Hash

Hyoungshick Kim
College of Software
Sungkyunkwan University

Dictionary

- Collection of pairs.
  - (key, element)
  - Pairs have different keys.
- Operations.
  - Search(theKey)
  - Delete(theKey)
  - Insert(theKey, theElement)

- A <u>linear list</u> is an ordered sequence of elements.
- A <u>dictionary</u> is just a collection of pairs.

Hash Tables

- A dictionary data structure
  - Collection of pairs (key, element)
  - Search, Delete, Insert operations
- Worst-case time for Search, Insert, and Delete is O(size).
- Expected time is O(1).

Main Idea of Hash

- Associate key with index

k ? $\longrightarrow$ A[f(k)]

Ideal Hashing

- Uses an 1D array (or table) 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].
  - f(k) is the home bucket for key k.
- Every dictionary pair (key, element) is stored in its home bucket table[f[key]].

Ideal Hashing Example

- Pairs are: (22,a), (33,c), (3,d), (73,e), (85,f).
- Hash table is table[0:7], b = 8.
- Hash function is key/11.
- Pairs are stored in table as below:

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]
(3,d)		(22,a)	(33,c)			(73,e)	(85,f)

- Search, Insert, and Delete take O(1) time.

What Can We Go Wrong?

(3,d)		(22,a)	(33,c)			(73,e)	(85,f)
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]

- Where does (26,g) go?
- Keys that have the same home bucket are synonyms.
  - 22 and 26 are synonyms with respect to the hash function that is in use.
- The home bucket for (26,g) is already occupied.
- Where does (100,h) go?

What Can We Go Wrong?

(3,d)		(22,a)	(33,c)			(73,e)	(85,f)

- A collision occurs when the home bucket for a new pair is occupied by a pair with a different key.
- An overflow occurs when there is no space in the home bucket for the new pair.
- When a bucket can hold only one pair, collisions and overflows occur together.
- We need a method to handle overflows.

Hash Table Issues

- Choice of hash function.
- Overflow handling method.

Hash Functions

- Two parts:
  - Convert key into a nonnegative integer in case the key is not an integer.
    - Done by the function hash().
  - Map an integer into a home bucket.
    - f(k) is an integer in the range [0, b-1] where b is the number of buckets in the table.

String to Integer

- Each character is 1 byte long.
- An int is 4 bytes.
- A 2 character string key[0:1] may be converted into a unique 4 byte non-negative int using the code:

```
number = (int)key[0];
number <<= 8;
number += (int)key[1];
```

- Strings that are longer than 4 characters do not have a unique non-negative int representation.

String to Integer

```
unsigned int stringToInt(char *key)
{
    int number = 0;
    while(*key)
    {
        number += (int)*key++;
        number <<= 8;
    }
    return number;
}
```

Number is shifted left 8 bits after adding to number.

Map into a Home Bucket

(3,d)		(22,a)	(33,c)			(73,e)	(85,f)
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]

- Most common method is to use modular division.

homeBucket = hash(theKey) % divisor;

- divisor equals number of buckets b.
- 0 <= homeBucket < divisor = b

Uniform Hash Function

(3,d)		(22,a)	(33,c)			(73,e)	(85,f)
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]

- Let keySpace be the set of all possible keys.
- A uniform hash function maps the keys in keySpace into buckets such that approximately the same number of keys get mapped into each bucket.
- Our goal is to use a uniform hash function.

Uniform Hash Function

(3,d)		(22,a)	(33,c)			(73,e)	(85,f)
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]

- Equivalently, the probability that a randomly selected key has bucket i as its home bucket is 1/b, 0 <= i < b.
- A uniform hash function minimizes the likelihood of an overflow when keys are selected at random.

Hashing by Division

- keySpace = all ints.
- For every b, the number of ints that get mapped (hashed) into bucket i is approximately $2^{32}/b$.
- Therefore, the division method results in a uniform hash function when keySpace = all ints.
- In practice, however, keys tend to be correlated.
- So, the choice of the divisor b affects the distribution of home buckets.

