Hash index

overview

- Hash-based Indexing
- Static Hashing
- Dynamic Hashing
  - Query
  - Insertion
- Comparison of Ordered Indexing and Hashbased Indexing

Hashing

Static hashing

一个**bucket**包含了一系列records s (a bucket is typically a disk block).

key经过哈希函数映射成bucket地址

不同的record可能会有同一个hash值，在同一个桶里，因此需要线性寻找record

hash function

最差情况下，分布过于集中

理想状态：uniform and random

- 每个bucket中的key数量相近
- 随机分布，不依赖key实际分布

$h(value) = (a * value + b) mod N$

Hash File Organisation

- Hash file organisation of *instructor* file, using *dept_name* as key; assume there are 8 buckets
- The binary representation of the *i* th character is assumed to be the integer *I*

A	B	C	D	E	F	G	H	I	G	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	25	25	26

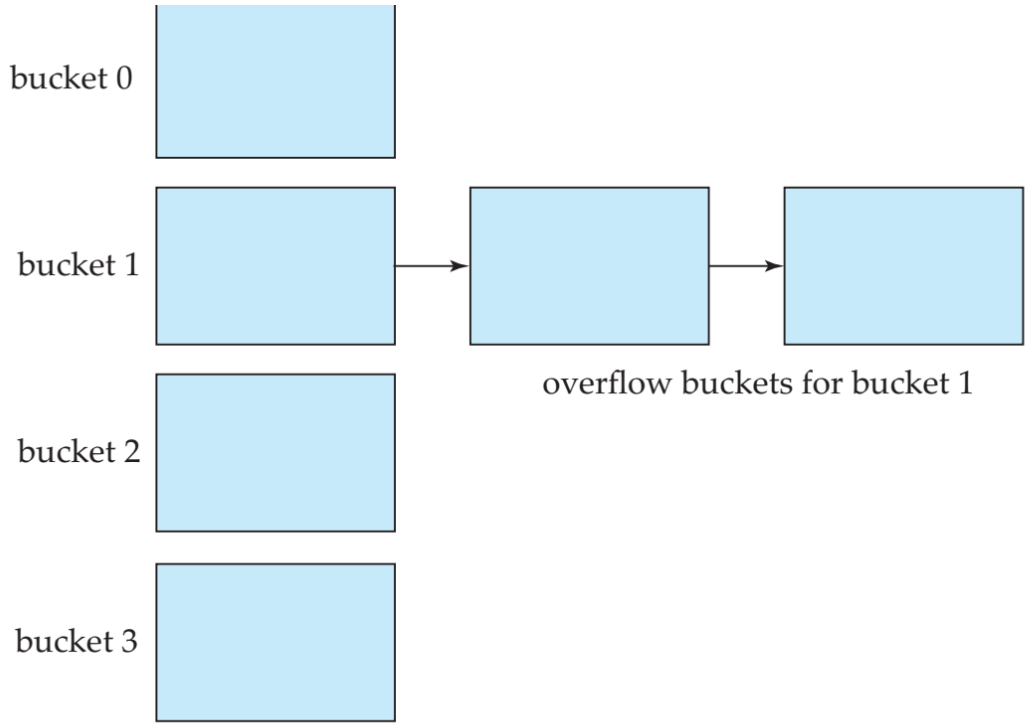
- The hash function returns the sum of the binary representations of the characters modulo 8
  - e.g. h(Music) = 1; h(History) = 2; h(Physics) = 3; h(Elec. Eng.) = 3
  - h(Music) = (13+21+19+9+3) mod 8 = 65 mod 8 = 1

字符串的二进制和进行取余

bucket overflow

- 桶不够
- record分布不均 (后面的例子)
  - 同一个key的record太多
  - hash function导致key值分布不均

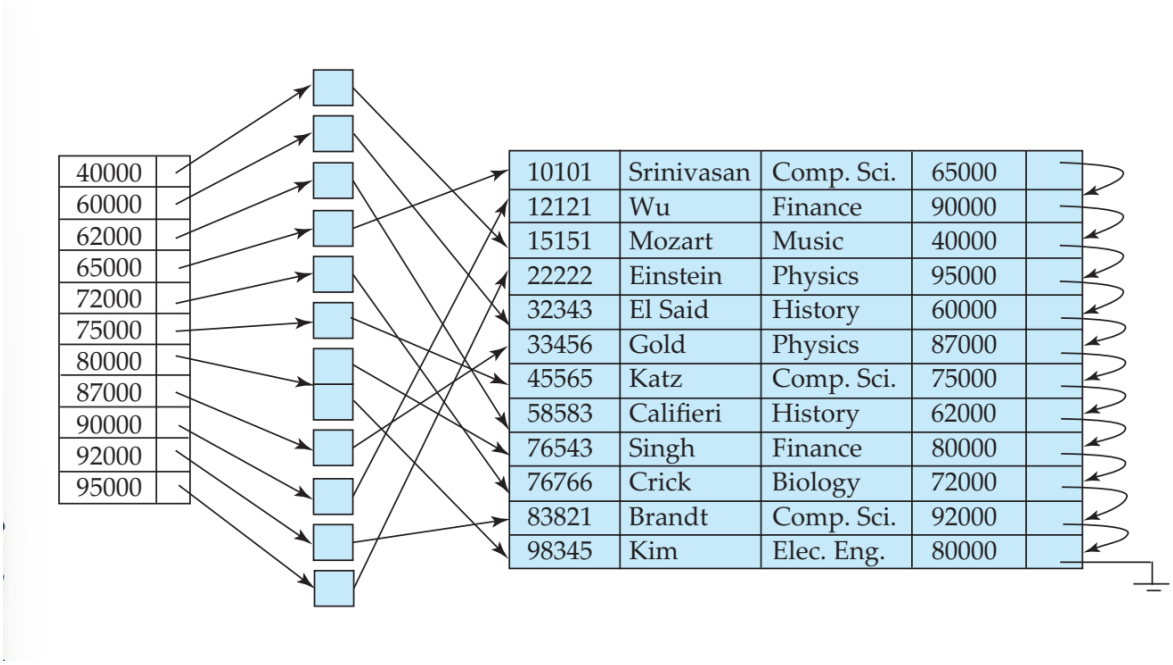
solution：使用**overflow bucket**， 用链表把溢出桶连起来



