Computer Science 331

Hash Tables with Chaining

Mike Jacobson

Department of Computer Science
University of Calgary

Lecture #19

Outline

Common Situation

We wish to use a dictionary (or mapping), under the following circumstances:

- The "universe" of possible values for keys is extremely large.
- We have a much smaller bound on the (maximal) size of the dictionary we will need to support.
- The only dictionary operations we need are
  - initialization of an empty dictionary,
  - searches for items in the dictionary,
  - insertions of new items into the dictionary,
  - deletions of items from the dictionary.

What is a Hash Table?

A **hash table** is a generalization of an ordinary array.

Features:

- Array size is generally chosen to be comparable to (perhaps, a small *multiple* of) our bound on dictionary size
- Worst-case performance is generally poor
- However, the average-case performance is extremely good — better than that of the other implementations of a dictionary we have considered!

Notation and Definitions

U: **Universe**: The set of possible values for keys

m: **Table Size**: The size of the array used to build a hash table

T: The array that is used.

h: **Hash Function**: A function

$$h : U \to \{0, 1, \ldots, m-1\}$$

used to map keys to array locations

Idea: try to store element x with key k in location $T[h(k)]$.

General Difficulty: Collisions

More terminology:

- A key k **hashes to** an array location ℓ if $h(k) = \ell$.
- A **collision** occurs if two keys k_1 and k_2 (used in the dictionary) hash to the same location, that is,

$$h(k_1) = h(k_2) \ .$$

Note: Collisions are unavoidable if the size of the dictionary is greater than the table size.

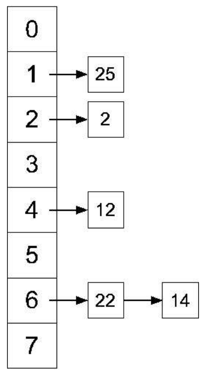
There are several different kinds of hash tables that use different ways to deal with collisions.

Collision Resolution with Chaining

In a hash table with **chaining**:

- we put all the keys (used in the dictionary) that hash to the same location ℓ into a linked list.
- For $0 \le \ell < m$, $T[\ell]$ is a pointer/reference to the head of the linked list for location ℓ.
- *Abuse of Notation:* Sometimes $T[\ell]$ will be used as the name for the above linked list (instead of a pointer to it).

Example



U: $\{1, 2, \ldots, 200\}$

m: 8

T: As shown to the left.

h: Function such that

$$h : \{1, 2, \ldots, 200\} \to \{0, 1, \ldots, 7\},$$

eg. $h(k) = k \bmod 8$ for $k \in U$.

Dictionary Operations

Search for an item with key k;

- Search for k in the linked list $T[h(k)]$.

Insertion of an item I with key k;

- Search for k in the linked list $T[h(k)]$.
- If the search was unsuccessful, insert I onto the **front** of this linked list.

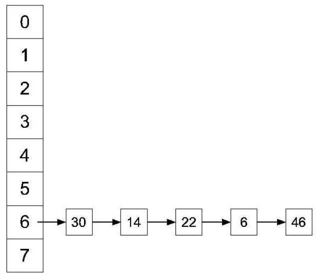
Deletion of an item with key k;

- Perform a deletion of an item with key k from the linked list $T[h(k)]$.

Worst-Case Analysis

Cost of an operation involving a key k is essentially the cost the same operation involving k, using the linked list $T[h(k)]$.

Problem:



It is possible for *all* dictionary items to be part of this linked list!

Average Case Analysis

We will consider the average cost of dictionary operations when a hash table is used to represent a dictionary with n elements.

The average cost of these operations depends on

- the likelihood of each kind of operation, and
- the *shape* of the hash table.

The shape of the hash table only depends on the *locations* to which keys are hashed — not on the values of the keys, themselves.

Simple Uniform Hashing

Assumption: **Simple Uniform Hashing:**

- Each key is hashed to location ℓ with the same probability, $\frac{1}{m}$, for $0 \leq \ell < m$.
- Furthermore, each key is hashed to a location *independently* of where any other key is hashed to.

