Hashing - III

Instructor: Ashok Singh Sairam

Lecture Plan

- Double Hashing (continued from previous lecture)
- #Probes
  - Unsuccessful Search
  - Insertion
  - Successful retrieval

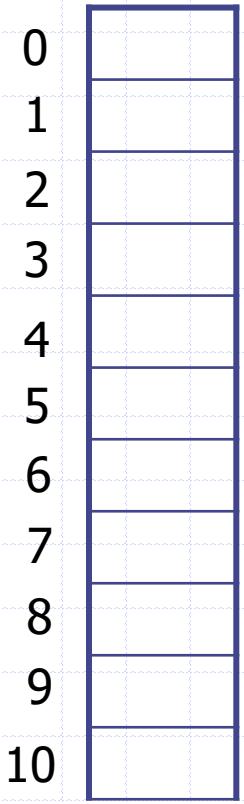
Review

11.4-1

Consider inserting the keys $10, 22, 31, 4, 15, 28, 17, 88, 59$ into a hash table of length $m = 11$ using open addressing with the auxiliary hash function $h'(k) = k$. Illustrate the result of inserting these keys using linear probing, using quadratic probing with $c_1 = 1$ and $c_2 = 3$, and using double hashing with $h_1(k) = k$ and $h_2(k) = 1 + (k \bmod (m - 1))$.

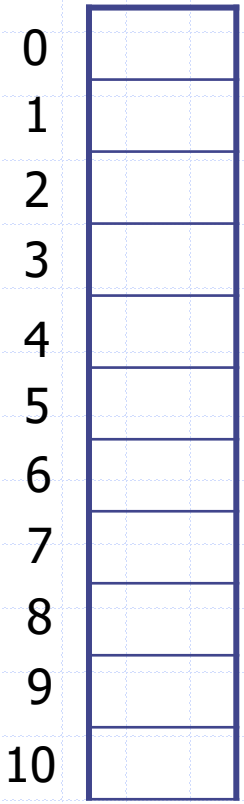
Ex: Linear Probing

- Keys = (10, 22, 31, 4, 15, 28, 17, 88, 59)
- $h(k,i)=(h_1(k)+i) \bmod m$, given $h_1(k)=k$; $m = 11$



Ex: Quadratic Probing

- Keys = (10, 22, 31, 4, 15, 28, 17, 88, 59)
- $h(k,i)=(h_1(k)+c_1 i+c_2 i^2) \bmod m$,
- $h_1(k)=k$; $c_1=1$; $c_2=3$; $m = 11$



Open addressing methods – contd.

- We will examine three common techniques used to compute the probe sequences required for open addressing

1. Linear probing
2. Quadratic probing
3. Double hashing

- Note: None of these methods can generate more than m^2 different probing sequences!

Double Hashing

- One of the best methods available for open addressing

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

  where both h_1 and h_2 are auxiliary hash functions
- Initial probe: $h_1(k)$
- Second probe is offset by $h_2(k)$ **mod** m, so on ...
- Advantage: avoids clustering
- Disadvantage: harder to delete an element
- Can generate m^2 probe sequences maximum

Double Hashing: Example

$h_1(k) = k \bmod 13$

$h_2(k) = 1 + (k \bmod 11)$

$h(k,i) = (h_1(k) + i\, h_2(k)) \bmod 13$

- Insert key 14:

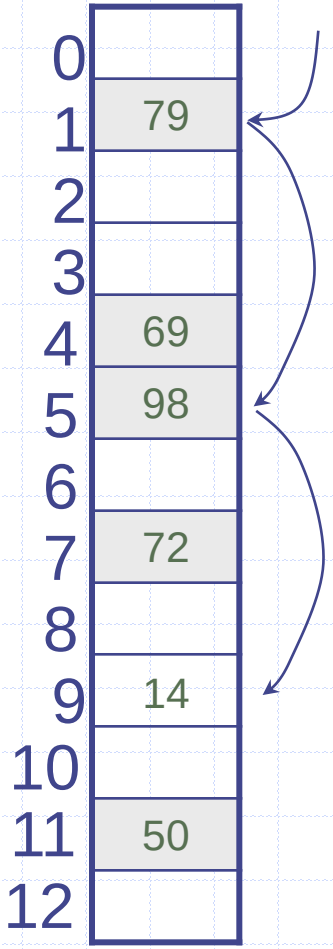
$h(14,0) = 14 \bmod 13 = 1$

$h(14,1) = (h_1(14) + h_2(14)) \bmod 13$

$= (1 + 4) \bmod 13 = 5$

$h(14,2) = (h_1(14) + 2\, h_2(14)) \bmod 13$

$= (1 + 8) \bmod 13 = 9$



#Probes for unsuccessful search

- In open addressing m $\geq$ n
- Load factor $\alpha = {}^{n}/_{m} \leq 1$
- Thm: Expected #probes for unsuccessful retrieval is $\leq \frac{1}{1-\alpha}$
- Proof:

