Hash tables: A generalization of notion of an ordinary array

Hash table

Chapter 12 Sections 12.1, 12.2, 12.3

Hash Table

CSE 310: Data Structures and Algorithms

Chapter 12, Section 12.2, Sections 12.1, 12.2, 12.3

Professor Phil S. Room 104
Engineering Hall (Shu-se-kh-Cola) Center for Academic Excellence/203-310/

CourseContactMail: Tel: (631) 472-9411

Worse, when 1/2 or more of keys is small

When no element in set with key = x/2, TX = NT

Show 8 problems to the element in set with key = x

Assumption: no two elements have the same key

hash function from x to m = [x]m
[1, 2, ..., m - 1]
Search time remains in $O(1)$, however, in average.

Hash Table

- Use a hash table
- Do not use a direct-access table
- When $n > h$, loss of space wasted
- When $n < h$, loss of space is impractical

Direct-access Table

Dictionary operations

Flowchart:

Open addressing

- Chaining

Techniques

- When $n < h$, no collection is impossible
- $h$ must be determined given $n$, $n$ is finite
- Where $h$ appears to be random: avoid coalescence collisions

Avoiding collisions

Two keys may hash to the same slot (when $h$ is not infinite)

Hash Table: stores element with key $k$ in slot $h(k)$

Direct-access Table: stores element with key $k$ in slot $k$
Dictionary operations in collision resolution by chaining

To analyze hashing with chaining, examine search, delete, and search have same running time.

Search: best-case: O(1) if x is key in head of chain.
Worst-case: O(n) if x is key in last element of chain.

Delete: has more complex analysis (as delete can cause new headers to be created).

Collision resolution by chaining

When two elements share a hash value, one element takes the next position in its chain.

For each element in the list of all elements that hash to k, if the next position is available, move that element to the next position in its chain.
Theorem 12.1: In a hash table in which collisions are resolved
by chaining an unsuccessful search takes time $\Theta(1 + \alpha)$ on the average...

Theorem 12.2: ...a successful search takes time $\Theta(1 + \alpha)$, on the average...

Proof (cont.)

The expected number of elements examined during a successful search is $1 + \alpha$ number of elements was inserted.
Thus, we take the average over the $n$ searches in table 1-

Expected length of list is $(i-1)/m$. Expected number of elements examined is $\sum_{j=0}^{m-1} \frac{1}{m} (n-1-j)$

Thus, total time required for a successful search is $\Theta((2 + \alpha) - \frac{1}{m}) = \Theta(1 + \alpha)$

Assume: Simple uniform hashing.
Hash Functions

However, $f$ is usually unknown.

For $f = 0, 1, \ldots, m - 1$, $m! / f(m) = \prod_{i=0}^{m-1} (i+1)$.

Definition: Single hash function.

Assume each key is drawn independently from $\mathcal{L}$ with probability $1/k$.

A good hash function satisfies (approximately) the single hash function hypothesis:

- Can be computed quickly.
- A good hash function.

Division method

Example: $m = 12$ and $n = 100 \equiv 100 \mod m$.

$\mathcal{O}(1)$

Example: $m = 11$ and $n = 1011$ in the ASCII character set.

Example: $p$ can be represented as $111_{10} = 1101_{16}$.

ASCII characters can be interpreted by interpreting each character as such.

Assumption: Keys are random numbers in $\mathcal{L}$.

1. Division method

2. Multiplication method

Interpretation of Theorem 2.1 and 2.2

All division operations can be completed in $O(n)$.

Operation in $O(1)$, definition in $O(1)$ (exponentiation here).

Since dividing blocks is expensive, divide $u$ by $m$, the number of blocks, in a least significant-to-most significant order.

Assume $\gamma$ is a random number in $\mathcal{L}$, $\gamma \in \mathcal{L}$.
There are many ways to choose a hash function that is random independent of any hash function that is random independent of the keys to be hashed.

### Universal Hashing

We can choose m to make this function easy to implement.

1. Choose m to be any prime
2. Choose p = m - 1
3. Choose h(0) = 0 and h(1) = 1

Two steps:

1. Multiply h(x) by a constant (e.g., k > 1), and extract the low order m bits.
2. Take the low order m bits of the result.

5. Choose a prime p such that p - 1 = 2q, where q is a prime.

### Pigeonhole Principle

No matter what keys are chosen by adversary

The adversary is allowed unlimited hashing

5. Choose a hash function that is random independent of any hash function that is random independent of the keys to be hashed.

We can choose m to make this function easy to implement.

1. Choose m to be any prime
2. Choose p = m - 1
3. Choose h(0) = 0 and h(1) = 1

Two steps:

1. Multiply h(x) by a constant (e.g., k > 1), and extract the low order m bits.
2. Take the low order m bits of the result.

5. Choose a prime p such that p - 1 = 2q, where q is a prime.

### Pigeonhole Principle

No matter what keys are chosen by adversary

The adversary is allowed unlimited hashing

5. Choose a hash function that is random independent of any hash function that is random independent of the keys to be hashed.