Hashing

Instructor: Dr. Sunho Lim (Ph.D., Assistant Professor)

Lecture 19

sunho.lim@ttu.edu

Adapted partially from Data Structures and Algorithms in C++, Adam Drozdek, 4th Edition, Cengage Learning; and Algorithms and Data Structures, Douglas Wilhelm Harder, Mmath

Introduction

- Searching??
  - comparing keys
- e.g., In sequential search,
  - search the table (or array) storing the elements in order, O(n)
  - key comparisons to determine a match
- e.g., In binary search trees
  - determine the direction to take in the tree by comparing keys in the nodes, O(log n)
- A different way to search?
  - calculate the position of the key in the table, based on the key's value
  - O(1)?

Introduction (cont.)

- For example,
  - A small company of 100 employees, assigned an employ id in the range of 0 – 99
  - employ id → index into the array
  - **direct access** the record of any employee, if employ id is known

Key	Array of Employees' Records
Key 0 → [0]	Employee record with Emp_ID 0
Key 1 → [1]	Employee record with Emp_ID 1
Key 2 → [2]	Employee record with Emp_ID 2
.....	.....
.....	.....
Key 98 → [98]	Employee record with Emp_ID 98
Key 99 → [99]	Employee record with Emp_ID 99

Introduction (cont.)

- For example (cont.),
  - what if, **five-digit** employ id used as the primary key?
  - key value ranging from 00000 to 99999 → 100,000 array size
    - actually using 100 elements…
  - just use last **two digits** of the key to identify each employee

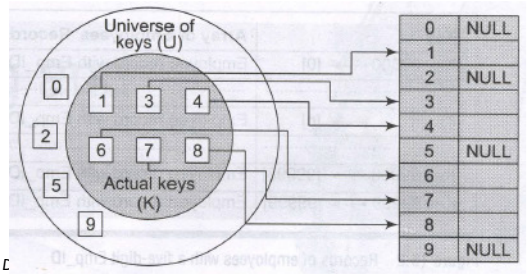
Key	Array of Employees' Records
Key 00000 → [0]	Employee record with Emp_ID 00000
.....	.....
Key n → [n]	Employee record with Emp_ID n
.....	.....
Key 99998 → [99998]	Employee record with Emp_ID 99998
Key 99999 → [99999]	Employee record with Emp_ID 99999

Introduction (cont.)

- For example (cont.),
  - convert a five-digit key number to a two-digit array index?
    - need a **function**..
    - e.g., Emp_ID 79439 → index 39
    - e.g., Emp_ID 12345 → index 45
- Terminologies,
  - **hash table** ←→ an array
  - **hash function** ←→ carry out the transformation
- Hash function, *h*
  - transform a key, *K*, into an index for a table used to store items of the same type as *K*

Hash Tables

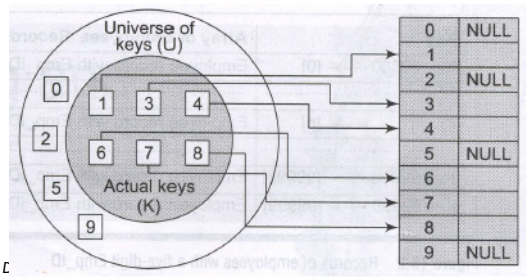
- A data structure, where
  - keys are mapped to array positions by a **hash function**
- For example, a **direct correspondence** between the keys and the indices of the array
  - useful when the total universe of keys is small
  - useful when most of the keys are actually used from the whole set of keys



storage requirement for a hash table, O(k), k is the number of keys actually used

Hash Tables (cont.)

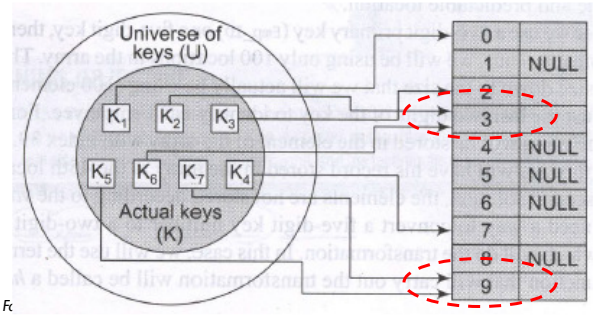
- **Hashing**
  - process of mapping the **keys** to appropriate **locations (or indices)** in a hash table
  - e.g., an element with **key k** stored at **index h(k)** , **NOT k**
    - use a hash function h



storage requirement for a hash table, O(k), k is the number of keys actually used

Hash Tables (cont.)

- **Collision**
  - two or more keys map to the same memory location
  - e.g., k2 and k6 point to the same memory location
  - e.g., k5 and k7 point to the same memory location



Hash Functions

- A mathematical formula,
  - apply to a key (numeric or alphanumeric (i.e., ASCII)), and
  - produce an integer used as an **index** for the key in the hash table
  - ideally produce a unique set of integers to reduce the number of **collisions**
- A good hash function?
  - minimize the number of collisions by spreading the elements **uniformly** throughout the array
  - low cost – the cost of executing a hash function
  - **determinism** – the same hash value must be generated for a given same input value
  - **uniformity** – must map the keys as evenly as possible over output range
    - minimize the number of collisions

Hash Functions (cont.): Division

- Division modulo,
  - h(x) = x mod T_size, where T_size = sizeof(table)
  - best choice if T_size is a prime number
- For example, calculate the hash values of keys 1234 and 5462. Here, T_size = 97
  - h(1234) = 1234 % 97 = 70
  - h(5642) = 5642 % 97 = 16

Hash Functions (cont.): Folding

- Two steps:
  - **divide** the key value into a number of parts, k1, k2, …, kn
    - each part is same number of digits except the last part
  - **add** individual parts
    - k1 + k2 + … + kn
    - ignore the last carry, if any
- For example, given a hash table of 100 locations, calculate the hash value using folding method for keys 5678 and 34567
  - key: 5678 → parts: 56 and 78 → sum: 134
    - hash value: 34 (ignore the last carry)
  - key: 34567 → parts: 34, 56, and 7 → sum: 97
    - hash value: 97

Hash Functions (cont.): Mid-Square Function

- Two steps:
  - **square** the value of key, **k^2**
  - **extract** the middle **r digits** of the result
  - h(k) = x, where x is obtained by selecting r digits from k^2
- A **good hash function**,
  - most or all digits of the key value contribute to the result
  - not dominated by the distribution of the bottom digits or the top digits of the original key value
- For example, given a hash table of 100 locations, calculate the hash value for keys 1234 and 5642
  - 100 memory locations → indices vary from 0 to 99
    - need only two digits to map the key to a location in the hash table, r = 2
  - k = 1234 → k^2 = 1522756 → h(1234) = 27
  - k = 5642 → k^2 = 31832164 → h(5642) = 32