Hashing

Tyler Moore

CS 2123, The University of Tulsa

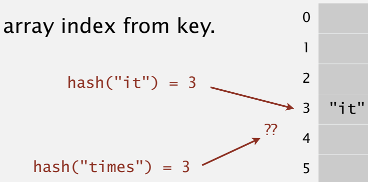
Some slides created by or adapted from Dr. Kevin Wayne. For more information see

http://www.cs.princeton.edu/courses/archive/fall12/cos226/lectures.php.

Hashing: basic plan

Save items in a key-indexed table (index is a function of the key).

Hash function. Method for computing array index from key.



Issues.

- Computing the hash function.
- Equality test: Method for checking whether two keys are equal.
- Collision resolution: Algorithm and data structure to handle two keys that hash to the same array index.

Classic space-time tradeoff.

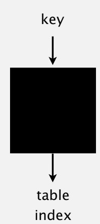
- No space limitation: trivial hash function with key as index.
- No time limitation: trivial collision resolution with sequential search.
- Space and time limitations: hashing (the real world).

Computing the hash function

Idealistic goal. Scramble the keys uniformly to produce a table index.

- Efficiently computable.
- Each table index equally likely for each key.

thoroughly researched problem, still problematic in practical applications



Ex 1. Phone numbers.

- Bad: first three digits.
- Better: last three digits.

Ex 2. Social Security numbers.

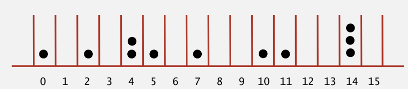
- Bad: first three digits. ← 573 = California, 574 = Alaska (assigned in chronological order within geographic region)
- Better: last three digits.

Practical challenge. Need different approach for each key type.

Uniform hashing assumption

Uniform hashing assumption. Each key is equally likely to hash to an integer between 0 and $M - 1$.

Bins and balls. Throw balls uniformly at random into M bins.



Birthday problem. Expect two balls in the same bin after $\sim\sqrt{\pi M / 2}$ tosses.

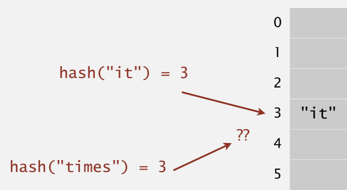
Coupon collector. Expect every bin has ≥ 1 ball after $\sim M \ln M$ tosses.

Load balancing. After M tosses, expect most loaded bin has $\Theta(\log M / \log \log M)$ balls.

Collisions

Collision. Two distinct keys hashing to same index.

- Birthday problem $\Rightarrow$ can't avoid collisions unless you have a ridiculous (quadratic) amount of memory.
- Coupon collector + load balancing $\Rightarrow$ collisions are evenly distributed.



Challenge. Deal with collisions efficiently.

Options for dealing with collisions

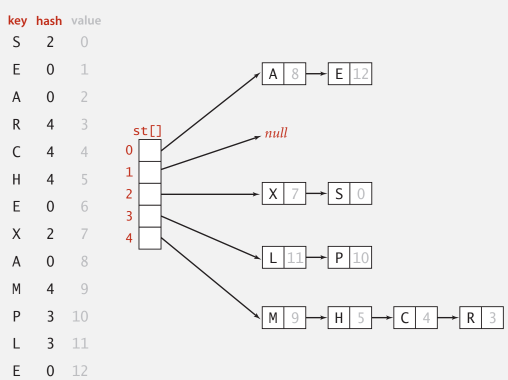
1. **Open hashing** aka separate chaining: store collisions in a linked list
2. **Closed hashing** aka open addressing: keep keys in the table, shift to unused space
   - Collision resolution policies

   1. Linear probing
   2. Quadratic probing aka quadratic residue search
   3. Double hashing

Separate chaining symbol table

Use an array of $M < N$ linked lists. [H. P. Luhn, IBM 1953]

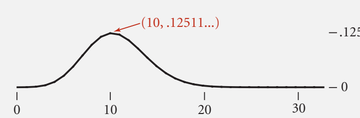
- Hash: map key to integer i between 0 and $M - 1$.
- Insert: put at front of i^{th} chain (if not already there).
- Search: need to search only i^{th} chain.



Analysis of separate chaining

Proposition. Under uniform hashing assumption, prob. that the number of keys in a list is within a constant factor of N / M is extremely close to 1.

Pf sketch. Distribution of list size obeys a binomial distribution.



