Hashing

What is a database?

- Structured collection of data.

Student ID	First Name	Last Name	Email	Major	Faculty
200120	Kate	West	kwest@email.com	Music	Arts
200121	Julie	McLain	jmclain@email.com	Finance	Business
200122	Tom	Erlich	terlich@email.com	Sculpture	Arts
200123	Mark	Smith	msmith@email.com	Biology	Science
200124	Jen	Foster	jfoster@email.com	Physics	Science
200125	Matt	Knight	mknight@email.com	Finance	Business
200126	Karen	Weaver	kweaver@email.com	Music	Arts
200127	John	Smith	jsmith@email.com	Sculpture	Arts
200128	Allison	Page	apage@email.com	History	Humanities
200129	Craig	Cambell	ccambell@email.com	Music	Arts
200130	Steve	Edwards	sedwards@email.com	Biology	Science
200131	Mike	Williams	mwilliams@email.com	Linguistics	Humanities
200132	Jane	Reid	jreid@email.com	Music	Arts

The Dictionary ADT

- a dictionary (table) is an abstract model of a database
- a dictionary stores key-element pairs
- the main operation supported by a dictionary is searching by key

Examples

- Telephone directory
- Library catalogue
- Books in print: key ISBN

- Any other example?

Applications

- Keeping track of customer account information at a bank
  - Search through records to check balances and perform transactions
- Keep track of reservations on flights
  - Search to find empty seats, cancel/modify reservations
- Search engine
  - Looks for all documents containing a given word

Main Issues

- Size
- Operations: search, insert, delete
- What will be stored in the dictionary?
- How will items be identified?

The Dictionary ADT

- simple container methods:
  - size()
  - isEmpty()
  - elements()
- query methods:
  - findElement(k)
  - findAllElements(k)

The Dictionary ADT

- update methods:
  - insertItem(k, e)
  - removeElement(k)
  - removeAllElements(k)

A case scenario

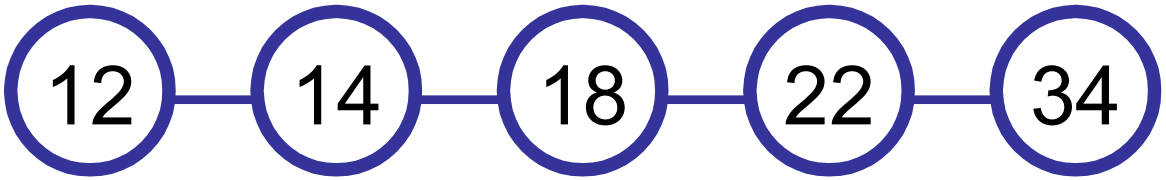
- ATCL is a large phone company, and they want to provide enhanced caller ID capability:
  - given a phone number, return the caller's name
  - phone numbers are in the range 0 to $R = 10^{10}-1$
  - n is the number of phone numbers used
  - want to do this as efficiently as possible

Implementing a Dictionary

- ***unordered sequence***
  - searching and removing takes n steps
  - inserting takes only constant time, one step
  - applications to log files (frequent insertions, rare searches and removals) 34 14 12 22 18

Implementing a Dictionary

- ***array-based ordered sequence***
  (assumes keys can be ordered)
  - searching can be efficient
  - inserting and removing takes n steps
  -application to look-up tables
  (frequent searches, rare insertions and removals)



A case scenario

- We know two ways to design this dictionary:
- ***A balanced search tree*** (AVL, b trees) --- good space usage and search time, but can we reduce the search time to constant?

