第18讲 静态查找表

信息学院（智能应用研究院）
欧 新 宇

建议课时：2

第18讲 静态查找表

考核要点

- **考核大纲**

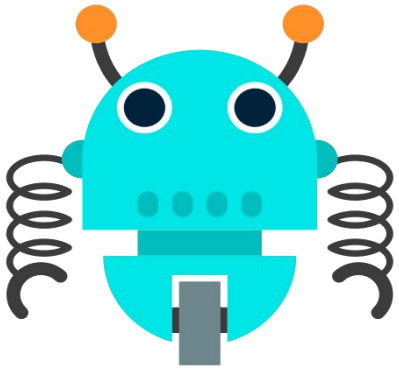
查找的基本概念、顺序查找、索引查找、折半查找、二叉搜索树、平衡二叉树、红黑树、B树和B+树、散列表以及各种查找算法的分析及应用

- **复习要点**
  - 熟悉顺序查找和索引查找的基本方法
  - 掌握折半查找的过程、构造判定树、分析平均查找长度
  - 掌握二叉排序树、二叉平衡树和红黑树的概念、性质和相关操作
  - 掌握B树的插入、删除和查找操作的过程，了解B+树的基本概念和性质
  - 掌握散列表的构造和冲突处理以及散列查找的特征和查找成功、失败的性能分析

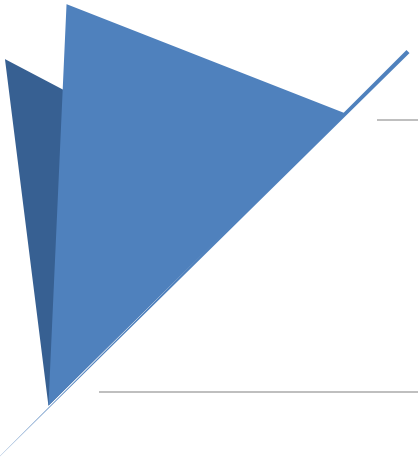
第18讲 静态查找表

本讲作业

- **必做题**
  - ✓ 【课堂互动18.1】顺序查找
  - ✓ 【课堂互动18.2】折半查找
  - ✓ 【课堂互动18.3】索引查找
  - ✓ 【课后作业18】[项目007] 查找的综合应用
- **选做题**
  - ✓ 【课前自测18】静态查找表
  - ✓ 【扩展练习18】静态查找表



- 查找的基本概念
- 顺序查找
- 折半查找
- 索引查找



查找的基本概念

/ 基本概念

/ 评价指标

查找的基本概念

数据源，由具有相同数据结构的元素构成

查找的基本概念

- **查找表**：由同一类型的数据元素（或记录）构成的集合

- **静态查找表**：仅做查找操作
- **动态查找表**：查找的同时对查找表进行修改（例如插入和删除等）

- **关键字**：数据元素中某个数据项的值，用于标识数据记录
  - **主关键字**：唯一标识数据元素的关键字
  - **次关键字**：可以标识若干个数据元素的关键字

查找的基本概念

查找算法的评价指标

搜索算法的**时间复杂度**分为搜索成功和搜索失败两种。为了确定一个指定关键字的值在记录表中的位置，需要进行关键字的比较匹配。比较次数的期望值称为搜索算法 **平均搜索长度（Average Search Length, ASL）**。

- **搜索成功**的平均搜索长度

$$\boldsymbol{ASL}_s = p_1 \times c_1 + p_2 \times c_2 + \ldots + p_n \times c_n$$

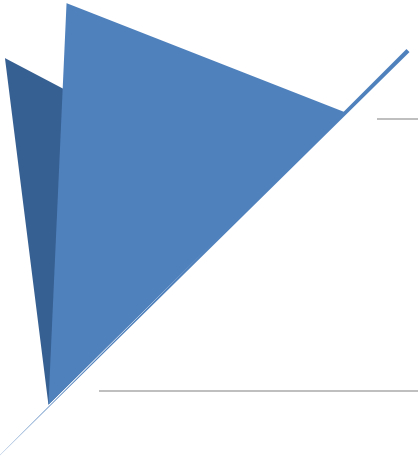
其中，p_i 是查找第i个元素的概率，p_i 是查找第i个元素所需要的比较次数。若每个元素的搜索概率相等，即 $p_i = 1/n$，则搜索成功时的平均搜索长度为：

$$\boldsymbol{ASL}_s = \sum_{i=1}^{n} i \times p_i = \frac{1}{n}\sum_{i=1}^{n} i_i = \frac{n+1}{2}$$

- **搜索失败**的平均搜索长度

搜索失败时，通常需要消耗最长搜索时间，因此总要进行 n 次关键字比较。

$$\boldsymbol{ASL}_s = n$$



顺序查找

/ 无序表的顺序查找

/ 有序表的顺序查找

/ 代码实现

/ 性能分析

/ 算法特点

顺序查找

顺序查找的基本概念

顺序查找（Sequential Search）是指查找序列表时，从表的一端开始，依次将记录的关键字和给定值进行比较，若某个记录的关键字和给定值**相等**，则查找成功；若查找完整个表后，仍**未找到**关键字和给定值相等的记录，则查找失败。

- 适用于顺序存储结构，也适用于链式存储结构
- 适用于有序表，也适用于无序表

顺序查找中数据元素的数据结构

```
typedef struct ElemType {
    KeyType key;
    InfoType otherinfo;
} ElemType
```

顺序查找中查找表的数据结构

```
typedef struct SSTable {
    ElemType *R;
    int length;
} SSTable
```