Selecting the Divisor

- Because of this correlation, applications tend to have a bias towards keys that map into odd integers (or into even ones).
- When the 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
- The <u>bias in the keys</u> results in a bias toward either the odd or even home buckets.

Selecting the Divisor

- When the 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
- The bias in the keys does not result in a bias toward either the odd or even home buckets.
- Better chance of uniformly distributed home buckets.
- So <u>do not use an even divisor</u>.

Selecting the Divisor

- Similar biased distribution of home buckets is seen, in practice, when the divisor is a multiple of prime numbers such as 3, 5, 7, …
- The effect of each prime divisor p of b decreases as p gets larger.
- Ideally, choose b so that it is a **prime number**.
- Alternatively, choose b so that it has no prime factor smaller than 20.

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.
    - Rehashing (or Double hashing)
  - 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 – Get and Insert

- divisor = b (number of buckets) = 17.
- Home bucket = key % 17.

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

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

Linear Probing – Delete

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

- Delete(0)

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

- Search cluster for pair (if any) to fill vacated bucket.

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

Linear Probing – Delete(34)

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

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

- Search cluster for pair (if any) to fill vacated bucket.

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

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

Linear Probing – Delete(29)

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

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

- Search cluster for pair (if any) to fill vacated bucket.

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

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

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

Performance of Linear Probing

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

- Cluster is a contiguous block of items.
- Insert and search cost depend on length of cluster
- Worst-case find/insert/erase time is $\Theta(n)$ where n is the number of pairs in the table.
  - This happens when all pairs are in the same cluster

Expected Performance

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

- Trivial: average length of cluster = α = loading density = (number of pairs)/b.
  - $\alpha = 12/17$.
- [Knuth 1962] Let $\alpha < 1$ be an average length of list

Search: $\frac{1}{2}(1 + 1/(1 - \alpha))$

Insert: $\frac{1}{2}(1 + 1/(1 - \alpha)^2)$

$\alpha <= 0.75$ is recommended.

Quadratic Probing

- Examine the hash table buckets

  $ht[f(x) \% b]$,

  $ht[(f(x) + i^2) \% b]$,

  $ht[(f(x) - i^2) \% b]$,

  for $0 \le i \le (b-1)/2$,

- reduce the average number of probes

Quadratic Probing – Insert

- divisor = b (number of buckets) = 17.
- Home bucket = key % 17.

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

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

Rehashing

- use a series of hashing functions

  $f_1, f_2, \cdots, f_m$

- bucket $f_i(x)$ is examined for

  $i = 1, 2, \cdots, m$

Rehashing – Insert

- F_1 = key % 17.
- F_2 = key % 7.
- F_3 = key % 7 + i.

