DATA STRUCTURE

Chapter 9: Hashing

Problem Set 3 (Exam Oriented, with Solutions)

9.1 Hash Functions | 9.2 Deleting Items from Hash Tables

This set contains a comparison of the three probing methods, rehashing with a threshold, table sizing, wrap-around deletion, chaining programs and a True/False section with reasons. Empty slots are shown blank and D means a DELETED slot (tombstone).

Section A: Problems on 9.1 Hash Functions

Problem 1. Using h(k) = k mod 23, find the addresses of 46, 69, 150, 275 and 321. Which keys collide?

Solution:

- 46 mod 23 = 0 (23 × 2 = 46)
- 69 mod 23 = 0 (23 × 3 = 69)
- 150 mod 23 = 12 (23 × 6 = 138)
- 275 mod 23 = 22 (23 × 11 = 253)
- 321 mod 23 = 22 (23 × 13 = 299)

Answer: the addresses are 0, 0, 12, 22 and 22. There are two collisions: **46 and 69** at address 0, and **275 and 321** at address 22.

Problem 2. Using the multiplication method with A = 0.6180339887 and m = 8, find the addresses of the keys 10, 20, 30 and 40. What property of the method is seen in the result?

Solution:

- k = 10: 10 × A = 6.18034, fractional part = 0.18034, 8 × 0.18034 = 1.443, so h = **1**.
- k = 20: 20 × A = 12.36068, fractional part = 0.36068, 8 × 0.36068 = 2.885, so h = **2**.
- k = 30: 30 × A = 18.54102, fractional part = 0.54102, 8 × 0.54102 = 4.328, so h = **4**.
- k = 40: 40 × A = 24.72136, fractional part = 0.72136, 8 × 0.72136 = 5.771, so h = **5**.

Answer: the addresses are 1, 2, 4 and 5. Although the keys are in a regular sequence (spaced by 10), the addresses are spread over the table without any collision. Because of the golden-ratio constant A, evenly spaced keys are scattered well.

Problem 3. Find the mid-square hash address (middle two digits of the square, table size 100) for the keys 53, 76, 91 and 38.

Solution:

- 53^2 = 2809. The digits are 2 **8 0** 9, so h = **80**.
- 76^2 = 5776. The digits are 5 **7 7** 6, so h = **77**.
- 91^2 = 8281. The digits are 8 **2 8** 1, so h = **28**.
- 38^2 = 1444. The digits are 1 **4 4** 4, so h = **44**.

Answer: the addresses are 80, 77, 28 and 44.

Problem 4. The key 9812345678 is hashed into a table of size 1000 using folding, with 3-digit parts taken from the left. Find the address using (a) shift folding and (b) boundary folding.

Solution:
The parts are 981, 234, 567 and 8.

- (a) Shift folding: 981 + 234 + 567 + 8 = 1790. Ignoring the carry (keeping 3 digits), h = **790**.
- (b) Boundary folding: reverse the 2nd and 4th parts (234 becomes 432, and 8 stays 8). 981 + 432 + 567 + 8 = 1988. Ignoring the carry, h = **988**.

Problem 5. A table of size 13 uses h(k) = k mod 13 and linear probing. Insert 26, 39, 14, 27, 40, 15 and 52. Show the table, the number of probes for each insertion, the total probes and the load factor. Then find the number of probes for an unsuccessful search of (a) 65 and (b) 20.

Solution:
The home addresses are: 26 → 0, 39 → 0, 14 → 1, 27 → 1, 40 → 1, 15 → 2, 52 → 0.

- 26: slot 0 is free. Placed at slot 0 (1 probe).
- 39: slot 0 is occupied, so slot 1 (2 probes).
- 14: slot 1 is occupied, so slot 2 (2 probes).
- 27: slots 1, 2 are occupied, so slot 3 (3 probes).
- 40: slots 1, 2, 3 are occupied, so slot 4 (4 probes).
- 15: slots 2, 3, 4 are occupied, so slot 5 (4 probes).
- 52: slots 0 to 5 are occupied, so slot 6 (7 probes).