- Direct Addressing

Direct Addressing

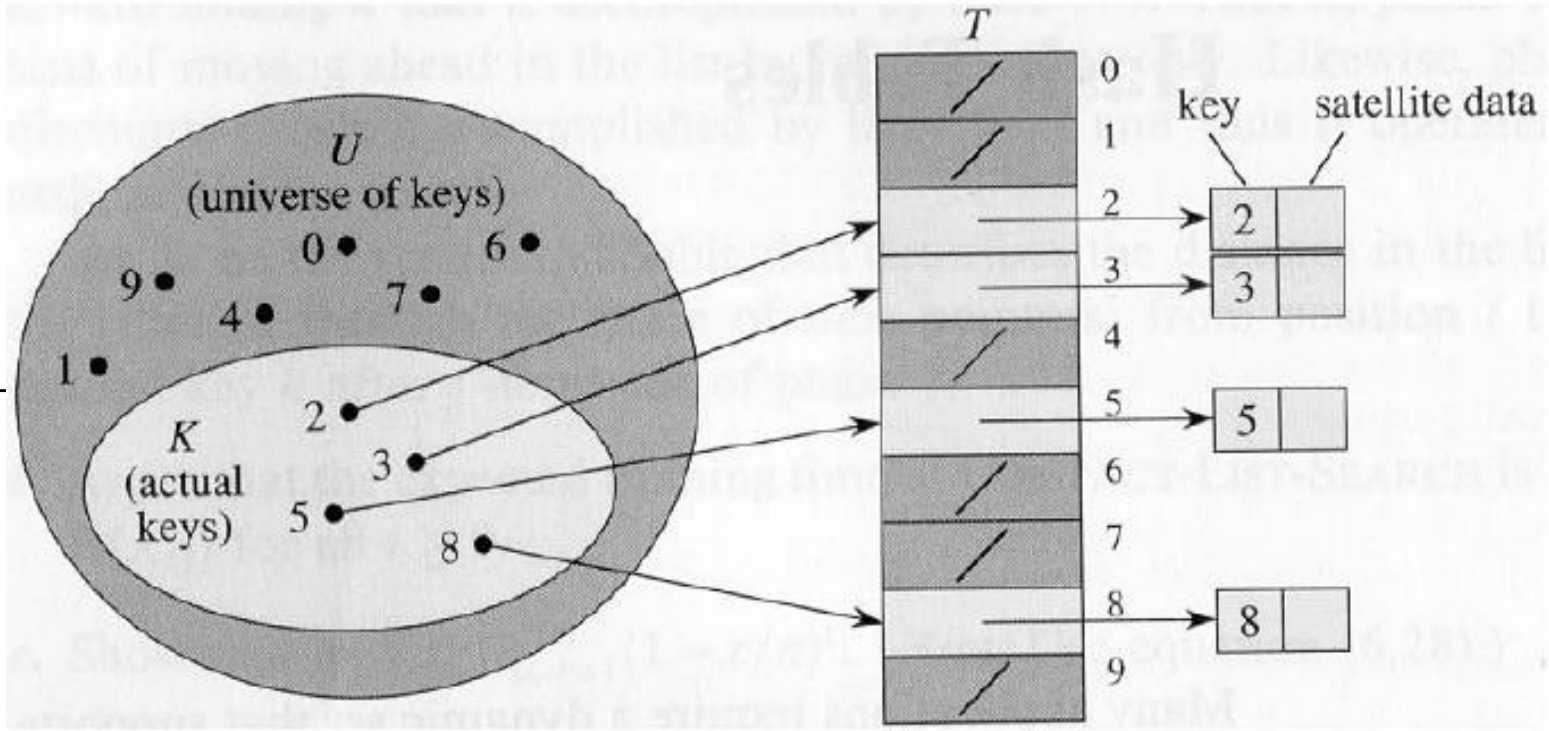
- Assumptions:
  - Key values are distinct
  - Each key is drawn from a universe $U = \{0, 1, \ldots, m - 1\}$
- Idea:
  - Store the items in an array, indexed by keys

- **Direct-address table** representation:
  - An array T[0 . . . m - 1]
  - Each **slot**, or position, in T corresponds to a key in U
  - For an element x with key k, a pointer to x (or x itself) will be placed in location T[k]
  - If there are no elements with key k in the set, T[k] is empty, represented by NIL

Direct Addressing

- Each cell is thought of as a bucket or a container
  - Holds key element pairs
  - In array A of size N, an element *e* with key *k* is inserted in *A[k]*.

