Hashing as a Dictionary Implementation

Chapter 22

Contents

Contents

- A Dictionary Implementation That Uses Hashing
  - Entries in the Hash Table
  - Data Fields and Constructors
  - The Methods **`getValue`**, **`remove`**, and **`add`**
  - Iterators
- Java Class Library: The Class **`HashMap`**
- Jave Class Library: The Class **`HashSet`**

Objectives

- Describe relative efficiencies of various collision resolution techniques
- Describe hash table's load factor
- Describe rehashing and why necessary
- Use hashing to implement ADT dictionary

Efficiency of Hashing

- Observations
  - Successful retrieval or removal has same efficiency as successful search
  - Unsuccessful retrieval or removal has same efficiency as unsuccessful search
  - A successful addition has same efficiency as unsuccessful search
  - An unsuccessful addition has same efficiency as successful search

Efficiency of Hashing

- Load factor $$\lambda = \frac{Number\ of\ entries\ in\ the\ dicionary}{Number\ of\ locations\ in\ the\ hash\ table}$$
- Minimum load factor = 0
  - When dictionary is empty
  - It is never negative
- Maximum load factor
  - Depends on type of collision resolution used
  - Cannot exceed 1

Open Addressing

- Average number of collisions for *linear* probing
  - Unsuccessful search $\frac{1}{2}\left\{1 + \frac{1}{(1-\lambda)^2}\right\}$
  - Successful search $\frac{1}{2}\left\{1 + \frac{1}{(1-\lambda)}\right\}$

λ	Unsuccessful Search	Successful Search
0.1	1.1	1.1
0.3	1.5	1.2
0.5	2.5	1.5
0.7	6.1	2.2
0.9	50.5	5.5

Figure 22-1 The average number of comparisons required by a search of the hash table for given values of the load factor λ when using linear probing

Open Addressing

- Average number of collisions for *quadratic probing* or *double hashing*
  - Unsuccessful search $\frac{1}{(1-\lambda)}$
  - Successful search $\frac{1}{\lambda}\log\left(\frac{1}{1-\lambda}\right)$

λ	Unsuccessful Search	Successful Search
0.1	1.1	1.1
0.3	1.4	1.2
0.5	2.0	1.4
0.7	3.3	1.7
0.9	10.0	2.6

Figure 22-2 The average number of comparisons required by a search of the hash table for given values of the load factor λ when using either quadratic probing or double hashing

Separate Chaining

- Load factor is $\lambda = \frac{Number\ of\ entries\ in\ the\ dicionary}{Number\ of\ chains}$

- Average number of comparisons
  - Unsuccessful search λ
  - Successful search $1 + \frac{\lambda}{2}$

λ	Unsuccessful Search	Successful Search
0.1	0.1	1.1
0.3	0.3	1.2
0.5	0.5	1.3
0.7	0.7	1.4
0.9	0.9	1.5
1.1	1.1	1.6
1.3	1.3	1.7
1.5	1.5	1.8
1.7	1.7	1.9
1.9	1.9	2.0
2.0	2.0	2.0

Figure 22-3 The average number of comparisons required by a search of the hash table for given values of the load factor λ when using separate chaining

Rehashing

- When load factor gets too high
  - Resize array to a prime number at least twice former size
  - Must rehash to different locations *c % n*, based on new size, *n*, of array
- Note – this is more work than simply increasing the size of an array
  - Not a task to be done often

