Hashing

Ch. 8

Hash tables

- A table to store dictionary pairs
  - A dictionary pair includes (key, element).
  - Different pairs has different keys.

- Operations
  - Search(key)
  - Insert(key, element)
  - Delete(key)

  Expected time: $O(1)$

Hash tables

- A hash table *ht* contains
  - *b* buckets: *ht*[0], *ht*[1], …, *ht*[*b*-1]
    - A bucket has *s* slots.
    - A slot hold a dictionary pair.
    - *s* == 1 → Each bucket can hold only one pair.
- A hash function *h*(*k*) determines the location of a pair whose key is *k*.
  - *h*(*k*) is an integer in the range [0, b-1].
  - *h*(*k*) is the home address for key *k*.
- Ideally, every dictionary pair (key, element) is stored in the home bucket *ht*[*h*(key)].

Example

Insert the pairs with the following keys to the hash table.

Asus, Canon, Zyxel, Epson, Ericsson, Apple, Dell

- Hash table *ht*:
  - 26 buckets (b=26)
  - 2 slots per bucket (s=2)
- Hash function $h(k)$:
  - Map the first character of a key to 0-25.

	Slot 0	Slot 1
0	Asus	Apple
1		
2	Canon	
3	Dell	
4	Epson	Ericsson
…		
25	Zyxel	

Loading density of a hash table

- Loading density

  $$\alpha = n/sb$$

  - n = number of pairs in the table
  - s = number of slots per bucket
  - b = number of buckets

- α in the Example:
  - $\alpha = 7 / (2 * 26)$

	Slot 0	Slot 1
0	Asus	Apple
1		
2	Canon	
3	Dell	
4	Epson	Ericsson
…		
25	Zyxel	

Synonyms

- Two keys are synonyms if $h(k_1)=h(k_2)$
  - In this example, Asus and Apple are synonyms.

- When inserting dictionary pairs,
  - Example1: (Cisco , 1000)
    The home bucket is not empty. → Collision
  - Example2: (Acer, 1000)
    The home bucket is full. → Overflow

Collisions and overflows occur simultaneously when each bucket has 1 slot.

	Slot 0	Slot 1
0	Asus	Apple
1		
2	Canon	Cisco
3	Dell	
4	Epson	Ericsson
…		
25	Zyxel	

Issues of hash tables

- Hash functions
- Overflow handling
- Size of hash table

Ideal hash functions

- Easy to compute
- Minimize number of collision
- No biased use of hash table
  - h(k) is independently and uniformly at random from 0 to b-1.
  - The probability of a bucket to be selected is 1/b.

Hash functions: Division

- Division: $h(k) = k \% D$
  - Use the reminder
- Example:
  - Inserting pairs (22,a), (34,c), (3,d), (73,e), (86,f)
  - Hash table with 11 slots, *ht*[0:10]
  - Hash function: key % 11

(22,a)	(34,c)		(3,d)				(73,e)		(86,f)	
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]

Hash functions: Folding

- Partition the key into several parts P_0, P_1, …, P_i and add all partitions together

- Shift folding: $h(k) = P_0 + P_1 + \dots + P_i$

  k = 123|203|241|112|20 and we partition it into 3 digits long.

  $h(k) = 123 + 203 + 241 + 112 + 20 = 699$

- Folding at the boundaries: Reversing every other partition and then adding.