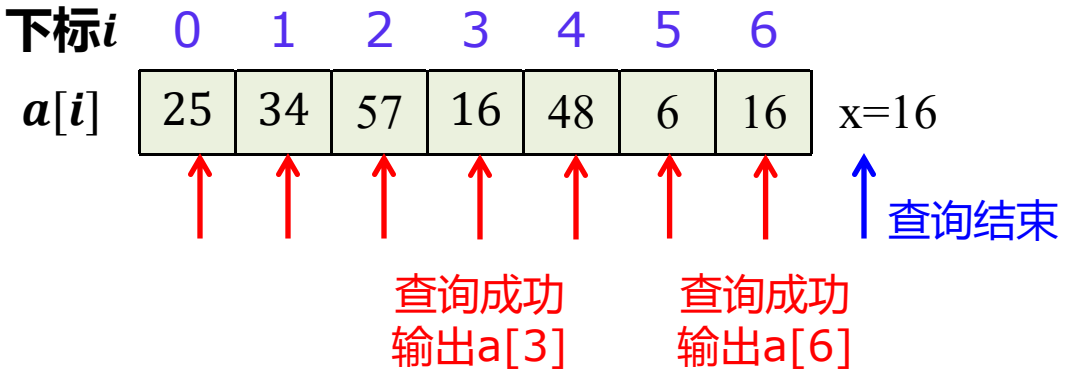
无序表的顺序查找

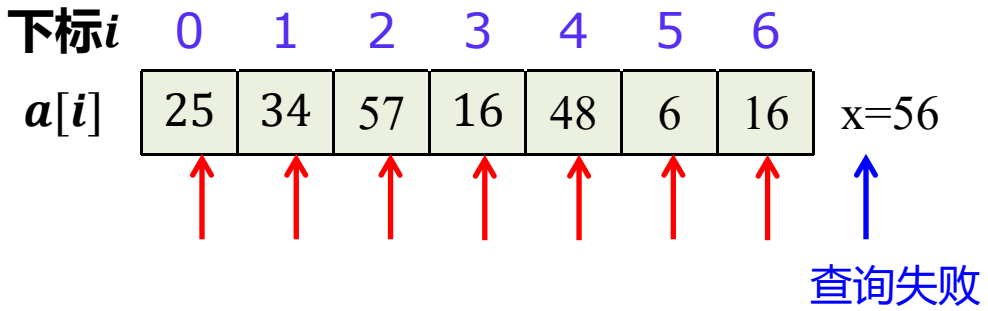
无序表顺序查询的基本思想

从查询表的开始，依次将待查询元素的关键字与查询表中元素的关键字比较。

- 若相等，则查询成功；
- 若搜索完整个表，不存在与待查询元素关键字值相等的元素，则查询失败。
- 待查询整个查询表检查完毕，则查询结束。

例1：在下列无序表中查找所有值为16和56的元素的存储位置





无序表的顺序查找

无序表顺序查询的代码实现

【算法步骤】

1. 从第一个元素起，依次将其和待查询值query相比较；
2. 若找到的值与query相等，则查找成功，返回该元素的序号i；
3. 若未查找到相同的值，则查找失败，返回0。

顺序查询无序表

```
int SeqSearch (SSTable ST, KeyType key){
    int i;
    for (i = 0; i < ST.length; i++) {
        if (ST.R[i] == key)    // 查找成功，返回i
            print(i);
        else
            print(-1);         // 搜索失败
    }
}
```