Direct Addressing (cont'd)



(insert/delete in O(1) time)

Operations

Alg.: DIRECT-ADDRESS-SEARCH(T, k)

return T[k]

Alg.: DIRECT-ADDRESS-INSERT(T, x)

T[key[x]] ← x

Alg.: DIRECT-ADDRESS-DELETE(T, x)

T[key[x]] ← NIL

- Running time for these operations: $O(1)$

Advantages with Direct Addressing

- Direct Addressing is the most efficient way to access the data since
  - It takes only single step for any operation on direct address table.
  - It works well when the Universe U of keys is reasonable small.

Difficulty with Direct Addressing

When the universe *U* is very large…

- Storing a table *T* of size *U* may be impractical, given the memory available on a typical computer.
- The set *K* of the keys actually stored may be so small relative to *U* that most of the space allocated for *T* would be wasted.

An Example

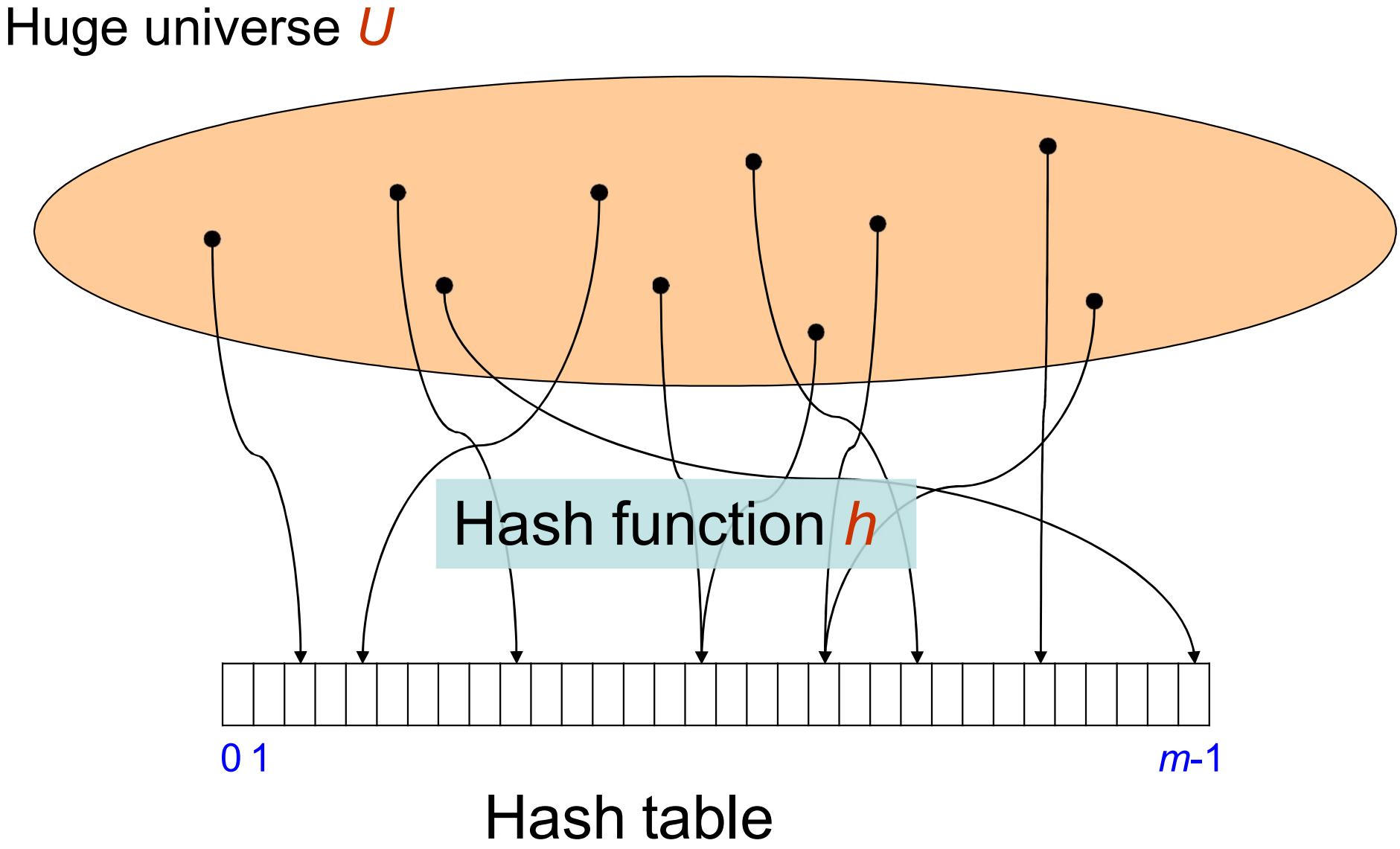
- A table, 50 students in a class.
- The key, 9 digit *SSN*, used to identify each student.
- Number of different 9 digit number=10^9
- The fraction of actual keys needed. $50/10^9$, **0.000005%**
- Percent of the memory allocated for table wasted, **99.999995%**

