Foundations of Algorithm SCS1308

Dr. Dinuni Fernando PhD

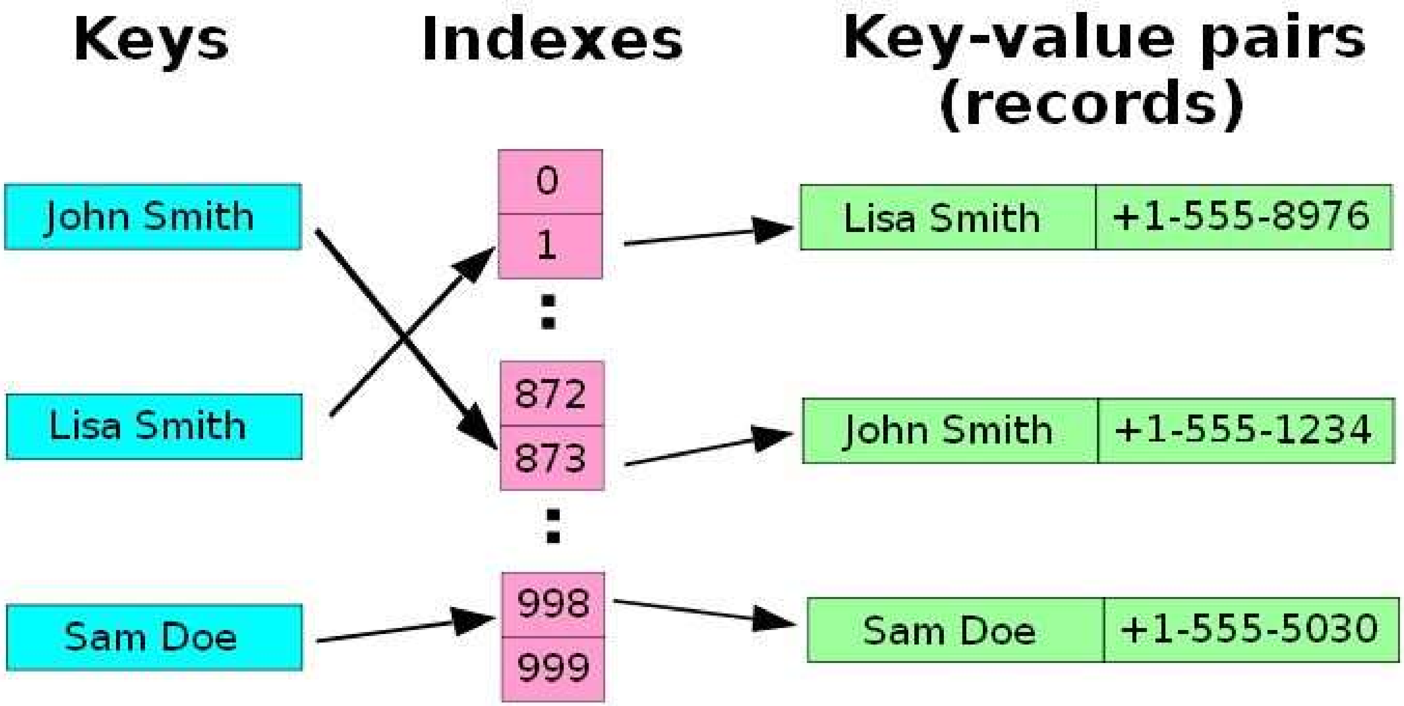
Senior Lecturer

Hashing

Concept of Hashing

- A **hash table** is a data structure that associates keys (names) with values (attributes).
  - Look-Up Table
  - Dictionary
  - Cache
  - Memcached: a.k.a. distributed hash table

Example



A small phone book as a hash table.

(Figure from Wikipedia)

Dictionaries

- Collection of pairs
  - (key, value)
  - Each pair has a unique key
- Operations.
  - Get(key)
  - Delete(key)
  - Insert(key, value)

Overall Idea

- Hash table :
  - Collection of pairs,
  - Lookup function (Hash function)
- Hash tables are often used to implement associative arrays,
  - Worst-case time for Get, Insert, and Delete is O(size).
  - Expected time is O(1).

