数据结构

第十五讲 字典的散列表示

孙猛

http://www.math.pku.edu.cn/teachers/sunm

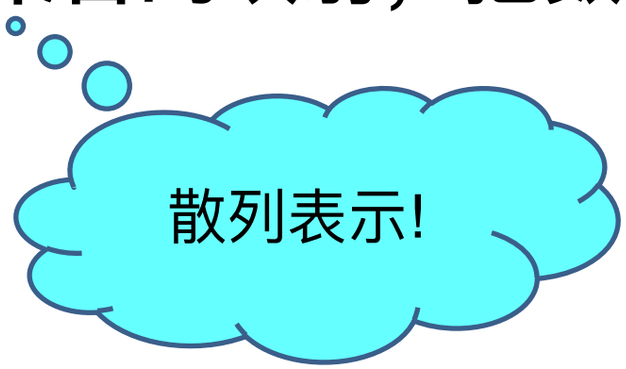
2017年11月27日

课程内容

- 字典的散列表示

字典的散列表示

- 动机:
  - 如果关键码是存储字典元素的数组下标，则可以直接找到字典元素!
  - 关键码未必总是整数!
    - Windows 序列号
- 关键码即使是整数，也未必适合做数组下标!
    - 北大学生学号10位数字，取值范围100亿
- 如何建立从关键码集合到适当整数集合的映射，把数据存放在映射值对应的位置。



基本概念

- **散列法（Hashing），也称为杂凑法或关键码－地址转换法。**

关键码 key —— 散列函数 h ——→ 散列地址 $h(key)$

- **这个 h 就称为散列函数，又称哈希（hash）函数或杂凑函数，是从可能的关键码到一个整数区间里（下标）的映射。**
- **存入关键码为key的数据时，将其存入表中第$h(key)$个位置。**
- **以key为关键码检索数据时，直接查第$h(key)$个位置的元素。**
- **散列函数h的值域应该是可以使用的整个地址空间，称为基本区域。**

基本概念

- **散列法（Hashing），也称为杂凑法或关键码－地址转换法。**

关键码 key —— 散列函数 h ⟶ 散列地址 $h(key)$

- **这个 h 就称为散列函数，又称哈希（hash）函数或杂凑函数，是从可能的关键码到一个整数区间里（下标）的映射。**
- **负载因子：α = 字典中结点数 / 基本区域能容纳的结点数**
- **碰撞： 如果 $key1 \neq key2$，而 $h(key1) = h(key2)$。**
- **发生碰撞的关键码称为同义词。**
- **散列表: 采用散列法表示的字典。**

设计散列表示

- 主要关注两个问题:
  - 散列函数的选择
    - 使得散列地址的分布尽可能均匀;
    - 理想的负载因子$\alpha>0.5$。
  - 碰撞的处理
    - 依据存储条件，同义词可以存放在基本区域中未被占用的单元;
    - 也可以在基本区域以外另开辟区域，例如$\alpha>1$时。

散列函数

- 不存在一种普遍适用的最佳的散列函数!
- 具体问题具体设计!
- 常用设计方法
  - 数字分析法
  - 折叠法
  - 中平方法
  - 基数转换法
  - 除余法