Comparing Schemes

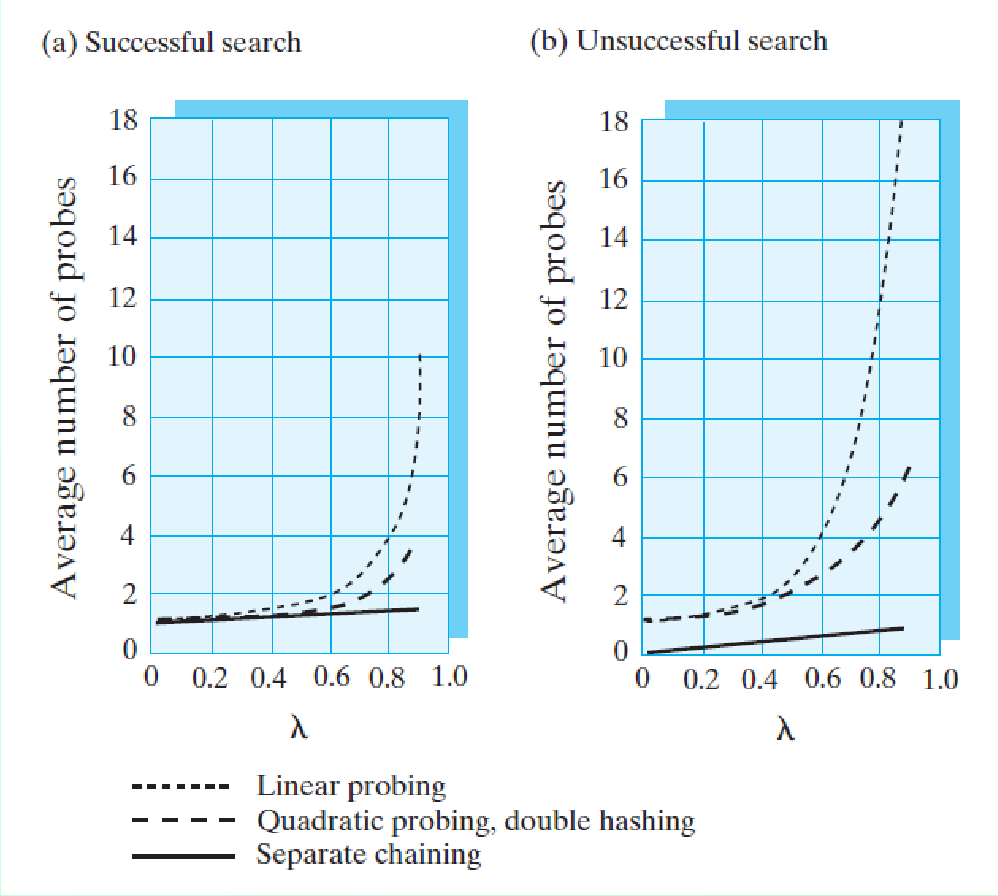

Figure 22-4 The average number of comparisons required by a search of the hash table versus the load factor λ for four collision resolution techniques when the search is
(a) successful; (b) unsuccessful

Dictionary Implementation That Uses Hashing

- We implement linear probing
  - Other open addressing strategies involve few changes
- Note source code of class **`HashedDirectory`**, Listing 22-1

Note: Code listing files must be in same folder as PowerPoint files for links to work

