第7章　查找

教学要求

相关知识点

- 顺序查找、折半查找算法
- 二叉排序树的插入和查找算法
- 哈希表、哈希函数、哈希地址、装填因子
- 哈希函数的选取原则及产生冲突的原因
- 采用开放地址法和链地址法解决冲突时，哈希表的建表方法、查找过程

学习重点

- 掌握顺序查找、折半查找、二叉排序树查找以及哈希表查找的基本思想和算法实现。

目录

7.1 查找的基本概念

7.1 查找的基本概念

查找的基本概念

1.关键字（key）

- 关键字(Key)是数据元素中唯一标识该元素的某个数据项的值，使用基于关键字的查找，查找结果是唯一的。

2.查找

- 根据给定的某个值，在数据集合中寻找一个其关键字等于给定值的数据元素。结果分两种情况：

（1）查找成功：数据集合中存在相应的数据元素；

（2）查找失败：数据集合中不存在关键字等于给定值的数据元素。

3.查找表（search table）

- 由同一数据类型的数据元素构成的数据集合，可以是一个数组或一个链表等数据类型。

(1) 静态查找表(Static Search Table)

如果一个查找表的操作只涉及查询某个特定的数据元素是否在查找表中或者检索满足条件的某个特定的数据元素的各属性值，无须修改查找表，这类查找表称为静态查找表。

静态查找表适用于：查找表一经生成，便只对其进行查找，而不进行插入和删除操作，或经过一段时间的查找之后，集中地进行插入和删除等修改操作。

(2) 动态查找表(Dynamic Search Table)

如果对一个查找表的操作在查找的同时需要动态地插入或删除的查找表则称为动态查找表。

动态查找表适用于：查找与插入或删除操作在同一个阶段进行，例如，在某些问题中，当查找成功时，要删除查找的记录，当查找不成功时，要插入被查找的记录。

4.查找算法的时间复杂度

- 衡量查找算法的优劣，主要是看要查找的值与关键字的比较次数。
- 通常把查找过程中对关键字的最多比较次数和平均比较次数作为衡量一个查找算法效率优劣两个基本技术指标。
- 前者叫做最大查找长度（**Maximun Search Length**），简记为**MSL**。
- 后者叫做平均查找长度（**Average Search Length**），简记**ASL**，计算公式为： **ASL=** $\sum_{i=1}^{n} P_i C_i$

其中： n是查找表长度；P_i是查找第i个数据元素的概率；C_i是找到第i个数据元素所需进行的比较次数。

5. 查找结构

- 各种数据结构都涉及查找操作，如线性表、队列、栈、树与图等。
- 为提高查找效率，需要专门为查找操作设置数据结构，这种面向查找操作的数据结构成为查找结构(**Search Structure**)。
- 本章讨论的查找结构有：静态查找表、动态查找表和哈希表。

7.2 静态查找算法

7.2 静态查找算法

静态查找算法通常采用顺序查找和折半查找。

顺序查找

- 顺序查找(Sequnce Search)又称线性查找(Linear Search)，主要用于在线性表中进行查找。顺序查找通常分为无序表的顺序查找和有序顺序表的顺序查找。

7.2 静态查找算法

1. 无序表的顺序查找

（1）基本思想

- 从线性表的一端开始，逐个检查关键字是否满足给定的条件。若查到某个元素的关键字满足给定条件，则查找成功，返回该元素在线性表中的位置；若已经查到表的另一端，还没有找到符合给定条件的元素，则返回查找失败的信息。
- 为了提高查找速度，可在算法中设置“哨兵”。哨兵就是待检查值，将它放在查找方向的“尽头”处，这样免去了在查找过程中每一次比较完之后都要判断查找位置是否越界，从而节省了时间。

(2) 静态查找表的顺序存储结构

```
#define MAXSIZE 100
typedef int keyType;
typedef struct
{   keyType key;    /*关键字域*/
}SElemType;
typedef struct
{   SElemType *elem;//数据元素存储空间基址
     int length;          /*表长度*/
} SeqTable;
```

(3) 顺序查找算法

```
int Search_Seq(SSTable ST,ElemType key)
/*在顺序表ST中顺序查找关键字等于key的数据元素。
若找到返回该元素在表中的位置；否则返回-1*/
{   ST.elem[ST.TableLen]=key;  /*"哨兵"*/
    for(i=0;ST.elem[i].key!=key;++i);
        /*从前往后找*/
    if(i<ST.TableLen) return i;
    else return -1;      /*未找到，返回-1*/
}
```

