Symbol Table and Hashing

- A (symbol) table is a set of table entries, (K,V)
- Each entry contains:
  - a unique key, K, and
  - a value (information), V
- Each key uniquely identifies its entry
- Table searching:
  - Given: a search key, K
  - Find: the table entry, (K,V)

Symbol Table and Hashing

- Once the entry (K,V) is found, its value V, may be updated, it may be retrieved, or the entire entry, (K,V) , may be removed from the table
- If no entry with key K exists in the table, a new table entry having K as its key may be inserted in the table
- Hashing is a technique of storing values in the tables and searching for them in linear, $O(n)$, worst-case and extremely fast, $O(1)$, average-case time

Basic Features of Hashing

- Hashing computes an integer, called the hash code, for each object
- The computation, called the hash function, $h(K)$, maps objects (e.g., keys K) to the array indices (e.g., 0, 1, …, $i_{\max}$)
- An object having a key value K should be stored at location $h(K)$, and the hash function must always return a valid index for the array

Basic Features of Hashing

- A perfect hash function produces a different index value for every key. But such a function cannot be always found.
- **Collision:** if two distinct keys, $K_1 \neq K_2$, map to the same table address, $h(K_1) = h(K_2)$
- **Collision resolution policy:** how to find additional storage in which to store one of the collided table entries

How Common Are Collisions?

- **Von Mises Birthday Paradox:**

  if there are more than 23 people in a room, the chance is greater than 50% that two or more of them will have the same birthday
- Thus, in the table that is only 6.3% full (since $23/365 = 0.063$) there is better than 50–50 chance of a collision!

How Common Are Collisions?

- **Probability $Q_N(n)$ of no collision** (that is, that none of the n items collides, being randomly tossed into a table with N slots):

$$Q_N(1) = 1 \equiv \frac{N}{N}; \quad Q_N(2) = Q_N(1)\frac{N-1}{N} \equiv \frac{N(N-1)}{N^2};$$

$$Q_N(3) = Q_N(2)\frac{N-2}{N} \equiv \frac{N(N-1)(N-2)}{N^3}; \quad \ldots$$

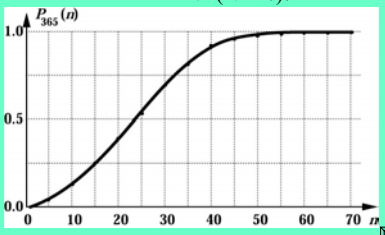
$$Q_N(n) = Q_N(n-1)\frac{N-n+1}{N} \equiv \frac{N(N-1)\ldots(N-n+1)}{N^n}$$

The University of Auckland

Probability $P_N(n)$ of One or More Collisions

$$P_N(n) = 1 - Q_N(n) = 1 - \frac{N!}{N^n(N-n)!}$$

n	%	$P_{365}(n)$
10	2.7	0.1169
20	5.5	0.4114
30	8.2	0.7063
40	11.0	0.8912
50	13.7	0.9704
60	16.4	0.9941



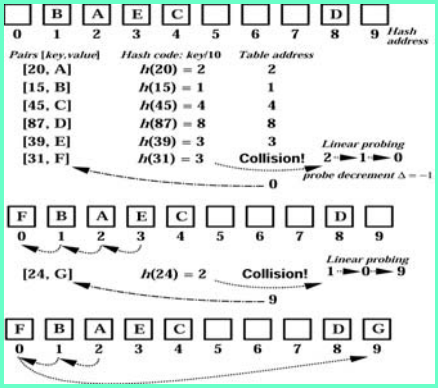
The University of Auckland

Open Addressing with Linear Probing (OALP)

- The simplest collision resolution policy:
  - to successively search for the first empty entry at a lower location
  - if no such entry, then ``wrap around'' the table
- **Drawbacks:** clustering of keys in the table

The University of Auckland

OALP example: $n = 5..7$ $N = 10$



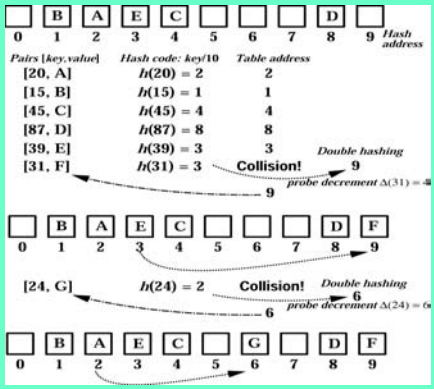
The University of Auckland

Open Addressing with Double Hashing (OADH)

- Better collision resolution policy reducing the likelihood of clustering:
  - to hash the collided key again using a different hash function and
  - to use the result of the second hashing as an increment for probing table locations (including wraparound)

The University of Auckland

OADH example: $n = 5..7$ $N = 10$



The University of Auckland

Two More Collision Resolution Techniques

- Open addressing has a problem when significant number of items need to be deleted as logically deleted items must remain in the table until the table can be reorganised
- Two techniques to attenuate this drawback:
  - Chaining
  - Hash bucket

Chaining and Hash Bucket

- **Chaining:** all keys collided at a single hash address are placed on a linked list, or chain, started at that address
- **Hash bucket:** a big hash table is divided into a number of small sub-tables, or buckets
  - the hush function maps a key into one of the buckets
  - the keys are stored in each bucket sequentially in increasing order