#Probes for insertion

- **Corollary**: Inserting an element into an open-addressing hash table with load factor α requires at most $1/(1-\alpha)$ probes on average, assuming uniform hashing
- Proof: Inserting a key when load factor is α
  1. Is exactly like unsuccessful search
  2. Upper bound on #probes is $1/(1-\alpha)$

#Probes for successful retrieval

- Expected #probes for successful retrieval when n keys are inserted into an empty table is $\leq \frac{1}{\alpha} ln \frac{1}{1-\alpha}$

Proof:

Successful search

- Searching for a key takes as many probes as inserting *that particular key*
- Each inserted key increases the load factor, so the inserted key number *i* + 1 is expected to take no more than

$$\frac{1}{1-\frac{i}{m}} = \frac{m}{m-i} \text{ probes}$$

- Averaging over all n keys, $\frac{1}{n}\sum_{i=0}^{n-1}\frac{m}{m-i}$

$$= \frac{m}{n}\sum_{i=0}^{n-1}\frac{1}{m-i}$$

[Sum is $\frac{1}{m}+\frac{1}{m-1}+\frac{1}{m-2}+\cdots+\frac{1}{m-n+1}$, a decreasing fn.]

$$= \frac{m}{n}\sum_{i=m-n+1}^{m}\frac{1}{i}$$

$$\leq \frac{m}{n}\int_{m-n+}^{m}\frac{1}{i}\,dx$$ [Upper bound on decreasing fn]

$$= \frac{m}{n}\left(\ln m - \ln(m-n)\right)$$

$$= \frac{1}{\alpha}\ln\left(\frac{m}{m-n}\right) = \frac{1}{\alpha}\ln\left(\frac{1}{1-\alpha}\right)$$

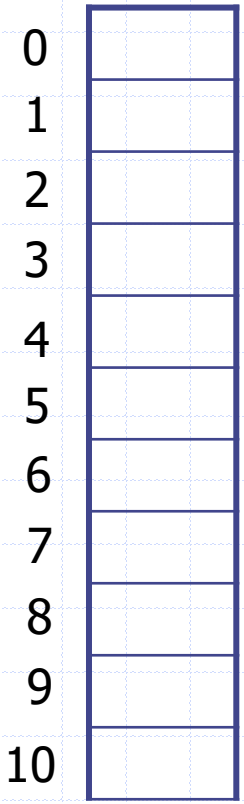
Review

11.4-1

Consider inserting the keys $10, 22, 31, 4, 15, 28, 17, 88, 59$ into a hash table of length $m = 11$ using open addressing with the auxiliary hash function $h'(k) = k$. Illustrate the result of inserting these keys using linear probing, using quadratic probing with $c_1 = 1$ and $c_2 = 3$, and using double hashing with $h_1(k) = k$ and $h_2(k) = 1 + (k \bmod (m - 1))$.

Ex: Double Hashing

- Keys = (10, 22, 31, 4, 15, 28, 17, 88, 59)
- $h(k,i)=(h_1(k)+i\ h_2(k)) \bmod m,\ m = 11$
- $h_1(k)=k$; $h_2(k)=(1 + k) \bmod 10$

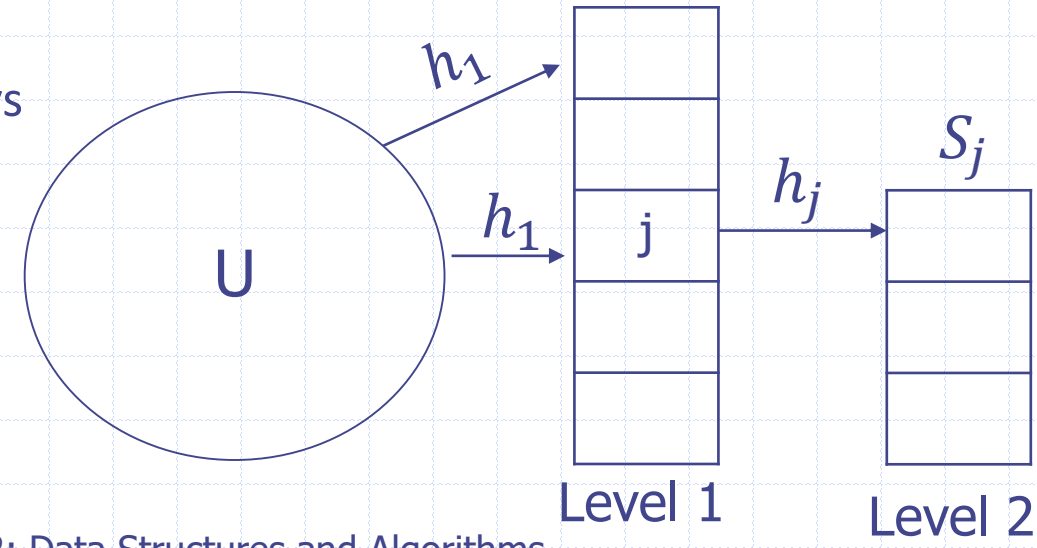


Perfect Hashing

- Used to solve the static hashing problem
  - The set of keys is static (never changes) such as set of reserved words in a programming language
- Given a set of n keys, construct a static hash table of size m=O(n), such that search takes $\Theta(1)$ in worst case
  - Space complexity: $\Theta(n)$
  - Build time: Polynomial or exponential with high probability