That is: If $k_1, k_2, \ldots, k_n$ are the keys in the dictionary then

$$h(k_1) = \ell_1 \quad \text{and} \quad h(k_2) = \ell_2 \quad \text{and} \quad \cdots \quad \text{and} \quad h(k_n) = \ell_n$$

with probability $(1/m)^n$, for *each* choice of locations $\ell_1, \ell_2, \ldots, \ell_n$.

Load Factor

Load Factor of T: The average λ of the lengths of the linked lists (or "chains") $T[0], T[1], \dots, T[m-1]$.

Claim:

$\lambda = n/m$ (hash table has m locations, dictionary has n elements).

Proof.

Suppose $T[i]$ has length n_i for $0 \le i < m$.

- Then $\lambda = \frac{1}{m}(n_0 + n_1 + \cdots + n_{m-1})$ (by definition).
- However, since each key is hashed to exactly one location, and there are n keys, $n_0 + n_1 + \cdots + n_{m-1} = n$, so $\lambda = n/m$.

□

Average Case Analysis: Summary

Expected Numbers of Comparisons Required:

Unsuccessful Search for a key k :

- Assumption: No additional assumptions required.
- Expected Cost:

Successful Search:

- Assumption: Search for each key with probability $\frac{1}{n}$
- Expected Cost:

Insertion of **New** Element:

Deletion of Existing Element:

Concepts from Probability Theory

Sample Space: Finite set S of *events* in which we are interested.

Probability Distribution: Function $\Pr : S \to \mathbb{R}$ such that

$$0 \le \Pr(s) \le 1 \text{ for all } s \in S \quad \text{and} \quad \sum_{s \in S} \Pr(s) = 1 \ .$$

Random Variable: A real valued function of S. That is, a function $X : S \to \mathbb{R}$.

Expected Value of a Random Variable: The *expected value* of a random variable X is

$$\mathrm{E}[X] = \sum_{s \in S} \Pr(s) \cdot X(s) \ .$$

Application to Hash Tables

If we are interested in analyzing the shape of the hash table including keys $k_1, k_2, \dots, k_n$ then the **sample space** S includes n-tuples

$$(\ell_1, \ell_2, \dots, \ell_n)$$

of *locations* of these keys (in the hash table).

The "event" $(\ell_1, \ell_2, \dots, \ell_n)$ occurs if

$$h(k_1) = \ell_1 \quad \text{and} \quad h(k_2) = \ell_2 \quad \text{and} \cdots \text{and} \quad h(k_n) = \ell_n .$$

One random variable of interest: n_i, the length of $T[i]$

More About Random Variables

Suppose that a random variable X can only have values $0, 1, 2, \ldots, t$.

Notation: For each integer i, write $\Pr(X = i) = \sum_{\substack{s \in S \\ X(s)=i}} \Pr(s)$.

Claim:

If the only possible values for X are $0, 1, 2, \ldots, t$ then

$$E[X] = \sum_{i=0}^{t} i \cdot Pr(X = i).$$

Proof.

Exercise. □

Linearity of Expectation

If a random variable X is a sum of t other random variables,

$$X = X_1 + X_2 + \cdots + X_t,$$

then

$$\begin{aligned} \mathrm{E}[X] &= \mathrm{E}[X_1 + X_2 + \cdots + X_t] \\ &= \mathrm{E}[X_1] + \mathrm{E}[X_2] + \cdots + \mathrm{E}[X_t] \ . \end{aligned}$$

Application: We can find the expected value of X by finding the expected values of each of $X_1, X_2, \ldots, X_t$ and then adding these together.

