Hash Tables

CSE 2020 Computer Science II

Learning Objectives

- Define hash table ADT
- Compare 3 different collision resolution strategies, linear probing, quadratic probing, and double hashing.

Hash Table ADT

- The insertion, deletion and search operations can be performed in constant expected time.
- It supports only a subset of the operations allowed by binary search tree. Tree operations that require any ordering information among elements are not supported, such as `findMin`, `findMax`, print all elements in ascending order.

General Idea

- Search is performed on keys. Keys should be unique. An element consists of a key and additional data members, such as employee id, name, and dept.
- A table has the fixed number of cells, TableSize. Each cell has a unique index.
- Hash function *hash()* converts/maps a key into an index of a table cell.
- Each key is mapped into an index in the range 0 to TableSize - 1 and placed in the appropriate cell.
- The keys can be distributed evenly among the cells.

Example

- entries:

  (1005, “Bob”, “CSE”)

  (1002, “Joe”, “Art”)

  (1001, “Alice”, “Art”)

  (1007, “Mary”, “CSE”)

  (1009, “Emma”, “CSE”)

- function

  i = key % 10

0	
1	(1001, “Alice”, “Art”)
2	(1002, “Joe”, “Art”)
3	
4	
5	(1005, “Bob”, ”CSE”)
6	
7	(1007, “Mary”, “CSE”)
8	
9	(1009, “Emma”, “CSE”)

Implementation of Hash Table

- Use Array to implement Hash Table
- Three problems to be solved:
  - TableSize, the capacity of array
    - a prime
  - hash function *hash()*
    - simple to compute
    - distribute the keys evenly
  - collision resolution

Collision Resolution

- Collision - entry (k, v), i = h(k), and a[i] is occupied by an entry
- Try alternative cells until an empty cell is found, cells $h_0(k)$, $h_1(k)$, $h_2(k)$, … are tried in succession, where

$$h_i(k) = (hash(k) + f(i))\ \%\ TableSize$$

- Function `f()` is collision resolution strategy, `f(0) = 0`
- 3 common collision resolution strategies

Linear Probing

- $h_i(k) = (hash(k) + f(i)) \, \% \, TableSize$
- Function f() is a linear function of i, that is,

$$f(i) = i$$

- Example keys (89, 18, 49, 58, 69)
  - TableSize = 10
  - hash(k) = k % 10
  - f(0) = 0, f(1) = 1, f(2) = 2, …
  - $h_0(k) = (hash(k) + f(0)) \, \% \, 10$
  - $h_1(k) = (hash(k) + f(1)) \, \% \, 10$
  - $h_2(k) = (hash(k) + f(2)) \, \% \, 10$

0	49
1	58
2	69
3	
4	
5	
6	
7	
8	18
9	89

Quadratic Probing

- $h_i(k) = (hash(k) + f(i)) \% TableSize$
- Function f() is a quadratic function of i, that is,

$$f(i) = i^2$$

- Example keys (89, 18, 49, 58, 69)
  - TableSize = 10
  - hash(k) = k % 10
  - f(0) = 0, f(1) = 1, f(2) = 4, …
  - $h_0(k) = (hash(k) + f(0)) \% 10$
  - $h_1(k) = (hash(k) + f(1)) \% 10$
  - $h_2(k) = (hash(k) + f(2)) \% 10$

0	49
1	
2	58
3	69
4	
5	
6	
7	
8	18
9	89

Double Hashing

- $h_i(k) = (hash(k) + f(i)) \% TableSize$
- Function $f(i) = i * hash_2(k)$, $hash_2()$ is a second hash function
- Example keys (89, 18, 49, 58, 69)
  - TableSize = 10
  - hash(k) = k % 10
  - $hash_2(k) = 7 - (k \% 7)$
  - $f(0) = 0$, $f(1) = hash_2(k)$,
    $f(2) = 2hash_2(k)$, …
  - $h_0(k) = (hash(k) + 0) \% 10$
  - $h_1(k) = (hash(k) + 1 * hash_2(k)) \% 10$
  - $h_2(k) = (hash(k) + 2 * hash_2(k)) \% 10$

0	69
1	
2	
3	58
4	
5	
6	49
7	
8	18
9	89

Hash Tables in STL

- In C++11, the Standard Template Library (STL) includes hash table implementation of sets and maps, namely, `unordered_set` and `unordered_map`
- `unordered_set` and `unordered_map` can be instantiated with function objects that provide a hash function and equality operator
- `unordered_set`
  https://www.geeksforgeeks.org/unordered_set-in-cpp-stl/
- `unordered_map`
- https://www.geeksforgeeks.org/unordered_map-in-cpp-stl/