7.2 静态查找算法

- 对于有n个元素的表，给定值key与表中第i个元素的关键字相等，即定位第i个元素时，需要进行n-i+1次关键字的比较，即$C_i=n-i+1$，等概率$P_i=1/n$。
- 查找成功时，平均长度为：

$$ASL_{成功}=\sum_{i=1}^{n}P_iC_i=\sum_{i=1}^{n}P_i(n-i+1)=\frac{1}{n}*\frac{n(n+1)}{2}=\frac{n+1}{2}$$

- 查找不成功时，与表中各关键字的比较次数显然是n+1次，从而顺序查找不成功的平均查找长度为$ASL_{失败}=n+1$。
- 平均查找长度为：$ASL_{平均}=(ASL_{成功}+ASL_{失败})/2=(n+1)*3/4$

7.2 静态查找算法

- 缺点：当元素个数较大时，平均查找长度较大，查找效率低，它的时间复杂度为**O(n)**。
- 优点：对数据元素的存储没有要求，顺序存储或链式存储皆可。对表中数据元素关键字的大小是否有序也没有要求，无论数据元素是否按关键字有序均可应用。
- 同时还需要注意，对线性链表只能进行顺序查找。

2. 有序顺序表的顺序查找

- 如果在查找之前已知表是按关键字排序的，那么当查找失败时可以不必再比较到表的另一端就能返回查找失败的信息，这样能降低顺序查找失败的平均查找长度。
- 若表**L**是按关键字从小到大排列的，查找的顺序是从前往后，待查找元素的关键字为**key**，当查找到第**i**个元素时，发现第**i**个元素对应的关键字小于**key**，但第**i+1**个元素对应的关键字大于**key**，这时就可以返回查找失败的信息了，因为第**i**个元素之后的元素的关键字均大于**key**，所以表中不存在关键字为**key**的元素。

