阅读：3,335　　作者：解学武

静态树表查找算法及C语言实现

前面章节所介绍的有关在静态查找表中对特定关键字进行顺序查找、折半查找或者分块查找，都是在查找表中各关键字被查找概率相同的前提下进行的。

> 例如查找表中有 n 个关键字，表中每个关键字被查找的概率都是 1/n。在等概率的情况，使用折半查找算法的性能最优。

而在某些情况下，查找表中各关键字被查找的概率是不同的。例如水果商店中有很多种水果，对于不同的顾客来说，由于口味不同，各种水果可能被选择的概率是不同的。假设该顾客喜吃酸，那么相对于苹果和橘子，选择橘子的概率肯定要更高一些。

在查找表中各关键字查找概率不相同的情况下，对于使用折半查找算法，按照之前的方式进行，其查找的效率并不一定是最优的。例如，某查找表中有 5 个关键字，各关键字被查找到的概率分别为：0.1，0.2，0.1，0.4，0.2（全部关键字被查找概率和为 1 ），则根据之前介绍的折半查找算法，建立相应的判定树为（树中各关键字用概率表示）：

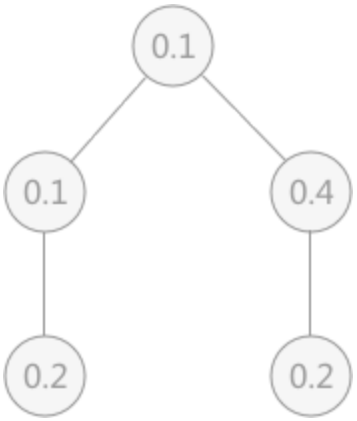


图 1 折半查找对应的判定树

折半查找查找成功时的平均查找长度的计算方式为：

ASL = 判定树中各结点的查找概率*所在层次

所以该平均查找长度为：

ASL=0.1*1 + 0.1*2 + 0.4*2 + 0.2*3 + 0.2*3 = 2.3

由于各关键字被查找的概率是不相同的，所以若在查找时遵循被查找关键字先和查找概率大的关键字进行比对，建立的判定树为：

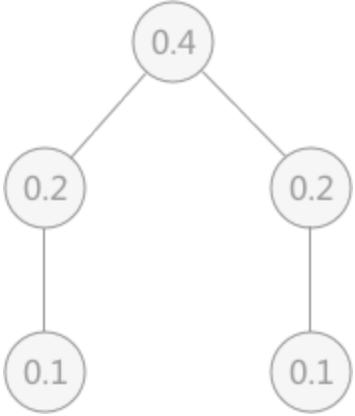


图 2 折半查找对应的新判定树

相应的平均查找长度为：

ASL=0.4*1 + 0.2*2 + 0.2*2 + 0.1*3 + 0.1*3=1.8

后者折半查找的效率要比前者高，所以在查找表中各关键字查找概率不同时，要考虑建立一棵查找性能最佳的判定树。若在只考虑查找成功的情况下，描述查找过程的判定树其带权路径长度之和（用 PH 表示）最小时，查找性能最优，称该二叉树为静态最优查找树。

> 带权路径之和的计算公式为：PH = 所有结点所在的层次数 * 每个结点对应的概率值。

但是由于构造最优查找树花费的时间代价较高，而且有一种构造方式创建的判定树的查找性能同最优查找树仅差 1% - 2%，称这种极度接近于最优查找树的二叉树为次优查找树。

次优查找树的构建方法

首先取出查找表中每个关键字及其对应的权值，采用如下公式计算出每个关键字对应的一个值：

$$\Delta P_i = \left| \sum_{j=i+1}^{h} w_j - \sum_{j=1}^{i-1} w_j \right|$$

其中 w_j 表示每个关键字的权值（被查找到的概率），h 表示关键字的个数。

表中有多少关键字，就会有多少个 $\triangle P_i$，取其中最小的做为次优查找树的根结点，然后将表中关键字从第 i 个关键字的位置分成两部分，分别作为该根结点的左子树和右子树。同理，左子树和右子树也这么处理，直到最后构成次优查找树完成。

