UCS1302
DATA STRUCTURES

Hashing

Session Objectives

- To learn about hashing algorithms
  - Hash tables
  - Collision resolution

Session Outcomes

- At the end of this session, participants will be able to
  - Understand the hash tables
  - Understand the hashing algorithms
  - Understand the collision resolution
    - **Separate chaining**
    - **Open addressing**
      - Linear Probing
      - Quadratic Probing
      - Double Hashing

Agenda

- Hash table
- Simple hashing algorithm
- Collision resolution
  - Separate chaining
  - Open addressing
    - – Linear Probing
    - – Quadratic Probing
    - – Double Hashing

Hashing

Dr. B. Bharathi
SSNCE

September 17, 2019

Hash Tables

- Consider an array or an array list.
  - To get a value from an array, need to specify an integer index.
  - The array "maps" the index to a data value stored in the array.
    - The mapping function is very efficient.
  - As long as the index value is within range, there is a strict one-to-one correspondence between an index value and a stored data value.
  - We can consider the index value to be the "key" to obtaining the corresponding data value.

Hash Tables

- A hash table also stores data values.
  - Use a key to obtain the corresponding data value.
  - The key does not have to be an integer value.
    - For example, the key could be a string.
  - There might not be a one-to-one correspondence between keys and data values.
  - The mapping function may not be trivial.

Hash Tables

- We can implement a hash table as an array of cells.
  - Refer to its size as TableSize
- If the hash table's mapping function maps a key value into an integer value in the range 0 to TableSize – 1, then we can use this integer value as the index into the underlying array.

Hash Tables

- Suppose we're storing employee data records into a hash table.
  - We want to use an employee's name as the key.
- Further suppose that the name *john* hashes (maps) to 3, *phil* hashes to 4, *dave* hashes to 6, and *mary* hashes to 7.
  - This is an ideal situation because each employee record ended up in a different table cell.

0	
1	
2	
3	john 25000
4	phil 31250
5	
6	dave 27500
7	mary 28200
8	
9	

An ideal hash table

Hash function example

Elements = Integers

h(i) = i % 10

41, 34, 7, and 18

0	
1	41
2	
3	
4	34
5	
6	
7	7
8	18
9	

Hash collisions

- **Collision**: the event that two hash table elements map into the same slot in the array
- Example: add 41, 34, 7, 18, then 21
  - 21 hashes into the same slot as 41!
  - 21 should not replace 41 in the hash table;
    they should both be there
- **Collision resolution**: means for fixing collisions in a hash table

0	
1	21
2	
3	
4	34
5	
6	
7	7
8	18
9	

Collision Resolution techniques

- If, when an element is inserted, it hashes to the same value as an already inserted element, then we have a collision and need to resolve it.
- There are several methods for dealing with this:
  - **Separate chaining**
  - **Open addressing**
    - Linear Probing
    - Quadratic Probing
    - Double Hashing

