。
 - ❑ **pop_heap函数**：弹出堆底元素，比较次数最多 $2 * \log(last - first)$ 。
 - ❑ **push_heap函数**：向堆中添加元素，比较次数最多 $\log(last - first)$ 。
 - ❑ **sort_heap函数**。



STL堆—MSDN例程



```
#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;

void main( )
{
    const int VECTOR_SIZE = 8;

    //Define a template class vector of int
    typedef vector<int> IntVector;

    //Define an iterator for template class vector of strings
    typedef IntVector::iterator IntVectorIt;

    IntVector Numbers(VECTOR_SIZE);

    IntVectorIt it ;

    //Initialize vector Numbers
    Numbers[0] = 4;  Numbers[1] = 10;  Numbers[2] = 70;  Numbers[3] = 10;
    Numbers[4] = 30;  Numbers[5] = 69;  Numbers[6] = 96;  Numbers[7] = 100;
```





```
//print content of Numbers
cout << "Numbers { ";
for(it = Numbers.begin(); it != Numbers.end(); it++)
    cout << *it << " ";
cout << " }\n" << endl ;

//convert Numbers into a heap
make_heap(Numbers.begin(), Numbers.end()) ;

cout << "After calling make_heap\n" << endl ;

//print content of Numbers
cout << "Numbers { ";
for(it = Numbers.begin(); it != Numbers.end(); it++)
    cout << *it << " ";
cout << " }\n" << endl ;

//sort the heapified sequence Numbers
sort_heap(Numbers.begin(), Numbers.end()) ;

cout << "After calling sort_heap\n" << endl ;

//print content of Numbers
cout << "Numbers { ";
for(it = Numbers.begin(); it != Numbers.end(); it++)
    cout << *it << " ";
cout << " }\n" << endl ;
```





```
//insert an element in the heap
Numbers.push_back(7) ;
push_heap(Numbers.begin(), Numbers.end()) ;

// you need to call make_heap to re-assert the heap property
make_heap(Numbers.begin(), Numbers.end()) ;

cout << "After calling push_heap and make_heap\n" << endl ;

//print content of Numbers
cout << "Numbers { " ;
for(it = Numbers.begin(); it != Numbers.end(); it++)
    cout << *it << " " ;
cout << " }\n" << endl ;

//remove the root element from the heap Numbers
pop_heap(Numbers.begin(), Numbers.end()) ;

cout << "After calling pop_heap\n" << endl ;

//print content of Numbers
cout << "Numbers { " ;
for(it = Numbers.begin(); it != Numbers.end(); it++)
    cout << *it << " " ;
cout << " }\n" << endl ;
```



}



STL中的优先队列

包含头文件：#include <queue>

使用命名空间：using namespace std;

定义优先队列的方法：

priority_queue<int> q1; //优先队列中的结点为整型数据

priority_queue<node> q2; //优先队列中的结点为自定义
类node对象

使用方法：优先队列和队列的使用方法基本一致。但要注意，因为优先队列需要根据结点的大小关系确定结点的位置，因此，如果优先队列中的结点是自定义类对象，则在该类中**必须重载关系运算符“<”**，以实现结点的大小比较运算。



STL优先队列例子



```
#include <iostream>
#include <queue>
using namespace std;
int main( )
{
    priority_queue<int> q1;
    q1.push(-25);
    q1.push(17);
    cout << q1.top() << endl; //输出17
    return 0;
}
```



ZUFE_OJ1150 优先队列

2009年数据结构课程专项竞赛1题目



```
#include <iostream>
#include <queue>
using namespace std;
struct node
{
    int no; //进入队列的序号
    int pri; //结点的优先级
    bool operator < (const node &b ) const
    {
        return b.pri<pri;
    }
};
int main( )
{
    ...
    priority_queue<node> nodes;
    ...
    return 0;
}
```



5 最大堆的应用



例1：POJ2051 Argus

- █ 比较高效的算法可以维护一个最小堆，不断的取最小值，然后将其进行更新。
- █ 建堆的时间复杂度是 $O(n\log n)$ ， n 为指令数目。操作过程的时间复杂度为 $O(k^*\log n)$ ， k 是前面产生的id个数。空间复杂度为 $O(n)$ 。

样例输入：

Register 2004 200

Register 2005 300

#

5

样例输出：

2004

2005

2004

2004

2005





例2：积水

《算法艺术与信息学竞赛》，P89

- █ 从边界入手：边界上积水一定为0.
- █ 从边界上最低的格子x开始做一次floodfill，填充它周围相邻的所有格子（将它们加入堆中）。
- █ 每个格子最多只能被加入一次，总时间复杂度 $O(mn\log mn)$ 。





例3：赛车，POJ2274 The Race

《算法艺术与信息学竞赛》，P89



练习题：最大堆的应用



1. ZOJ1200 Mining
2. POJ3110 Jenny's First Exam