Hashing

- Suppose we were to come up with a "magic function" that, given a value to search for, would tell us exactly where in the array to look
  - If it's in that location, it's in the array
  - If it's not in that location, it's not in the array

- This function would have no other purpose
- This function is called a hash function because it "makes hash" of its inputs

Hashing



Hashing

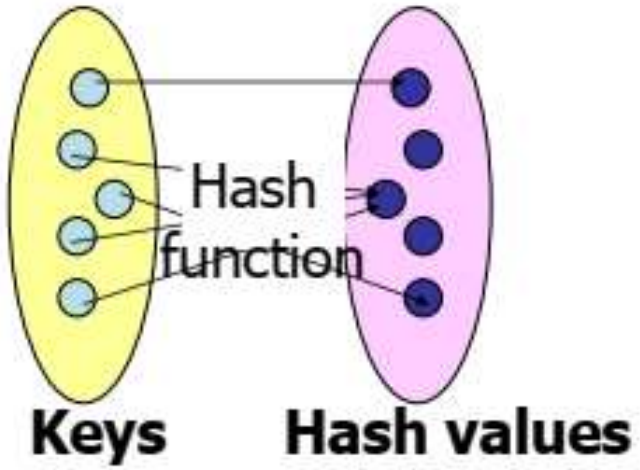
Hashing is a technique where we can compute the location of the desired record in order to retrieve it in a single access (or without comparison)

Hash Function

Map key values to hash table addresses

keys -> hash table address

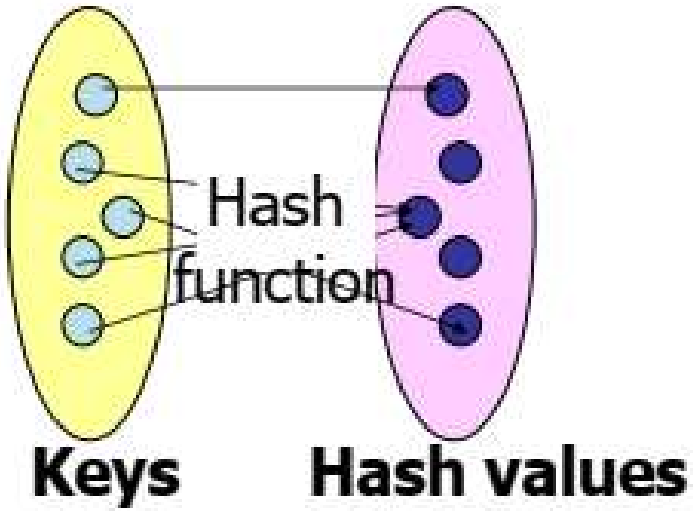
This applies to find, insert, and remove



Hash Function

The basic idea of hash function is the transformation of the key into the corresponding location in the hash table