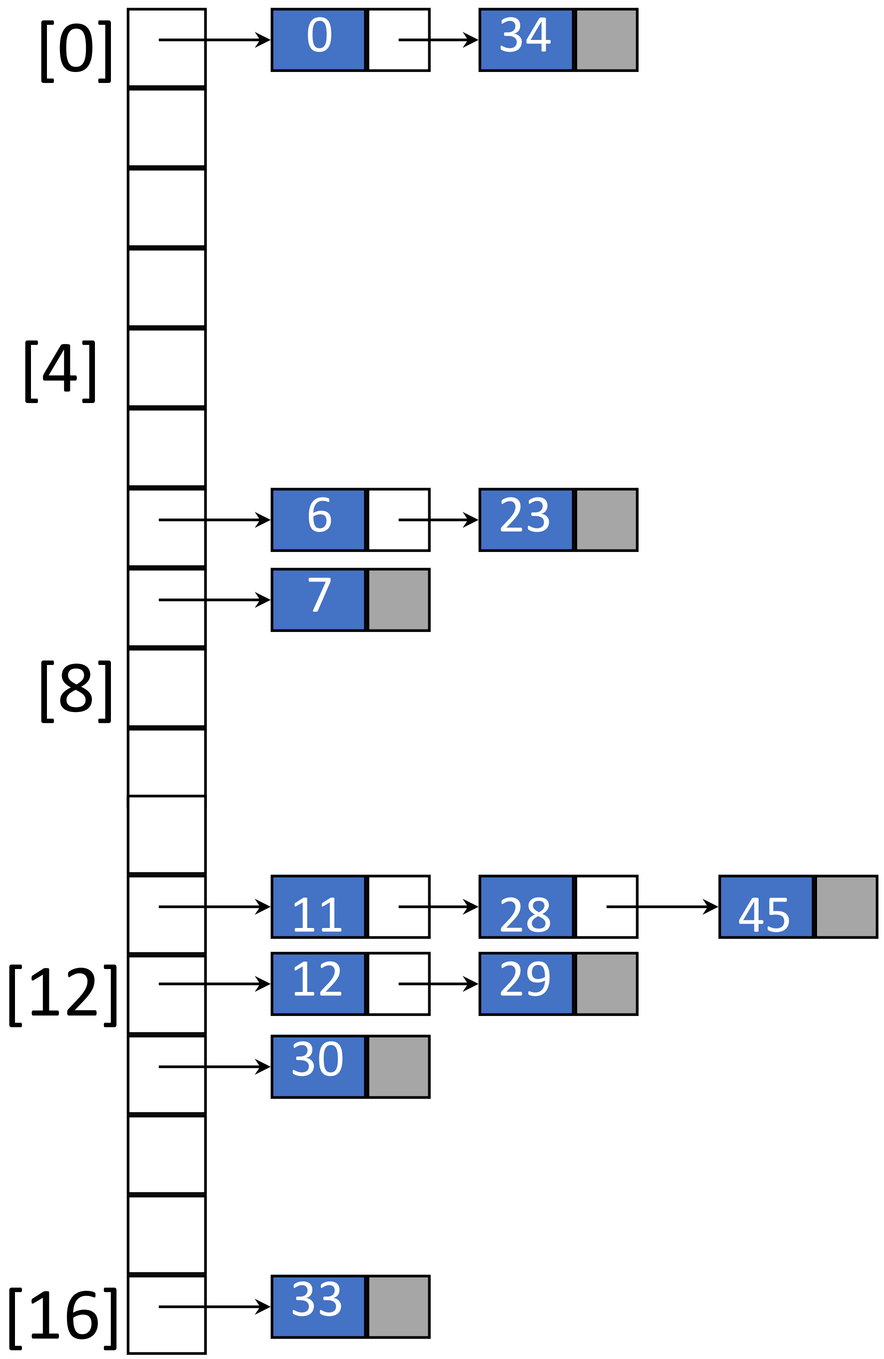
  k = 123|203|241|112|20 and we partition it into 3 digits long.

  $h(k) = 123 + 302 + 241 + 211 + 20 = 897$

Overflow handling (1)

Chaining

- A linked list per bucket
- Each list contains all the synonyms.
- Example:
  - Inserting pairs whose keys are 6, 12, 34, 29, 28, 11, 23, 7, 0, 33, 30, 45
  - Hash function = key % 17
- Averaged chain length: n/b



Overflow handling (2)

Open addressing

- Search the hash table in some systematic fashion for a bucket that is not full
  - Linear probing (linear open addressing)
  - Quadratic probing
  - Rehashing
  - Random probing

Linear probing: Search() and Insert()

Pseudocode: Please see program 8.3 in textbook.