在无序表中，由于可能存在多个与待查询关键字相同的元素，
因此总是需要遍历完整个查询表。所以，**时间复杂度$ASL_s = ASL_f = O(n)$**

有序表的顺序查找

有序顺序表的定义

有序表：若线性表 L = (a_0, a_1,…,a_{n-1}) 中的元素关键字存在如下关系：

$key_i \leq key_{i+1}$

≤ 代表有序表中可能存在多个相同关键字的元素

则称线性表L为**有序表（递增）**，其中key_i表示a_i的关键字域。有序表可以视为一个已按关键字排序的有序集。

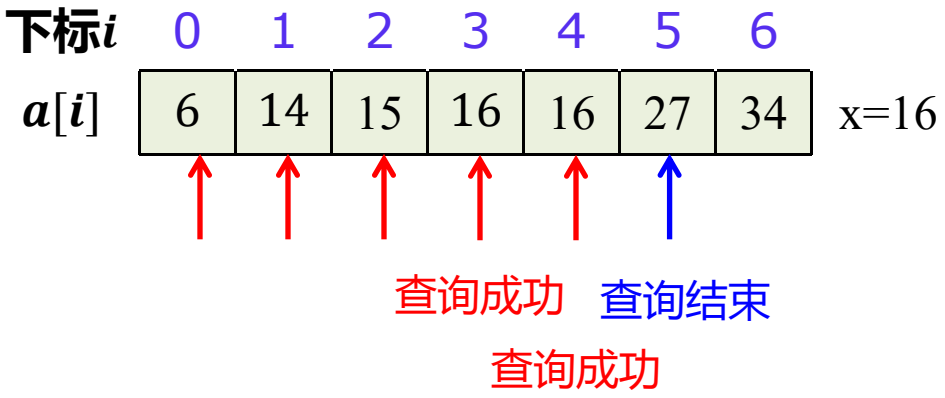
有序表的顺序查找

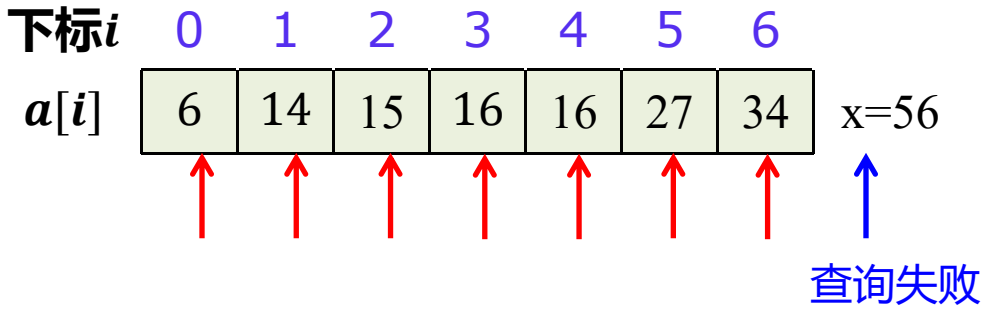
有序顺序表的查询步骤

从查询表的开始，依次将待查询元素的关键字与查询表中元素的关键字比较。

- 若相等，则查询成功；
- 若搜索到某个元素关键字大于指定元素x时，退出查询；
- 若搜索完整个查询表，仍不存在与待查询元素关键字值相等的元素，则查询失败。

例2：在下列有序表中查找所有值为16和56的元素的存储位置





有序表的顺序查找

有序表顺序查询的代码实现

【算法步骤】

1. 从第一个元素起，依次将其和待查询值query相比较；
2. 若找到的值与待查询元素相等的关键字，则输出元素序号；
3. 若找到的值比待查询元素关键字大的元素，则退出查询，并返回0；
4. 若遍历完整个程序表，仍未查找到相同的值，则查找失败，返回-1。

顺序查询有序表

```
int SeqSearch (SSTable ST, KeyType key){
    int i;
    for (i = 0; i < ST.length; i++) {
        if (L.R[i] == key)        // 查找成功，返回i
            print(i);
        else if (L.R[i] > key)    // 查询结束
            print(0);
    }
    print(-1);                    // 查询失败
}
```

每一轮都需要进行边缘判断，浪费时间

有序表的顺序查找

带“哨兵”的有序顺序表查询的代码实现

通过“哨兵”减少一层条件判断，在问题规模足够大时，将有效提高查询速度

顺序查询有序表

```
int SeqSearch (SSTable ST, KeyType key){
    int i;
    for (i = 0; i < ST.length; i++) {
        if (ST.R[i] == key)        // 查找成功，返回i
            print(i);
        else if (ST.R[i] > key)    // 查询结束
            print(0);
    }
    print(-1);                     // 查询失败
}
```

带“哨兵”的顺序查询

```
int SeqSearch (SSTable ST, KeyType key){
    int i;
    e = 0.0001    // e等于一个极小的数
    i  = ST.length;
    while (ST.R[i].key > key-e){
        if (ST.R[i] == key)    // 查找成功，返回i
            print(i);
        i--;
    }
    print(-1);                 // 查询失败
}
```

哨兵的设置方法是多样的

增加一个“哨兵”用于探测终止点，若循环过程中遇到“哨兵”，则停止探测。

顺序查找

顺序查找的性能分析

- 查询表中确定只存在一个符合条件的元素
  - 搜索成功：$ASL_s = \frac{1}{n}\sum_{i=1}^{n} i_i = \frac{n+1}{2}$
  - 搜索失败：$ASL_f = n$

查找概率相等

- 查询表中存在多个符合条件的元素
  - 搜索成功：
    - ✓ 无序表：$ASL_s = n$；
    - ✓ 有序表：$ASL_s = \frac{1}{n}\sum_{i=1}^{n} i_i = \frac{n+1}{2}$
  - 搜索失败：$ASL_f = n$

