Symbol Table and Hashing

- (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 entry with k as its key may be inserted to the table
- **Hashing:**
  - 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 is called the hash function, $h(k)$
  - It maps objects (e.g., keys k) to the array indices (e.g., $0, 1, \ldots, i_{\max}$)
- An object with a key k has to be stored at location $h(k)$
  - The hash function must always return a valid index for the array

Basic Features of Hashing

- **Perfect hash function** $\rightarrow$ 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 to store one of the collided table entries
- **Load factor** λ – fraction of the already occupied entries (m occupied entries in the table of size $n \rightarrow \lambda = {}^m/_n$)

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

- In the only 6.3% full table (since $23/365 = 0.063$) there is better than 50% chance of a collision!
  - Therefore: 50% chance of collision if $\lambda = 0.063$

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:

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

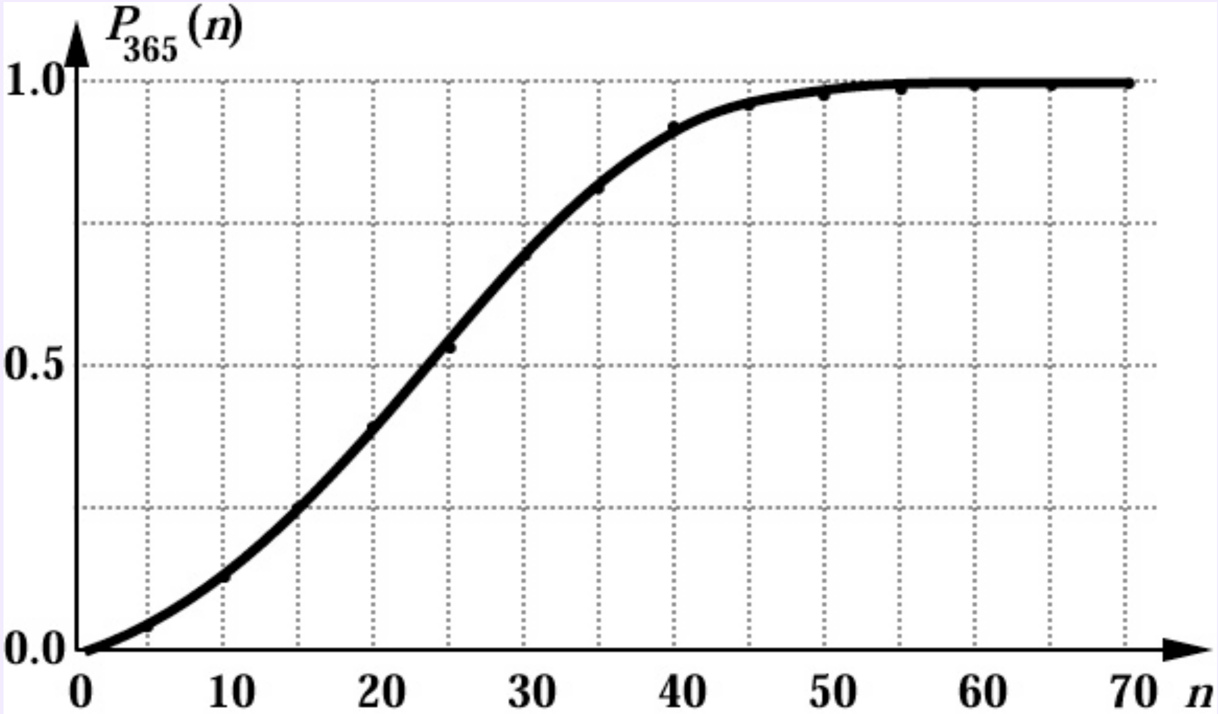
$$Q_N(3) = Q_N(2)\frac{N-2}{N} \equiv \frac{N(N-1)(N-2)}{N^3}; \qquad \ldots$$

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

Probability $\mathrm{P}_N(n)$ of One or More Collisions

$$\mathrm{P}_N(n) = 1 - \mathrm{Q}_N(n) = 1 - \frac{N!}{N^n (N-n)!}$$

n	%	$\mathrm{P}_{365}(n)$
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



Open Addressing with Linear Probing (OALP)

- The simplest collision resolution policy:
  - Successive search for the first empty entry at a lower location
  - If no such entry, then "wrap around" the table
- **Lemma 3.33**: The average number of probes for successful, $T_{\text{ss}}(\lambda)$, and unsuccessful, $T_{\text{us}}(\lambda)$, search in a hash table with load factor $\lambda = {}^{m}/_{n}$ is, respectively,