- Search the hash table in the following order:

$$ht[h(k) \,\%\, b],\ ht[(h(k)+1) \,\%\, b],\ \ldots,\ ht[(h(k)+\mathrm{b}) \,\%\, b]$$

h: hash function, b: number of buckets

- The search terminates when we reach the first unfilled bucket.
  - Insert the pair into that bucket.

- Example: Hash function = key % 17, b = 17. insert pairs whose keys are 6, 12, 34, 29, 28, 11, 23, 7, 0, 33, 30, 45.

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

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

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

- Delete(34)

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	45					6	23	7			28	12	29	11	30	33

Linear probing: Delete

- Delete(29)

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

34	0	45				6	23	7			28	12	11	30		33

Cluster for 0

Cluster for 16

34	0					6	23	7			28	12	11	30	45	33

Exercise

Please reply your answers of Q6-Q7 via the following link:

Group members: 1~3 people

- Q6: Hash function = key % 13, number of buckets = 13. Using linear probing to insert pairs whose keys are 6, 13, 34, 29, 41, 11, 23, 7, 0, 33, 30, 45. Write out the hash table.

- Q7: A hash function h defined as key % 7 with linear probing. Insert the keys 37, 38, 72, 48, 98, 11, and 56 into a table. Where will be 11 in the table?

Quadratic probing

- Search the hash table in the following order:

  $ht[h(k)\ \%\ b]$,

  $ht[(h(k)+1)\ \%\ b]$, $ht[(h(k)-1)\ \%\ b]$,

  $ht[(h(k)+2^2)\ \%\ b]$, $ht[(h(k)-2^2)\ \%\ b]$,

  $ht[(h(k)+3^2)\ \%\ b]$, $ht[(h(k)-3^2)\ \%\ b]$,…

  h: hash function, b: number of buckets

- b should be the prime number of the form $4j + 3$

$b = 4j + 3$

Prime	j
3	0
7	1
11	2
19	4

Rehashing

- Create a series of hash functions $h_1, h_2, \ldots, h_m$
- Examine buckets in the order of $h_1(k), h_2(k), \ldots, h_m(k)$.

Performance

- Worst-case find/insert/delete time is $O(n)$
  n: The number of pairs in the table.

- Open addressing: This happens when all pairs are in the same cluster.
- Chaining: This happens when all pairs are in the same chain.

Expected performance of chaining

Chained hash table with uniform hash function

- Averaged chain length: n/b
  - n: number of data items in hash table
  - b: number of buckets (number of chains)

When n is large.

- U_n = expected number of key comparisons in an unsuccessful search
  = expected number of keys on a chain

$$U_n = n/b$$

- S_n = expected number of key comparisons in a successful search.
  - When the i-th key is being inserted, the expected number of keys in a chain is $(i-1)/b$.
  - The expected number of comparisons needed to search for k_i is $1 + (i-1)/b$.

(assume that new entry will be insert)

Find the i-th key, averaged over $1 \le i \le n$

$$S_n = \frac{1}{n}\sum_{i=1}^{n}\left\{1 + \frac{i-1}{b}\right\} \approx 1 + \frac{\alpha}{2}$$

Expected performance of linear probing

- Loading density of a hash table: $\alpha = n/b$
  - n: number of data items in hash table
  - b: number of buckets

When n is large and $0 \leq \alpha \leq 1$,

- U_n = expected number of key comparisons in an unsuccessful search

$$U_n \approx \frac{1}{2}\left\{1 + \frac{1}{(1-\alpha)^2}\right\}$$

- S_n = expected number of key comparisons in a successful search.

$$S_n \approx \frac{1}{2}\left\{1 + \frac{1}{1-\alpha}\right\}$$

Proven by Knuth, 1962

$\alpha \leq 0.75$ is recommended.

α	S_n	U_n
0.50	1.5	2.5
0.75	2.5	8.5
0.90	5.5	50.7

Summary

- What is hash table?
- Choices of hash functions
- Overflow handling
  - Chaining
  - Open addressing
  - Expected performance