SE2205: Algorithms and Data Structures for Object-Oriented Design

Hash Tables

Instructor: Pirathayini Srikantha

University of Toronto

March 27, 2019

Readings/References

- Goodric (10.2)

Table of Contents

Table of Contents

Abstract Data Types

- A **table** is an ADT that consists of table entries each containing a **unique** key K and associated information I
- Each table entry is then a **key pair (K,I)**
- How can tables be represented?
  - Arrays
  - Binary search trees
  - AVL trees
- **Hashing** is an alternative that is very efficient under **some circumstances**
- **Hash functions** $h(K)$ map a key K to an address in a table
- If the function h provides a **unique one-to-one mapping** of K to an address, then table entries can be accessed directly
- If not, it is necessary to evoke **collision resolution policies**

Example: Hash Table

	Key (K)	Information (I)
0	24	X
1	19	S
2	2	B
3		
4	10	J
5	23	W
6		

Table of Contents

Collisions and Resolutions

- Collisions occur when $h(K) = h(K')$ where K and K' are different keys (i.e. not a one-to-one mapping)
- Collisions can be resolved via **collision resolution policies**
- **Open addressing** (linear probe or double hashing): When a collision occurs, a probe for other available locations is performed
- **Separate chaining**: All colliding keys are stored in a linked list
- Hashing with **buckets**: Each bucket will represent a sub-table

Open Addressing

Open Addressing

- First, the location at which (K, I) is to be stored is determined through the hash function $h(K)$
- One assumption with **open addressing** is that at least one empty spot available for the storage of (K, I) (i.e. table is not full)
- If there is a collision, then a sequence of probing is performed to search for the next available spot
- **Linear probing**: Search decrement is constant for all keys
  - Every consecutive table entry is examined until an available spot is found
  - **Probe sequence** for every colliding key can overlap
- **Double hashing**: Search decrement is not the same for all keys
  - Probe sequence has a higher chance of being different for colliding keys
  - Can find an empty spot faster

Example: Inserting into a Hash Table

	Key (K)	Information (I)
0		
1		
2		
3		
4		
5		
6		

Hash Function
h(K)=k%7

Information to be stored:
(2, B)
(22, V)
(19, S)
(21, U)
(20, T)

Example: Inserting into a Hash Table

	Key (K)	Information (I)
0		
1		
2		
3		
4		
5		
6		

Hash Function
h(K)=k%7

Information to be stored:
(2, B) maps to 2
(22, V) maps to 1
(19, S) maps to 5
(21, U) maps to 0
(20, T) maps to 6

Example: Inserting into a Hash Table

	Key (K)	Information (I)
0	21	U
1	22	V
2	2	B
3		
4		
5	19	S
6	20	T

Hash Function

h(K)=k%7

Information to be stored:

(2, B) maps to 2

(22, V) maps to 1

(19, S) maps to 5

(21, U) maps to 0

(20, T) maps to 6

Linear Probing

Example: Linear Probing

	Key (K)	Information (I)
0	21	U
1	22	V
2	2	B
3		
4		
5	19	S
6	20	T

Hash Function
h(K)=k%7

Information to be stored:
(2, B) maps to 2
(22, V) maps to 1
(19, S) maps to 5
(21, U) maps to 0
(20, T) maps to 6
(16,P) maps to 2

Example: Linear Probing

	Key (K)	Information (I)
0	21	U
1	22	V
2	2	B
3		
4		
5	19	S
6	20	T

Hash Function
h(K)=k%7

Information to be stored:
(16,P) maps to 2

Collision Resolution via Linear Probing:
Check location 1

Example: Linear Probing

	Key (K)	Information (I)
0	21	U
1	22	V
2	2	B
3		
4		
5	19	S
6	20	T

Hash Function
h(K)=k%7

Information to be stored:
(16,P) maps to 2

Collision Resolution via Linear Probing:
Check location 1
Check location 0

Example: Linear Probing

	Key (K)	Information (I)
0	21	U
1	22	V
2	2	B
3		
4		
5	19	S
6	20	T

Hash Function
h(K)=k%7

Information to be stored:
(16,P) maps to 2

Collision Resolution via Linear Probing:
Check location 1
Check location 0
Check location 6

Example: Linear Probing

	Key (K)	Information (I)
0	21	U
1	22	V
2	2	B
3		
4		
5	19	S
6	20	T

Hash Function
h(K)=k%7

Information to be stored:
(16,P) maps to 2

Collision Resolution via Linear Probing:
Check location 1
Check location 0
Check location 6
Check location 5

Example: Linear Probing

	Key (K)	Information (I)
0	21	U
1	22	V
2	2	B
3		
4		
5	19	S
6	20	T

Hash Function
h(K)=k%7

Information to be stored:
(16,P) maps to 2

Collision Resolution via Linear Probing:
Check location 1
Check location 0
Check location 6
Check location 5
Check location 4