Index	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
Key	26	39	14	27	40	15	52						

- **Total probes** = 1 + 2 + 2 + 3 + 4 + 4 + 7 = **23** (average 23 / 7 ≈ 3.29 per insertion).
- **Load factor** α = 7 / 13 ≈ **0.54**.
- **(a) Search 65:** h(65) = 0. Slots 0 to 6 are occupied and none matches. Slot 7 is EMPTY. So **8 probes**.
- **(b) Search 20:** h(20) = 7. Slot 7 is EMPTY, so the search ends in **1 probe**.

An unsuccessful search takes very different times depending on where the key lands. A key that hashes into the cluster (slots 0 to 6) is slow, while a key that hashes outside the cluster is fast.

Problem 6. Insert the keys 10, 17, 24 and 31 in this order into a table of size 7 with h(k) = k mod 7 using (a) linear probing, (b) quadratic probing with $(h + i^2)$ mod 7 and (c) double hashing with h2(k) = 5 − (k mod 5). Compare the total number of probes.

Solution:
All four keys have h(k) = 3.

(a) Linear probing: 10 goes to slot 3; 17 to slot 4; 24 to slot 5; 31 to slot 6. Probes: 1 + 2 + 3 + 4 = **10**.

Index	0	1	2	3	4	5	6
Key				10	17	24	31

(b) Quadratic probing:

- 10 goes to slot 3.
- 17: slot 3 is occupied. i = 1: (3 + 1) = 4. Placed at slot 4.
- 24: slots 3, 4 are occupied. i = 2: (3 + 4) mod 7 = 0. Placed at slot 0.
- 31: slots 3, 4, 0 are occupied. i = 3: (3 + 9) mod 7 = 5. Placed at slot 5.

Probes: 1 + 2 + 3 + 4 = **10**.

Index	0	1	2	3	4	5	6
Key	24			10	17	31	

(c) Double hashing:

- 10 goes to slot 3.
- 17: h2 = 5 − (17 mod 5) = 5 − 2 = 3. Slot 3 is occupied. i = 1: (3 + 3) = 6. Placed at slot 6.
- 24: h2 = 5 − (24 mod 5) = 5 − 4 = 1. Slot 3 is occupied. i = 1: (3 + 1) = 4. Placed at slot 4.
- 31: h2 = 5 − (31 mod 5) = 5 − 1 = 4. Slot 3 is occupied. i = 1: (3 + 4) mod 7 = 0. Placed at slot 0.

Probes: 1 + 2 + 2 + 2 = **7**.

Index	0	1	2	3	4	5	6
Key	31			10	24		17

Comparison: linear and quadratic probing both need 10 probes, while double hashing needs only 7. In double hashing each key has its own step size, so the keys that collide at slot 3 go to different places. In linear probing they follow each other in one cluster, and in quadratic probing they all follow the same jump pattern.

Problem 7. A table starts with size 5 and uses h(k) = k mod m with linear probing. Whenever the load factor exceeds 0.6 after an insertion, the table is rehashed into a new table of size 2m + 1 (11 for m = 5). Insert 7, 12, 19, 26 and then 40 and show the tables.

Solution:

- 7: 7 mod 5 = 2. Placed at slot 2. α = 1 / 5 = 0.2.
- 12: 12 mod 5 = 2, occupied, so slot 3. α = 2 / 5 = 0.4.
- 19: 19 mod 5 = 4. Placed at slot 4. α = 3 / 5 = 0.6, which does not exceed 0.6.
- 26: 26 mod 5 = 1. Placed at slot 1. α = 4 / 5 = 0.8, which **exceeds 0.6**, so rehashing is done.

