Data Structures

Hashing and File Structure

Rutvij H. Jhaveri
Computer Science & Engineering

Outline

- What is Hashing
- Hash-table
- Hashing methods
- Hash collision
- Hash resolution techniques
  - Open hashing
  - Closed hashign

[Darshan]

What is Hashing?

- Sequential search usually requires more comparisons to search an element with complexity O(n)
  - Increased number of comparisons are not desirable for a large database
- Binary search usually requires fewer comparisons with complexity O (log n)
  - There is an additional requirement that the data should be sorted. Even with best sorting algorithm, sorting of elements require O(n log n) comparisons.
- **Hashing** is another widely used technique for storing data which doesn't require sorted data and still provides constant time search, insert and delete operations on average.

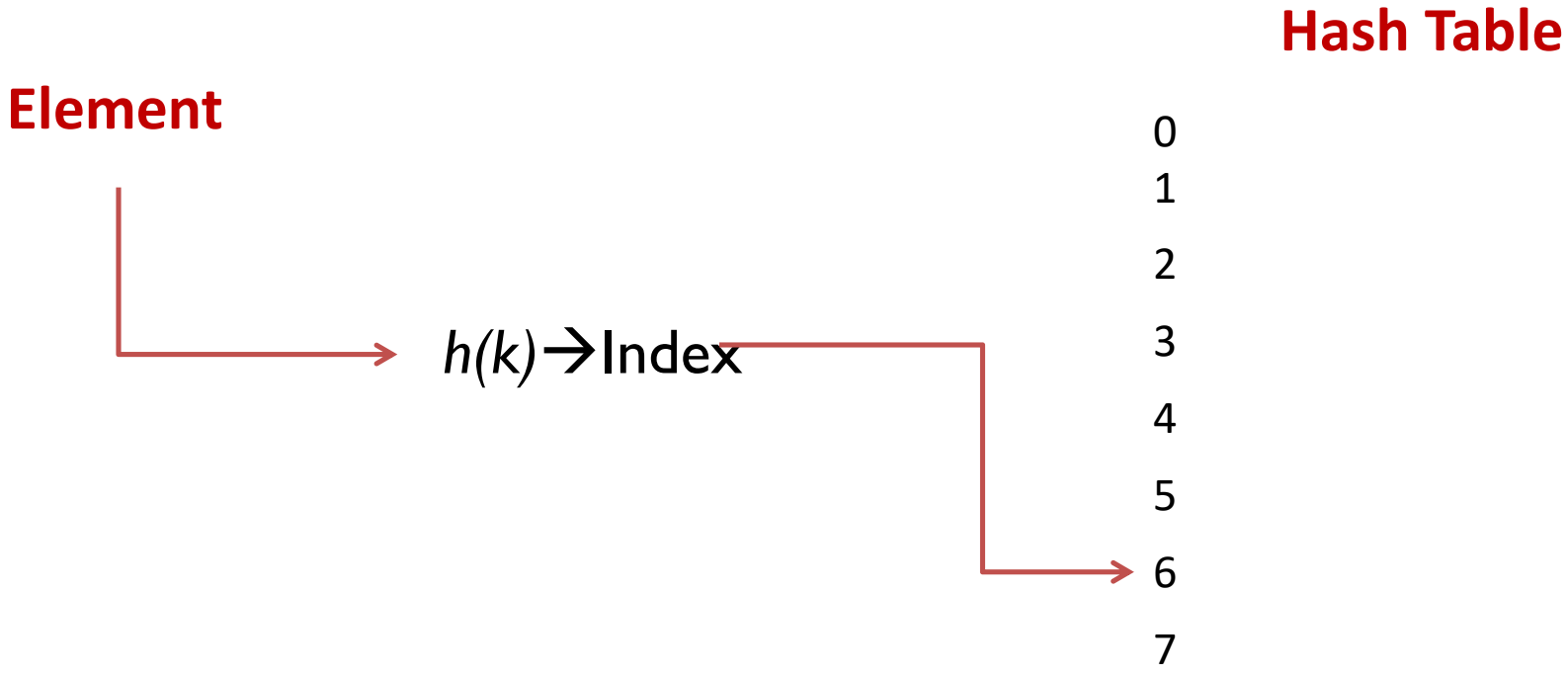
How Hashing Works

- Data structure called **Hash Table** uses an array as a storage medium and uses hash technique to generate an index where an element is to be inserted or is to be located from.
- In Hash Tables, insertion, deletion and search operations are very fast irrespective of the size of the data.
- Hashing converts a range of key values into a range of indices.
- The element (or record) for a key value is directly accessed by calculating the address from the key value.
- Address or location of an element is obtained by computing some arithmetic function *h*.
- *h(key)* provides the address of a record in the table.

