Chapter 11
Hashing and the Table ADT

Outline

Motivation

- Tables
- O(1) performance!

Hashing

- Open addressing and separate chaining
- Collisions, load factors, and clusters

Hash Functions

- Covering the table, primary clustering
- Performance analysis

Table ADT

- Table entry as ordered pair (K, I)
  - K: a unique key for each entry in table
  - I: associated information with key
- Operations on table
  - Table searching given a search key, K
  - Retrieve or update K's information
  - Delete entry
  - Enumerate all entries in some order
- Representations
  - Arrays, linked-lists, AVL tree
  - Hashing can provide significant performance improvement

Examples

- Clemson student ID's: 9 digits
  - There are 10^9 (1 billion) numbers
  - Enrollment: ~17,000 students, or 0.0017%
  - Organize as binary search tree
    - O(log n) search, insert, delete
    - $C_n = 2\log_2 17{,}000 - 3 \ \approx 25$ comparisons
  - We will show that if we use a table of size 20,000
    - ~2.2 comparisons on average
    - Ten times improvement
- Clemson login names: $\sum_{i=1}^{7} 36^i$ ~ 80 billion

Hash Function

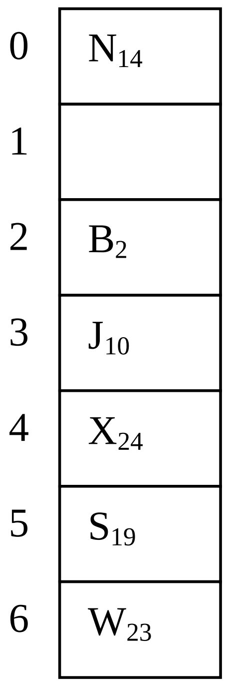
- A mapping of keys into table addresses
- For table entry (K, I) define h(K): translates K into an address in the table
- Collision when $K_1 \neq K_2$ but $h(K_1) = h(K_2)$
  - An ideal hash function never has collisions
  - We will find collisions are common
- Collision resolution policies
  - Chaining
  - Open addressing
  - Buckets

Simple Example

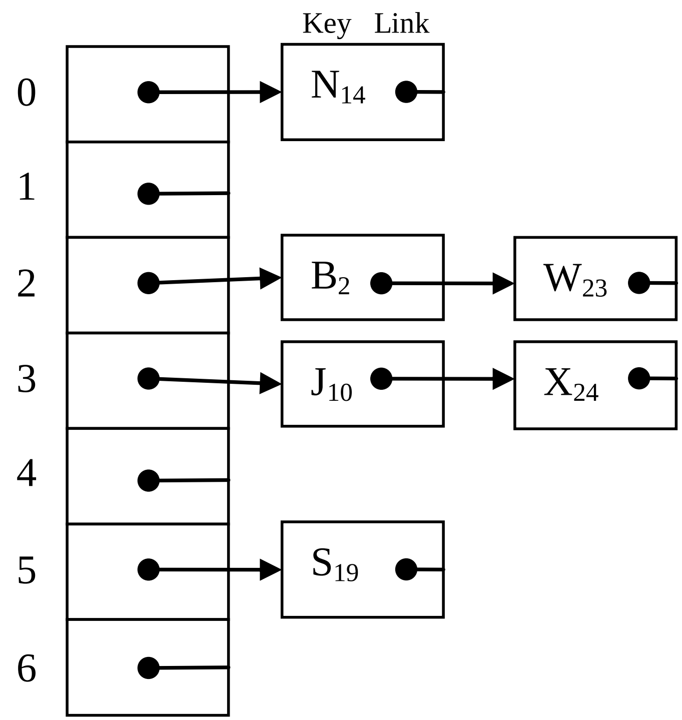
- Table size: 7
- Key is single letter L_n : n is letter's position in alphabetical order
- Hash function $h(L_n) = n\%7$
- Probe decrement $p(L_n)$
- Resolve collisions using open addressing
  - Linear probing (leads to clusters) $p(L_n) = 1$
  - Double hashing example: $p(L_n) = \max(1, \frac{n}{7})$

Collision Resolution

Open addressing



Separate chaining



Collisions, Load Factors, and Clusters

- Collisions are common
  - Von Mises Birthday Argument
    - If there are 22 or more people in a room chances are greater than 50% that two or more will have the same birthday
    - 22/365 = 0.06 → the table is 6% full
  - While collision are frequent, there should be an empty table location that is "nearby" or can be found quickly
- Load factor: α
  - Table size: M
  - Number of occupied entries: N

