Hashing (part 2)

CSE 2011

Winter 2011

Collision Handling

- Separate chaining
- Probing (open addressing)
  - Linear probing
  - Quadratic probing
  - Double hashing

Quadratic Probing

- Linear probing:
  Insert item (k, e)
  i = h(k)
  A[i] is occupied
  Try A[(i+1) mod N]: used
  Try A[(i+2) mod N]
  and so on until
  an empty cell is found

- Quadratic probing
  A[i] is occupied
  Try A[(i+1) mod N]: used
  Try A[(i+2^2) mod N]: used
  Try A[(i+3^2) mod N]
  and so on

- May not be able to find an empty cell if *N* is not prime, or the hash table is at least half full

3

Double Hashing

- Double hashing uses a secondary hash function $d(k)$ and handles collisions by placing an item in the first available cell of the series
  $$(i + j \times d(k)) \bmod N$$
  for $j = 0, 1, \ldots, N-1$
- The secondary hash function $d(k)$ cannot have zero values
- The table size N must be a prime to allow probing of all the cells

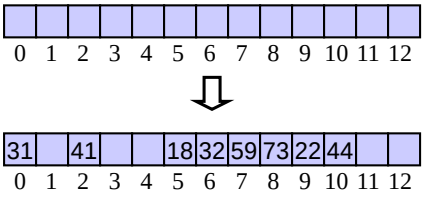
Insert item (k, e)
i = h(k)
A[i] is occupied
Try A[(i+d(k))mod N]: used
Try A[(i+2d(k))mod N]: used
Try A[(i+3d(k))mod N]
and so on until
an empty cell is found

4

Example of Double Hashing

- Consider a hash table storing integer keys that handles collision with double hashing
  - $N = 13$
  - $h(k) = k \bmod 13$
  - $d(k) = 7 - k \bmod 7$
- Insert keys 18, 41, 22, 44, 59, 32, 31, 73, in this order

k	$h(k)$	$d(k)$	Probes		
18	5	3	5		
41	2	1	2		
22	9	6	9		
44	5	5	5	10	
59	7	4	7		
32	6	3	6		
31	5	4	5	9	0
73	8	4	8		



5

Double Hashing (2)

- $d(k)$ should be chosen to minimize clustering
- Common choice of compression map for the secondary hash function:

  $d(k) = q - k \bmod q$

  where
  - $q < N$
  - q is a prime
- The possible values for $d(k)$ are

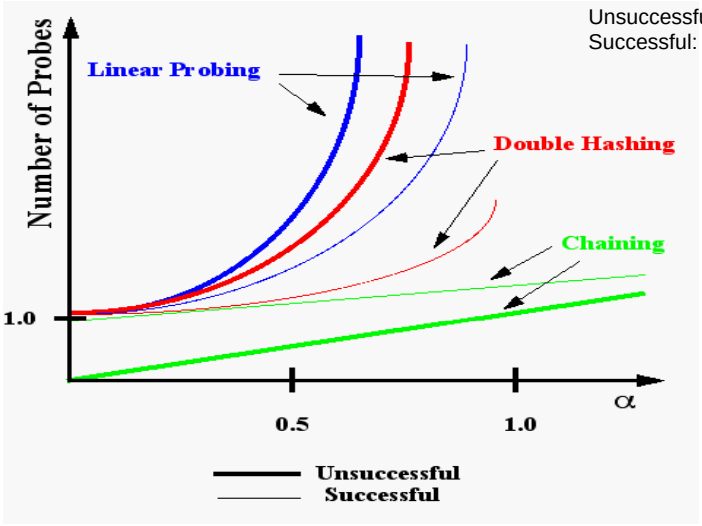
  $1, 2, \ldots, q$

- Note: linear probing has $d(k) = 1$.

6

Comparing Collision Handling Schemes

$\alpha = n/N$
Unsuccessful: key not found
Successful: key found



7

Comparing Collision Handling Schemes (2)

- Separate chaining:
  - simple implementation
  - faster than open addressing in general
  - using more memory
- Open addressing:
  - using less memory
  - slower than chaining in general
  - more complex removals
- Linear probing: items are clustered into contiguous runs (primary clustering).
- Quadratic probing: secondary clustering.
- Double hashing: distributes keys more uniformly than linear probing does.

8

Performance of Hashing

- In the worst case, searches, insertions and removals on a hash table take $O(n)$ time.
- The worst case occurs when all the keys inserted into the map collide.
- The load factor $\alpha = n/N$ affects the performance of a hash table.
- Assuming that the hash values are like random numbers, it can be shown that the expected number of probes for an insertion with open addressing is
  $1 / (1 - \alpha)$
- The expected running time of all the dictionary ADT operations in a hash table is $O(1)$.
- In practice, hashing is very fast provided the load factor is not close to 100%.
  $\Rightarrow$ rehashing.
- Applications of hash tables:
  - small databases
  - compilers
  - browser caches

  $\Rightarrow$ converting non-integer keys to integers.

9

Keys That Are Strings

- We need to convert a string to an integer before hashing.
- One option is to add up the ASCII values of the characters in the string.
  - Is this a good strategy?
- Polynomial accumulation:
  - We partition the bits of the key into a sequence of components of fixed length (e.g., 8, 16 or 32 bits).
    $x_0 x_1 \ldots x_{n-1}$
  - We evaluate the polynomial
    $p(z) = x_0 + x_1 z + x_2 z^2 + \ldots + x_{n-1} z^{n-1}$
    at a fixed value z, *ignoring overflows*.

10

Polynomial Accumulation

- Polynomial $p(z)$ can be evaluated in $O(n)$ time using Horner's rule:
  - The following polynomials are successively computed, each from the previous one in $O(1)$ time

$$p_0(z) = x_{n-1}$$
$$p_i(z) = x_{n-i-1} + zp_{i-1}(z)$$
$$(i = 1, 2, \ldots, n-1)$$

- We have $p(z) = p_{n-1}(z)$
- Good z values: 33, 37, 39, 41.
- Especially suitable for strings
  - $z = 33$ gives at most 6 collisions on a set of 50,000 English words.

11

Summary

- Purpose of hash tables: to obtain O(1) ***expected*** query time using O(n+N) space.
- If the keys are not integers, convert them to integer keys.
- Map integer keys to the hash table entries using a *compression map* function.
- If collision occurs, use one of the collision handling schemes, taking into account available memory space.
- If the load factor $\lambda = n/N$ approaches the specified threshold, *rehash*.

12

Next time…

- Graphs (chapter 13)

13