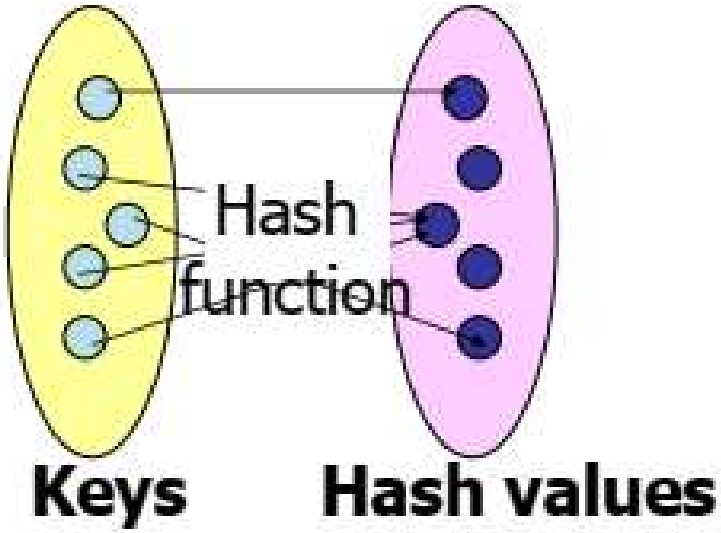
A Hash function H can be defined as a function that takes key as input and transforms it into a hash table index



A Hash Table

a hash table is a data structure

that uses a hash function to efficiently translate certain keys (e.g., person ids) into associated values.



Example

Let there is a table of n employee records and each employee record is defined by a unique employee code, which is a key to the record and employee name,

If the key (or employee code) is used as the array index, then the record can be accessed by the key directly

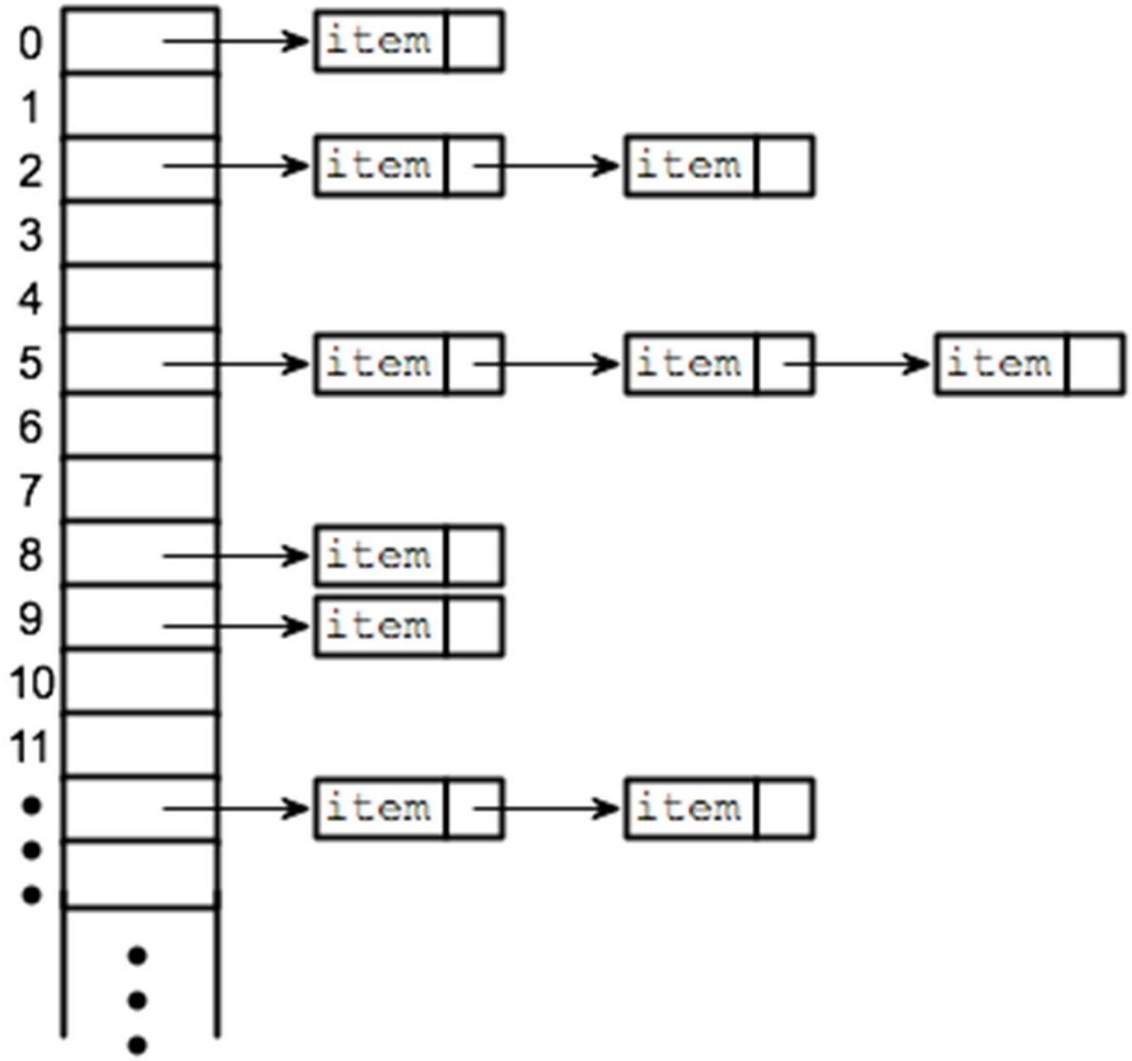
Hash Address	*Employee Code (keys)*	*Employee Name and other Detail*
0		
1		
..		
..		
..		
22	2103	Talha
..		
..		
62	3750	Dawood
..		
..		
..		
97	4147	Faizan
..		
99		