Applications

- **Database systems**: Specifically, those that require efficient random access. Hash tables are an important part of efficient random access because they provide a way to locate data in a constant amount of time.
- **Symbol tables**: The tables used by compilers to maintain information about symbols from a program. Compilers access information about symbols frequently. Therefore, it is important that symbol tables be implemented very efficiently.
- **Data dictionaries**: Although the operations of a hash table and a data dictionary are similar, other data structures may be used to implement data dictionaries. However, using a hash table is particularly efficient.
- **Network processing algorithms**: Hash tables are fundamental components of several network processing algorithms and applications, including route lookup, packet classification, and network monitoring.
- **Browser Cashes**: Hash tables are used to implement browser cashes.

Mapping an Element with Hash Function *h*



Hashing Functions

Characteristics of a good hash function

- minimizes collisions.
- uses all the information provided in the key.
- tends to spread keys evenly in the array.
- easy and quick to compute.

Hashing methods

1. Division-Method
2. Midsquare Methods
3. Folding Method
4. Digit Analysis
5. Length Dependent Method
6. Radix Conversion
7. Multiplicative Hashing …

Division-Method

- Modular arithmetic system is used to divide the key value by some integer divisor *m* (may be table size).
- It gives us the location value, where the element can be placed.
- We can write, *h(k)* = (k mod m) + 1,
  - k = key value
  - m = table size/number of slots in file
- Suppose, k = 43, m = 10 then
  - *h(k)* = (43 mod 10) + 1= 3 + 1=4
  - The key whose value is 43 is placed in 4^{th} location.

Midsquare Method

- The value of key is squared and the number of digits required to form an address, are taken from the middle position of squared value.
- Suppose a key value is 12
  - Square is 144
  - Consider that we want address of two digits
  - The address selected as 44 (i.e. two digits starting from middle of 1**44**)

Folding Method

- Most machines have a small number of primitive data types for which there are arithmetic instructions
- Key to be used may not fit easily into one of these data types
- It is not possible to discard the portion of the key that does not fit into such an arithmetic data type
- The solution is to combine various parts of the key in such a way that all parts of the key affect in the final result (such an operation is termed as *folding* of the key)
- The key is partitioned into number of parts, each part having the same length as that of the required address
- Add the value of each parts, ignoring the final carry to get the required address

…Folding Method

- Two ways

Fold-shifting:

Here actual values of each parts of key are added

- E.g. the key is : 12345678, and the required address is of two digits
- Break the key into: 12, 34, 56, 78
- Add these, we get 12 + 34 + 56 + 78 : 180, ignore first 1 we get 80 as location

Fold-boundary: Here the reversed values of outer parts of key are added

- E.g.the key is : 12345678, and the required address is of two digits
- Beak the key into: **2**1, 34, 56, **8**7
- Add these, we get 21 + 34 + 56 + 87 : 198, ignore first 1 we get 98 as location

Digit Analysis Method

- This hashing method is a distribution-dependent
- Statistical analysis of digits of the key is performed, and those digits (of fixed position) are selected which occur quite frequently
- Then reverse or shift the digits to get the address
- Consider the key is : 9871534
  - If the statistical analysis has revealed that the third and fifth position digits occur quite frequently,
    - We choose the digits in these positions from the key
  - Thus, we get 75. Reversing it we get 57 as the address

Length Dependent Method

- The key and the length of the key are combined in some way to form either the index itself or an intermediate value that is then used with one of our other methods to form the index.
- Consider the key is 8765,
  - we might multiply the first two digits by the length (87x4=348)
  - then divide by the last digit, yielding 69.
  - If our table size is 10, we would then use the division method (*as discussed earlier*), resulting in an index of 9

Radix Conversion

- Transforms a key into another number base to obtain the hash value.
- Typically use number base other than base 10 and base 2 to calculate the hash addresses.
- To map the key 55354 in the range 0 to 9999 using base 11 we have:

$$55354_{10} = 3\underline{8652}_{11}$$

- We may truncate the high-order 3 to yield 8652 as our hash address within 0 to 9999.