顺序查找

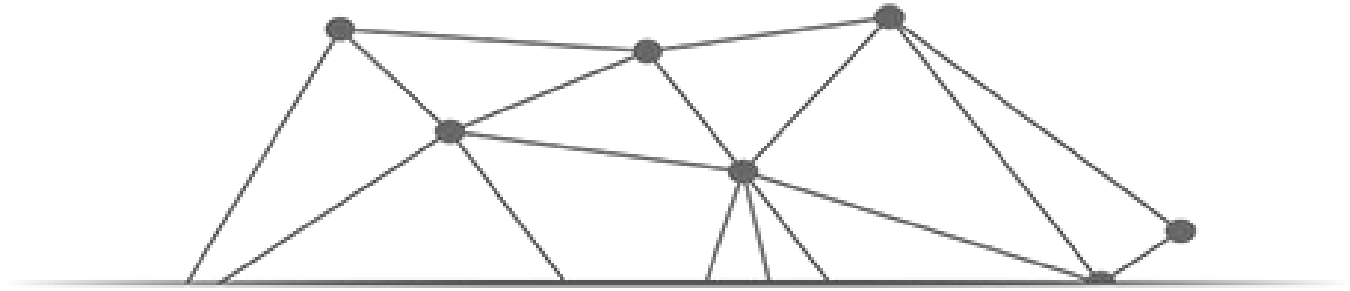
顺序查找算法的特点

算法简单，对表结构无任何要求（顺序和链式）

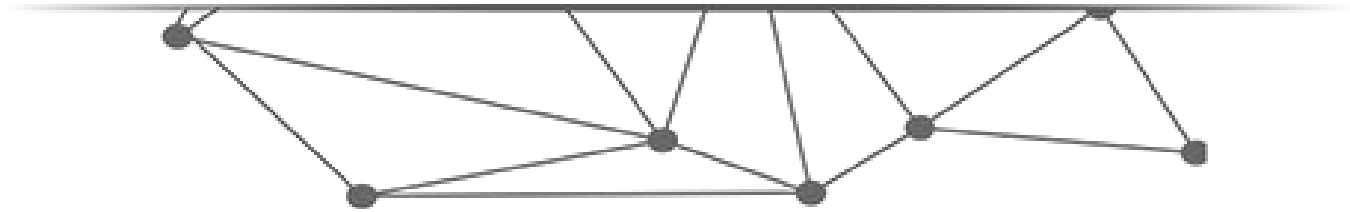
n很大时查找效率较低（时间长）

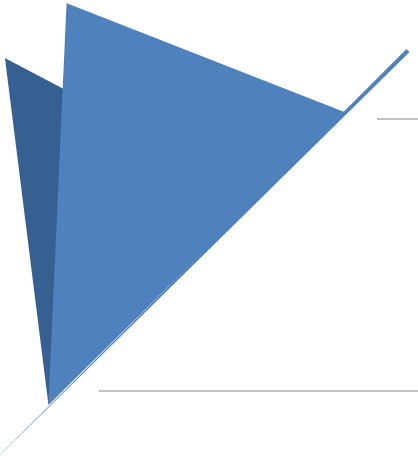
改进措施：非等概率查找时，可按照查找概率进行排序。

- ✓ 查找概率相等时，ASL相同；
- ✓ 查找概率不等时，如果从前向后查找，则按查找概率由大到小排列的有序表其ASL要比无序表ASL小。



课堂互动 17.1





折半查找

/ 基本原理

/ 查找过程

/ 代码实现

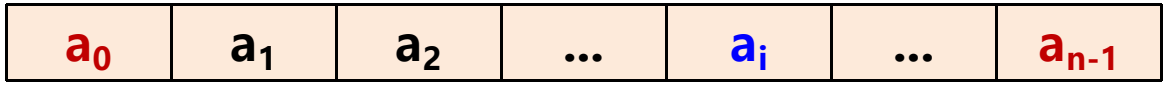
/ 性能分析

/ 算法特点

折半查找

二分查找基本思想

二分查找有序表 (a_0, a_1, a_2,… , a_{n-1})



- 如果有序表长 ≤ 0，搜索失败，返回
- 如果有序表长 > 0，取表中某个元素 a_i 与 x 进行比较
  - 如果a_i.key = x.key，搜索成功，返回
  - 如果a_i.key > x.key，**二分搜索**有序表 (a_0, a_1, a_2,… , a_i)
  - 如果a_i.key < x.key，**二分搜索**有序表 (a_i, a_{i+1}, a_{i+2},… , a_{n-1})

折半查找

二分查找基本思想

折半查找有序表 (a_0, a_1, a_2,… , a_{n-1})

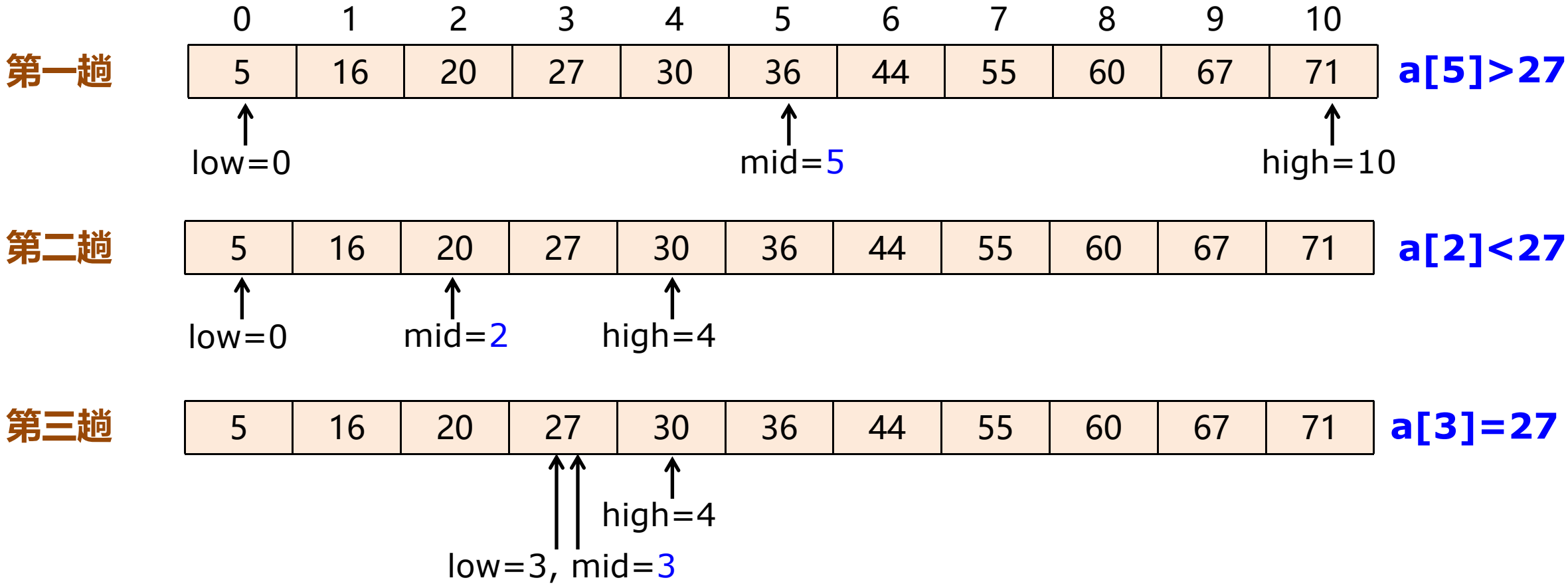
折半查找是二分查找的特殊形式，分割点为表的中点元素

- 如果有序表长 ≤ 0，搜索失败，返回
- 如果有序表长 > 0，取表中某个元素 $a_{(low+high)/2}$ 与 x 进行比较
  - 如果a_i.key = x.key，搜索成功，返回
  - 如果a_i.key > x.key，**折半查找**有序表 (a_0, a_1, a_2,… , $a_{(low+high)/2-1}$)
  - 如果a_i.key < x.key，**折半查找**有序表 ($a_{(low+high)/2+1}$,… , a_{n-1})