对于非整数关键码，常见的方式是先设计一种方式把它转换到整数，而后再用整数散列的方法。

数字分析法

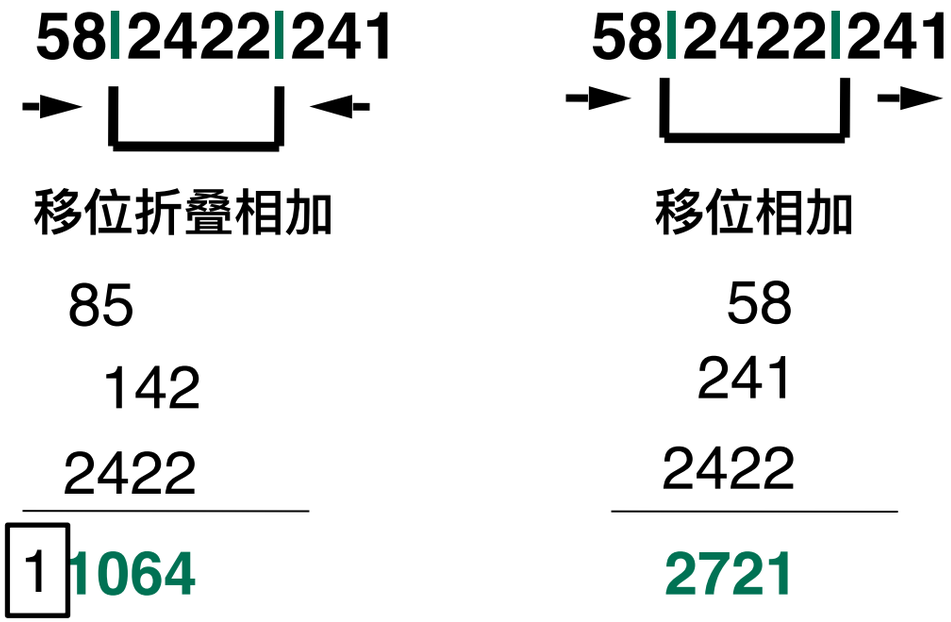
- 如果关键码位数比基本区的地址码位数多，丢掉分布不均匀的位留下均匀的位作为地址。
- 例如:

key	*h*(*key*)
136xxxx5678	**5678**
136xxxx0496	**0496**
130xxxx5555	**5555**
180xxxx1532	**1532**

- 过分依赖于关键码集合，通常适合静态的字典。

折叠法

- 如果关键码位数比地址码位数多,且分布比较均匀,则将关键码从某些地方断开,分为几部分,以几部分的叠加和(舍弃进位)作为地址.
- 例如：key =582422241



中平方法

- 先求出关键码的平方，然后取中间几位作为地址。
- 例如:

key = 4731,

4731^2=22382361,

h(4731)=**382**.

基数转换法

- 把关键码看成基数为*r1*的数，将它转换成基数为*r2*的数，用数字分析法取中间几位作为散列地址。*r1*和*r2*互素。

key =236075, $r1$=13, $r2$=10,

$(236075)_{13} = 2\times13^5+3\times13^4+6\times13^3+7\times13+5$

$= (841547)_{10}$

$h(key) =$ **4154**

除余法

- 关键码除以某个不大于基本区域大小(m)的数p后的余数为散列地址，即

  h(key)=(int)key%p;

- 数 p 一般选素数。
- 常用素数：
  - m = 128, 256, 512, 1024,
  - p选择 127, 251, 503, 1019。
- 除余法使用最广，常用于动态字典和关键码没有规律出现的情况。
- 也可以用于将其他散列函数得到值归缩到所需基本区域。

散列函数选取的原则

- 一般而言，散列函数的选取应根据具体问题具体分析的原则，针对具体字典元素关键码集合的特性，加上空间、时间的条件，去构造相对理想的散列函数。
- 散列函数计算出的地址应该尽可能均匀地分布在希望的地址空间中。
- 散列函数应该尽可能简单，以提高关键码到地址的转换速度。

碰撞的处理

- 散列表的插入和检索都要考虑对碰撞的处理。
  - 如果以$h(key)$为地址的空间未被占用:
    - 执行检索, 检索失败;
    - 执行插入, 将元素存入该存储空间。
  - 如果以$h(key)$为地址的空间已被占用:
    - 执行检索,则需要检查已经存放的元素(甚至同义词)是否是需要检索的元素;
      - 检查同义词需要知道处理碰撞的方法
    - 如果执行插入, 发生碰撞。
      - 涉及对同义词的处理
- 碰撞处理方法:
  - 开地址法
  - 拉链法

开地址法

- 基本想法：
  - 出现冲突时元素无法保存在散列函数确定的位置，需设法为它在基本区域内另外安排一个位置。
  - 采用探查方式作为系统化的位置安排方式。
- 开地址法解决碰撞：
  - 在基本区域内形成一个同义词的探测序列，沿着探测序列逐个查找，直到满足下列条件为止：
    - 找到查找的元素(检索成功)，或者
    - 碰到一个未被占用的地址(可以插入同义词或者检索失败)。
  - 显然基本区域的负载因子必须小于1。