hash index

不仅可以用于文件组织，还可以用于index

hash index 总是 secondary index（与实际顺序无关



dynamic hashing

静态hash没办法调整bucket的数量，过少会导致的overflow降低性能，过多浪费空间

引入动态哈希，可以随着数据量变动而变动，hash function可以动态修改

Extendable hashing 是一种动态哈希，使用可变长度的哈希值前缀来对数据进行分桶

- 只用prefix来计算哈希值
- **Bucket address table** size = 2^i，innitially i = 0，i 会动态增减
- 可能指向同一个bucket
- 桶的实际数量小于 2^i，会随着桶的合并和分裂动态变化

只看二进制前 i 位

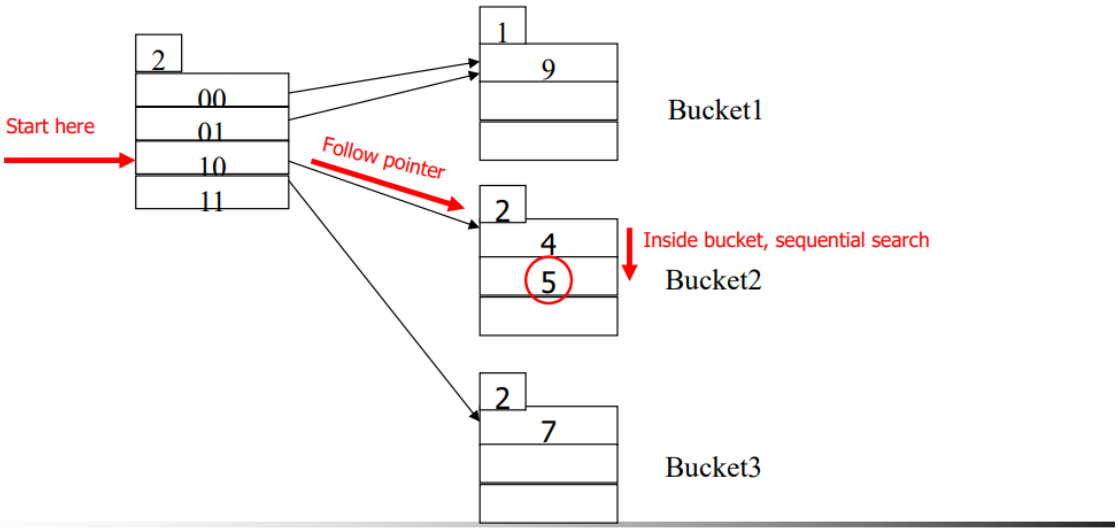
- i 写在表的左上角
- i_j 写在桶的左上角，
- 指向同一个桶的entry在前ij位相等
- 指向桶j的指针数 2^{i-ij}

Queries

1. 计算hash value
2. 取前i位查找，进入相应的桶

Query Example

- Assume $h(x) = x \bmod 8$; look up search key 5
- $h(5) = 5 \bmod 8 = 5 = 101$
- i=2; only look at first two bits (always from the left), 101



insertion

查找后如果桶里有空就插入，如果没空就分裂并重新尝试插入。

分裂：

if i > ij （同一个桶至少有两个指针）桶分裂

- 分配一个新bucket z，j++， z++
- 重新分配指针，一半指向新桶
- 移除原来桶的全部entry, reinsert
- 仍然桶满则再次分裂。

if i = ij （桶只有一个指针）表倍增

- if i reach limit，或者多次分裂依然无法插入
  - 恢复到原始索引，创建overflow bucket
- Else
  - i++，table倍增，指针数量倍增
  - back to first case

优缺点

pro：

- 性能不会随着文件增长而降低
- 空间开销小

con：

- 二级索引导致的额外查找

- 桶地址表过大（大于内存）
    - solution：在桶地址表中用b+树结构寻找
- 修改桶地址表的操作开销大

Odered and hashing index

取决于

- 定期充足的开销
- 插入和删除的频率
- 查找的类型（exact / range）

实际应用中hash在内存中更常用，在磁盘中不常用