折半查找

折半查找的查找过程

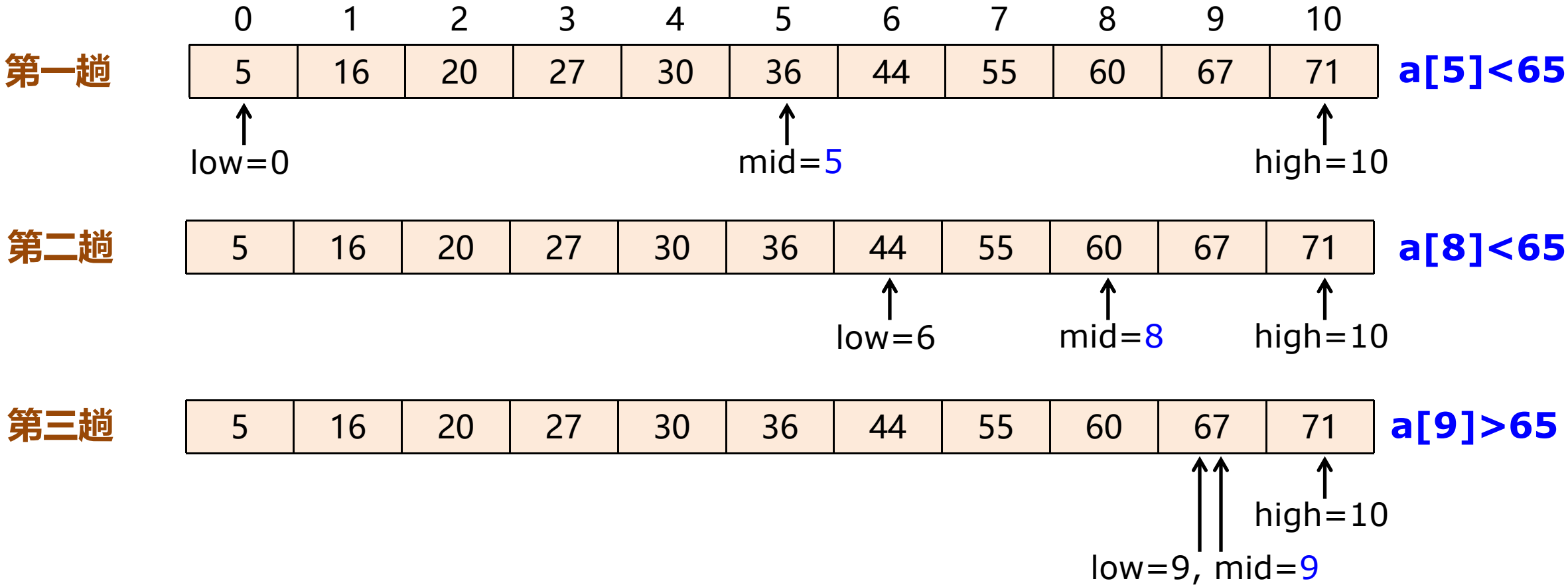
例2：在下列有序表中查找值为27和65的元素的存储位置　查询27



折半查找

折半查找的查找过程

例2：在下列有序表中查找值为27和65的元素的存储位置　查询65

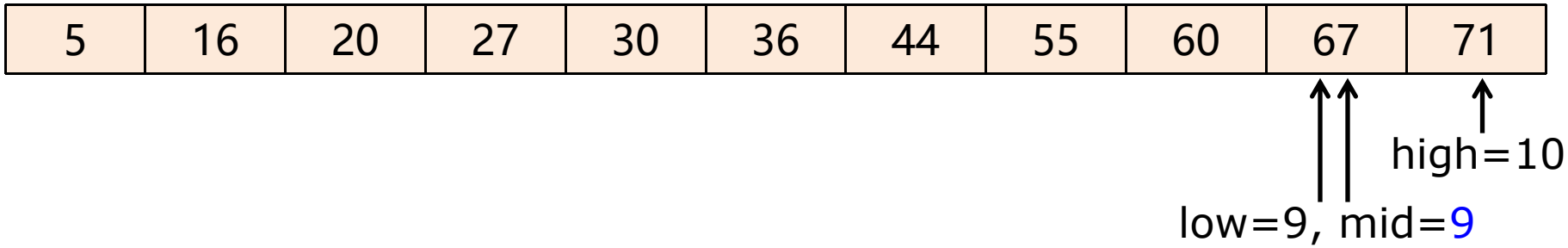


折半查找

折半查找的查找过程

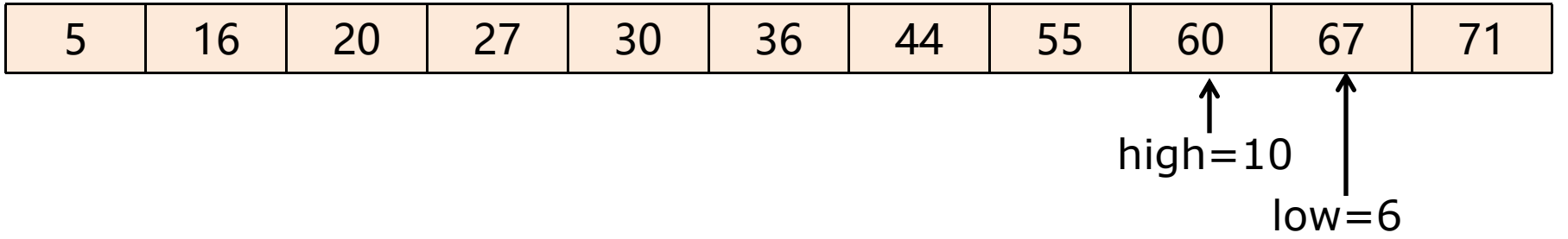
例2：在下列有序表中查找值为27和65的元素的存储位置　查询65

第三趟



a[9]>65

第四趟



low>high

查找失败

折半查找

折半查找算法的代码实现（迭代-非递归）

【算法步骤】

0. 设查询表长为n, low, high和mid分别指向待查元素所在区间的上界、下界和中点，key为给定值；
1. 初始时，令low=1, high=n, mid=(low+high)/2；
2. 让key与mid指向的记录比较：
   - ✓ 若key == R[mid].key，则查找成功；
   - ✓ 若key < R[mid].key，则high=mid-1；
   - ✓ 若key > R[mid].key，则low=mid+1。
3. 重复步骤2，直至low>high时，查找失败

在有序表中折半查找关键字等于Key的元素

```
int BinSearch (SSTable ST, KeyType key){
    low =1; high=ST.length;
    while(low<=high) {
        mid = (low+high)/2;      // 对半分割
        if(key == ST.R[mid].key)    return mid;
        else if(key<ST.R[mid].key)
            high = mid-1;
        else
            low = mid+1;
    }