Origins of the Term

- The term "hash" comes by way of analogy with its standard meaning in the physical world, to "chop and mix." **D. Knuth** notes that **Hans Peter Luhn** of IBM appears to have been the first to use the concept, in a memo dated January 1953; the term hash came into use some ten years later.

Search vs. Hashing

- Search tree methods: key comparisons
  - Time complexity: O(size) or O(log n)
- Hashing methods: hash functions
  - Expected time: O(1)
- Types
  - Static hashing
  - Dynamic hashing

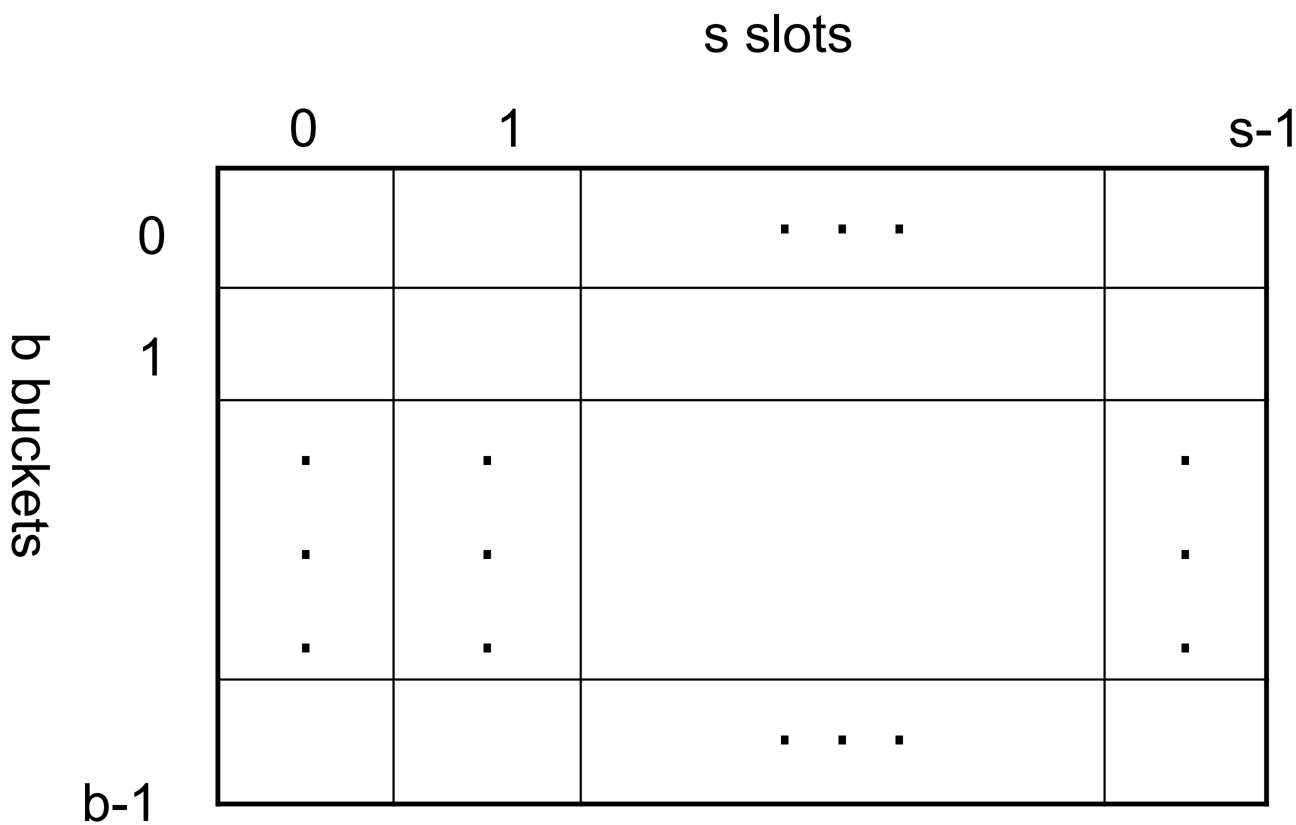
Types of Hash functions

1. Cryptographic Hash Functions:
   - Designed to be secure and are used in encryption, digital signatures, and data integrity checks.
   - Examples: MD5, SHA-1, SHA-256, SHA-3.
2. Non-Cryptographic Hash Functions:
   - Used in applications like hash tables, databases, and checksums.
   - Examples: MurmurHash, CityHash, DJB2.