7.2 静态查找算法

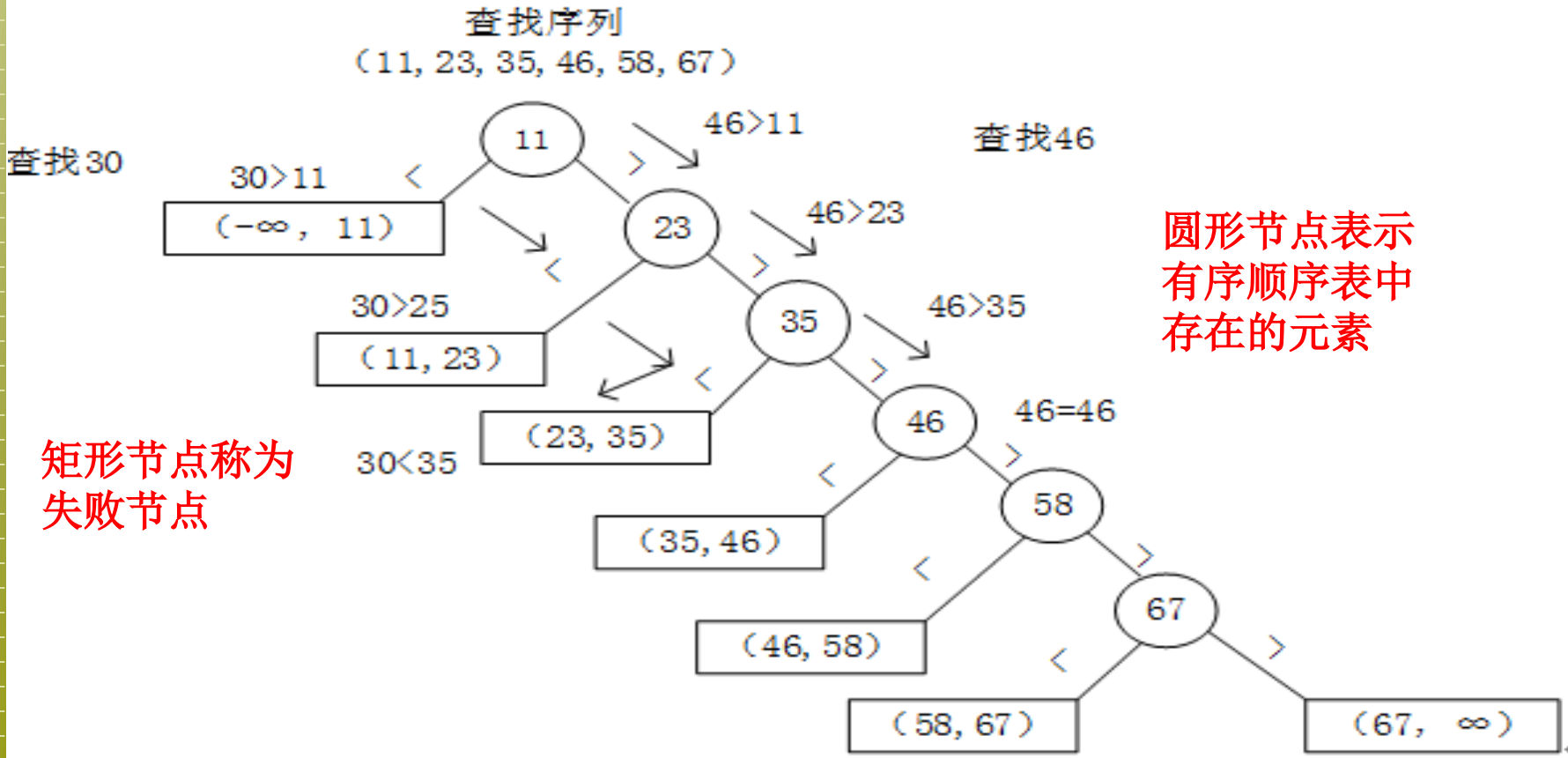

图 7.1 · · 有序顺序表的顺序查找判定树

7.2 静态查找算法

折半查找

- 折半查找(Binary Search)查找又称为二分查找，它的效率较高。但有一定的条件限制：要求线性表必须采用顺序存储结构，且表中元素必须按关键字有序。

1.折半查找的基本思想

- 首先将给定值key与有序表中间位置元素的关键字相比较，若相等，则查找成功，返回该元素的存储位置；若不等，则所需查找的元素只能在中间元素以外的前半部分或后半部分中，然后在缩小的范围中继续进行同样的查找，如此重复直到找到为止，或者确定表中没有所需要查找的元素，查找不成功，返回查找失败的信息。

2.折半查找的具体操作过程

- 假设顺序表ST是有序的。设两个指针，一个是low，指示查找表第一个记录的位置；另一个是high，指示查找表最后一个记录的位置。初始时low=0，high=ST.TableLen-1。设要查找的记录的关键字为key。当low<=high时，反复执行以下步骤：
- (1) 计算中间记录的位置mid，mid=(low+high)/2;

7.2 静态查找算法

- (2) 将待查记录的关键字key和r[mid].key进行比较：
- ① 若key==r[mid].key，查找成功，mid所指元素即为要查找的元素。
- ② 若key<r[mid].key，说明若存在要查找的元素，该元素一定在查找表的前半部分。修改查找范围的上界：high=mid-1，转①；
- ③ 若key>r[mid].key，说明若存在要查找的元素，该元素一定在查找表的后半部分。修改查找范围的下界：low=mid+1，转①；
- (3) 重复以上过程，当low>high时，表示查找失败。

7.2 静态查找算法

- 【例7.1】已知由11个数据元素构成的有序表(关键字即为数据元素的值)如下：

{4，12，20，21，35，58，63，77，81，87，98}

现要查找值为**21**的数据元素。

假设指针**low**和**high**分别指示待检查元素所在范围的下界和上界，指针**mid**指示区间的中间位置，即**mid=(low+high)/2=(0+10)/2=5**。

则在此例中，**low**和**high**的初值分别为**0**和**10**，即**[0,10]**为待检查范围。

- key=21时的查找过程

0	1	2	3	4	5	6	7	8	9	10
4	12	20	21	35	58	63	77	81	87	98
↑low				↑mid						↑high

查找范围中间位置的数据元素的关键字与给定值key相比较，因为前者大，说明待查找元素若存在，必在区间[low,mid-1]的范围内，令指针high指向第mid-1个元素，重新求得mid=(0+4)/2=2。

0	1	2	3	4	5	6	7	8	9	10
4	12	20	21	35	58	63	77	81	87	98
↑low		↑mid		↑high						

7.2 静态查找算法

仍以ST.elem[mid].key和key相比，因为ST.elem[mid].key<key，说明待检查元素若存在，必在[mid+1,high]范围内，则令指针low指向第mid+1个元素，求得mid的新值为3；

ST.elem[mid].key和key相等，则查找成功，所查元素在表中的序号等于指针mid的值。

0	1	2	3	4	5	6	7	8	9	10
4	12	20	21	35	58	63	77	81	87	98
			↑low	↑high						
			↑mid							

3.折半查找算法

```
int Binary_Search(SSTable L,ElemType key)
    /*在有序表L中查找关键字为key的元素，若存在则返回其位置，不存在则返回-1*/
{   int low=0,high=L.TableLen-1;//置区间初值
    int mid;
    while(low<=high)
    {    mid=(low+high)/2;  /*取中间位置*/
```

```
        if(L.elem[mid].key==key)
            return mid; /*查找成功返回所在位置*/
        else if(L.elem[mid].key>key)
            high=mid-1;  /*从前半部分继续查找*/
        else low=mid+1; /*从后半部分继续查找*/
    }
    return -1;  /*顺序表中不存在待查元素*/
}
```

