13 第五讲 树（下）

笔记本:	浙江大学《数据结构》		
创建时间:	2025/4/6 16:59	**更新时间:**	2025/4/14 21:22
作者:	panhengye@163.com		
URL:	vscode-file://vscode-app/c:/Users/Lenovo/AppData/Local/Programs/cursor/reso...		

什么是堆?

优先队列（priority queue）：特殊的“队列”，取出元素的顺序是依照元素的优先权（关键字）大小，而不是元素进入队列的先后顺序

堆（heap）

> *A* **heap** *of things is usually untidy, and often has the shape of a hill or mound. Now, the house is a heap of rubble. A* **stack** *is usually tidy, and often consists of flat objects placed directly on top of each other. ...a neat stack of dishes. A* **pile** *of things can be tidy or untidy....a neat pile of clothes.*
>
> *heap* 通常指杂乱的、呈小山状的一堆东西，如：*Now, the house is a heap of rubble*（现在，房子成了一堆瓦砾）。*stack*通常是整齐的一叠，指扁平物体叠放起来，如：*a neat stack of dishes*（整齐的一叠盘子）。*pile* 既可指整齐的一叠，也可指杂乱的一堆，如：*a neat pile of clothes*（整齐的一叠衣服）。

堆有两个特性：

- 结构性：用数组表示的完全二叉树
- 有序性：任一结点的关键字是其子树所有结点的最大值（或最小值）
    - 最大堆（maxheap），也称为大顶堆：最大值
    - 最小堆（minheap），也称为小顶堆：最小值

堆的插入

☞ 最大堆的创建

```
typedef struct HeapStruct *MaxHeap;
struct HeapStruct {
        ElementType *Elements; /* 存储堆元素的数组 */
        int Size;              /* 堆的当前元素个数 */
        int Capacity;          /* 堆的最大容量     */
};
```

```
MaxHeap Create( int MaxSize )
{       /* 创建容量为MaxSize的空的最大堆 */
  MaxHeap H = malloc( sizeof( struct HeapStruct ) );
  H->Elements = malloc( (MaxSize+1) * sizeof(ElementType));
  H->Size = 0;
  H->Capacity = MaxSize;
  H->Elements[0] = MaxData;
     /* 定义“哨兵”为大于堆中所有可能元素的值，便于以后更快操作 */
  return H;
}
```

❖ 算法：将新增结点插入到从其父结点到根结点的有序序列中

```
void Insert( MaxHeap H, ElementType item )
{ /* 将元素item 插入最大堆H，其中H->Elements[0]已经定义为哨兵 */
    int i;
    if ( IsFull(H) ) {
        printf("最大堆已满");
        return;
    }
    i = ++H->Size; /* i指向插入后堆中的最后一个元素的位置 */
    for ( ; H->Elements[i/2] < item; i/=2 )
        H->Elements[i] = H->Elements[i/2]; /* 向下过滤结点 */
    H->Elements[i] = item; /* 将item 插入 */
}
```

比交换数据要快

堆的删除

取出根结点（最大值）元素，同时删除堆的一个结点

- 删除最后一个元素，放到堆顶
- 比较该元素和儿子的大小，将较大的一个往上挪，直到确保了完全二叉树的有序性

堆的建立

将已经存在的N个元素按最大堆的要求存放在一个一维数组中

两种思路

- N log N：按照插入的思路完成，即将元素一个一个插入到初始为空的堆中
- 线性复杂度：
    - 将N个元素按照输入顺序存入，先满足完全二叉树的**结构特性**
    - 调整各结点位置，以满足最大堆的**有序特性**

> 思考题：建堆时，最坏情况下需要挪动元素次数是等于树中各结点的高度和。问：对于元素个数为12的堆，其各结点的高度之和是多少？

对于元素个数为 12 的堆（完全二叉树），按结点高度（从结点到叶子的最长路径边数）计算各结点高度之和：

- **第 1 层（结点 1）**：到叶子最长路径边数为 3，高度 3。
- **第 2 层（结点 2、3）**：
    - 结点 2：到叶子最长路径边数为 2，高度 2。
    - 结点 3：到叶子最长路径边数为 2，高度 2。
- **第 3 层（结点 4、5、6、7）**：
    - 结点 4、5、6：到叶子最长路径边数为 1，高度均为 1。
    - 结点 7：无子孙（叶子），高度 0。
- **第 4 层（结点 $8 \sim 12$）**：均为叶子，高度 0。