Static Hashing

- Key-value pairs are stored in a fixed size table called a *hash table*.
  - A hash table is partitioned into many ***buckets***.
  - Each bucket has many ***slots***.
  - Each slot holds one record.
  - A hash function f(x) transforms the identifier (key) into an address in the hash table

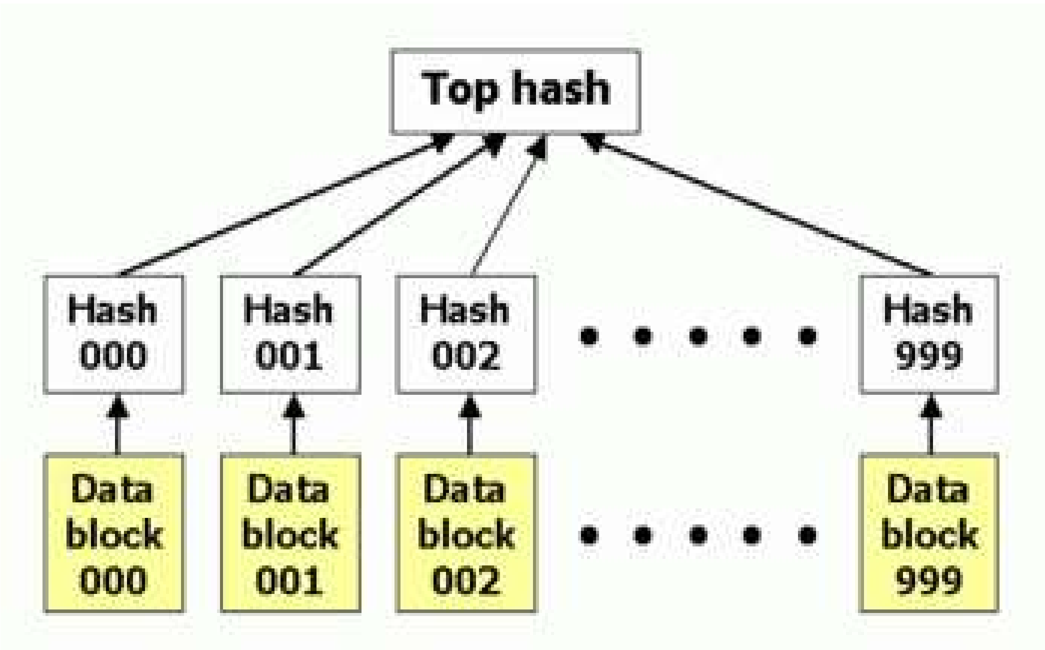
Hash Table



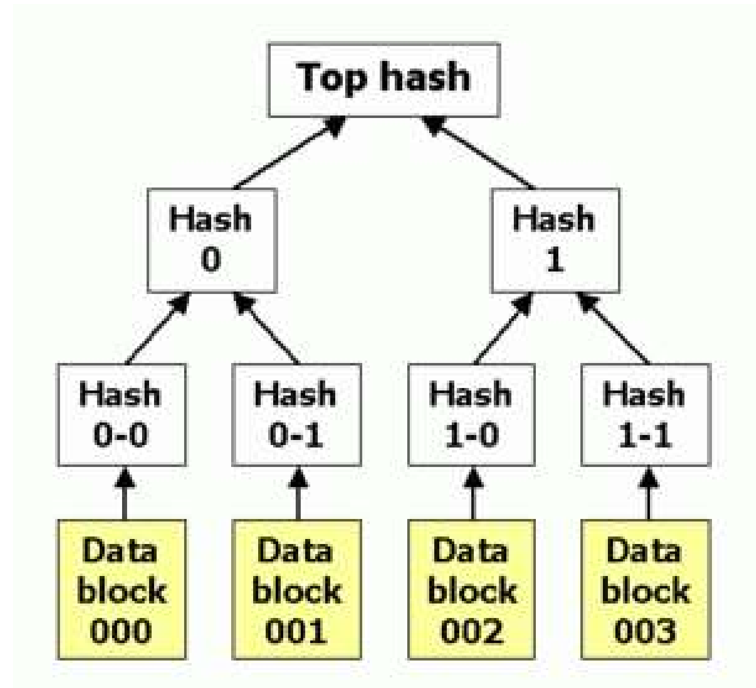
Data Structure for Hash Table

```
#define MAX_CHAR  10
#define TABLE_SIZE  17
typedef struct {
    char key[MAX_CHAR];
    /* other fields */
} element;
element hash_table[TABLE_SIZE];
```

