CSE 373: Hash Tables
(open addressing and applications)

Chapter 5

Open Addressing

Goal: Use available space in table to store collisions rather than lists or resizing

- linear probing
- quadratic probing
- double hashing

Open Addressing Requirements

- The selection of alternate slots must be recomputable and deterministic
  - so that we can `Find()` data that we've inserted
- Deletion from the table must be "lazy"
  - similar to binary search trees
  - don't remove data, simply mark it as being deleted

Open Addressing: General Form

Open addressing is generally expressed as:

(hash(**key**) + $f(i)$) mod **tablesize**, for $i = 0, 1, 2, \ldots$

The hashing procedure is therefore:

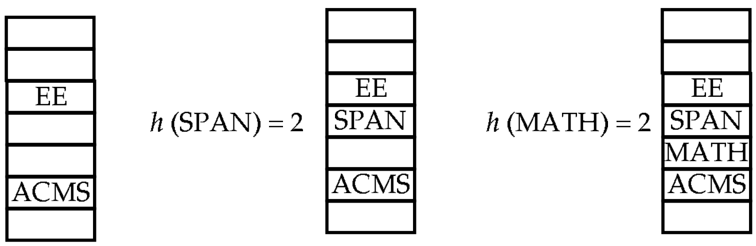
1) Try (hash(**key**) + $f(0)$) mod **tablesize**
2) If it's full, try (hash(**key**) + $f(1)$) mod **tablesize**
3) Continue until you find an empty slot

Design decision: what to use for $f()$?

Linear Probing

Uses $f(i) = i$

Insert(T,SPAN) **Insert(T,MATH)**

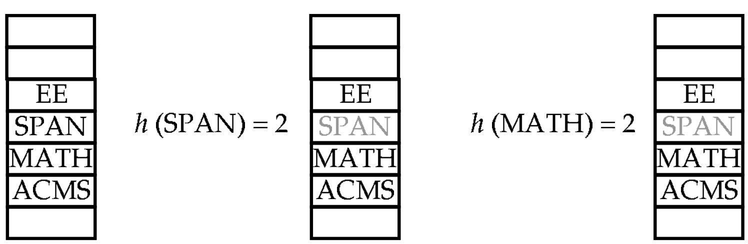


Note that if there is an open slot in the table, linear probing will always find it (eventually)

UW, Spring 1999 CSE 373 – Data Structures and Algorithms Brad Chamberlain

Finding, Deleting w/ Linear Probing

Find(T,SPAN) **Delete(T,SPAN)** **Find(T,MATH)**

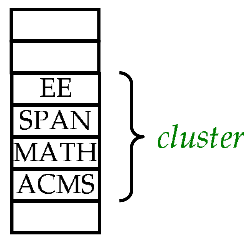


UW, Spring 1999 CSE 373 – Data Structures and Algorithms Brad Chamberlain

Primary Clustering

Linear probing has the tendency to result in clusters of data in the table

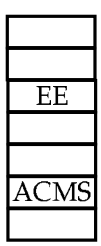
– increases search time for values hashing to that area



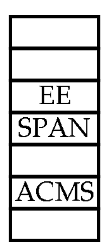
Quadratic Probing

Uses $f(i) = i^2$

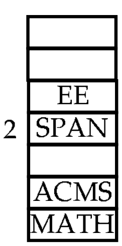
`Insert(T,SPAN)` `Insert(T,MATH)`



h (SPAN) = 2



h (MATH) = 2



Quadratic Probing: Evaluation

- *Intuition:* spreads things out more, so primary clustering should not be as much of a problem

- It can be proven that quadratic probing is guaranteed to find a free slot if…
  - number of slots is prime
  - table is less than half full
  - (therefore, resize when $\lambda = 0.5$)

Double Hashing

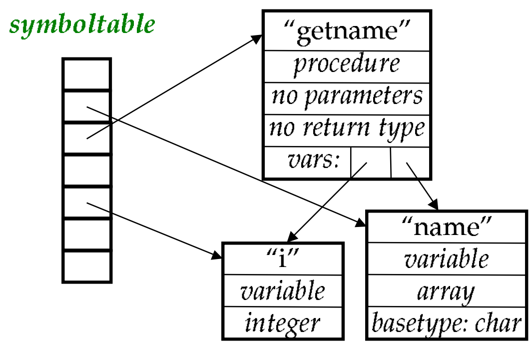
$f(i) = i \cdot \text{hash}_2(\mathbf{key})$

Intuition: since good hash functions result in fairly random distributions, this spreads values out in a less predictable pattern

Applications: Compilers

Compilers use hash tables to store information about all user-defined identifiers

```
void getname() {
  int i;
  char name[20];
  …
}
```



Applications: AI

- Create a hash function for a game's position
- Store "good moves" from each position as they are discovered
- While playing, can quickly check if there is a good move from the current position

UW, Spring 1999 CSE 373 – Data Structures and Algorithms Brad Chamberlain