Binomial distribution ($N = 10^4$, $M = 10^3$, $\alpha = 10$)

equals() and hashCode()

Consequence. Number of probes for search/insert is proportional to N / M.

- M too large $\Rightarrow$ too many empty chains.
- M too small $\Rightarrow$ chains too long.
- Typical choice: $M \sim N / 5$ $\Rightarrow$ constant-time ops.

M times faster than sequential search

Closed hashing

- Records stored directly in table of size M at hash index $h(x)$ for key x
- When a collision occurs:
  - Hashes to occupied home position
  - Record stored in first available slot based on repeatable collision resolution policy
  - Formally, for each i collisions $h_0(x), h_1(x), \ldots h_i(x)$ tried in succession where $h_i(x) = (h(x) + f(i)) \mod M$

Closed hashing: insert

```
Hash(key) into table at position i
Repeat up to the size of the table {
   If entry at position i in table is blank or marked as deleted
     then insert and exit
   Let i be the next position using the collision resolution function
}
```


Closed hashing: search

```
Hash(key) into table at position i
Repeat up to the size of the table {
   If entry at position i in table matches key and not marked as deleted
     then found and exit
   If entry at position i in table is blank
     then not found and exit
   Let i be the next position using the collision resolution function
} Not found and exit
```


Closed hashing: delete

```
Hash(key) into table at position i
Repeat up to the size of the table {
   If entry at position i in table matches key
     then mark as deleted and exit
   If entry at position i in table is blank
     then not found and exit
   Let i be the next position using the collision resolution function
} Not found and exit
```

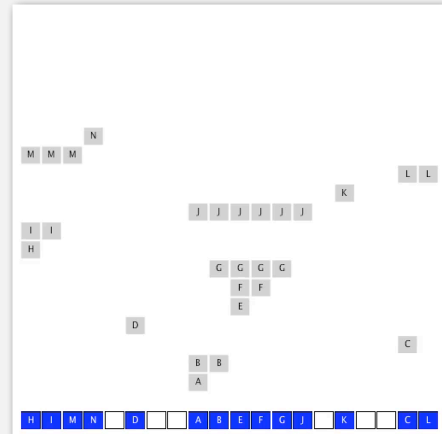
Linear probing

- Collision resolution function $f(i) = i$: $h_i(x) = (h(x) + i) \mod M$
- Work example

Clustering

Cluster. A contiguous block of items.

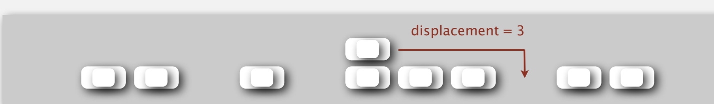
Observation. New keys likely to hash into middle of big clusters.



Knuth's parking problem

Model. Cars arrive at one-way street with M parking spaces.
Each desires a random space i: if space i is taken, try $i+1$, $i+2$, etc.

Q. What is mean displacement of a car?



Half-full. With $M/2$ cars, mean displacement is ~ $3/2$.

Full. With M cars, mean displacement is ~ $\sqrt{\pi M / 8}$.

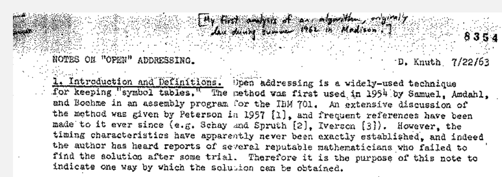
Analysis of linear probing

Proposition. Under uniform hashing assumption, the average # of probes in a linear probing hash table of size M that contains $N = \alpha M$ keys is:

$$\sim \frac{1}{2}\left(1 + \frac{1}{1-\alpha}\right) \qquad \sim \frac{1}{2}\left(1 + \frac{1}{(1-\alpha)^2}\right)$$

search hit — search miss / insert

Pf.

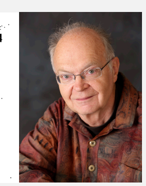


[My first analysis of an algorithm, originally done during Summer 1962 in Madison.]

8354

NOTES ON "OPEN" ADDRESSING. D. Knuth 7/22/63

1. Introduction and Definitions. Open addressing is a widely-used technique for keeping "symbol tables." The method was first used in 1954 by Samuel, Amdahl, and Boehme in an assembly program for the IBM 701. An extensive discussion of the method was given by Peterson in 1957 [1], and frequent references have been made to it ever since (e.g. Schay and Spruth [2], Iverson [3]). However, the timing characteristics have apparently never been exactly established, and indeed the author has heard reports of several reputable mathematicians who failed to find the solution after some trial. Therefore it is the purpose of this note to indicate one way by which the solution can be obtained.