Computing the Average Length Another Way

Consider the random variable n_i (length of list $T[i]$):

- $n_i = X_{i,1} + X_{i,2} + \ldots, + X_{i,n}$ where

$$X_{i,j} = \begin{cases} 1 & \text{if } h(k_j) = i \\ 0 & \text{if } h(k_j) \neq i, \end{cases}$$

  and $k_1, k_2, \ldots, k_n$ are the keys in the dictionary.
- Since $X_{i,j} \in \{0, 1\}$, $\mathrm{E}[X_{i,j}] = \Pr(X_{i,j} = 1) = 1/m$ by the *Simple Uniform Hashing* assumption.
- *Linearity of Expectation* can be used to show that

$$\mathrm{E}[n_i] = \mathrm{E}\left[\sum_{j=1}^{n} X_{i,j}\right] = \sum_{j=1}^{n} \mathrm{E}[X_{i,j}] = \frac{n}{m}.$$

Expected Cost of an Unsuccessful Search

Suppose that:

- $k_1, k_2, \ldots, k_n$ are keys in the dictionary, and
- we perform an unsuccessful search for a key k.

If we do not include comparisons to the null pointer, then the number of comparisons for an unsuccessful search for k is

$$X_1 + X_2 + \cdots + X_n$$

where

$$X_i = \begin{cases} 1 & \text{if } h(k) = h(k_i) \\ 0 & \text{if } h(k) \neq h(k_i) \end{cases}$$

Expected Cost of an Unsuccessful Search

The *Uniform Simple Hashing* assumption can be used to show that

$$\mathrm{E}[X_i] = \tfrac{1}{m},$$

no matter what value $h(k)$ has.

Linearity of Expectation can be used to show that the expected number of comparisons is

$$\mathrm{E}[X_1] + \mathrm{E}[X_2] + \cdots + \mathrm{E}[X_n] = \tfrac{n}{m} = \lambda.$$

Expected Cost of a Successful Search for k_i

Suppose keys were introduced in order

$$k_1, k_2, \ldots, k_n.$$

Consider a *successful search* for k_i.

Note: k_i appears *before* any of

$$k_1, k_2, \ldots, k_{i-1}$$

that are in the same linked list, and *after* any of

$$k_{i+1}, k_{i+2}, \ldots, k_n$$

that are in the same linked list.

Expected Cost of a Successful Search for k_i

Number of comparisons to search for k_i is, therefore,

$$Y_i = 1 + X_{i+1} + X_{i+2} + \cdots + X_n$$

where

$$X_j = \begin{cases} 1 & \text{if } h(k_j) = h(k_i) \\ 0 & \text{if } h(k_j) \neq h(k_i) \end{cases}$$

Under the *Uniform Simple Hashing* assumption, $\mathrm{E}[X_j] = \frac{1}{m}$.

By *Linearity of Expectations*,

$$\mathrm{E}[Y_i] = 1 + (n-i) \cdot \left(\tfrac{1}{m}\right) = 1 + \tfrac{n-i}{m}.$$

Expected Cost of a Successful Search

Additional Assumption: We search for k_i with probability $\frac{1}{n}$.

One can show that the expected cost of a successful search is

$$\tfrac{1}{n}\left(\mathrm{E}[Y_1] + \mathrm{E}[Y_2] + \cdots + \mathrm{E}[Y_n]\right) = 1 + \tfrac{\lambda}{2} - \tfrac{\lambda}{2n},$$

under these assumptions, as claimed.

A Variation

Suppose that the universe U is ordered, so that we can also ask whether $k_1 \leq k_2$ for any two keys k_1 and k_2.

In this case we could maintain the keys in each of our lists in *sorted order*.

- *Worst case* costs for operations are unchanged.
- *Expected cost for a successful search* using the usual assumptions is also unchanged
- However, the expected cost for an *unsuccessful* search is somewhat reduced — because we can use the list ordering to end an unsuccessful search a bit earlier.
- The overhead to maintain sorted order is insignificant, so this optimization is worthwhile.

References

- Textbook, Section 9.2 — description of hash tables as a data structure for implementing Java's `Map` interfaces (recall that `Map` is similar to `Dictionary`).
- *Introduction to Algorithms*, Section 11.2 — additional information about hash tables with chaining (including much of the material in these notes)
- *Introduction to Algorithms*, Appendix C — more information about useful concepts from probability and statistics