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

Developing Algorithms

- Insert: program 11.16
- Search: program 11.17
- Standish uses a fixed sized table (in MP you will `malloc()` table!)

```
typedef int KeyType
typedef struct info_type_tag {
      // some members for information
      // associated with search keys
} InfoType;
```

```
typedef struct table_entry_tag {
      KeyType   Key;
      InfoType  Info;
} TableEntry;
typedef TableEntry Table[M];
```

Initialize Table

```
Table T;
for(j=0; j<M; j++)
      T[j].Key = EmptyKey;
```

Developing Algorithms

- Airport code example with table size M=11
  - $h(K) = \text{Base26ValueOf}(K)\%11$
  - $p(K) = \max\left(1, \left(\frac{\text{Base26ValueOf}(K)}{11}\right)\%11\right)$
- For 3-letter Airport Codes
  - $K = X_2X_1X_0$
  - $\text{Base26ValueOf}(K) = X_2 26^2 + X_1 26^1 + X_0 26^0$
  - A=0, B=1, C=2, …, Z=25
  - K=DCL (D=3, C=2, L=11)

$$\boxed{3 \cdot 26^2 + 2 \cdot 26^1 + 11 \cdot 26^0 = 2091}$$

$$\boxed{2091 = 190 \cdot 11 + 1 \ \rightarrow h(DCL) = 1}$$

$$\boxed{190 = 17 \cdot 11 + 3 \ \rightarrow p(DCL) = 3}$$

Key	h(K)	p(K) double	p(K) linear
PHL	4	4	1
ORY	8	1	1
GCM	6	1	1
HKG	4	3	1
GLA	8	9	1
AKL	7	2	1
FRA	5	6	1
LAX	1	7	1
DCA	1	2	1

Probe Sequence

- What is worst case scenario for probing?
  - All keys hash to same table location
  - We need to probe all table locations
- Probe sequence in range 1, 2, …, M-1
- The location visited by the probe sequence is $[h(K) - i \times p(K)]\%M$ for $i = 0, 1, 2, \ldots, M - 1$
- Does the probe sequence cover all address in the table exactly once?

Example Probe Decrements

- Choose *M* to be a prime and probe decrement as integer in range 1:*M*-1
- Choose *M* to be a power of two and probe decrement as any odd integer in range 1:*M*-1

$p(K)$ must be relatively prime to the table size *M*

Performance Analysis

- C_n : Average number of probe addresses examined during a successful search
- C'_n : same for unsuccessful search
  - Notice C'_n is also the number of probes required during the insertion of a new key (but we have not considered deletions yet)

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

- Linear probing

$$C_n = \frac{1}{2}\left(1 + \frac{1}{1-\alpha}\right) \qquad C'_n = \frac{1}{2}\left(1 + \left(\frac{1}{1-\alpha}\right)^2\right)$$

Formulas for linear probing apply if $\alpha \leq 0.7$

Performance Analysis (con't)

- Open addressing with double hashing

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

$$C_n = \frac{1}{2}\ln\left(\frac{1}{1-\alpha}\right) \qquad C'_n = \frac{1}{1-\alpha}$$

- Separate chaining

$$C_n = 1 + \frac{\alpha}{2} \qquad C'_n = \alpha$$

Performance Comparisons

Representation	Initialize	Search, Retrieve, Update	Insert	Delete	Enumerate
Sorted array of structs	O(1)	O(log n)	O(n)	O(n)	O(n)
AVL tree of structs	O(1)	O(log n)	O(log n)	O(log n)	O(n)
Hash table	O(n)	O(1)	O(1)	O(1)	O(n log n)

Why can we claim hash table searches occur in time O(1)?

Keep table less than half full: $\alpha < 0.5$

Rehash into new larger table when needed

Double hashing $\rightarrow C_n = 1.39$ and $C_n' = 2.0$

Deletions Can Be Troublesome

- For separate chaining no problem
- For open addressing
  - If delete by leaving table entry with an empty key, then destroy the validity of searches
  - So, mark table entry to be deleted with special key
    - Search: probe past entries marked as deleted, treating them as if not deleted
    - Insertion: insert new entry in place of any entry marked as deleted
  - Table becomes clogged with entries marked as deleted
    - Makes searches slower
    - If problem, rehash the table keeping only keys not marked as deleted

Design of Hash Functions

- Challenge: need function for long keys (often strings)
- Examples of poor functions
- Methods if table size is a prime number
- Folding
  - Additive, XOR, rotating
- Middle squaring
- Truncation (almost always poor)
- Extensive discussion of art of hashing