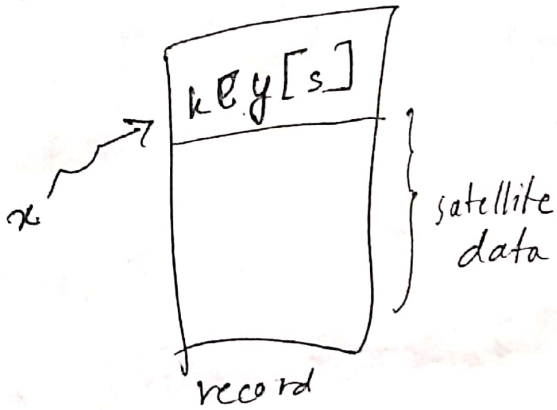
(22/11/2020)(0)

Lecture-7

Hashing I

Symbol table problem:

Table S holding n records



Operations:

dynamic

Insert(S, x) : $S \leftarrow S \cup \{x\}$

Delete(S, x) : $S \leftarrow S - \{x\}$

Search{S, k} : return x $ key[x] = k or nil, if no such x

direct access table

Suppose key are drawn from $U = \{0, 1, \ldots, m-1\}$

Assume keys are distinct

Set up array $T[0, \ldots, m-1]$ to represent dynamic set S

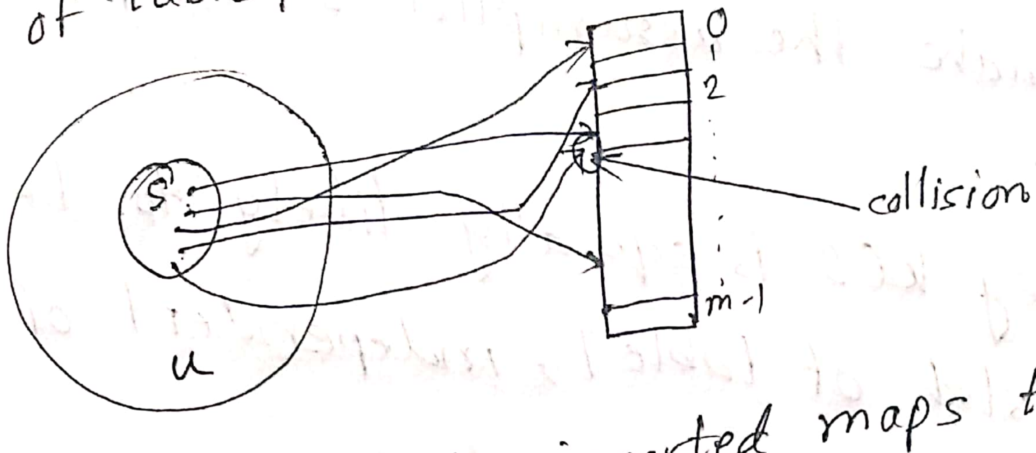
$$T[k] = \begin{cases} x & \text{if } x \in S \text{ and } key[x] = k \\ nil & \text{otherwise} \end{cases}$$

The operations take $\Theta(1)$ time

Problem: The range of keys can be large.

Hashing:

Hash function h maps keys "randomly" into slots of table T



- When a record to be inserted maps to an already occupied slot T, a collision occurs

When a record to be inserted maps to an already occupied slot a collision occurs

Resolving collisions by chaining:

Idea: link records in same slot into list

- **Worst case:** Every key hashes to the same slot.

Access time $\Theta(n)$ if $|s| = n$

Average case analysis of chaining

- We make the assumption of simple uniform hashing.

Each key $k \in S$ is equally likely to be hashed ata any slot of table T, independent of where other keys are hashed.

Let n be the number of key in the table and m be the number of slots

Define the load number of T to be

$\alpha = n/m$ = average number of key per slot

- The expected time for an unsuccessful search for a record with a given key is

$\Theta(1 + \alpha)$

search the lis

apply hash function and access the slot

Expected search time $= \Theta(1)$ if $\alpha = \Theta(1)$ or equivalently $n = \Theta(m)$

A successful search has same asymptotic bound, but a rigorous argument is a little more complicated.