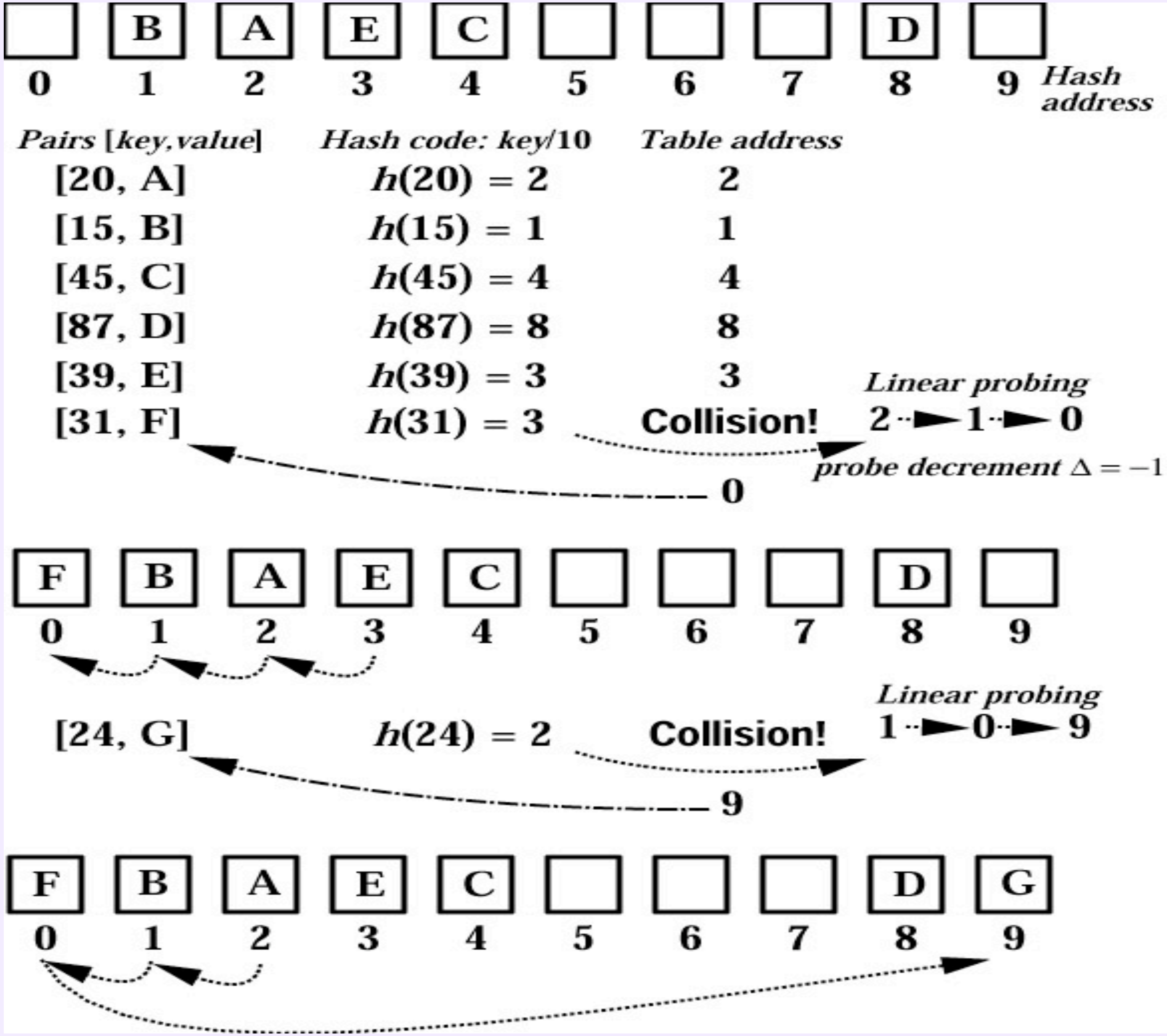
$$T_{\text{ss}}(\lambda) = 0.5\left(1 + \frac{1}{1-\lambda}\right) \quad \text{and} \quad T_{\text{us}}(\lambda) = 0.5\left(1 + \left(\frac{1}{1-\lambda}\right)^2\right)$$

- **Drawbacks:** clustering of keys in the table

OALP example:

$n = 5..7$

$N = 10$



Open Addressing with Double Hashing (OADH)