线性探查与双散列探查

- 基本区域内的一个**探查序列**:

$$H_i = (h(key)+d_i)\%m,$$

其中:m为表长, d_i为增量序列, $i=1,2,\ \ldots,k\ \ (k \le m\text{-}1)$。

- 如果增量序列满足:

$$d_i=i,$$

则为**线性探查序列**。

- 如果增量序列满足:

$$d_i=i\times h_2(key)$$

则为**双散列探查序列**。

线性探查序列

- **对关键码集合**

  K={18,73,10,5,68,99,22,32,46,58,25},

- **我们取**h(key)=key%13, **则** ***h*(18)=5**, ***h*(73)=8**, h(10)=10, ***h*(5)=5**, h(68)=3, ***h*(99)=8**, h(22)=9, ***h*(32)=6**, h(46)=7, ***h*(58)=6**, h(25)=12。

散列地址	0	1	2	3	4	5	6	7	8	9	10	11	12
关键码	58	25		68		18	5	32	73	99	10	22	46
						5	32	46	99	22			25
							58						

- **从例子中看到线性探测容易出现堆积现象，即在处理同义词的过程中又添加了非同义词冲突(其他探查法也可能有类似情况)。**

双散列函数探查序列

- **对关键码集合**

  K={18,10,73,7},

- **我们取**h(key)=key%5，h_2(key) =key%3+1，**则**

  h(18)=3，h(10)=0，h(73)=3，h(7)=2;

  h_2(18)=1，h_2(10)=2，h_2(73)=2，h_2(7)=2。

散列地址	0	1	2	3	4
关键码	10		73	18	7
	73		7	73	

开地址法散列表上的检索

- **对给定key值：**

1. **根据散列函数求出散列地址**
2. **若该位置无记录，则检索失败结束；**
3. **否则比较元素关键码，若与key 相等则检索成功结束；**
4. **否则根据本散列表设计的冲突处理方法找出“下一地址”，回到步骤2用这个位置检查。**

- 对关键码集合K={18,10,73,7},
- 我们取h(key)=key%5，h_2(key) =key%3+1,则 h(18)=3，h(10)=0，h(73)=3，h(7)=2；h_2(18)=1，h_2(10)=2，h_2(73)=2，h_2(7)=2.
- 检索关键码73。

散列地址	0	1	2	3	4
关键码	10		73	18	7

开地址法散列表的实现

- 存储结构
- 实现算法
  - 创建空散列表
  - 与顺序表的创建算法类似;
  - 假定有效的关键码值都大于0，因此创建空字典的时候都用0初始化关键码。
- 散列表的检索算法—用线性探查法解决碰撞;
- 散列表的插入算法—用线性探查法解决碰撞。

存储结构

```
typedef int KeyType;
typedef int DataType;

typedef struct {
    KeyType key;                /* 字典元素的关键码字段*/
    DataType value;             /* 字典元素的属性字段*/
} DicElement;

typedef struct {
    int m;                      /* m为基本区域长度 */
    DicElement  *elements;
} HashDictionary;

typedef HashDictionary *PHashDictionary;
```

散列表的检索算法

```
int linearSearch(HashDictionary *phash, KeyType key, int *position) {
    int d = h(key), inc;                                    /* 设 h 为散列函数  */
    for (inc = 0; inc < phash->m; inc++) {
        if (phash->elements[d].key == key) {
            *position = d; return 1;                        /* 检索成功 */
        }
        else if (phash->elements[d].key == 0) {
            *position = d; return 0;                        /* 检索失败，找到空位置 */
        }
        d = (d+1) % phash->m;
    }
    *position = -1;                                         /* 散列表溢出 */
    return 0;
}
```

散列表的插入算法

首先调用检索算法，如果检索失败，再根据提供的位置信息将元素ele插入。

```
int linearInsert(HashDictionary *phash, DicElement ele) {
    int position;
    if (linearSearch(phash, key, &position) ==1)
        printf("Key exists \n");                    /* 已有关键码为key的结点*/
    else if (position != -1)                        /*插入结点*/
        phash->elements[position] =ele ;
    else  return 0;                                 /* 散列表溢出 */
    return 1;
}
```

关于开地址法的讨论

- 对于用开地址法构造的散列表，删除结点时不能简单地将要删除的结点空间置为0;
- 这会引起检索方面的问题。
- 怎么处理?!