Other Extensions



Hash List



Hash Tree / Merkle Tree

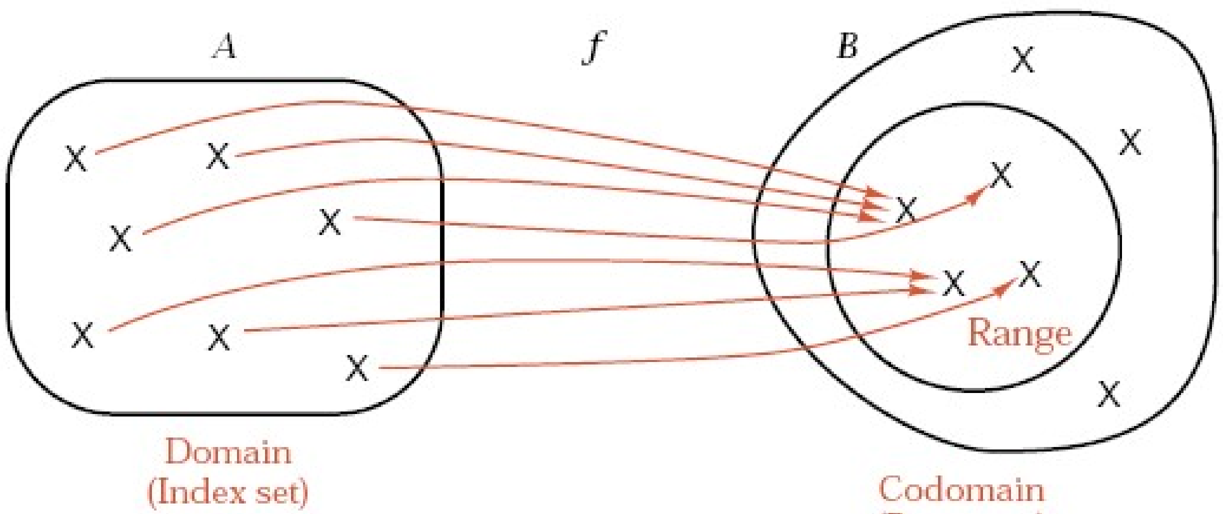
(Figure is from Wikipedia)

Hash list vs. Hash tree

	Hash List	Hash Tree
Structure	Linear list of hashes	Hierarchical tree of hashes
Efficiency	Verifying the entire dataset requires all hashes	Verifying a chunk requires only a few hashes
Scalability	Limited scalability for large datasets	Better scalability for large datasets
Use case	File integrity verification	Blockchain, distributed systems, peer-to-peer networks

Formal Definition

- Hash Function : mapping between 2 sets.
  - In addition, one-to-one / onto



All possible inputs to the function (f) originates.

Set of al potential outputs of f.

- Hash function maps inputs to a fixed set of outputs.
- Hash collisions – when 2 inputs map to the same output.
- Perfect one-to-one mapping hashing is not practical.
- Perfect hashing – need to satisfy both one-to-one and onto properties.

Ideal Hashing

- Uses an array table[0:b-1].
  - Each position of this array is a bucket.
  - A bucket can normally hold only one dictionary pair.
- Uses a hash function f that converts each key k into an index in the range [0, b-1].
- Every dictionary pair (key, element) is stored in its home bucket table[f[key]].

What Can Go Wrong?