Hashing

Suppose the company has100 employee records then two digit indices of array will be sufficient.

But if the 100 employee have 5-digit of code then we have to use 5-digit indices of array but store only 100 records will waste the memory.

Sample Hash Table



The properties of a good hash function

- Rule1: The hash value is fully determined by the data being hashed.
- Rule2: The hash function uses all the input data.
- Rule3: The hash function uniformly distributes the data across the entire set of possible hash values.
- Rule4: The hash function generates very different hash values for similar strings.(Minimize collisions)

Following are the most popular hash functions:

1. Division method
2. Mid Square method
3. Folding method

Division Method

TABLE is an array of database file where the employee details are stored

- Choose a number m, which is larger than the number of keys k.
- i.e., m is greater than the total number of records in the TABLE
- The number m is usually chosen to be prime number to minimize the collision

The hash function H is defined by

$$H(k) = k \bmod m$$

Where H(k) is the hash address and k mod m means the remainder when k is divided by m

Hash Address	*Employee Code (keys)*	*Employee Name and other Details*
0		
1		
..		
..		
..		
22	2103	Talha
..		
..		
62	3750	Dawood
..		
..		
..		
97	4147	Faizan
..		
99		

For Example

Let a company has 90 employees and 00, 01, 02, ... 99 be the two digit 100 memory address (or index or hash address) to store the records

We have employee code as the key

Choose m in such a way that it is greater than 90

Suppose m = 91. Then for the following employee code (or key k) :

H(k) = H(2103) = 2103 mod 91 = 10

H(k) = H(6147) = 6147 mod 91 = 50

H(k) = H(3750) = 3750 mod 91 = 19

So if you enter the employee code to the hash function, we can directly retrieve TABLE[H(k)] details directly

Mid-Square Method

In this method, the key is squared and the address selected from the middle of the squared number

The hash function H is defined by:

$$H(k) = k2 = l$$

Where l is obtained by digits from both the end of k2 starting from left

Limitation

The most obvious limitation of this method is the size of the key
Given a key of 6 digits, the product will be 12 digits, which may be beyond the maximum integer size of many computers
Same number of digits must be used for all of the keys

For Example

Consider following keys in the table and its hash index :

K	4147	3750	2103
K^2	17197609	14062500	4422609
H(*k*)	97	62	22

Hash Address	*Employee Code (keys)*	*Employee Name and other Details*
0		
1		
..		
..		
..		
22	2103	Talha
..		
..		
62	3750	Dawood
..		
..		
..		
97	4147	Faizan
..		
99		

Hash Table with Mid-Square Division

Folding Method

In this method, the key K is partitioned into number of parts, k1, k2,..... kr

The parts have same number of digits as the required hash address, except possibly for the last part

Then the parts are added together, ignoring the last carry

H(k) = k1 + k2 + ... + kr

Here we are dealing with a hash table with index from 00 to 99, i.e., two-digit hash table