    return -1;                          // 搜索失败
}
```

折半查找

折半查找算法的代码实现（递归）

【算法步骤】

0. 设查询表长为n, low, high和mid分别指向待查元素所在区间的上界、下界和中点，key为给定值；
1. 初始时，令low=1, high=n, mid=(low+high)/2；
2. 让key与mid指向的记录比较：
   - ✓ 若key == R[mid].key，则查找成功；
   - ✓ 若key < R[mid].key，递归调用BinSearch；
   - ✓ 若key > R[mid].key，递归调用BinSearch。
3. 重复步骤2，直至low>high时，查找失败

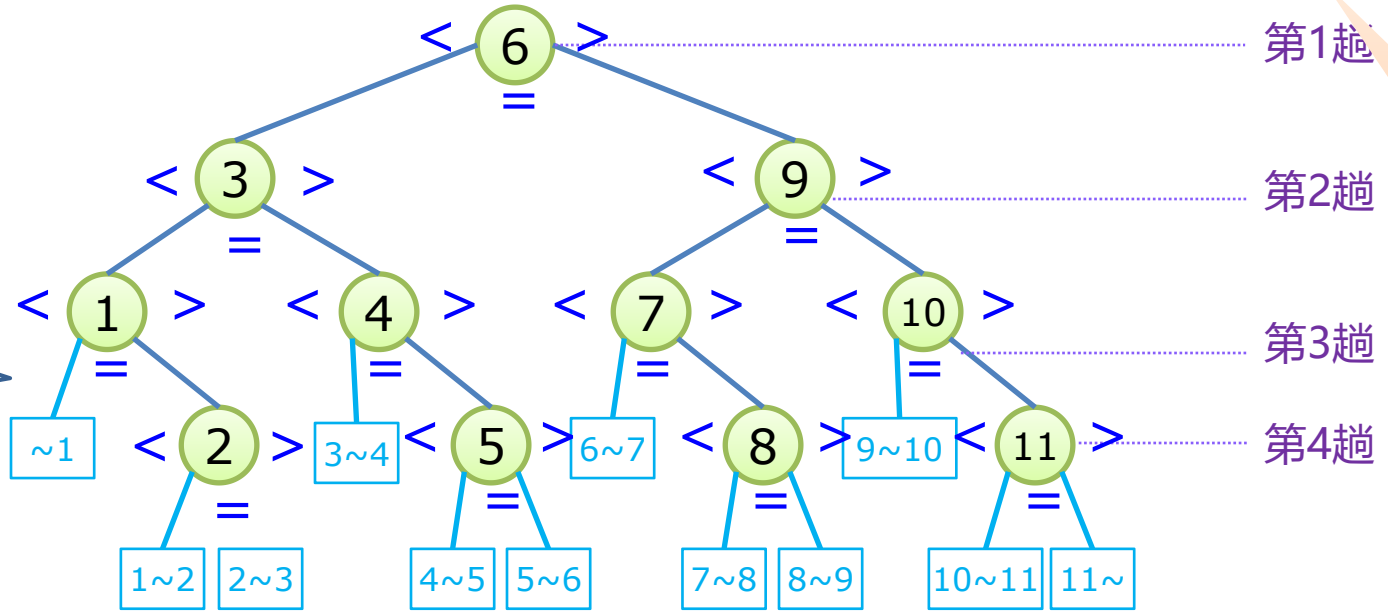
在有序表中折半查找关键字等于Key的元素

```
int BinSearch (SSTable ST, KeyType key, int low, int high){
    if (low<=high) {
        int mid = (low+high)/2;  // 对半分割
        if(key<ST.R[mid].key)
            return BinSearch(ST, x, low, mid-1);
        else if
            return BinSearch(ST, x, mid-1, high);
        else    return mid;        // 搜索成功
    }
    return -1;                     // 搜索失败
}
```

折半查找

折半查找的性能分析

假定每个元素的查找概率相等，折半查找可以用二叉树来描述。树中每个结点对应于一个记录，但结点值不是记录的关键字，而是记录在表中的位置序号。当前查找区间的中间位置做为根，左子表和右子表分别作为根的左子树和右子树，由此可得的二叉树称为**折半查找的判定树（平衡二叉树）**。