- More than one keys can be hashed to the same bucket
- Keys that have the same home bucket are synonyms
- How to deal with this?
  - First, choose a good hash function to minimize collisions
  - Second, handle overflow efficiently

Some Issues

- **Choice of hash function**
  - ***Really tricky!***
  - To avoid collision where two different pairs are in the same bucket
  - Size (number of buckets) of hash table is desired to be small
- **Overflow handling method**
  - Overflow: there is no space in the bucket for the new pair.

Example

synonyms:
char, ceil,
clock, ctime

↑

overflow

	Slot 0	Slot 1
0	acos	atan synonyms
1		
2	char	ceil synonyms
3	define	
4	exp	
5	float	floor
6		
…		
25		

Choice of Hash Function

- Requirements
  - easy to compute
  - minimal number of collisions
- If a hashing function groups key values together, this is called clustering of the keys
- A good hashing function distributes key values uniformly throughout the range
- Ideally, simple uniform hashing

Hash table operations

- Hash : key is processed by a hash function to generate a hash value.
- Index calculation
  - The hash value is mapped to an index in a fixed-size array.
  - Example: index = hash(key) % table_size
- Storage / Retrieval
  - For insertion, the value is stored at the computed index.
  - For retrieval, the same hash function and index calculation are used to locate the value.

Characteristics of Hash Functions for Hash Tables

- **Efficiency**: Should be computationally fast.
- **Uniform Distribution**: Should spread keys evenly across the table to minimize clustering.
- **Determinism**: The same input key must always produce the same index.
- **Minimized Collisions**: Should avoid producing the same index for different keys.

Examples of hash functions

1. Middle of square hash function (mid-square hash)
   - Input: Take the input key x
   - Square the Input: Compute x^2
   - Extract the Middle Digits: Choose a specific number of middle digits from the squared value to form the hash value. The number of digits depends on the desired hash table size.

Examples of hash functions

3. Middle of square hash function
   - Eg: Suppose we are hashing the number x = 1234, and we want a hash value with 4 digits.
   - Square the input = $x^2 = 1234^2 = 1522756$
   - Extract Middle Digits: The middle 4 digits of 1522756 are 2275.
   - Result: the hash value is H(x) = 2275.
   - Index = H(x) % table_size
   - **Usecases** :
     - Small hash tables with integer keys.
     - Educational or demonstration purposes to show the fundamentals of hashing.
     - Systems where key values are not uniformly distributed.

Examples of hash functions

2. Multiplication method

- h(key) = floor(table_size * (key * A % 1))| $h(k)=\lfloor m\cdot(k\cdot A \bmod 1)\rfloor$
- A is a constant (commonly chosen as a fractional number like 0.618).
- Produces better distribution than the division method

Multiplicative Hashing Method

- Multiplicative:
  1. Choose A where 0 < A < 1 $\quad A = \frac{\sqrt{5}-1}{2}$
  2. Multiply key k by A : k x A where k is the input key
  3. Extract the fractional part of k*A (eg: KxA = 23.4567; fractional part is 0.4567
  4. Multiply the fractional part by the number of slots m
  5. Take the floor of the result
- Pro: Value of m is not critical
- Con: slower than division

Multiplicative Hashing Method: example

- Let's calculate the hash value for a key $k=1234$, table size $m=10$, and $A=0.618033$:

1. $k \cdot A = 1234 \times 0.618033 = 762.307422$
2. Fractional part: 0.307422
3. Multiply by m: $0.307422 \times 10 = 3.07422$
4. Take the floor: $\lfloor 3.07422 \rfloor = 3$
5. The hash value is $h(1234)=3$.

Examples of hash functions

4. Folding hash function:

- A key is divided into smaller parts (called "folds"), and these parts are combined (usually added together) to compute the hash value.
- Steps in folding method

1. Divide the key

- Break the key into smaller, fixed-size parts.
  - If the key is a number, you can divide it into groups of digits.
  - If the key is a string, convert it to numeric parts (e.g., ASCII values) and then divide.

2. Combine the parts
   - Add or XOR (exclusive OR) the parts together.
   - Handle any carries or overflows (if addition is used).