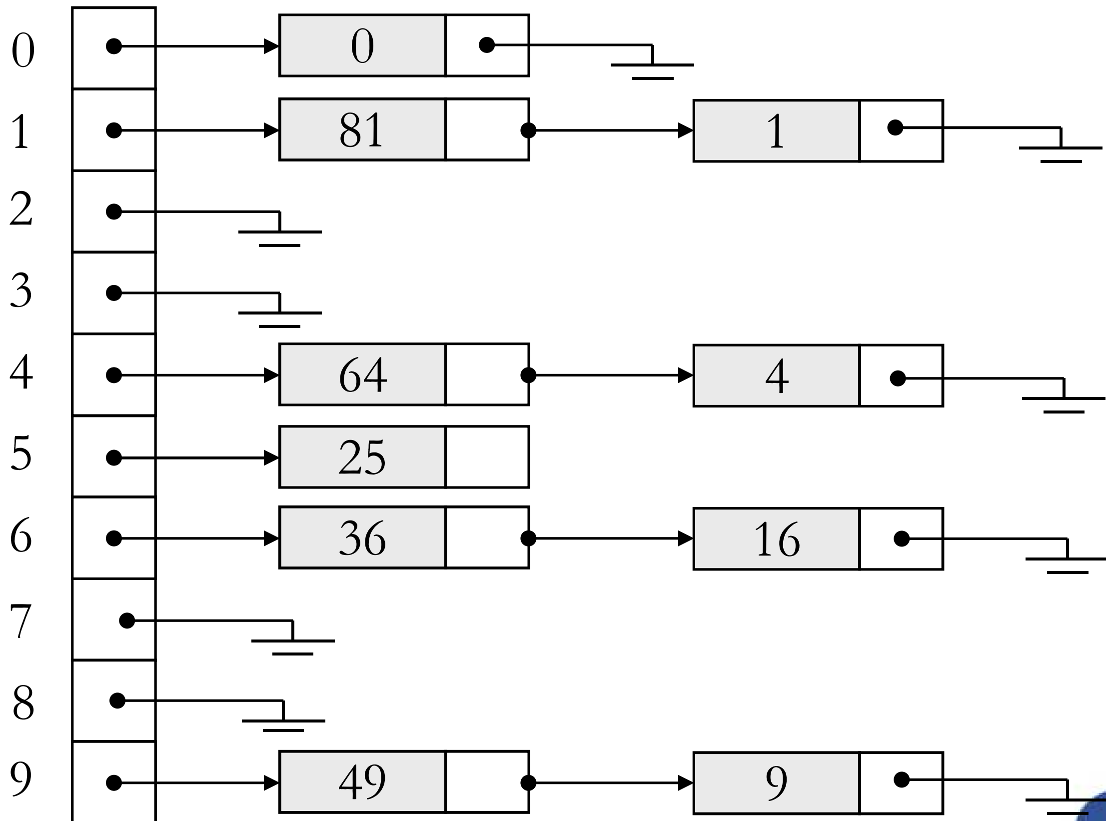
Separate Chaining

- The idea is to keep a list of all elements that hash to the same value.
  - The array elements are pointers to the first nodes of the lists.
  - A new item is inserted to the front of the list.
- Advantages:
  - Better space utilization for large items.
  - Simple collision handling: searching linked list.
  - Overflow: we can store more items than the hash table size.
  - Deletion is quick and easy: deletion from the linked list.

Example

Keys: 0, 1, 4, 9, 16, 25, 36, 49, 64, 81

$h(i) = i \% 10$



Collision Resolution with Open Addressing

- Separate chaining has the disadvantage of using linked lists.
  - Requires the implementation of a second data structure.
- In an open addressing hashing system, all the data go inside the table.
  - Thus, a bigger table is needed.
  - If a collision occurs, alternative cells are tried until an empty cell is found.
- There are three common collision resolution strategies:
  - Linear Probing
  - Quadratic probing
  - Double hashing

Linear Probing

- In linear probing, collisions are resolved by sequentially scanning an array (with wraparound) until an empty cell is found.
  - i.e. *f* is a linear function of *i*, typically *f(i)= i*.
- $h_i(x)$ = (hash(x) + F(i)) mod tablesize where F is the collision resolution strategy
- Here F(i) = i
  - add 41, 34, 7, 18, then 21, then 57
    - 21 collides (41 is already there), so we search ahead until we find empty slot 2
    - 57 collides (7 is already there), so we search ahead twice until we find empty slot 9

0	
1	41
2	21
3	
4	34
5	
6	
7	7
8	18
9	57

Linear Probing

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

Hash table with linear probing, after each insertion

The first collision occurs with 49:
Put 49 into cell 0.

58 collides with
18, 89, and 49 → cell 1.

69 → cell 2

Linear Probing

- add 89, 18, 49, 58, 69
  - 49 collides (89 is already there), so we search ahead by +1 to empty slot 0
  - 58 collides (18 is already there), so we search ahead by +1 to occupied slot 9, then +3 to empty slot 2
  - 9 collides (89 is already there), so we search ahead by +1 to occupied slot 0, then +4 to empty slot 3

Primary Clustering

- As long as table is big enough, a free cell can always be found, but the time to do so can get quite large.
- Worse, even if the table is relatively empty, blocks of occupied cells start forming.
- This effect is known as *primary clustering*.
- Any key that hashes into the cluster will require several attempts to resolve the collision, and then it will add to the cluster.

Quadratic Probing

- Quadratic Probing eliminates primary clustering problem of linear probing.

- Collision function is quadratic.
  - The popular choice is $f(i) = i^2$.

- If the hash function evaluates to h and a search in cell h is inconclusive, we try cells $h + 1^2$, $h+2^2$, … $h + i^2$.
  - i.e. It examines cells 1,4,9,16 and so on away from the original probe.

Quadratic Probing

$F(i) = i^2$

Add 89, 18, 49, 58, 69

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

Problem

- We may not be sure that we will probe all locations in the table (i.e. there is no guarantee to find an empty cell if table is more than half full.)
- If the hash table size is not prime this problem will be much severe.

Double Hashing

- A second hash function is used to drive the collision resolution.
  - $f(i) = i * hash_2(x)$

- We apply a second hash function to x and probe at a distance $hash_2(x)$, $2*hash_2(x)$, … and so on.

- A function such as $hash_2(x) = R - (x \bmod R)$ with R a prime smaller than TableSize will work well.
  - e.g. try R = 7 for the previous example.(7 - x mod 7)

Double Hashing

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

Rehashing

- Hash Table may get full
  - No more insertions possible
- Hash table may get *too* full
  - Insertions, deletions, search take longer time
- Solution: Rehash
  - Build another table that is twice as big and has a new hash function
  - Move all elements from smaller table to bigger table

Rehashing Example

h(x)=x mod 7

h(x)=x mod 17

Original Hash Table

0	6
1	15
2	
3	24
4	
5	
6	13

Input 13,15,6,24

After Inserting 23

0	6
1	15
2	23
3	24
4	
5	
6	13

After Rehashing

0	
1	
2	
3	
4	
5	
6	6
7	23
8	24
9	
10	
11	
12	
13	13
14	
15	15
16	

Hashing Applications

- Compilers use hash tables to implement the *symbol table* (a data structure to keep track of declared variables).
- Game programs use hash tables to keep track of positions it has encountered (*transposition table*)
- Online spelling checkers.
- Looking up Passwords
- Routing Table

Summary

- Hash table
- Simple hashing algorithm
- Collision resolution
  - Separate chaining
  - Open addressing
    - Linear Probing
    - Quadratic Probing
    - Double Hashing