Example: Linear Probing

	Key (K)	Information (I)
0	21	U
1	22	V
2	2	B
3		
4	16	P
5	19	S
6	20	T

Hash Function
h(K)=k%7

Information to be stored:
(16,P) maps to 2

Collision Resolution via Linear Probing:
Check location 1
Check location 0
Check location 6
Check location 5
Check location 4

Double Hashing

Example: Double Hashing

	Key (K)	Information (I)
0	21	U
1	22	V
2	2	B
3		
4		
5	19	S
6	20	T

Hash Function
h(K)=k%7

Information to be stored:
(2, B) maps to 2
(22, V) maps to 1
(19, S) maps to 5
(21, U) maps to 0
(20, T) maps to 6
(16,P) maps to 2

Example: Double Hashing

	Key (K)	Information (I)
0	21	U
1	22	V
2	2	B
3		
4		
5	19	S
6	20	T

Hash Function
h(K)=k%7
p(K)=max(1,floor(K/7))

Information to be stored:
(2, B) maps to 2
(22, V) maps to 1
(19, S) maps to 5
(21, U) maps to 0
(20, T) maps to 6
(16,P) maps to 2

Example: Double Hashing

	Key (K)	Information (I)
0	21	U
1	22	V
2	2	B
3		
4		
5	19	S
6	20	T

Hash Function
h(K)=k%7=2
p(K)=max(1,floor(K/7))=2

Information to be stored:
(16,P) maps to 2

Collision Resolution via Double Hashing:
Check location 0

Example: Double Hashing

	Key (K)	Information (I)
0	21	U
1	22	V
2	2	B
3		
4		
5	19	S
6	20	T

Hash Function
h(K)=k%7=2
p(K)=max(1,floor(K/7))=2

Information to be stored:
(16,P) maps to 2

Collision Resolution via Double Hashing:
Check location 0
Check location 5

Example: Double Hashing

	Key (K)	Information (I)
0	21	U
1	22	V
2	2	B
3		
4		
5	19	S
6	20	T

Hash Function

h(K)=k%7=2

p(K)=max(1,floor(K/7))=2

Information to be stored:

(16,P) maps to 2

Collision Resolution via Double Hashing:

Check location 0

Check location 5

Check location 3

Example: Double Hashing

	Key (K)	Information (I)
0	21	U
1	22	V
2	2	B
3	16	P
4		
5	19	S
6	20	T

Hash Function
h(K)=k%7=2
p(K)=max(1,floor(K/7))=2

Information to be stored:
(16,P) maps to 2

Collision Resolution via Double Hashing:
Check location 0
Check location 5
Check location 3

An Implementation of Open Addressing Hash Table

Hash Table Structures

Data Structures:

- Class for key:
  - Key<K>
- Class information:
  - Info<I>
- Classes for table:
  - Table<T>

Hash Function and Probe Decrement Definitions

```
public int h(Key<K> k){
    return k.getValue()% M;
}

public int p(Key<K> k, int M){
    int quo=k.getValue/M;
    if (quo !=0) {
        return quo;
    }
    else{
        return 1;
    }
}
```

Key Insertions

```
public void insertKey(Key<K> k, Info<I> info, Table<T> t){
    int index=h(K);
    int probeDecr=p(K);
    while (t.get(index)!=null) {
      index=index-probeDecr;
      if(index<0)
        index=index+M;
    }
    t.set(index,k,info);
}
```

Key Searching

```
public boolean searchKey(Key<K> k, Table<T> t){
    int index=h(K);
    int probeDecr=p(K);
    while (t.get(index)!=null) {
      index=index-probeDecr;
      if(index<0)
        index=index+M;
      }
    t.set(index,k,info);
    if (t.get(index)!=null)
      return false;
    else
      return true;
}
```

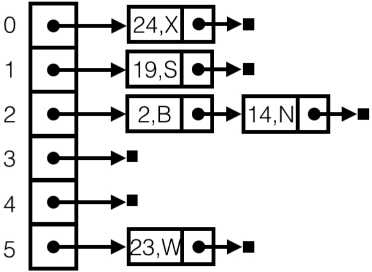
Chaining

Chaining

- With the chaining method for collision resolution, key pairs that map to the identical location in the table are stored in a linked list in an ascending order of the key value

Hash Function
h(K)=k%6

Information to be stored:
(2, B) maps to 2
(19, S) maps to 1
(24, X) maps to 0
(23,W) maps to 5
(14,N) maps to 2



Buckets

Chaining

- If the amount of table entries is large, then the table is divided into buckets where each bucket represents a subtable
- Each subtable can be assigned a linked list in which key pairs are stored in ascending order
- Allows for conservation of space