Index	0	1	2	3	4
Key		26	7	12	19

Rehashing into size 11: take the keys in the order of the old table (26, 7, 12, 19) and recompute k mod 11: 26 → 4, 7 → 7, 12 → 1, 19 → 8.

Index	0	1	2	3	4	5	6	7	8	9	10
Key		12			26			7	19		

The load factor drops to 4 / 11 ≈ 0.36. **Insert 40:** 40 mod 11 = 7, which is occupied by 7. Slot 8 is occupied by 19, and slot 9 is free. So 40 is placed at slot **9** (3 probes). Now α = 5 / 11 ≈ 0.45, which does not exceed 0.6, so no rehashing is needed.

Problem 8. A chained hash table has to store n = 1000 keys so that the average chain length does not exceed 0.5. What table size should be used? Also find the average chain length for a table of size 400.

Solution:

- The average chain length is the load factor α = n / m.
- For α ≤ 0.5: m ≥ 1000 / 0.5 = 2000. A prime is preferred, so the next prime above 2000, which is **2003**, is chosen. Then α = 1000 / 2003 ≈ 0.499.
- For m = 400: α = 1000 / 400 = **2.5**, so the average chain has 2.5 keys. Chaining still works at α > 1, but each search needs to traverse a longer chain.

Problem 9. Write C code for the hash table with separate chaining (division method, table size 5 for the example): the node structure, insertion at the head of the chain, and search. Trace the insertion of 5, 10 and 15.

Solution:

```
#define M 5
struct node {
    int key;
    struct node *next;
};
struct node *table[M];   /* all NULL at the start */

void insert(int k) {
    int idx = k % M;
    struct node *n = malloc(sizeof(struct node));
    n->key = k;
    n->next = table[idx];    /* new node goes at the head */
    table[idx] = n;
}

struct node *search(int k) {
    struct node *p = table[k % M];
    while (p != NULL && p->key != k)
        p = p->next;
    return p;                /* NULL if not found */
}
```

Trace: 5, 10 and 15 all give index 0. After inserting 5, chain 0 is 5. After inserting 10 at the head, it is 10 → 5. After inserting 15 at the head, it is 15 → 10 → 5.

- Insertion at the head takes **O(1)** time. Search takes O(1 + α) on average and O(n) in the worst case.

Section B: Problems on 9.2 Deleting Items from Hash Tables

Problem 10. Write a C function to delete a key from the chained hash table of Problem 9. Trace it on the chain 15 → 10 → 5 for deleting 10 and then 15.

Solution:

```
int delete_key(int k) {
    int idx = k % M;
    struct node *cur = table[idx], *prev = NULL;
    while (cur != NULL && cur->key != k) {
        prev = cur;
        cur = cur->next;
    }
    if (cur == NULL) return 0;        /* key not found */
    if (prev == NULL)
        table[idx] = cur->next;       /* deleting the head */
    else
        prev->next = cur->next;       /* bypass the node */
    free(cur);
    return 1;
}
```

- **Delete 10:** idx = 0. cur = 15, no match (prev = 15). cur = 10, match, and prev is not NULL. So 15 is linked directly to 5, and the node 10 is freed. The chain is now 15 → 5.
- **Delete 15:** idx = 0. cur = 15 is a match and prev is NULL, so it is the head node. The slot table[0] is set to point to 5, and the node 15 is freed. The chain is now just 5.
- Two pointers (cur and prev) are needed because a singly linked list cannot go back to the previous node. **Time:** O(1 + α) on average.

Problem 11. A table of size 11 uses double hashing with h1(k) = k mod 11 and h2(k) = 7 − (k mod 7), with tombstones. Insert 25, 36, 14, 47 and 20, then delete 36, search 20, and insert 31. Show why the slot of 36 must not be emptied.

Solution:
The probe sequence is (h1(k) + i × h2(k)) mod 11.

