Collision Resolution

Def'n: 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 $M \approx N$: $O(N/M) = O(1)$.

Collision Resolution

Use empty places in the 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

def:n: 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

def:n: contiguous group of occupied table cells

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

What is best case/worst case distribution?
- **Best Case:** every other cell is empty
- **Worst Case:** first half is full, second half is empty
Cluster
Consider table that is half-full ($M = 2N$)

Pop Quiz
- 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’

Possible Probe Outcomes (Revisited)
- 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 (\( \alpha \))

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

\[
\frac{1}{2} \left( 1 + \frac{1}{1 - \alpha} \right) \text{ for hits}
\]

\[
\frac{1}{2} \left( 1 + \frac{1}{(1 - \alpha)^2} \right) \text{ for misses}
\]

Examples
Examples

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 \( j = 1, 2, \ldots \)

Collision Resolution

Double Hashing
Apply additional hash function if collision occurs

- \( h(key) \mod M \Rightarrow addr \)
- if bucket \( addr \) is full, then try
  - \( (h(key) + j \times h'(key)) \mod M \), where
    - \( j = 1, 2, 3, \ldots \) and
    - \( h'(k) = q - (k \mod q) \) for some prime number \( q < M \)
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’ (more than half full)
- 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 certain operations require as much time as previous cost of building 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 \( M \)
- Insert \( M/2 \)-1 keys
- Each insertion in a table <= \( \frac{1}{2} \) full
  - costs avg 2.5 probes (from table)
- Insert \( M/2\)-1 keys
  - \( 2.5 \times (M/2)-1 ) = O(M) \)

Amortized Analysis: Applied

- Insert \((M/2)^{th}\) key
- Build new table, size \( 2M \)
  - remove keys from old table, insert in new
  - each insert <= \( \frac{1}{4} \) full, costs avg 1.5 probes (from table)
  - \( 1.5 \times M/2 = O(M) \)
- \( O(M) + O(M) = O(M) \)
  - linear time to insert \( M \) keys, but last one is a doozy
Summary: Hashing

- **Collision Resolution**
  - Linear Probing uses empty places in table to resolve collisions

- **Dynamic Hashing**
  - Modify size of hash table when it is x% full

---

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