查找成功时的比较次数为该结点在判定树上的层次数，不超过树的深度

$d = \lfloor \log_2 n + 1 \rfloor$

左子树永远比根小，右子树永远比根大

成　功： $ASL = \frac{1}{11}(1\times1 + 2\times2 + 3\times4 + 4\times4) = 3$

不成功： $ASL = \frac{1}{12}(3\times4 + 4\times8) = 11/3$

折半查找

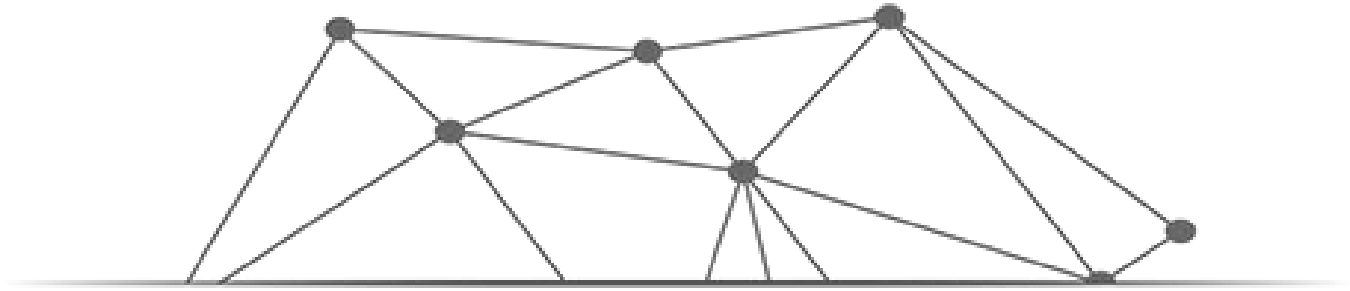
折半查找算法的特点

优点：比较次数少，查找效率高。

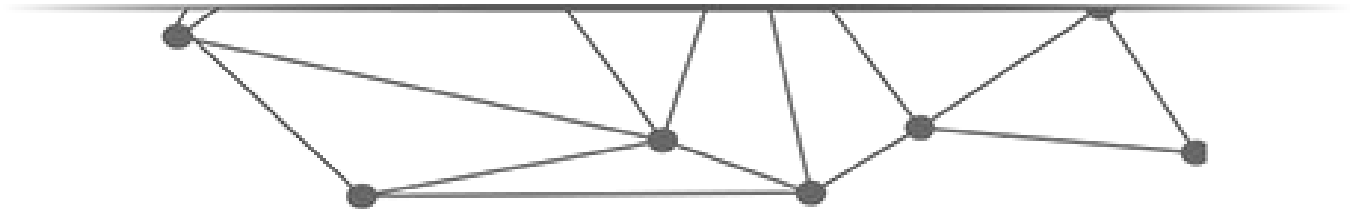
缺点：对表结构要求高，只能用于顺序存储的有序表，不宜于链式结构。

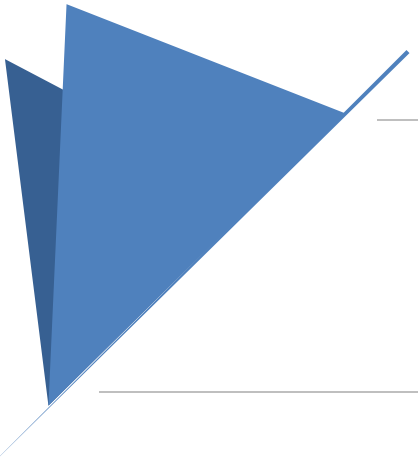
特点：折半查找每次将待查记录所在区间缩小一半，比顺序查找效率高，时间复杂度为$O(log_2n)$。

适用情况：折半查找要求先对元素进行排序，而排序是一种费时的运算。为了保证有序，每次进行插入和删除时，平均比较和移动表中一半元素，也是一种费时运算。不适用于经常变动的线性表。