Universal Classes of Hash Functions

- Universal hashing: a random choice of the hash function from a large class of hash functions in order to avoid bad performance on certain sets of input
- Let **K**, N, and **H** be a key set, a size of the range of the hash function, and a class of functions that map **K** to $0,\ldots,N-1$, respectively. Then **H** is universal if, for any distinct $k,\kappa \in \mathbf{K}$, it holds that $\left|h \in \mathbf{H} : h(k) = h(\kappa)\right| / \left|\mathbf{H}\right| \le 1/N$
- **H** is a universal class if no pair of distinct keys collide under more than $1/N$ of the functions in the class

Choosing a hash function

- Four basic methods: division, folding, middle-squaring, and truncation
- **Division:**
  - choose a prime number as the table size N
  - convert keys, K, into integers
  - use the remainder $h(K) = K \bmod N$ as a hash value of the key K
  - find a double hashing decrement using the quotient,

$$\Delta K = \max\{1, (K/N) \bmod N\}$$

Choosing a hash function

- **Folding:**
  - divide the integer key, K, into sections
  - add, subtract, and/or multiply them together for combining into the final value, $h(K)$
- Example:

  K=013402122 → sections 013, 402, 122 → $h(K) = 013 + 402 + 122 = 537$

Choosing a hash function

- **Middle-squaring:**
  - choose a middle section of the integer key, K
  - square the chosen section
  - use a middle section of the result as $h(K)$
- Example:

  $K = 013402122$ → middle: 402 → 402^2=161404 → middle: $h(K) = 6140$

Choosing a hash function

- **Truncation:**
  - delete part of the key, K
  - use the remaining digits (bits, characters) as $h(K)$
- Example:

  K=013402122 → last 3 digits: $h(K) = 122$
- Notice that truncation does not spread keys uniformly into the table; thus it is often used in conjunction with other methods

Universal Class by Division

- **Theorem** (universal class of hash functions by division):
  - Let the size of a key set, **K**, be a prime number:
    $$|\mathbf{K}| = M$$
  - Let the members of **K** be regarded as the integers $\{0,\ldots,M-1\}$
  - For any numbers $a \in \{1,\ldots,M\text{-}1\}$; $b \in \{0,\ldots,M-1\}$ let
    $$h_{a,b}(k) = ((a \cdot k + b) \bmod M) \bmod N$$

Universal Class by Division

- Then $\mathbf{H} = \{h_{a,b}: \ 1 \le a < M \text{ and } 0 \le b < M\}$ is a universal class
- **Proof: [optional: see in the Coursebook…]**
- In practice:
  - let M be the next prime number larger than the size of the key set
  - Then choose randomly a and b such that $a > 0$ and use the hash function $h_{a,b}(k)$

Efficiency of Search in Hash Tables

- **Load factor** λ: if a table of size N has exactly M occupied entries, then $\lambda = M/N$
- Average numbers of probe addresses examined for a successful (S_λ) and unsuccessful (U_λ) search:

	OALP: $\lambda < 0.7$	OADH: $\lambda < 0.7$	SC
$S_\lambda \approx$	$0.5(1+1/(1-\lambda))$	$(1/\lambda)\ln(1/(1-\lambda))$	$1+\lambda/2$
$U_\lambda \approx$	$0.5(1+(1/(1-\lambda))^2)$	$1/(1-\lambda)$	λ

SC – separate chaining; λ may be higher than 1

Efficiency of Search: S_λ

λ ($N = 997$)	SC; 3 trials	OALP; 50 trials	OADH; 50 trials
0.10	1.05/1.04	1.06/1.05	1.05/1.05
0.25	1.12/1.12	1.17/1.16	1.15/1.15
0.50	1.25/1.25	1.50/1.46	1.39/1.37
0.75	1.37/1.37	2.50/2.42	1.85/1.85
0.90	1.45/1.44	5.50/4.94	2.56/2.63
0.99	1.49/1.49	50.5/16.4	4.65/4.79

Theoretical / average measured experimental results

Efficiency of Search: U_λ

λ ($N = 997$)	SC; 3 trials	OALP; 50 trials	OADH; 50 trials
0.10	0.10/0.10	1.12/1.11	1.11/1.11
0.25	0.25/0.21	1.39/1.37	1.33/1.33
0.50	0.50/0.47	2.50/2.38	2.00/2.01
0.75	0.75/0.80	8.50/8.36	4.00/4.10
0.90	0.90/0.93	50.5/39.1	10.0/10.9
0.99	0.99/0.97	5000/360.9	100.0/98.5

Theoretical / average measured experimental results

Table ADT Representations: Comparative Performance

Operation	Representation		
	Sorted array	AVL tree	Hash table
Initialize	O(N)	O(1)	O(N)
Is full?	O(1)	O(1)	O(1)
Search*)	O(log N)	O(log N)	O(1)
Insert	O(N)	O(log N)	O(1)
Delete	O(N)	O(log N)	O(1)
Enumerate	O(N)	O(N)	O(N log N)**)

*) also: **Retrieve, Update** **)To enumerate a hash table, entries must first be sorted in ascending order of keys that takes O(N log N) time