- 25: h1 = 3. Placed at slot **3**.
- 36: h1 = 3, occupied. h2 = 7 − (36 mod 7) = 7 − 1 = 6. i = 1: (3 + 6) = 9. Placed at slot **9**.
- 14: h1 = 3, occupied. h2 = 7 − (14 mod 7) = 7. i = 1: (3 + 7) = 10. Placed at slot **10**.
- 47: h1 = 3, occupied. h2 = 7 − (47 mod 7) = 7 − 5 = 2. i = 1: (3 + 2) = 5. Placed at slot **5**.
- 20: h1 = 20 mod 11 = 9, occupied (36). h2 = 7 − (20 mod 7) = 7 − 6 = 1. i = 1: slot 10, occupied (14). i = 2: (9 + 2) mod 11 = 0. Placed at slot **0** (3 probes).

Index	0	1	2	3	4	5	6	7	8	9	10
Key	20			25		47				36	14

Delete 36: h1 = 3 (25, no). i = 1: slot 9 (36, found). Mark slot 9 as D.

Index	0	1	2	3	4	5	6	7	8	9	10
Key	20			25		47				D	14

Search 20: h1 = 9. Slot 9 is D, so continue. i = 1: slot 10 (14, no). i = 2: slot 0 (20, found). **3 probes.**

If slot 9 had been made EMPTY: the search for 20 would stop at slot 9 (the first probe) and wrongly report "not found", although 20 is in slot 0.

Search 14 is unaffected: its sequence is slot 3 (25, no), then slot 10 (14, found), 2 probes. In double hashing only the keys whose probe sequence passes through the deleted slot are affected.

Insert 31: h1 = 31 mod 11 = 9. Slot 9 is D (note it as reusable and continue). h2 = 7 − (31 mod 7) = 7 − 3 = 4. i = 1: (9 + 4) mod 11 = 2, which is EMPTY. So 31 is not present, and it is inserted in the tombstone slot, **slot 9**.

Index	0	1	2	3	4	5	6	7	8	9	10
Key	20			25		47				31	14

Problem 12. A table of size 8 uses h(k) = k mod 8 with linear probing (with wrap-around). Insert 15, 23, 31 and 9, then delete 15 using backward shift deletion.

Solution:

- 15: 15 mod 8 = 7. Placed at slot 7.
- 23: home 7 is occupied. The next slot wraps around to slot 0. Placed at slot 0.
- 31: home 7 is occupied, slot 0 is occupied. Placed at slot 1.
- 9: home 1 is occupied (31). Placed at slot 2.

Index	0	1	2	3	4	5	6	7
Key	23	31	9					15

The home addresses are: 23 → 7, 31 → 7, 9 → 1. **Delete 15** (slot 7). The hole is at slot 7. Move a record into the hole if its home address is at or before the hole (counting cyclically, so the hole lies between the home and the current slot).

- Slot 0 holds 23 (home 7). The hole 7 is at its home, so move 23 to slot 7. The hole is now at slot 0.
- Slot 1 holds 31 (home 7). Going around the table from 7, the hole 0 comes before slot 1, so move 31 to slot 0. The hole is now at slot 1.

- Slot 2 holds 9 (home 1). Home 1 is at the hole, so move 9 to slot 1. The hole is now at slot 2.
- Slot 3 is EMPTY, so stop.

Index	0	1	2	3	4	5	6	7
Key	31	9						23

Check: search 31: h = 7, slot 7 (23, no), slot 0 (31, found). Search 9: h = 1, slot 1 (9, found). The wrap-around is the difficult part of this method, so it must be handled with modulo arithmetic.

Problem 13. A chained hash table stores n = 5000 keys in m = 1000 slots. Find the expected comparisons for deleting a key that is present and for a key that is absent (use 1 + α / 2 for a present key and α for an absent key). What table size (a prime near it) is needed to make the absent-key cost at most 2?