1. Mod by table size
   - Apply modulo operation to scale the hash value to range of the hash table size
   - h(k)=(Combined Value)%m

Folding hashing method :

- Key: 987654321, Table Size: 100

1. Divide into parts: 987, 654, 321.
2. Add the parts: 987 + 654 + 321 = 1962.
3. Modulo operation: 1962 % 100 = 62.

- Result: Hash value = 62.

Examples of hash functions

5. Digit analysis:

- Is a hashing method where specific digits or parts of a key analysed and used to generate a hash value.
- Goal is to select the digits with the most uniform distribution to reduce clustering in the hash table.
- This method assumes that the distribution of the keys is known in advance, which is often not the case in practice.

  Steps

1. Analyze key distribution : Look for digits with **skewed distributions** (digits that occur more frequently in the same position across keys).
2. Eliminate skewed digits : Ignore or delete the digits that are skewed and do not contribute to uniform hashing.
3. Use remaining digits : combine the remaining digits to form the hash address.

Digital analysis : example

Imagine you have the following keys:

- 123456
- 223457
- 323458
- 423459

- How do you generate hashes for above keys ?

Digital analysis : example

Imagine you have the following keys:

- 123456
- 223457
- 323458
- 423459
- Step1 : Observe the distribution of each digit:
    - The first digit (1, 2, 3, 4) is evenly distributed.
    - The last 3 digits (456, 457, 458, 459) are similar across keys.

Digital analysis : example

- **Step 2: Eliminate Skewed Digits**
- Remove the last 3 digits (as they are skewed) and focus on the first digit.
- **Step 3: Use Remaining Digits**
- Use the first digit (1, 2, 3, 4) as the basis for the hash value.

Resulting reduced keys:
- 1 from 123456
- 2 from 223457
- 3 from 323458
- 4 from 423459

Digital analysis : example

- **Step 3: Use Remaining Digits**
- Use the first digit (1, 2, 3, 4) as the basis for the hash value.
  - Apply a modulo operation if the hash table size is smaller.
  - Example: Hash table size m =3
  - h(k) = (first digit) % m

Resulting reduced keys:

- h(123456) = 1%3 =1
- h(223457) = 2%3 =2
- h(323458) = 3%3 =0
- h(423459) = 4%3 =1

Summary

- Simplifies key be removing less important/skewed digits.
- Focuses on uniformly distributed parts of the key.
- Reduces clustering but may still need adjustments eg: larger table size to handle collisions effectively.

Examples of hash functions

6. Division method
   - Simplest and most widely used hashing technique.
   - h(key) = key % table_size(m) Takes the remainder
   - Choosing good table size(m) is tricky
   - Bad example: m is power of 2
   - Generally, choose a prime that is not so close to power of 2 as m
     - To reduce clustering

Examples of hash functions : division method

- Example 1 : Using simple division
- Keys : 42, 56, 73, 101
- Table size (m) = 10

$h(k) = k \bmod m$

Q : Create your hash table

Examples of hash functions : division method

- Example 1 : Using simple division
- Keys : 42, 56, 73, 101
- Table size (m) = 10

$h(k) = k \bmod m$

h(42) = 42 mod 10 = 2
h(56) = 56 mod 10 = 6
h(73) = 73 mod 10 = 3
h(101) = 101 mod 10 = 1

Slot	Key
0	
1	101
2	42
3	73
4	
5	
6	56
7	
8	
9	

Examples of hash functions : division method

- Example 2 : Using prime table size
- Keys : 42, 56, 73, 101
- Table size (m) = 7 (a prime number) h(k) = k mod m

Q : Create your hash table

Examples of hash functions : division method

- Example 2 : Using prime table size
- Keys : 42, 56, 73, 101
- Table size (m) = 7 (a prime number)

h(k) = k mod m

h(42) = 42 mod 7 = 0
h(56) = 56 mod 7 = 0
h(73) = 73 mod 7 = 3
h(101) = 101 mod 7 = 3

Slot	Key
0	42,56
1	
2	
3	73,101
4	
5	
6	

Collisions occur when multiple keys map to the same slot.

Hashing By Division

- Domain is all integers.
- For a hash table of size *b*, the number of integers that get hashed into bucket *i* is approximately $\mathbf{2^{32}/b}$.
- The division method results in a uniform hash function that maps approximately the same number of keys into each bucket.