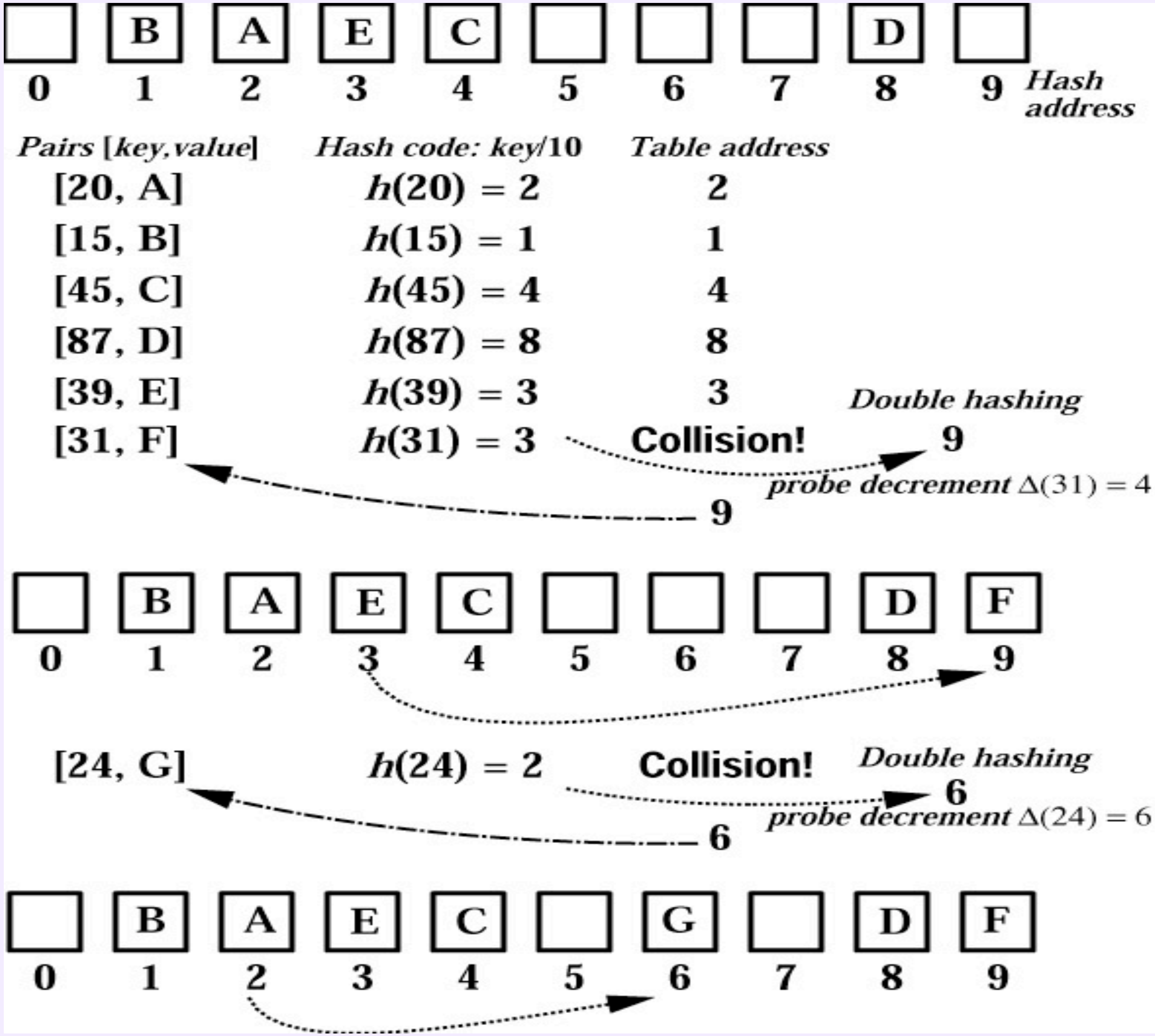
- Better collision resolution policy reducing the clustering:
  - hash the collided key again with a different hash function
  - use the result of the second hashing as an increment for probing table locations (including wraparound)
- **Lemma 3.35**: Assuming that OADH provides nearly uniform hashing, the average number of probes for successful, $T_{ss}(\lambda)$, and unsuccessful, $T_{us}(\lambda)$, search is, respectively,

$$T_{ss}(\lambda) = \frac{1}{\lambda}\ln\left(\frac{1}{1-\lambda}\right) \quad \text{and} \quad T_{us}(\lambda) = \frac{1}{1-\lambda}$$

OADH example:

$n = 5..7$

$N = 10$



Two More Collision Resolution Techniques

- Open addressing has a problem if significant number of items need to be deleted:
  - Logically deleted items must remain in the table until the table can be re-organised
- 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

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 k
  - get the 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, $h(k)$

  Ex.: $k = 013402122 \rightarrow 013, 402, 122 \rightarrow h(k) = 013 + 402 + 122 = 537$
- **Middle-squaring:**
  - choose a middle section of the integer key, k
  - square the chosen section
  - use a middle section of the result as $h(k)$

  Ex.: $k = 013402122 \rightarrow$ mid: $402 \rightarrow 402^2=161404 \rightarrow$ mid: $h(k) = 6140$

Choosing a hash function

Truncation:

- delete part of the key, k
- use the remaining digits (bits, characters) as $h(k)$

Example:

$k = 013402122 \rightarrow$ 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

Efficiency of Search in Hash Tables

Load factor λ: if a table of size n has exactly m occupied entries, then $\ddot{e} = m/n$

- Average numbers of probe addresses examined for a successful ($T_{ss}(\lambda)$) and unsuccessful ($T_{us}(\lambda)$) search:

	OALP: $\lambda < 0.7$	OADH: $\lambda < 0.7$	SC
$T_{ss}(\lambda)$	$0.5(1+1/(1-\lambda))$	$(1/\lambda)\ln(1/(1-\lambda))$	$1+\lambda/2$
$T_{us}(\lambda)$	$0.5(1+(1/(1-\lambda))^2)$	$1/(1-\lambda)$	λ

SC – separate chaining; λ may be higher than 1

Table Data Type 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