Multiplicative Hashing

- This method is based on obtaining an address of a key, based on the multiplication value.
- If k is the non-negative key, and a constant c, (0 < c < 1)
  - Compute kc mod 1, which is a fractional part of kc.
  - Multiply this fractional part by *m* (size of hash table) and take a floor value to get the address

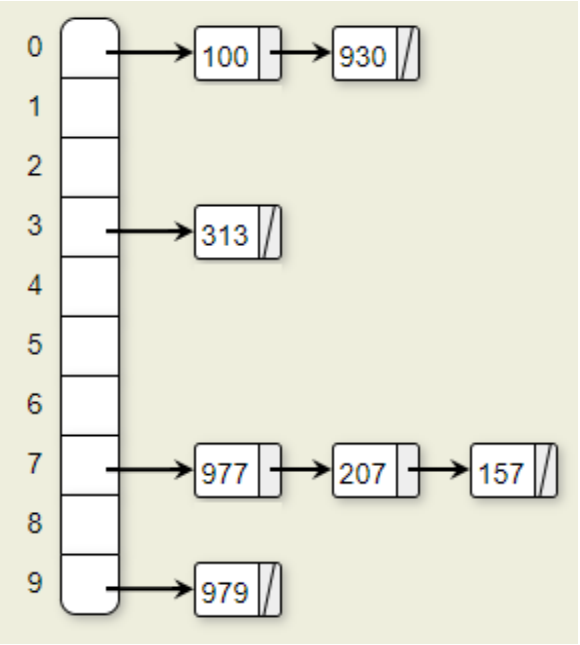
    h(k) = floor(m(kc mod 1))
  - Consider k=123 m=100 c=0.566
    - h(123) = 100 (123 * 0.566 mod 1)
      = 100 (69.618 mod 1)
      = 100 (0.618)
      = 61

Collision Resolution Strategies

- If the element to be inserted is mapped to the same location, where an element is already inserted then we have a collision and it must be resolved.
- Collision resolution techniques can be broken into two classes: open hashing (also called separate chaining) and closed hashing (also called open addressing).

Open Hashing

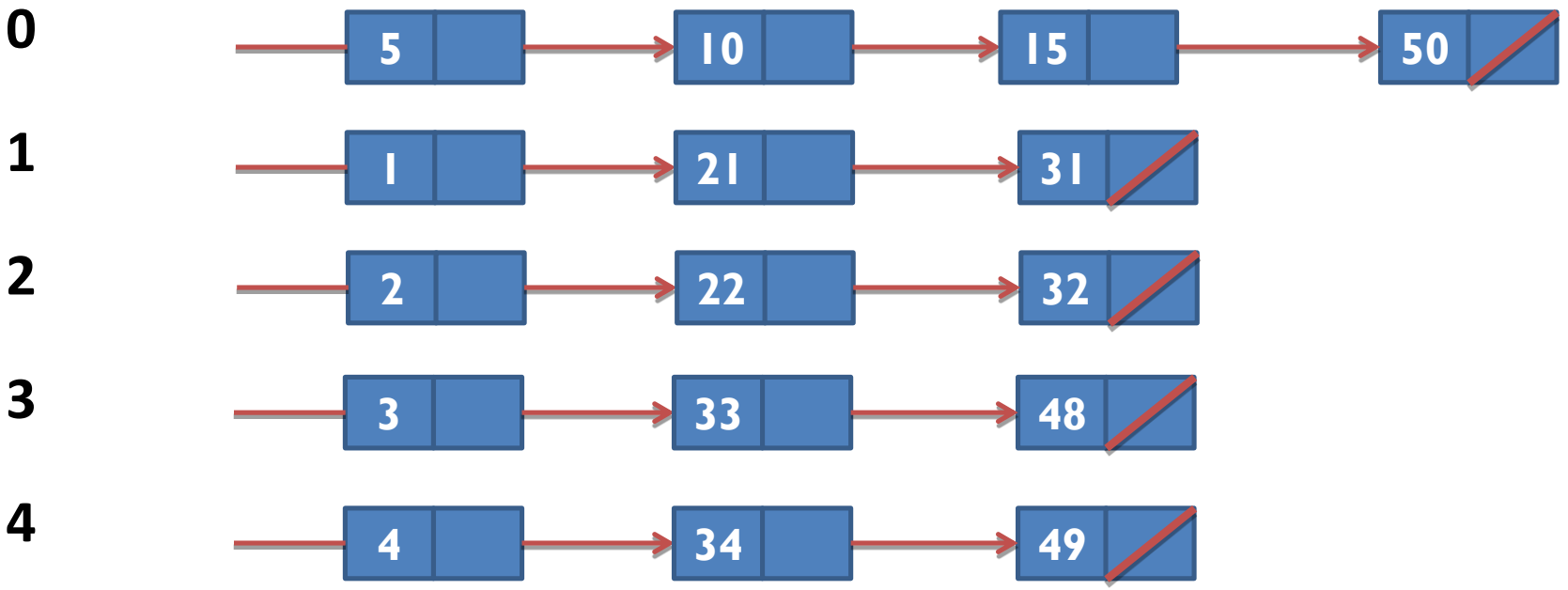
- The simplest form of open hashing defines each slot in the hash table to be the head of a linked list.
- All records that hash to a particular slot are placed on that slot's linked list. This method is also known as *Separate Chaining*.
- The following figure illustrates a hash table where each slot points to a linked list to hold the records associated with that slot.
- The hash function used is the simple mod function.