Hashing By Division II

- In practice, keys tend to be correlated.
  - If divisor is an even number, odd integers hash into odd home buckets and even integers into even home buckets.
    - 20%14 = 6, 30%14 = 2, 8%14 = 8
    - 15%14 = 1, 3%14 = 3, 23%14 = 9
  - divisor is an odd number, odd (even) integers may hash into any home.
    - 20%15 = 5, 30%15 = 0, 8%15 = 8
    - 15%15 = 0, 3%15 = 3, 23%15 = 8

Hashing By Division III

- Similar biased distribution of home buckets is seen in practice, when the divisor is a multiple of prime numbers such as 3, 5, 7, …
- Ideally, choose large prime number b.
- Alternatively, choose b so that it has no prime factors smaller than 20.

Hash Algorithm via Division

```
void init_table(element ht[])
{
  int i;
  for (i=0; i<TABLE_SIZE; i++)
    ht[i].key[0]=NULL;
}

int transform(char *key)
{
  int number=0;
  while (*key) number += *key++;
  return number;
}
```

```
int hash(char *key)
{
  return (transform(key)
              % TABLE_SIZE);
}
```

Criterion of Hash Table

- The key density (or identifier density) of a hash table is the ratio n/T
  - n is the number of keys in the table
  - T is the number of distinct possible keys
- The loading density or loading factor of a hash table is $\alpha = n/(sb)$
  - s is the number of slots
  - b is the number of buckets

Example

synonyms:
char, ceil,
clock, ctime

overflow

	Slot 0	Slot 1
0	acos	atan
1		
2	char	ceil
3	define	
4	exp	
5	float	floor
6		
…		
25		

synonyms

synonyms

$b=26$, $s=2$, $n=10$, $\alpha=10/52=0.19$, $f(x)$=the first char of x

Overflow Handling

- An overflow occurs when the home bucket for a new pair (key, element) is full.
- We may handle overflows by:
  - Search the hash table in some systematic fashion for a bucket that is not full.
    - Linear probing (linear open addressing).
    - Quadratic probing.
    - Random probing.
  - Eliminate overflows by permitting each bucket to keep a list of all pairs for which it is the home bucket.
    - Array linear list.
    - Chain.

Linear probing (linear open addressing)

- **Open addressing** ensures that all elements are stored directly into the hash table. It attempts to resolve collisions using various methods.

  - **Linear Probing** resolves collisions by placing the data into the next open slot in the table.

Linear Probing – Get And Insert

- Compute hash function H(k)
- If occupied, probe H(k) + 1, H(k)+ 2, …
- divisor = b (number of buckets) = 17.
- Home bucket = key % 17.

- Insert pairs whose keys are 6, 12, 34, 29, 28, 11, 23, 7, 0, 33, 30, 45

Q : Create your hash table

Linear probing example

- Keys : 6, 12, 34, 29, 28, 11, 23, 7, 0, 33, 30, 45
- h(k) = key % 17
- 6 % 17 = 6
- 12 % 17 = 12
- 34 % 17 = 0
- 29 % 17 = 12 collision = 12+1 = 13
- 28 % 17 = 11
- 11 % 17 = 11 collision = 11+1 = 12 collision = 13 -> 14
- 23% 17 = 7
- 7 % 17 = 7
- 0 % 17 = 0 : collision = 0+1 = 1
- 33 % 17 = 16
- 30 % 17 = 13 collision : 13+1 = 14 collision = 15
- 45 % 17 = 11 : collision 11+ 1 =12……1 =2

Slot	Key
0	34
1	0
2	45
3	
4	
5	
6	6
7	23
8	7
9	
10	
11	28
12	12
13	29
14	11
15	30
16	33

Performance Of Linear Probing

0				4				8				12				16
34	0	45				6	23	7			28	12	29	11	30	33

- Pro: Easy to implement
- Con: Primary clustering
  - Long runs of occupied slots build up. An empty slot preceded by i full slots gets filled next with probability (i+1)/m
- Worst case time is $\Theta(n)$. This happens when all pairs are in the same cluster