代码实现为：

```
01.  typedef int KeyType;//定义关键字类型
02.  typedef struct{
03.      KeyType key;
04.  }ElemType;//定义元素类型
05.  typedef struct BiTNode{
```

```
06.      ElemType data;
07.      struct BiTNode *lchild, *rchild;
08.  }BiTNode, *BiTree;
09.
10.  //定义变量
11.  int i;
12.  int min;
13.  int dw;
14.  //创建次优查找树，R数组为查找表，sw数组为存储的各关键字的概率（权值），low和high表示的sw数组中的权值的范
15.  void SecondOptimal(BiTree T, ElemType R[], float sw[], int low, int high){
16.      //由有序表R[low...high]及其累计权值表sw（其中sw[0]==0）递归构造次优查找树
17.      i = low;
18.      min = abs(sw[high] - sw[low]);
19.      dw = sw[high] + sw[low - 1];
20.      //选择最小的ΔPi值
21.      for (int j = low+1; j <=high; j++){
22.          if (abs(dw-sw[j]-sw[j-1])<min){
23.              i = j;
24.              min = abs(dw - sw[j] - sw[j - 1]);
25.          }
26.      }
27.
28.      T = (BiTree)malloc(sizeof(BiTNode));
29.      T->data = R[i];//生成结点（第一次生成根）
30.      if (i == low) T->lchild = NULL;//左子树空
31.      else SecondOptimal(T->lchild, R, sw, low, i - 1);//构造左子树
32.      if (i == high) T->rchild = NULL;//右子树空
33.      else SecondOptimal(T->rchild, R, sw, i + 1, high);//构造右子树
34.
35.  }
```

完整事例演示

例如，一含有 9 个关键字的查找表及其相应权值如下表所示：

关键字：	A	B	C	D	E	F	G	H	I
权值：	1	1	2	5	3	4	4	3	5

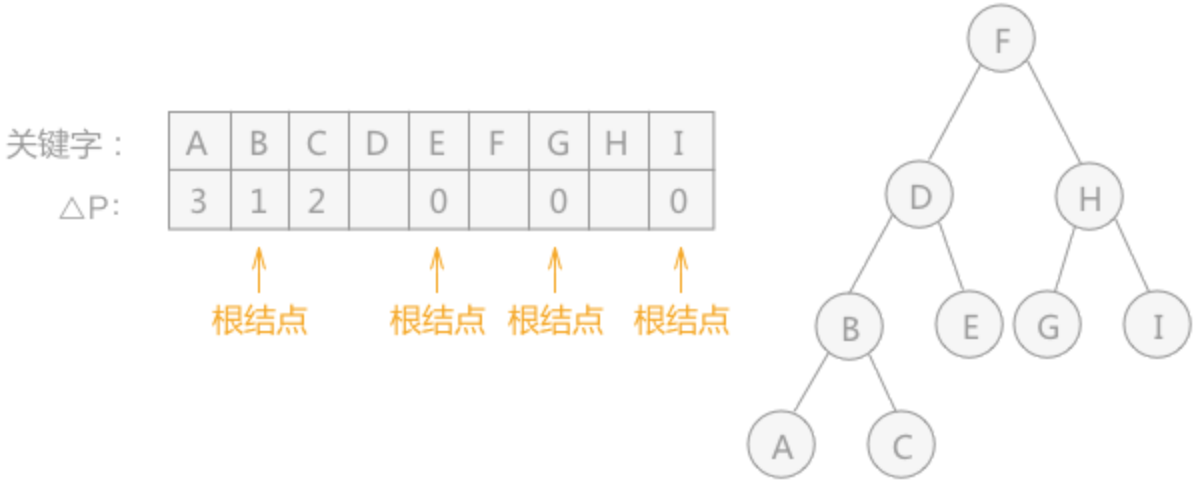
则构建次优查找树的过程如下：

首先求出查找表中所有的 ΔP 的值，找出整棵查找表的根结点：

> 例如，关键字 F 的 △P 的计算方式为：从 G 到 I 的权值和 - 从 A 到 E 的权值和 = 4+3+5-1-1-2-5-3 = 0。

通过上图左侧表格得知，根结点为 F，以 F 为分界线，左侧子表为 F 结点的左子树，右侧子表为 F 结点的右子树（如上图右侧所示），继续查找左右子树的根结点：

通过重新分别计算左右两查找子表的 △P 的值，得知左子树的根结点为 D，右子树的根结点为 H （如上图右侧所示），以两结点为分界线，继续判断两根结点的左右子树：



通过计算，构建的次优查找树如上图右侧二叉树所示。

后边还有一步，判断关键字 A 和 C 在树中的位置，最后一步两个关键字的权值为 0 ，分别作为结点 B 的左孩子和右孩子，这里不再用图表示。

> 注意：在建立次优查找树的过程中，由于只根据的各关键字的 P 的值进行构建，没有考虑单个关键字的相应权值的大小，有时会出现根结点的权值比孩子结点的权值还小，此时就需要适当调整两者的位置。

总结

由于使用次优查找树和最优查找树的性能差距很小，构造次优查找树的算法的时间复杂度为 O(nlogn)，因此可以使用次优查找树表示概率不等的查找表对应的静态查找表（又称为静态树表）。