散列地址	0	1	2	3	4
关键码	10		73	18	7

散列地址	0	1	2	3	4
关键码	0		73	18	7

检索73失败

拉链法

- 基本思想：
  - 元素存在基本区域之外；
  - 为每个关键码建立一个链接表，所有关键字为同义词的记录存储在同一个链表中。
- 所有元素可以统一处理(无论是否冲突)。
- 允许任意的负载因子。
- 最简单的方法是把新元素插在链接表头。
  - 如果要防止出现重复关键码，就需要检查整个链表。

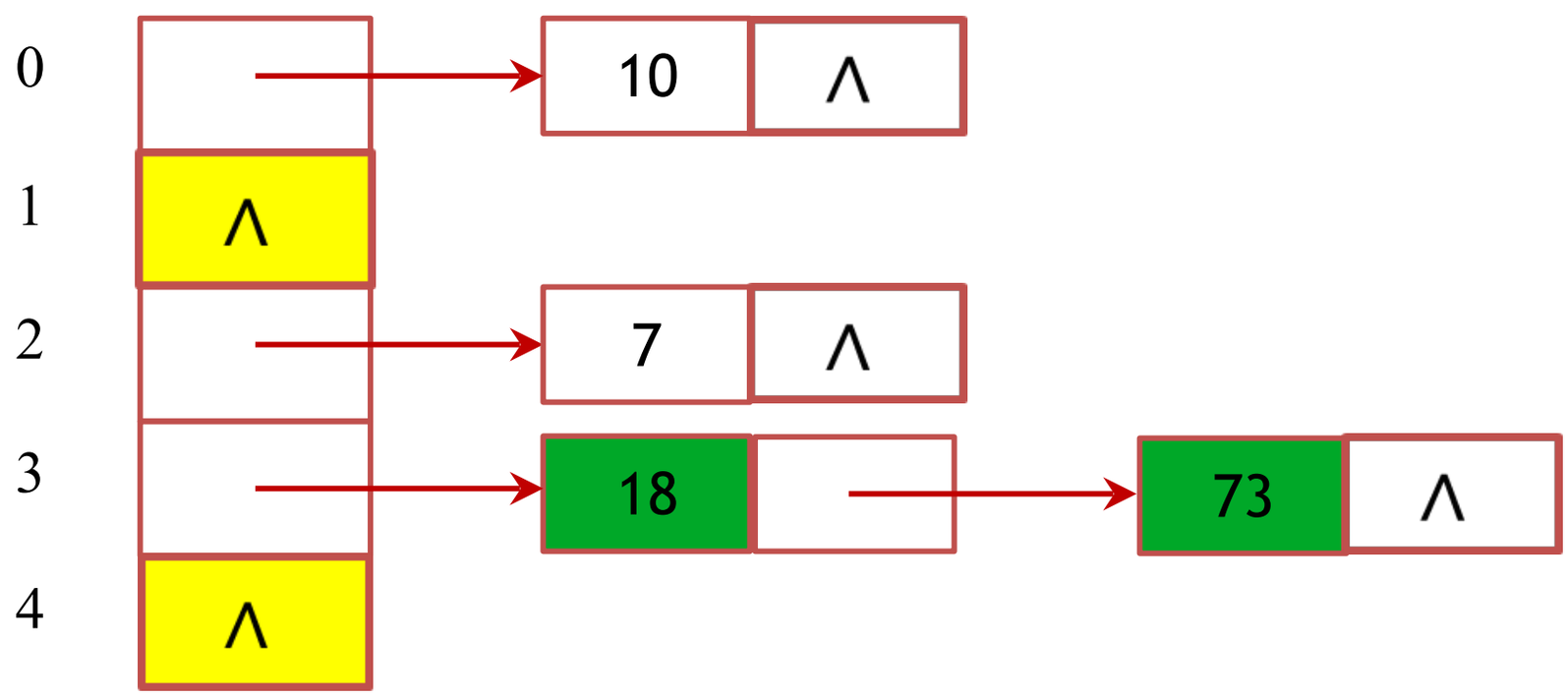
拉链法举例

对关键码集合K={18,10,73,7},