Choosing hash function:

-> should distributed keys uniformly into slots

-> Regularity in key distribution should not affect uniformity

Division method:

$h(k) = k \bmod m$

Definiciency

Don't pick m with small divisor d

Ex: $d = 2$ and all keys are even

=> odd slots are never use

Extreme deficiency

Ex: $m = 2^r$ => hash doesn't depend on all bits of k

Good heuristic:

Pick m = prime not too close to pow of 2 or 10

Annoyance:

Sometimes making the table size a prime is convenient.

Multiplication Method:

$m = 2^r$ computer has w bit words

$h(k) = (A \cdot k \bmod 2^w) \underbrace{\text{rsh}}_{\text{bit-wise right shift}} (w - r)$

A is an odd integer ~~be~~ in range $2^{w-1} < A < 2^w$

Don't pick A to close to 2^{w-1} or 2^w

Fast method: module 2^w is ~~fa~~ fast
'rsh' ~~is~~ operator is fast

$h(k) = (A \cdot k \bmod 2^w) \text{ rsh}(w - r)$

suppose that $m = 8 = 2^3$ and that our comp has $w = 7$-bit words

$$\begin{array}{r} 1011001 \quad = A \\ 1101011 \quad = k \\ \hline 1001010\underbrace{011}_{h(k)}0011 \end{array}$$

ignored

Resolving collision by open addressing

→ No storage ~~for links~~ is used outside of the hash table itself.

- Insertion systematically probes the table until an empty slot is found.
- The hash function depends on both the key and probe number

$$h: U \times \{0, 1, \ldots, m-1\} \to \{0, 1, \ldots, m-1\}$$

- The probe sequence $\langle h(k,0), h(k,1), \ldots, h(k,m-1) \rangle$ should be a permutation of $\{0, 1, \ldots, m-1\}$.
- The table may fill up, and deletion is difficult (but not impossible)

Search: same probe sequence

- successful - find record
- unsuccessful - find nil

Probing strategies:

Linear probing: ~~$h(k,i) = (h(k,0)+i) \bmod m$~~

$$h(k,i) = (h'(k)+i) \bmod m$$

This method though simple suffers from primary clustering, where long runs of occupied slots build up, increasing the average search time. Moreover, the long run of occupied slots tend to get longer.

Double hashing:

Given two ordinary hash function $h_1(k)$ and $h_2(k)$ double hashing uses the hash function

$$h(k,i) = (h_1(k) + i \cdot h_2(k)) \bmod m$$

This method generally produces excellent results but $h_2(k)$ must be relatively prime to m. One way is to make m a power of 2 and design $h_2(k)$ to produce only odd numbers.

Average case |

Analysis of open addressing:

We make assumptions of uniform hashing

→ Each key is equal to have any one of the $m!$ permutations as its probe sequence.

Theorem: Given an open addressed hash table with load factor $\alpha = n/m < 1$, the expected number of probes in an unsuccessful search is at most $1/(1-\alpha)$ $E[\#\text{probes}] = \frac{1}{1-\alpha}$

Proof of Theorem:

- At least one probe is always necessary
- With probability n/m, the first probe hits an occupied slot, and a second probe is necessary
- With probability $(n-1)/(m-1)$ the second probe hits an occupied slot and third probe is necessary

Observe $\frac{n-i}{m-i} < \frac{n}{m} = \alpha$ for $1, 2, \ldots, n$

Expected number of probes

$$1 + \frac{n}{m}\left(1 + \frac{n-1}{m-1}\left(1 + \frac{n-2}{m-2}\left(\cdots\left(1 + \frac{1}{m-n+1}\right)\cdots\right)\right)\right)$$

$$\leq 1 + \alpha\left(1 + \alpha\left(1 + \alpha\left(\cdots(1+\alpha)\cdots\right)\right)\right)$$

$$\leq 1 + \alpha + \alpha^2 + \alpha^2$$

$$= \sum_{i=0}^{\infty} \alpha^i$$

$$= \frac{1}{1-\alpha} \quad \blacksquare$$