So we divide the K into numbers of two digits

K	2103	7148	12345
$k_1\ k_2\ k_3$	21, 03	71, 48	12, 34, 5
H(k) $= k_1 + k_2 + k_3$	H(2103) $= 21+03 = 24$	H(7148) $= 71+48 = 19$	H(12345) $= 12+34+5 = 51$

Hash Collision

It is possible that two non-identical keys K1, K2 are hashed into the same hash address

This situation is called Hash Collision

Let us consider a hash table having 10 locations as shown in Figure

Location	*(Keys)*	*Records*
0	210	
1	111	
2		
3	883	
4	344	
5		
6		
7		
8	488	
9		

Collision handling

Collisions are almost impossible to avoid but it can be minimized considerably by introducing any one of the following techniques:

1. Open addressing
2. Chaining
3. Use a second hash function

   ...and a third, and a fourth, and a fifth, ...

Open Addressing

In open addressing method, when a key is colliding with another key, the collision is resolved by finding a nearest empty space by probing the cells.

Suppose a record R with key K has a hash address H(k) = h
then we will linearly search h + i (where i = 0, 1, 2, ... m) locations for free space (i.e., h, h + 1, h + 2, h + 3 ..... hash address)

Clustering

- One problem with the above technique is the tendency to form “clusters”
- A cluster is a group of items not containing any open slots
- The bigger a cluster gets, the more likely it is that new values will hash into the cluster, and make it ever bigger
- Clusters cause efficiency to degrade
- Here is a *non*-solution: instead of stepping one ahead, step n locations ahead
  - The clusters are still there, they’re just harder to see
  - some table locations may be never checked

Chaining

In chaining technique the entries in the hash table are dynamically allocated and entered into a linked list associated with each hash key

The hash table in Fig. A can be represented using linked list as in Fig. B

Location	*(Keys)*	*Records*
0	210	30
1	111	12
2		
3	883	14
4	344	18
5		
6	546	32
7		
8	488	31
9		

