Chapter 11 PRIORITY QUEUES AND HEAPS

优先级队列

```
template<class T, class Container = vector<T>, class Compar
e = less<Container::value_type>>

class priority_queue
{
}
```

- 优先级队列类是容器适配器
- T是元素值的类型
- **Container容器默认是vector<T>**
- **规定容器需要支持随机访问和方法front, push_back, pop_back**（故底层容器只能为向量/双端队列）
- **默认为less比较 值越大 优先级越高（最顶层）**
- greater比较 值越小 优先级越高

EXAMPLE **SUPPOSE** my_set **IS A** set **Container of doubles; the ordering of items is irrelevant. to construct a** priority_queue **container** pq **that has a copy of the items in** my_set:

priority_queue<**double**> pq (my_set);

the largest item in my_set **will be at the front of** pq **because** less<**double**> **is the default comparator.**

- pop()和push()中采用push_heap和pop_heap 使用堆的方式进行调整

堆

- 堆是完全二叉树（与二叉查找树完全无关！）
  - 根是值最大的（优先级最高）【默认less比较】
  - 左子树和右子树都是堆（完全二叉树）
- **从上到下有序（二叉查找树是从左至右）**
- 堆存储在**数组**中
  - 支持随机访问 通过索引
  - 对于堆（完全二叉树）中的每一项，都可以为它关联一个0至n(t)-1的位置，即索引
  - 堆（完全二叉树）可与数组相互转换
  - 找子女，索引i处项的子女位于2i+1和2i+2
  - 找父母，索引j处项的父母位于(j-1)/2
- push_heap方法的接口

```
//前置条件：从first（包括在内）到last-1（不包括在内）之间的项是一个堆。
//后置条件：位于last-1上的值被插入到堆中。
template<class RandomAccessIterator, class Compare>
inline void push_heap(RandomAccessIterator first,
                      RandomAccessIterator last, Compare comp);
```

  - 将位于last-1位置的item存入value中，此时hole处于last-1位置，若在value里的item比hole的父母节点的item大，则将hole上移
  - 最坏情况：插入项大于first项（根），于是while循环的迭代次数与树高成正比，又因为完全二叉树的高度和n成对数关系，所以最坏时间复杂度为logn
  - 平均情况，堆中大约一半的项比插入项小，又因为堆的性质，大部分比插入项小的项将位于或接近树的底层位置，即while循环迭代的平均数量小于3，所以平均时间复杂度为常数
- pop_heap方法的接口

```
//前置条件: 堆非空。
//后置条件: 从堆中删除last-1位置上的项。
template<class RandomAccessIterator, class Compare>
inline void pop_heap(RandomAccessIterator first,
                     RandomAccessIterator last, Compare comp);
```

- 过程一：最顶层的item是要被删除的节点，先将last-1位置的item存入value中，最顶层的item移至last-1位置，hole放在最顶层位置，将**hole的两个孩子item相比较，值更大的子树与hole交换位置（从上至下移动）**
- **注意last-1位置上的元素不是堆的一部分，当hole移到倒数第二层时，此时只有一个孩子（last-1位置不算），hole与唯一孩子交换位置**
- 过程二（相当于push_heap）：将value中的item放回堆中，**value与hole的父母item相比较**，hole移动到更大的父母节点的位置（从下至上移动）
- 最坏时间复杂度：logn
- 平均时间复杂度：logn

// 课堂笔记

堆的插入

- 插入元素实际为父母和孩子的交换 父母比孩子小 则孩子交换上去到父母的位置 通过(j-1)/2找到父母的位置
- 插入时间复杂度与树高有关 树高和元素为log 所以一般为O(logn) 最坏时间复杂度为O(n)

pop_heap 每次把最大的元素放到最后

比较子树中较大的 移动到hole的位置

当对整个堆做了pop_heap后，此时的数组呈现已排序

堆的排序最坏时间复杂度

[pop_heap最坏O(logn)+push_heap最坏O(logn)]*(n-1) = O(2logn*(n-1) ≈ O(logn*n)

堆的排序平均时间复杂度

[pop_heap平均O(logn)+push_heap平均O(1)]*(n-1) = O(2logn*(n-1) ≈ O(logn*n)

平均和最坏都要进行(n-1)次

堆的排序最坏和平均时间复杂度为O(logn*n)

应用：哈夫曼编码

重新编码使得信息压缩 减少浪费单位

但编码不通用 需要在特定系统和一一对应

避免二义性：prefix-free encoding（不会是前缀部分） 利用二叉树的二分 使得路径就是编码

e.g 010 和 10 即使后缀相同 但前序不同 使得二义性不会出现 因为在判断0时由于二分