求和：$3 + 2 + 2 + 1 + 1 + 1 + 0 + 0 + 0 + 0 + 0 + 0 = 10$。

因此，元素个数为 12 的堆，各结点高度之和是 10。

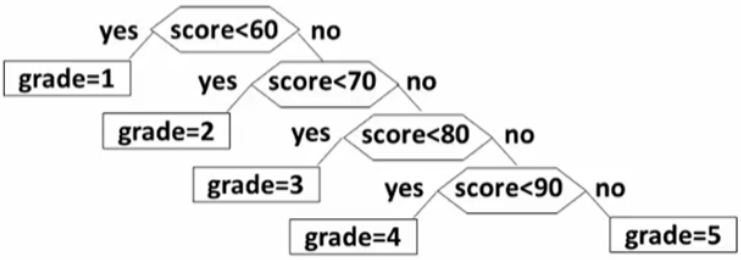
哈夫曼树与哈夫曼编码

什么是哈夫曼树（Huffman Tree）

[例] 将百分制的考试成绩转换成五分制的成绩

```
if( score < 60 )  grade =1;
else if( score < 70 ) grade =2;
else if( score < 80 ) grade =3;
else if( score < 90 ) grade =4;
else grade =5;
```

判定树：

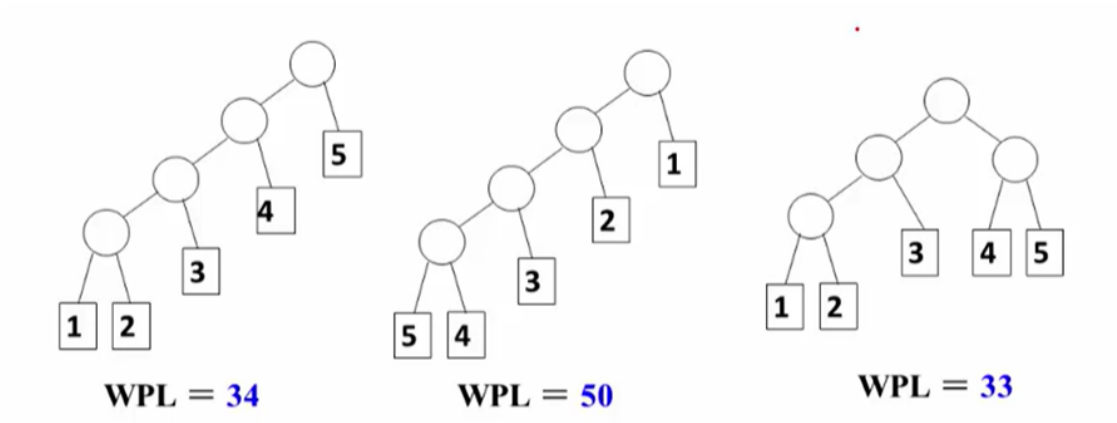


怎么根据不同的频率构造更有效的搜索树?