Parameters.

- M too large ⇒ too many empty array entries.
- M too small ⇒ search time blows up.
- Typical choice: $\alpha = N / M \sim ½$. ⟵ # probes for search hit is about 3/2; # probes for search miss is about 5/2

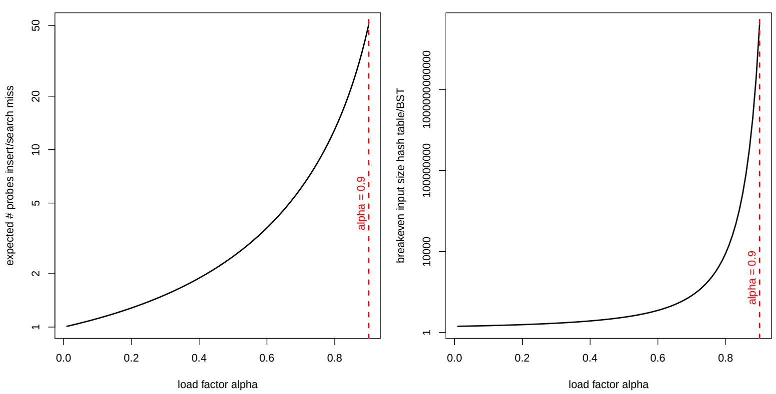
Performance comparison of search

Tree	Worst-case cost (after n inserts)			Avg.-case cost (after n inserts)			Ordered
	search	insert	delete	search	insert	delete	iteration?
Sequential search (unordered list)	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$	no
Binary search (ordered array)	$\Theta(\log(n))$	$\Theta(n)$	$\Theta(n)$	$\Theta(\log(n))$	$\Theta(n)$	$\Theta(n)$	yes
BST	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$	$\Theta(\log(n))$	$\Theta(\log(n))$	$\Theta(\log(n))$	yes
AVL	$\Theta(\log(n))$	$\Theta(\log(n))$	$\Theta(\log(n))$	$\Theta(\log(n))$	$\Theta(\log(n))$	$\Theta(\log(n))$	yes
B-tree	$\Theta(\log(n))$	$\Theta(\log(n))$	$\Theta(\log(n))$	$\Theta(\log(n))$	$\Theta(\log(n))$	$\Theta(\log(n))$	yes
Hash table	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$	no

Load factors and cost of probing

- What size hash table do we need when using linear probing and a load factor of $\alpha = 0.75$ for closed hashing to achieve a more efficient expected search time than a balanced binary search tree?
- Search hit: $\frac{1}{2}(1 + \frac{1}{1-3/4}) = 2.5$
- Search miss/insert: $\frac{1}{2}(1 + \frac{1}{(1-3/4)^2}) = 8.5$
- Thus we need a hash table of size M where $\log_2 M = 8.5$, so $M \geq 2^{8.5} = 362$

Load factors and cost of probing



Quadratic probing

- Collision resolution function $f(i) = \pm i^2$: $h_i(x) = (h(x) \pm i^2) \mod M$ for $1 \leq i \leq \frac{(M-1)}{2}$
- M is a prime number of the form $4j + 3$, which guarantees that the probe sequence is a permutation of the table address space
- Eliminates primary clustering (when collisions group together causing more collisions for keys that hash to different values)
- Work example

Double hashing

- With quadratic probing, secondary clustering remains: keys that collide must follow sequence of prior collisions to find an open spot
- Double hashing reduces both primary and secondary clustering: probe sequence is dependent on original key, not just one hash value
- Collision resolution function $f(i) = i \cdot h_b(x)$: $h_i(x) = (h_A(x) + i \cdot h_B(x)) \mod M$
- Works best if M is prime
- Our approach: $h_A(x) = x \mod M$, $h_B(x) = R - (x \mod R)$ where R is a prime $< M$.

Rehashing

- We have already seen how hash table performance falls rapidly as the table load factor approaches 1 (in practice, any load factor above $1/2$ should be avoided)
- To rehash: create a new table whose capacity M' is the first prime more than twice as large as M
- Scan through the old table and insert into the new table, ignoring cells marked as deleted
- Running time $\Theta(M)$
  - Relatively expensive operation on its own
  - But good hash table implementations will only rehash when the table is half full, then double in size, so the operation should be rare
  - Can even consider the cost amortized over the $M/2$ insertions as constant addition to the insertions