取h(key)=key%5,

则 h(18)=3，h(10)=0，h(73)=3，h(7)=2。

用拉链法解决碰撞：



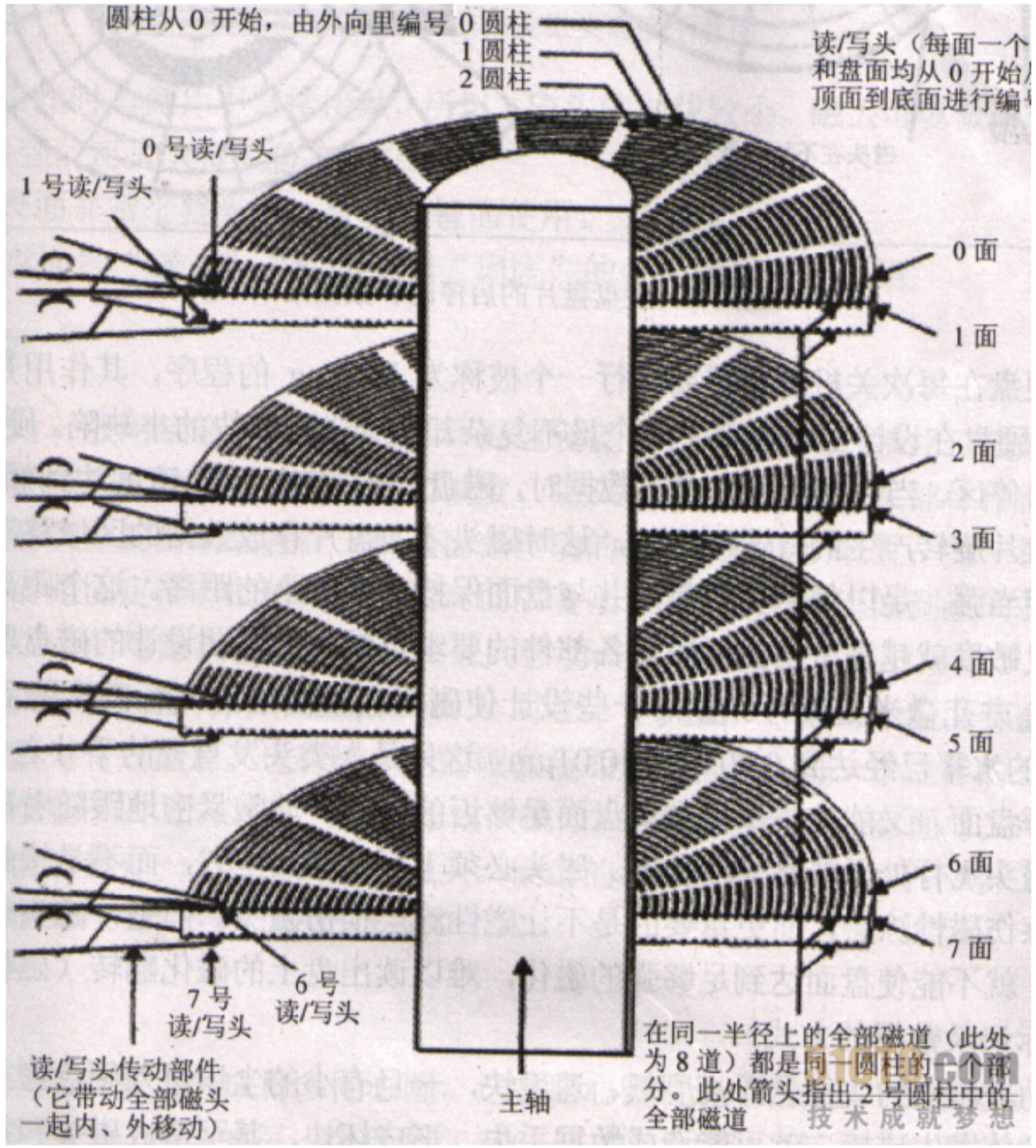
关于拉链法的讨论

- 拉链法的实现比较清楚：在单链表上检索、插入、删除。
- 每个同义词子表的平均长度为n/m，其中n为关键码个数，m为基本区域长度。
  - $n<m$，则一定存在若干空表；
  - 一般设计为 $1<n/m<10$。
- 各链接表上的结点动态申请，适合构造散列表前无法确定表长的情况；
- 不会造成堆积现象，结点的删除比较方便。