Perfect Hashing: The Idea

- Two-level scheme with universal hashing at both levels.
  - Level 1: Hash using h_1 (universal hashing). Same as hashing with collision
  - Level 2: Two keys hash to same slot (j). Create a secondary table S_j with associated hash function h_j
- No collisions at level 2
  - $m_j = n_j^2$, where n_j #keys hashing into slot j



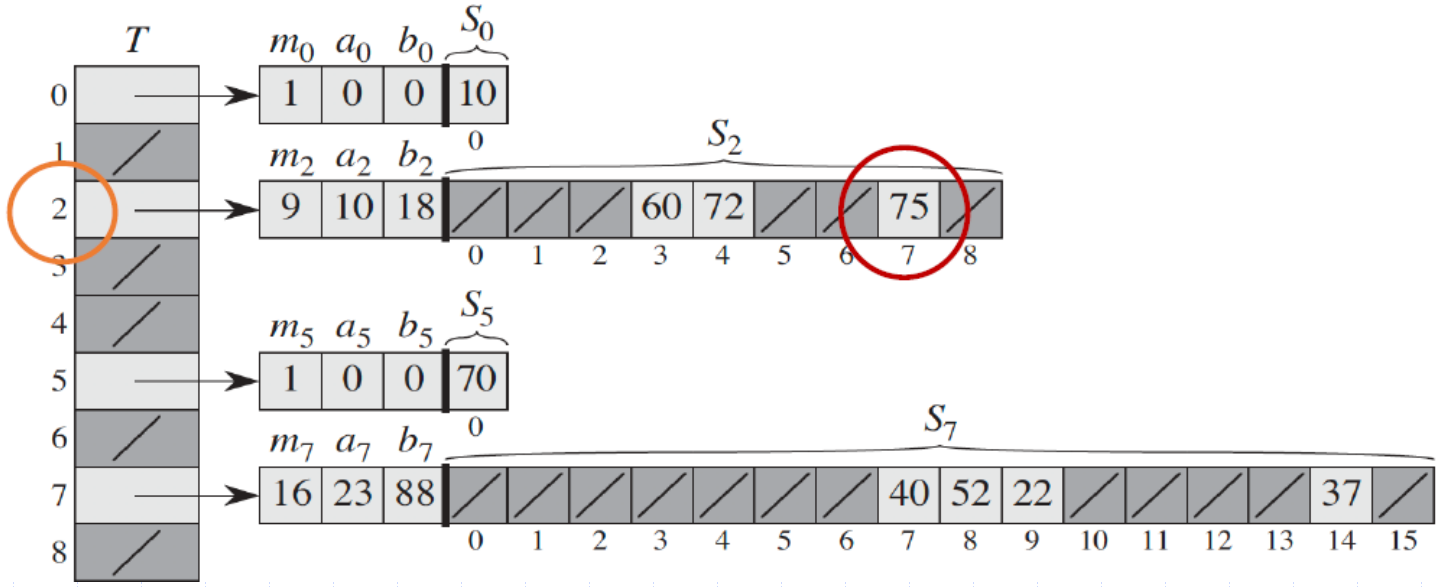
Perfect Hashing: Example

- $h_1(k) = \big((3k + 42)\ mod\ 101\big)\ mod\ 9$
- $h_2(k) = \Big(\big(a_j k + b_j\big)\ mod\ p\Big)\ mod\ m,$

K = {10, 22, 37, 40, 52, 60, 70, 72, 75}

$h(75) = 2$

$h_2(75) = 7$



Probability of collision

- If h ∈ H is universal and $m=n^2$, then Pr[no collision] < ½
- Proof: #pairs that may collide = $\binom{n}{2}$ [#distinct pairs]

$$\Pr[h(k_1) = h(k_2)] = \frac{1}{m} \text{ [By universality]}$$

$$\Pr[\text{Exists a collision}] = \binom{n}{2} \times \frac{1}{m} = \frac{n^2-n}{2}.\frac{1}{n^2}$$

$$< ½$$

- Try a random $h_j \in H$, if we get a collision pick a new h. On average, we will need to do this only twice.

Exercise

11-3 Quadratic probing

Suppose that we are given a key k to search for in a hash table with positions $0, 1, \ldots, m-1$, and suppose that we have a hash function h mapping the key space into the set $\{0, 1, \ldots, m-1\}$. The search scheme is as follows:

1. Compute the value $j = h(k)$, and set $i = 0$.
2. Probe in position j for the desired key k. If you find it, or if this position is empty, terminate the search.
3. Set $i = i + 1$. If i now equals m, the table is full, so terminate the search. Otherwise, set $j = (i + j) \bmod m$, and return to step 2.

Assume that m is a power of 2.

a. Show that this scheme is an instance of the general "quadratic probing" scheme by exhibiting the appropriate constants c_1 and c_2 for equation (11.5).

b. Prove that this algorithm examines every table position in the worst case.