Lecture 16
Hash Collision Resolution

EECS 281: Data Structures & Algorithms

Collision Resolution

Definition: method to handle case when two keys hash to same address

Methods of Collision Resolution

- Separate Chaining
- Linear Probing
- Quadratic Probing
- Double Hashing

Collision Resolution

Separate Chaining: scheme for collision resolution where we maintain *M* linked lists, one for each table address

Collision Resolution

Property: Separate chaining reduces the number of comparisons for sequential search by a factor of *M* (on average), using extra space for *M* links

Property: In a separate chaining hash table with *M* lists (table addresses) and *N* keys, the probability that the number of keys in each list is within a small constant factor of *N*/*M* is extremely close to 1 (O(1)) *if the hash function is good*

Collision Resolution

- Separate chaining
  - Insert: constant time
    - O(1)
  - Search: time proportional to *N*/*M*
    - O(*N*/*M*)
  - Remove: dependent upon Search
    - O(*N*/*M*)

This is why we choose M ≈ N: O(N/M) = O(1)

Collision Resolution

Use empty places in table to resolve collisions (known as *open-addressing*)

Probe: determination whether given table location is ‘occupied’

Linear Probing: when collision occurs, check the next position in the table

Possible Probe Outcomes

- Miss: probe finds empty cell in table, OR
- Hit: probe finds cell that contains item whose key matches search key, OR
- Full: probe finds cell has ‘occupant’, but key doesn’t match search key
- *If probe results in* full*, then probe table at next “higher” cell until* hit *(search ends successfully) or* miss *(search ends unsuccessfully)*

Cluster

Definition: contiguous group of occupied table cells

Consider table that is half-full ($M = 2N$)

What is best case/worst case distribution?

- Best Case:
- Worst Case:

Cluster

Consider table that is half-full ($M = 2N$)

<u>Pop Quiz</u>

- What is the *average* cost (in terms of N) to obtain a miss (find an empty cell) given the best case distribution?

- What is the *average* cost (in terms of N) to obtain a miss (find an empty cell) given the worst case distribution?

Linear Probing

- How to delete a key from a table built with linear probing?
  - Why is this hard?
- Option 1: remove it, re-hash rest of cluster
- Option 2: use a “dummy” element
  - Not an element, not empty either
  - We’ll call this ‘deleted’

“Deleted” Elements

- When an item is marked as deleted:
  - Insert considers it an empty spot and usable
  - Search considers it occupied (full) and keeps searching

Possible Probe Outcomes (Revised)

- Empty: probe finds cell that has never held item, OR
- Deleted: probe finds cell that once held item, but is not currently holding item, OR
- Hit: probe finds cell that contains item whose key matches search key, OR
- Full: probe finds cell has ‘occupant’, but key doesn’t match search key

Load Factor (α)

- $\alpha = N/M$, where N keys are placed in an M-sized table
- Separate Chaining
  - α is average number of items per list
  - α is sometimes larger than 1
- Linear Probing
  - α is percentage of table positions occupied
  - α is (must be) <= 1

Collision Resolution

When collisions are resolved with linear probing, the average number of probes required to search in a hash table of size *M* that contains *N* = α*M* keys is about

$$\frac{1}{2}\ 1+\frac{1}{1-}\div$$ for hits

$$\frac{1}{2}\ 1+\frac{1}{(1-\quad)^{2}}\div$$ for misses

Effect of Load on # of Probes
Linear Probing

α **(% Full)**	**Average Probes: Successful Search**	**Average Probes: Unsuccessful Search**
.1	1.1	1.1
.2	1.1	1.3
.3	1.2	1.5
.4	1.3	1.9
.5	1.5	2.5
.6	2.8	3.6
.7	2.2	6.1
.8	3.0	13.0
.9	5.5	50.5

Collision Resolution

Quadratic Probing

Try buckets at increasing 'distance' from hash table location

- h(*key*) mod $M \Rightarrow$ *addr*
- if bucket *addr* is full, then try
  - (h(*key*) + j^2) mod M for j = 1, 2, …

Collision Resolution

When collisions are resolved with quadratic probing, the average number of probes required to search in a hash table of size *M* that contains *N* = α*M* keys is about

$$1 - {}_{2} + \ln \frac{1}{1-} \div$$ for hits

$$\frac{1}{1-} + \ln \frac{1}{1} \div$$ for misses

Effect of Load on # of Probes Quadratic Probing

α **(% Full)**	**Average Probes: Successful Search**	**Average Probes: Unsuccessful Search**
.1	1.06	1.12
.2	1.12	1.27
.3	1.21	1.49
.4	1.31	1.78
.5	1.44	2.19
.6	1.62	2.82
.7	1.85	3.84
.8	2.21	5.81
.9	2.85	11.40

Collision Resolution

Double Hashing

Apply additional hash function if collision occurs

- $h(key) \bmod M \Rightarrow addr$
- If bucket *addr* is full, then try
  - $(h(key) + j * h'(key)) \bmod M$, where
    - $j = 1, 2, 3, \ldots$ and
    - $h'(k) = q - (k \bmod q)$ for some prime number $q < M$
  - Until an empty cell is found

New Topic: Dynamic Hashing

- As number of keys in hash table increases, search performance degrades
- Separate Chaining
  - Search time increases gradually
  - Double keys means double list length at each of *M* table locations
- Linear Probing
  - Search time increases dramatically as table fills
  - May reach point when no more keys can be inserted

Objective: Dynamic Hashing

- Double size of table when it ‘fills up’ (load factor is say one half)
- Expensive, but infrequent

Amortized Analysis

- Cannot guarantee that each and every operation will be fast, but can guarantee that average cost per operation will be low
- Total cost is low, but performance profile is erratic
- Most operations are extremely fast, but some operations require much more time to rebuild the table

Amortized Analysis: Concept

- Each insert
  - Pays (small constant) cost to actually insert
  - Deposits other small constant (“balance”) in a bank
- First *M*/2-1: build up “balance”
- (*M*/2)th insertion
  - Faced with a big (not small constant) bill
  - Finds a big (not small constant) balance
- Net result
  - Each insert charged small constant costs
  - Some costs deferred

Amortized Analysis: Applied

- Start with table of size M
- Each insertion in a table <= ½ full
  - Costs up to 2.5 probes (from table)
- Insert $M/2 - 1$ keys
  - $2.5 * (M/2-1) = O(M)$

Amortized Analysis: Applied

- Insert (M/2)th key
- Build new table, size 2M
  - Remove keys from old table, insert in new
  - Each insert <= ¼ full, costs a maximum of 1.5 probes (from table)
  - 1.5 * M/2 = O(M)
- O(M) + O(M) = O(M)
  - Linear time to insert M/2 keys, but last one is at a higher cost than the previous inserts

Summary: Hashing

- Collision Resolution
  - Separate Chaining creates a linked list for each table address
  - Linear Probing uses empty places in table to resolve collisions
  - Quadratic Probing looks for empty table address at increasing distance from original hash
  - Double Hashing applies additional hash function to original hash
- Dynamic Hashing increases the table size when it reaches some pre-determined load factor