❖4.折半查找的性能分析

圆形节点表示一个记录，节点中的值为该记录在表中的位置

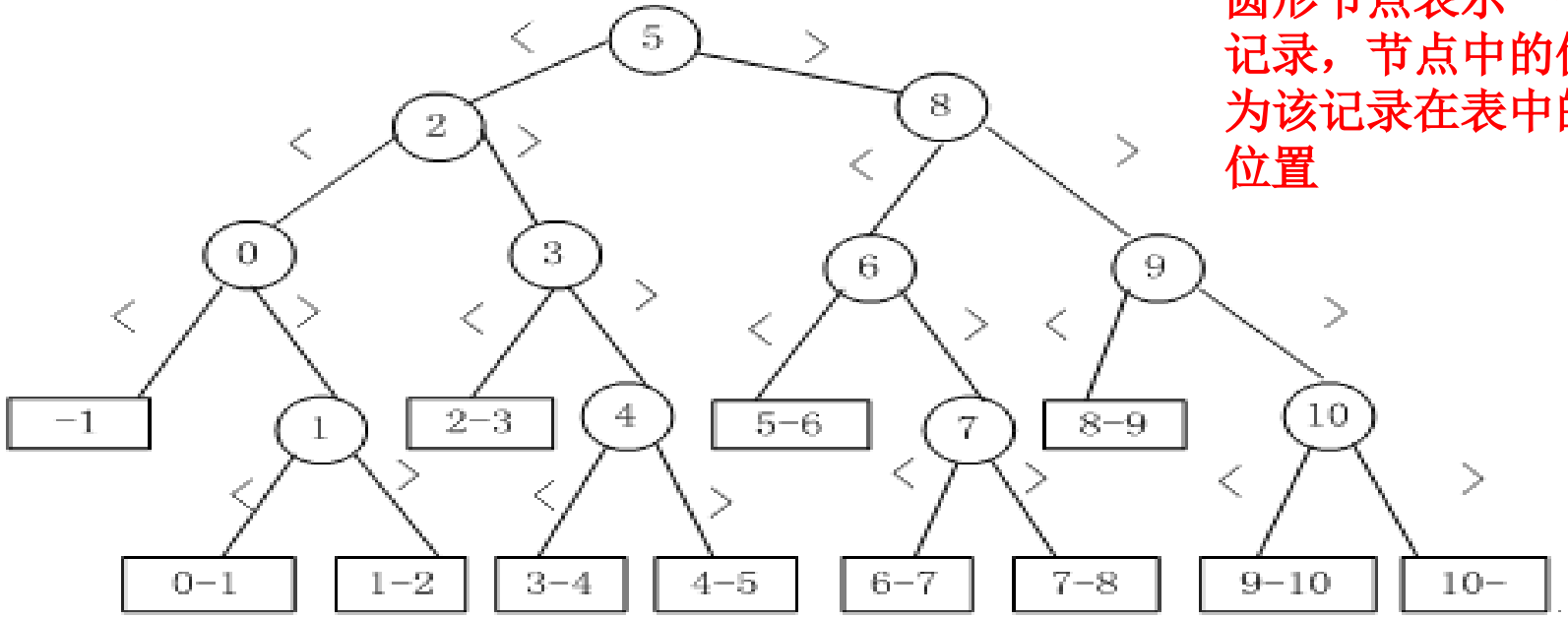

图 7.2 带外部节点的判定树

长方形节点为判定树外部节点，由判定树中所有节点的空指针域上的指针指向它叶节点表示查找不成功的情况

4.折半查找的性能分析

- 判定树特点：中序序列是一个有序序列，为二分查找的初始序列，所有的根节点值大于左子树而小于右子树。与根节点比较时若相等则查找成功，若待找的值小于根节点，则进入左子树继续查找，否则进入右子树查找，若找到叶子结点时还没有找到所需元素，则查找失败。

7.2 静态查找算法

- 查找成功时的查找长度为从根节点到目的节点的路径上的节点数，而查找失败时的查找长度为从根节点到对应失败节点的父节点的路径上的节点数。故用折半查找法查找到给定值的比较次数最多不会超过树的高度，时间复杂度为$O(\log_2^n)$。
- 折半查找的优点是比较次数较顺序查找要少，查找速度较快，执行效率较高；
- 缺点是表的存储结构只能为顺序存储，不能为链式存储，且表中元素必须是有序的。

7.3 动态查找表