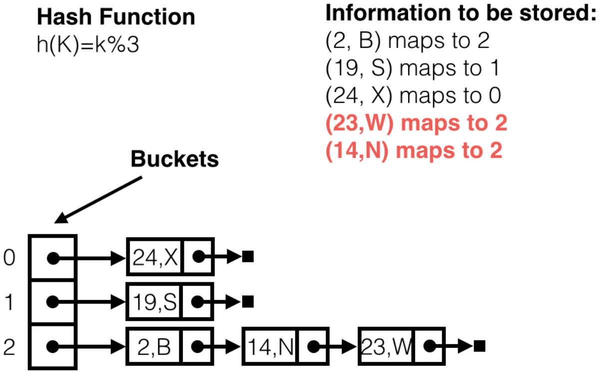


Table of Contents

Impact of a Hash Function

- A good hash function is one that maps keys in a **uniform or random manner**
- What this means is that the chance of mapping a key into any spot in the table is equally likely
  - Analogy: Tossing a ball into a slots with equal probability
- What do you think about how often a collision may take place in the table?
  - More frequent than what you think
  - Example: von Mises paradox

Selection of Hash Functions

Impact of a Poor Hash Function

- Suppose that your key consists of three characters and the table size is set to 256
- Each character is represented with 8 bits (i.e. 0 to 255)
- Suppose your hash function is $h(K) = K\%256$, then your address mapping will be biased towards the last character
- Clusters are preserved rather than dispersed

Some Options for Selecting Hash Functions

Dividing

- When double hashing is used, $h(K) = K\%M$, $p(K) = max(1, K/M)$ and M is a prime number
- $h(K)$ is the remainder and $p(K)$ is the quotient

Folding

- Key is divided into sections and these sections are added together (i.e. K=982 347 812, $h(K) = 982 + 347 + 812 = 2141$)

Middle-squaring

- Key is divided into sections and the middle section (i.e. K=982 347 812, $h(K) = 347^2$)

Truncation

- Key is divided into three sections and the last section is retained (i.e. K=982 347 812, $h(K) = 812$)

Open Addressing

Table Attributes

- When open addressing is used, the number of slots required to store M entries are reserved beforehand
- Available table locations for storage range from 0 to $M - 1$
- **Load factor $\boldsymbol{\alpha}$** $(0 \leq \alpha < 1)$: Suppose that that the table has N occupied entries

$$\alpha = \frac{N}{M}$$

- **Clustering**: sequence of adjacent occupied entries in the hash table (puddles)
- **Primary clustering**: Encountered in *linear probing* where colliding element expand the cluster
    - Once formed, puddles grow very fast (puddle formation, growth mergers) with linear probing but this is not the case for double hashing

Ensuring that Probing Sequence Covers the Entire Table

- **Linear** probe sequence will cover the entire table!
- How about **double hashing**?
- If you examine all the values $p(K)$ may take for table of size 7, 11, 13 ..., the probe sequences do cover the entire table
- These are all **prime numbers**
- It can be shown that as long as $p(K)$ and M are **relatively prime** (i.e. no common divisor except for 1) then probe sequences via double hashing will cover the entire table!
- How about setting $M = 2^k$ and forcing $p(K)$ to be odd numbers?
    - $p(K) = \{1, 3, 5, 7\}$ and $M = 7$

Table of Contents

Performance of Various Collision Resolution Techniques

Successful search refers to the key already existing in the table and unsuccessful search refers to the insertion of a new key into the table

- **Linear Probing**:

$$C_N^s \approx \frac{1}{2}(1 + \frac{1}{1-\alpha})$$
$$C_N^u \approx \frac{1}{2}(1 + (\frac{1}{1-\alpha})^2)$$

- **Double Hashing**:

$$C_N^s \approx \frac{1}{\alpha} ln(\frac{1}{1-\alpha})$$
$$C_N^u \approx (\frac{1}{1-\alpha})$$

- **Separate Chaining**:

$$C_N^s \approx 1 + \frac{1}{2}\alpha$$
$$C_N^u \approx \alpha$$

General Performance Observations

Small load factor:

- Successful search: All three techniques has the same performance
- Unsuccessful search: Separate chaining has better performance

Medium load factor:

- Successful search: Separate chaining is slightly better than the other two techniques
- Unsuccessful search: Separate chaining is the best and then double hashing

Large load factor:

- Successful search: Separate chaining is the best by a large gap
- Unsuccessful search: Separate chaining remains effective while the other two techniques degrade

General Performance Observations

What can be said in general about Hash Tables?

- If the load factor is lesser than 0.5, then the average number of comparisons ranges from 1.25 to 2
- Since the performance is **not dependent** on the size of the table n, the performance of an insertion or a search is essentially **O(1)**!!!
- If the table is guaranteed to be **half full** all the time, then the performance will remain at O(1)