Figure A

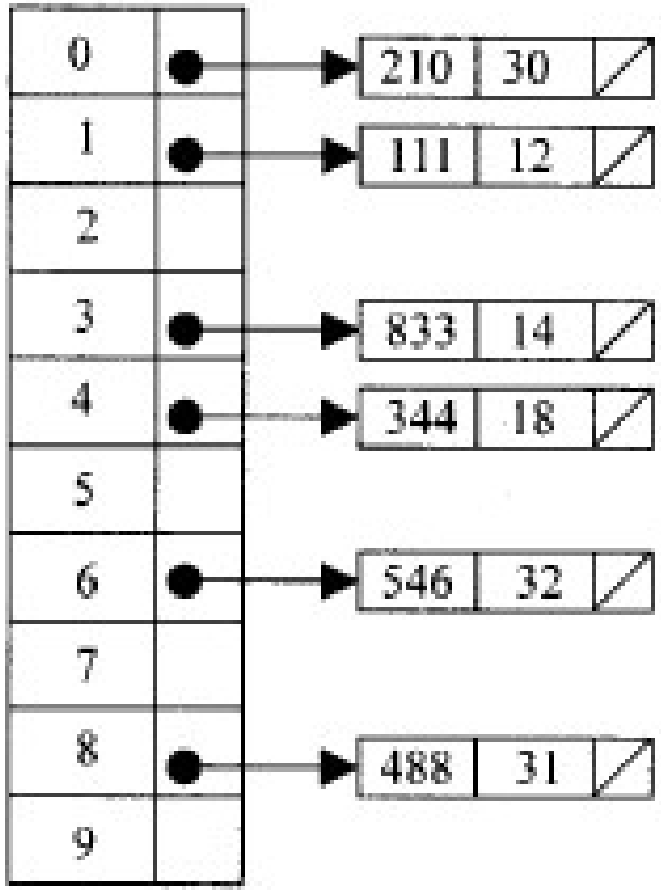
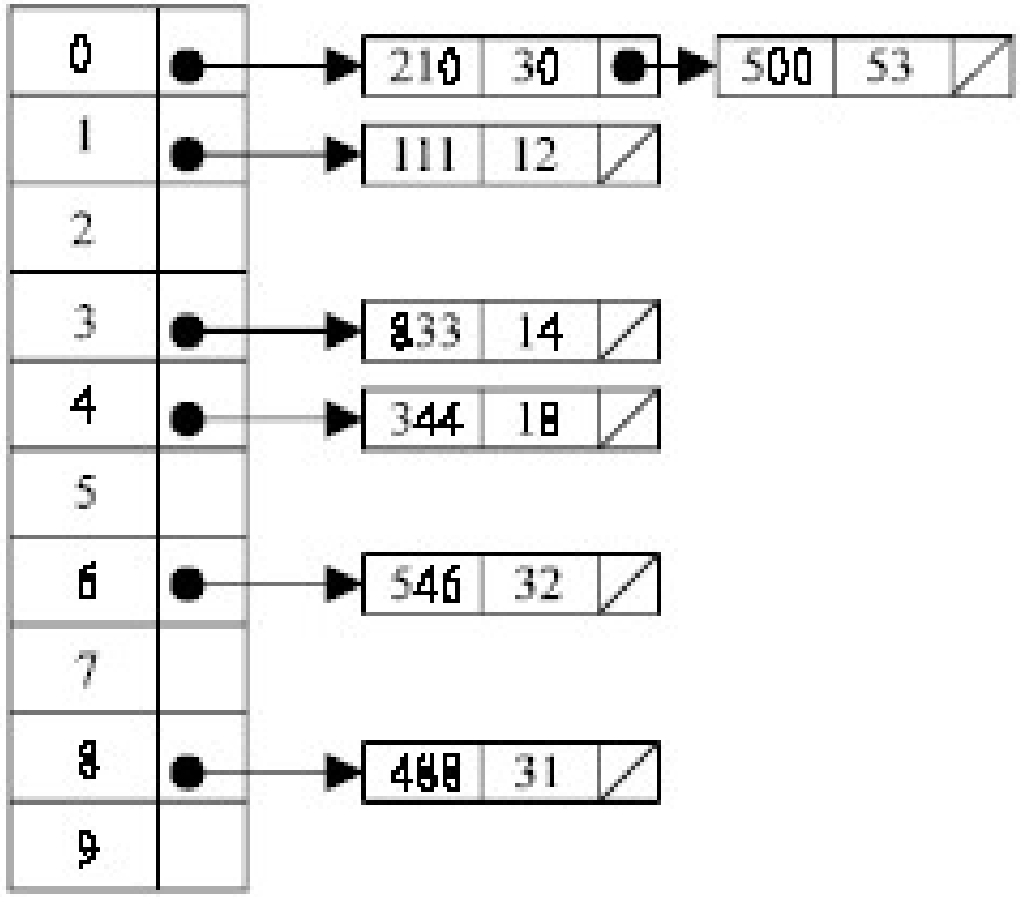


Figure B

If we try to insert a new record with a key 500 then H(500)= 500 (mod 10)=0
Then the collision occurs because there exist a record in the 0^{th} position
But in chaining corresponding linked list can be extended to accommodate the new record with the key as shown below



Efficiency

- Hash tables are actually surprisingly efficient
- Until the table is about 70% full, the number of probes (places looked at in the table) is typically only 2 or 3
- Sophisticated mathematical analysis is required to *prove* that the expected cost of inserting into a hash table, or looking something up in the hash table, is O(1)
- Even if the table is nearly full (leading to long searches), efficiency is usually still quite high

References (Homework)

Map STL

- http://www.cprogramming.com/tutorial/stl/stlmap.html

Hash Function in C++ (GCC)

- http://www.cplusplus.com/reference/functional/hash/
- http://stackoverflow.com/questions/19411742/what-is-the-default-hash-function-used-in-c-stdunordered-map
  - http://gcc.gnu.org/git/?p=gcc.git;a=blob_plain;f=libstdc%2b%2b-v3/libsupc%2b%2b/hash_bytes.cc;hb=HEAD

Finals are here !

- Practice all algorithms studied during the course (at least twice)
- Some practice problems:
  - http://www.spoj.com/problems/classical/
  - https://leetcode.com/problemset/algorithms/
  - http://www.codechef.com/problems/easy