Dictionary Implementation That Uses Hashing

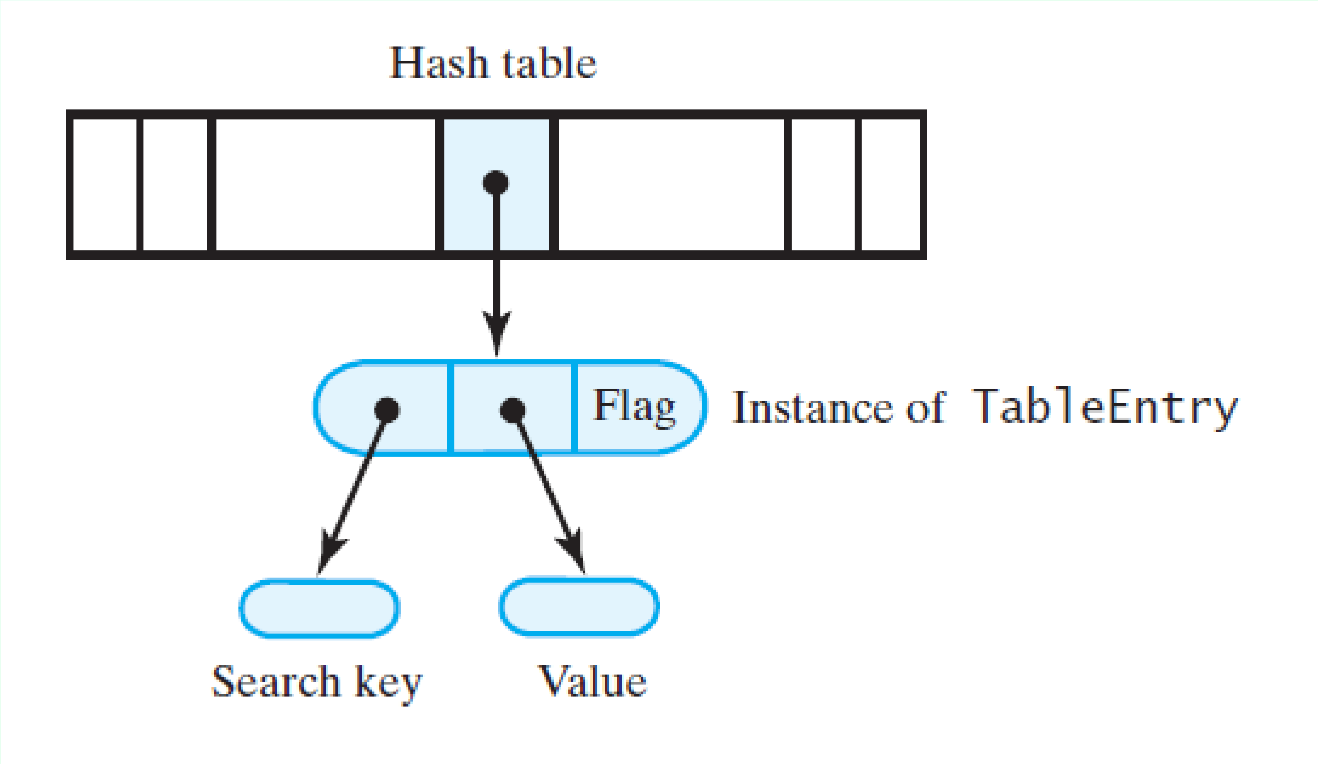

Figure 22-5 A hash table and one of its entry objects

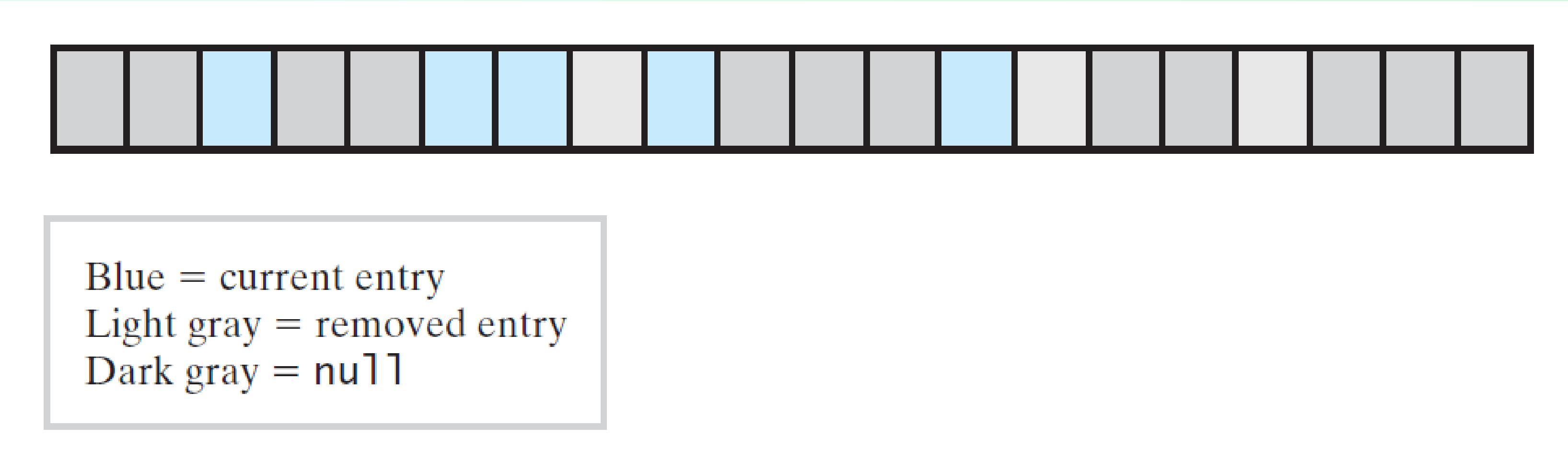

FIGURE 22-6 A hash table containing dictionary entries, removed entries, and **null** values

Java Class Library: The Class **HashMap**

- Standard package java.util contains the class **`HashMap<K, V>`**
- Table is a collection of buckets
- Constructors
  - **`public HashMap()`**
  - **`public HashMap(int initialSize)`**
  - **`public HashMap(int initialSize, float maxLoadFactor)`**
  - **`public HashMap(Map<? extends K,? extends V> table)`**

Java Class Library: The Class **`HashMap`**

- Design
  - Max λ = 0.75
  - Avoid necessity of rehashing by setting

$$Number\ of\ buckets > \frac{Max\ entries\ in\ dictionary}{\lambda_{max}}$$

Java Class Library: The Class **HashSet**

- Implements the interface **java.util.Set** of Chapter 1
  - Uses an instance of the class **HashMap** to contain entries in a set
- Constructors
  - **public HashSet()**
  - **public HashSet(int initialCapacity)**
  - **public HashSet(int initialCapacity, float loadFactor)**

End

Chapter 22

THIRD EDITION
Data Structures and Abstractions with Java™
FRANK M. CARRANO