散列文件

- 散列文件是存在外存的大型字典结构，把散列的思想用于文件:
  - 将散列函数作用到记录的关键码，确定记录在外存的存储地址;
  - 处理冲突常用适合外存分块存取的拉链法。

外存与文件


圆柱从0开始，由外向里编号
0圆柱
1圆柱
2圆柱
读/写头（每面一个
和盘面均从0开始
顶面到底面进行编
0号读/写头
1号读/写头
0面
1面
2面
3面
4面
5面
6面
7面
7号
读/写头
6号
读/写头
读/写头传动部件
（它带动全部磁头
一起内、外移动）
主轴
在同一半径上的全部磁道（此处为8道）都是同一圆柱的一个部分。此处箭头指出2号圆柱中的全部磁道

外存储器——磁盘结构

- 磁盘类似于多碟的CD，若干盘片（比如4片）串在一根主轴上形成一个盘组，当主轴旋转时带动各个盘片旋转。每个盘面上包含大小不同的许多同心圆。
- 每个圆圈称为一个**磁道**，各个盘面的半径相同的磁道合在一起构成一个**柱面**。
- 一个磁道又可分为若干段，每段是一个**页块**（物理记录）。
- 一个盘组上从大到小的**存储地址为：柱面、磁道和页块**。

磁盘上读写信息

1．选定柱面，这是机械动作，所以比较慢。
2．选定磁道，由电子线路实现，所以比较快。
3．找物理记录，这是机械动作，较慢。

- 实际读写信息的时间比定位时间少得多，访问磁盘中的数据比访问内存慢5－6个数量级。
- 主机对外存储器上的数据必须按页块存取（不能直接按字或者字节存取）。
- 页块是内存与外存进行交换的物理单位。

缓冲区与访外

读外存上的数据:

1,把外存上一个或者多个物理记录读到内存的指定的区域（缓冲区）。

2,在缓冲区中找到需要的逻辑记录进行处理。

- 写的过程则相反，先写到缓冲区中，然后通过主机与外存的接口，把缓冲区中的数据写到外存储器的物理记录中。

一次读写过程，通常称为一次访外。

提高外存的存取效率的关键在于减少访外的次数。

文件与逻辑记录

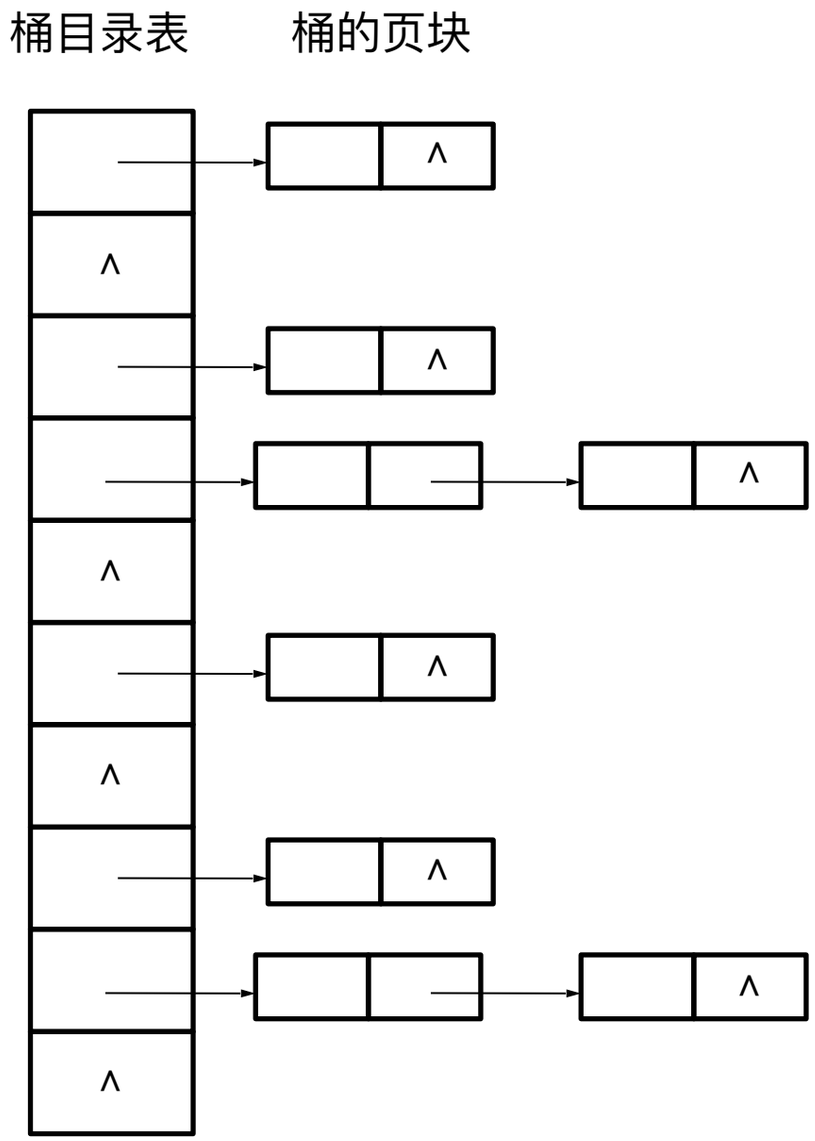
- 文件通常指存储在外存的数据，是逻辑记录的集合。
- 逻辑记录（简称记录）是应用程序需要进行内外存交换的逻辑单位。
- 每个记录可以包含若干数据项，
- 其中能够唯一标识该记录的数据项称为关键码。
- 逻辑记录在外存储器上的地址由两部分组成：
  - 逻辑记录所在物理记录的地址；
  - 逻辑记录在物理记录内的相对位置。
- 由于缓冲区的大小受到内存容量的限制，所以减少访外次数的有效方法是精心设计文件的结构，使外存中存放的记录，相互关联，以便于成批处理。

散列文件

- 散列文件是存放在外存的大型字典结构，把散列的思想用于文件：
  - 将散列函数作用到记录的关键码，确定记录在外存的存取地址
  - 处理冲突常用适合外存分块存取的拉链法

按桶散列的文件

- **桶: 拉链法中存放同义词的表.**
  - **这里桶是由0个或多个外存的页块通过链接而构成.**
- **实现散列文件的常见方法就是按桶散列:**
  - **只是这里存储桶里的一个链表结点对应于外存的一个页块,各种链接都直接采用外存的页块地址.**
  - **检索和插入、删除操作中涉及到访问外存.**
  - **桶目录表是一些链接的序列, 通常存储在内存中,每个链接指向一个桶的首页块.**



按桶散列的文件

检索：查找关键码为key的记录

首先计算 $h(key)$；

设 $h(key)=i$，查阅桶目录表的第 i 项，得到第 i 个桶的首页块地址，读入首页块，在块上进行顺序查找；

找到关键码为key的记录则结束，否则根据块之间的链接读入下一块，继续查找；

直到找到或者找遍全桶时结束。

插入：插入关键码为*key*的记录

先按上述方法进行查找；

找到时，说明本次插入是错误的或多余的；

若找不到(则此时读到内存中的页块恰好是新记录应插入的桶的最后一块)，在这个页块的空位置上填入新记录，并写回外存；

如果此页块已满，则申请一个新页块，其地址填入前一块的链接位置；

在新页块上填入新记录并把链接置空；最后把两个页块都写回外存。

桶数的选择

- 桶数的选择:
  - 桶数太少会使得桶中的页块较多, 增大访外次数。
  - 桶数太多,造成空间浪费。
  - 桶目录表本身的增大所带来的浪费不严重。
  - 主要的空间浪费: 不空但只存放很少几个记录的桶。
  - 确定桶的数目:

    经验规则:桶数与文件所能填满的页块数大致相等, 即

    $B \approx n/b$,

    其中B为桶数, n为文件记录数, b为每个页块能容纳的记录数。

按照上面取法,如果散列函数不太糟,

- 大多数桶中不超过 2 个页块,
- 检索时平均只需要 1 － 2 次访外,
- 每个页块中平均放入的记录数能使空间利用率达到半满的程度。