Problem of Linear Probing

- Identifiers tend to cluster together
- Adjacent cluster tend to coalesce
- Increase the search time

Quadratic Probing

- Compute H(k)
- If occupied, probe $H(k) + 1^2$, $H(k) + 2^2$, $H(k) + 3^2$, …
- Secondary clustering
  - If the initial hash value is the same for two different keys, their probe sequences are the same
  - Could be better than primary clustering though

Quadratic probing

- Keys to insert : 10,22,31,40,42,52
- Hash table size (m) = 7
- Hash function h(k) = k mod m
- Quadratic probing formula $h'(k,i) = (h(k)+i^2) \bmod m$

Q : Create your hash table

Quadratic probing: Example

- Keys to insert : 10,22,31,40,42,52
- Hash table size (m) = 7
- Hash function h(k) = k mod m
- Quadratic probing formula h’(k,i) = (h(k)+i^2) mod m
- 10 mod 7 =3
- 22 mod 7 = 1
- 31 mod 7 = 3 collision ($3+1^2$) = 4 mod 7 = 4
- 40 mod 7 = 5
- 42 mod 7 = 0
- 52 mod 7 = 3, collision ($3+1^2$) = 4 mod 7 = 4 collision ($3+2^2$) = 7 mod 7 = 0 collision, ($3+3^2$) = 12 mod 7 = 5 collision, ($3+4^2$) = 19 mod 7 = 5 collision, ($3+5^2$) = 28 mod 7 = 0 collision, ($3+6^2$) = 4 collision = 3 collision……

A **poor hash table size** choice or overcrowding.

Random Probing

- Is a collision resolution technique used in open addressing for hash tables.
- When a collision occurs, instead of using sequential (linear, quadratic) probing, a random probing sequence is generated to find an empty slot.
- The probing sequence is determined using a random number generator.
- Random Probing works incorporating with random numbers.
  - $H(x) := (H'(x) + S[i]) \% b$
  - S[i] is a table with size b-1
  - S[i] is a random permutation of integers 1,2, …, b-1.

Random Probing

- $h'(k) = (h(k) + r(i)) \bmod m$
- Where :
- $h(k)$: original hash value of the key
- i: probe attempt number ($i = 0,1,2,\ldots.$)
- $r(i)$ = a pseudo-random number generated for the ith probe
- m : size of the hash table.

Random Probing : example

- Keys : 10,22,31,40,42,52
- Hash table (m) = 7
- Hash function h(k) = k mod m
- Random numbers for each probe attempt
- r(0) = 0, r(1) = 2, r(2) = 4, r(3) = 1, r(4) = 3, r(5) = 6, r(6) = 5

Q : Create your hash table

Random Probing : example

- Keys : 10,22,31,40,42,52 | Hash table (m) = 7
- Hash function h(k) = k mod m
- r(0) = 0, r(1) = 2, r(2) = 4, r(3) = 1, r(4) = 3, r(5) = 6, r(6) = 5
- 10 mod 7 = 3
- 22 mod 7 = 1
- 31 mod 7 = 3 collision, random probing = 3+r(1) mod 7 = 5
- 40 mod 7 = 5
- 42 mod 7 = 0
- 52 mod 7 = 3 collision, random probing = 3+r(1) mod 7 = 5
  - (3+r(2)) mod 7 = 0 collision, (3+ r(3)) = 4

Random Probing : example

- Keys : 10,22,31,40,42,52 | Hash table (m) = 7
- Hash function h(k) = k mod m
- r(0) = 0, r(1) = 2, r(2) = 4, r(3) = 1, r(4) = 3, r(5) = 6, r(6) = 5
- 10 mod 7 = 3
- 22 mod 7 = 1
- 31 mod 7 = 3 collision, random probing = 3+r(1) mod 7 = 5
- 40 mod 7 = 5 collision; 5+r(1) = 0
- 42 mod 7 = 0 collision 0+r(1) = 2
- 52 mod 7 = 3 collision, random probing = 3+r(1) mod 7 = 5
  - (3+r(2)) mod 7 = 0 collision, (3+ r(3)) = 4

Slot	Key
0	40
1	22
2	42
3	10
4	52
5	31
6	6