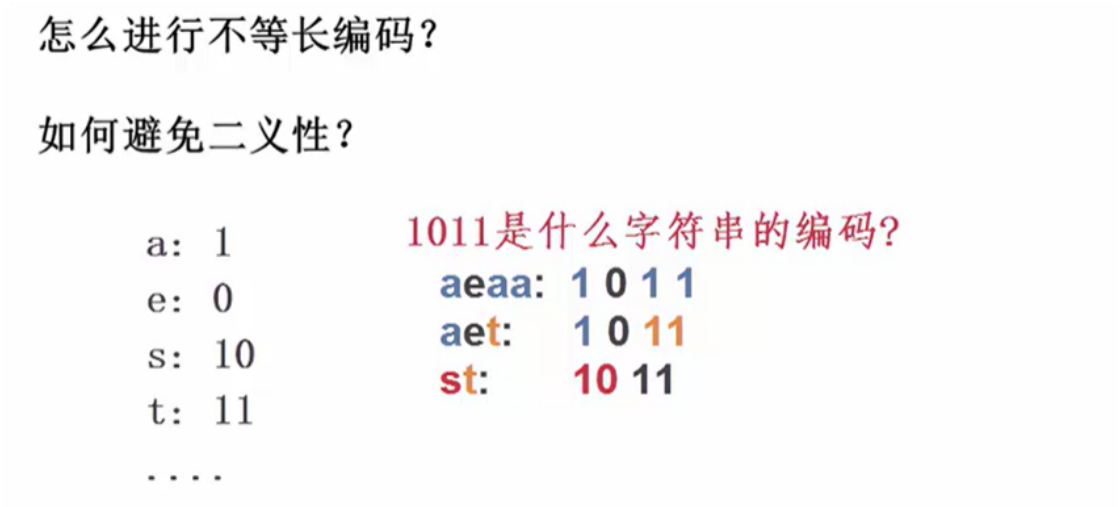
带权路径长度（Weighted Path Length，WPL）

- 在一棵树中，从根节点到某一节点的路径长度与该节点上权值的乘积，称为该节点的带权路径长度。而树中所有叶子节点的带权路径长度之和，称为这棵树的带权路径长度
  - **计算方法**：假设有一棵二叉树，它有 n 个叶子节点，每个叶子节点 k_i 都有对应的权值 w_i，从根节点到叶子节点 k_i 的路径长度为 l_i，那么这棵二叉树的带权路径长度 WPL 的计算公式为：$WPL = \sum_{i=1}^{n} w_i \times l_i$
-

最优二叉树 或者 哈夫曼树： **WPL最小的二叉树**



哈夫曼编码



前缀码 prefix code: 任何字符的编码都不是另一字符编码的前缀，即可以无二义地解码

> 对于一棵二叉树，当所有的信息都存放在叶结点的时候，就不能存在二义性

用哈夫曼树编码的思路：从列表中选择两个最小的合并，直到所有的元素都用完

判别是不是前缀码的思路

- **定义**： 前缀码的核心性质是，集合中的任何一个码字都不是另一个码字的前缀
- **算法步骤：**
  - 将给定的所有码字（字符串）按字典序（字母顺序）进行排序。
  - 遍历排序后的码字列表。从第一个码字开始，检查它是否是紧随其后的下一个码字的前缀。
  - 具体来说，对于列表中的第 `i` 个码字 `code[i]`，检查第 `i+1` 个码字 `code[i+1]` 是否以 `code[i]` 开头 (例如，在 Python 中使用 `startswith()` 方法)。
  - 如果在任何一步检查中发现 `code[i+1]` 是以 `code[i]` 开头的，那么这组编码就不是前缀码，算法可以立即停止并返回 `False`。
  - 如果遍历完所有相邻的码字对，都没有发现前一个码字是后一个码字的前缀的情况，那么这组编码就是前缀码，返回 `True`。
- **示例：**
  - 编码集：`{ "10", "0", "11", "100" }`
  - 排序后：`{ "0", "10", "100", "11" }`
  - 比较 "0" 和 "10"： "10" 不以 "0" 开头。
  - 比较 "10" 和 "100"： "100" 以 "10" 开头。 **发现前缀！**
    - 结论：该编码集不是前缀码。
  - 编码集：`{ "0", "10", "11" }`
  - 排序后：`{ "0", "10", "11" }`
  - 比较 "0" 和 "10"： "10" 不以 "0" 开头。
  - 比较 "10" 和 "11"： "11" 不以 "10" 开头。
  - 遍历完成。
    - 结论：该编码集是前缀码。

集合及运算

集合运算：交、并、补、差，判定一个元素是否属于某一集合

并查集：集合并、查元素属于什么集合

例子：有10台电脑{1, 2, 3, . . . , 9, 10}，已知下列电脑之间已经实现了连接：

1和2、2和4、3和5、4和7、5和8、6和9、6和10

问：2和7之间，5和9之间是否是连通的？

解决思路：

（1）将10台电脑看成10个集合{1}, {2}, {3}, . . . , {9}, {10}

（2）已知一种连接“x和y”，就将x和y对应的集合合并；

（3）查询“x和y是否是连通的”就是判别x和y是否属于同一集合。

查找的实现

（1）查找某个元素所在的集合（用根结点表示）

```
int Find( SetType S[ ], ElementType X )
{  /* 在数组S中查找值为X的元素所属的集合 */
   /* MaxSize是全局变量，为数组S的最大长度 */
   int i;
   for ( i=0; i < MaxSize && S[i].Data != X; i++) ;
   if( i >= MaxSize ) return -1; /* 未找到X，返回-1 */
   for( ; S[i].Parent >= 0; i = S[i].Parent ) ;
   return i; /* 找到X所属集合，返回树根结点在数组S中的下标 */
}
```

合并的实现

- 分别找到X1和X2两个元素所在集合树的根结点
- 如果它们不同根，则将其中一个根结点的父结点指针设置成另一个根结点的数组下标。

```
void Union( SetType S[ ], ElementType X1, ElementType X2 )
{
    int Root1, Root2;
    Root1 = Find(S, X1);
    Root2 = Find(S, X2);
    if ( Root1 != Root2 ) S[Root2].Parent = Root1;
}
```

当x1和x2不属于同一子集时，才需要合并