…Open Hashing

1, 2, 3, 4, 5, 10, 21, 22, 33, 34, 15, 32, 31, 48, 49, 50

h(k)=k mod 5



Hash Table

Closed Hashing

- All entry records are stored in the bucket array itself.
  - When a new entry has to be inserted, the buckets are examined, starting with the hashed-to slot and proceeding in some probe sequence, until an unoccupied slot is found.
  - When searching for an entry, the buckets are scanned in the same sequence. This method is also termed as *Open addressing*
- Well-known probe sequences include:
  - Linear probing,
  - Quadratic probing
  - Double hashing
- Drawback of all these schemes is that the number of stored entries cannot exceed the number of slots in the bucket array.

Linear Probing

- In linear probing, whenever there is a collision, cells are searched sequentially (with wraparound) for an empty cell.
- Fig. shows the result of inserting keys {5,18,55,78,35,15} using the hash function (f(key)= key%10) and linear probing strategy.

	Empty Table	After 5	After 18	After 55	After 78	After 35	After 15
0							15
1							
2							
3							
4							
5		5	5	5	5	5	5
6				55	55	55	55
7						35	35
8			18	18	18	18	18
9					78	78	78

…Linear Probing

- Linear probing is easy to implement but it suffers from "primary clustering“
- When many keys are mapped to the same location (clustering), linear probing will not distribute these keys evenly in the hash table.
- These keys will be stored in neighbourhood of the location where they are mapped.
- This will lead to clustering of keys around the point of collision

Quadratic probing

- One way of reducing "primary clustering" is to use quadratic probing to resolve collision.
- Suppose the "key" is mapped to the location *j* and the cell *j* is already occupied.
- In quadratic probing, the location j, (j+1), (j+4), (j+9), ... are examined to find the first empty cell where the key is to be inserted.
- This method reduces primary clustering.
- It does not ensure that all cells in the table will be examined to find an empty cell.
- Thus, it may be possible that key will not be inserted even if there is an empty cell in the table.

Double Hashing

- This method requires two hashing functions *h1(key)* and *h2 (key)*.
- Problem of clustering can easily be handled through double hashing.
- Function *h1(key)* is known as primary hash function.
- In case the address obtained by *h1(key)* is already occupied by a key, the function *h2(key)* is evaluated.
- The second function *h2(key)* is used to compute the increment to be added to the address obtained by primary function *h1(key)* in case of collision.
- The search for an empty location is made successively at the addresses
  - h1(key) + h2(key),
  - h1(key) + 2 * h2(key),
  - h1 (key) + 3 * h2(key),...

$h(x,i)=(h1(x)+ih2) \bmod m$

…Double Hashing Example

- T

Problems for Which Hash Tables are NOT Suitable

1. **Problems for which data ordering is required**

   Because a hash table is an unordered data structure, certain operations are difficult and expensive. There are hash table implementations that keep the keys in order, but they are far from efficient.

2. Problems having **multidimensional data**
3. **Prefix searching** especially if the keys are long and of variable-lengths.
4. **Problems that have dynamic data**

   Open-addressed hash tables are based on 1D-arrays, which are difficult to resize once they have been allocated. Unless you want to implement the table as a dynamic array and rehash all of the keys whenever the size changes. This is an incredibly expensive operation. An alternative is use a separate-chained hash tables or dynamic hashing.

5. **Problems in which the data does not have unique keys**

   Open-addressed hash tables cannot be used if the data does not have unique keys. An alternative is use separate-chained hash tables.

Summary

- Hashing
- Hash-table
- Different hashing methods
- Hash collision
- Hash functions
- Hash resolution techniques
  - Open hashing and Closed hashing
- Incapability of hash tables

Assignments

- P