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

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

Hash Table Design

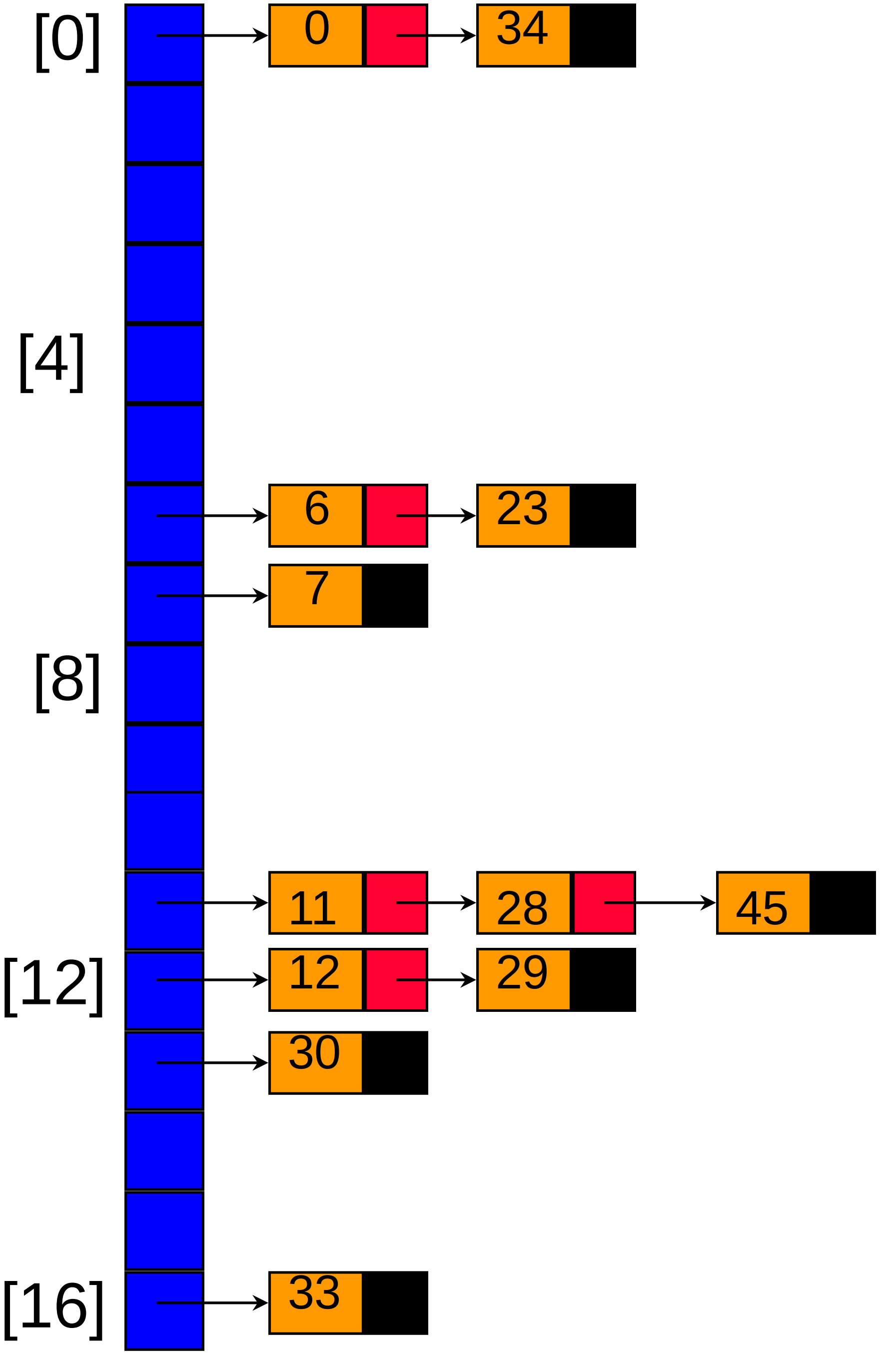
- Dynamic resizing of table.
  - Maintain the loading density <= 4/5
  - Whenever loading density exceeds a threshold (e.g., 4/5), rehash into a table of approximately twice the current size.
- Fixed table size.
  - When the maximum number of pairs is known.
  - Pick b (equal to divisor) to be a prime number or an odd number with no prime divisors smaller than 20.

Linear List of Synonyms

- Each bucket keeps a linear list of all pairs for which it is the home bucket.
- The linear list may or may not be sorted by key.
- The linear list may be an array linear list or a chain.

Sorted Chains

- Put in pairs whose keys are 6, 12, 34, 29, 28, 11, 23, 7, 0, 33, 30, 45
- Home bucket = key % 17.



Expected Performance

- Note that $\alpha >= 0$.
- Expected chain length is α.
- Search: $1 + \alpha/2$.
- Insert: $1 + \alpha/2$ for sorted, 1 for unsorted

Questions?


Sungkyunkwan
Campus
Sciences
established
School
Suwon
university
University
campus
Seoul
Natural
Graduate
edit
Korean
Social
English
Global
constructed
Building
Institute
Professors
Confucian
Education
Humanities
lecturers
Advanced
Korea
Zone
Joseon
located
Engineering
schools
Science
Administration
Academic
ASLE-Korea
also
city
two
four
Hall
Course
now
one
heart
trees
symbol