课堂互动 17.2





索引查找（分块查找）

/ 基本原理

/ 查找过程

/ 性能分析

/ 算法特点

索引查找（分块查找）

数据太多，杂乱无章，查找困难

建立索引

先分析数据规律，建立索引；再根据索引进行快速定位

再在定位的地方进行细致搜索

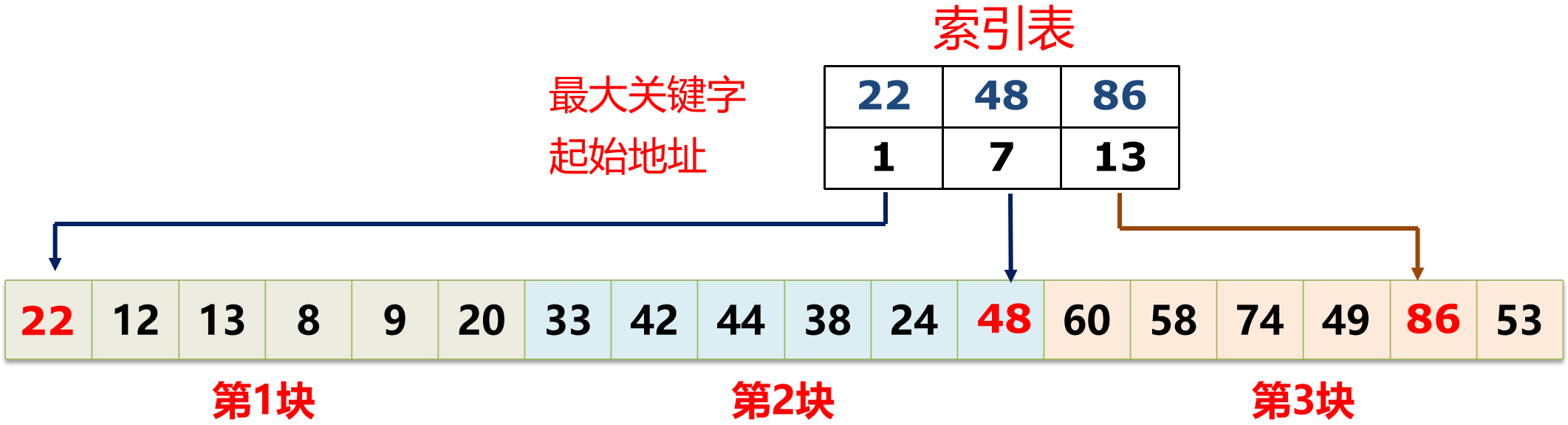
索引查找

索引查找的原理

分块有序：将查询表分成若干子表，并且每个子表的数值都比后一块中的数值更小

块内无序：子表内部**不要求有序**

索引表：将查询表各块中的最大关键字组成一个**索引表**，表中还包含每个子表的起始地址（即头指针）



索引查找

索引查找过程

索引表 01

对索引表使用折半查找。
因为索引表是有序表。

分块子表 02

确定待查关键字所在子表后，
在分块子表内部采用顺序查找。
因为各自表内部是无序的。

索引查找

索引查找性能分析

索引表查找　　块内查找

查找效率： $ASL_{bs} = L_b + L_w \approx \log_2(\frac{n}{s} + 1) + \frac{s}{2}$

S为每块内部的记录个数，n/s 即块的数目，当s = $\sqrt{n}$时，性能是最理想的，此时平均查找长度为$\sqrt{n} + 1$

例3：当n=9，s=3时，ASL_{bs}=3.5，而折半法为3.1，顺序法为5。

索引查找

索引查找算法的特点

优点：插入和删除比较容易，无需大量移动

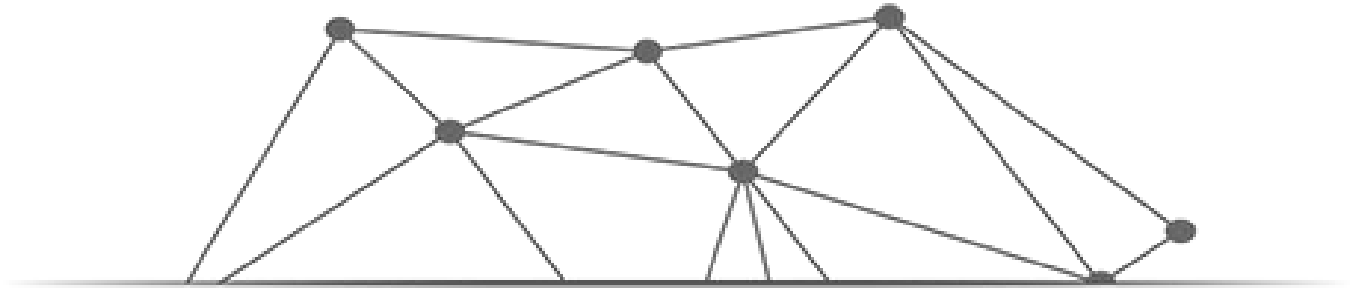
缺点：需要增加一个索引表的存储空间，并对初始索引表进行排序。

适用情况：如果线性表既要快速查找，又经常动态变化，则分块查找比较合适。

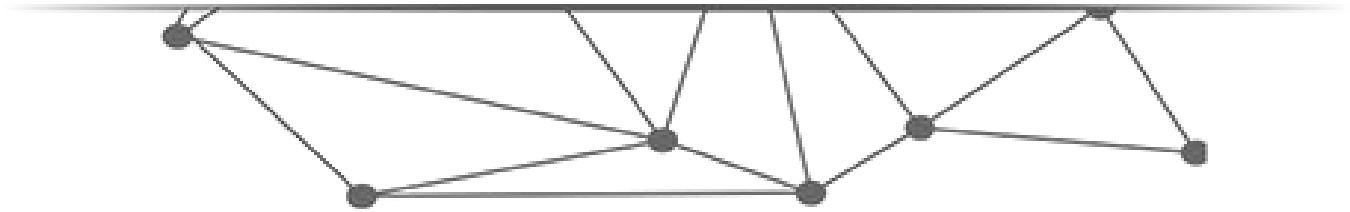
第18讲 静态查找表

小　结

	查找方法		
	顺序查找	**折半查找**	**索引查找**
查找时间复杂度	多个：O(n) 一个：$O(\frac{n+1}{2})$	$O(\log_2 n)$	与分块的查找方法有关 折半+顺序=$O(\log_2(\frac{n}{s}+1)+\frac{s}{2})$
特点	算法简单 对表结构无要求	对表结构要求高	对表结构有一定要求， 需要增加索引表的存储空间效率
查找效率	低	高	介于折半查找和顺序查找之间
适用情况	任何线性表	有序顺序表	块间有序、块内无序的顺序表
是否允许经常插入和删除操作	否	否	是



课堂互动 17.3



欧老师的联系方式

读万卷书 行万里路 只为最好的修炼

QQ：14777591 (宇宙骑士)

Email：ouxinyu@alumni.hust.edu.cn

Website: http://ouxinyu.cn

Tel：18687840023

地址：安宁校区 诚远楼201

南院 智能应用研究院A306-2