Solution:

- Load factor α = 5000 / 1000 = **5**.
- **Key present:** about 1 + α / 2 = 1 + 2.5 = **3.5** comparisons.
- **Key absent:** the whole chain has to be examined, about α = **5** comparisons.
- For the absent-key cost to be at most 2, α ≤ 2, so m ≥ 5000 / 2 = 2500. Choosing the prime **2503** gives α ≈ 1.998, so the absent-key cost is about 2 and the present-key cost is about 1 + 1 = 2 comparisons.

Conclusion: the cost of deletion in chaining depends only on the load factor, and it can be reduced by using a bigger table.

Section C: True or False (with Reason)

Problem 14. State whether each statement is true or false, with a one-line reason.

Solution:

- (1) The load factor of a table using open addressing can be greater than 1. **False.** All keys are stored in the table itself, so n cannot exceed m and α ≤ 1.
- (2) Deletion in separate chaining needs tombstones. **False.** A node is simply unlinked from its linked list.
- (3) In open addressing, a search must stop when it reaches a DELETED slot. **False.** It must continue; it stops only at an EMPTY slot or when the key is found.
- (4) Rehashing removes the tombstones from a table. **True.** Only the live keys are inserted into the new table.
- (5) The division method with m = 2^p uses only the lowest p bits of the key. **True.** k mod 2^p is just the last p bits, so the higher bits are ignored.
- (6) Double hashing largely avoids both primary and secondary clustering. **True.** The step size depends on the key, so keys with the same home address follow different paths.
- (7) The worst-case search time in a hash table is O(1). **False.** If all keys collide, the search takes O(n).

- (8) A hash function may give a different value for the same key on different calls. **False.** It must be deterministic, or the key could not be found again.
- (9) Backward shift deletion leaves no tombstones in the table. **True.** The records of the cluster are moved back to fill the hole.
- (10) A new key may be stored in a DELETED slot during insertion. **True.** But only after the probe sequence shows that the key is not already present.

Section D: Practice Problems (with Answers)

Solve these yourself and then compare with the answers.

Problem 15. Find h(k) = k mod 29 for 58, 87, 100 and 200.

> **Answer:** 0, 0, 13 and 26 (58 = 29 × 2; 87 = 29 × 3; 100 = 29 × 3 + 13; 200 = 29 × 6 + 26). Keys 58 and 87 collide.

Problem 16. Using A = 0.6180339887 and m = 20, find the multiplication method address of k = 10.

> **Answer:** 10 × A = 6.18034; fractional part = 0.18034; 20 × 0.18034 = 3.61, so h = 3.

Problem 17. Find the mid-square address of the key 66 for a table of size 100 (middle two digits of the square).

> **Answer:** $66^2 = 4356$; the middle two digits are 35, so the address is 35.

Problem 18. Key 4567891 is split into 2-digit parts (45, 67, 89, 1). Find the shift folding address for a table of size 100.

> **Answer:** 45 + 67 + 89 + 1 = 202; ignoring the carry gives 02, so the address is 2.

Problem 19. Insert 18, 27 and 36 into a table of size 9 with h(k) = k mod 9 using linear probing.

> **Answer:** All three hash to 0, so they go to slots 0, 1 and 2. (All the keys are multiples of 9, which shows why a poor m gives clustering.)

Problem 20. Insert 11, 22 and 33 into a table of size 11 using quadratic probing, $(h + i^2)$ mod 11.

> **Answer:** All hash to 0. 11 goes to slot 0; 22 to slot (0 + 1) = 1; 33 to slot (0 + 4) = 4.

Problem 21. Insert 3, 6, 9, 4 and 7 into a table of size 3 using chaining, h(k) = k mod 3, then delete 9. How many comparisons does the deletion need?

> **Answer:** Chain 0: 3 → 6 → 9; chain 1: 4 → 7. Deleting 9 needs 3 comparisons (3, 6, 9). Chain 0 becomes 3 → 6.

Problem 22. Table size 5, h(k) = k mod 5, linear probing with tombstones. Insert 5, 10, 15, 20; delete 5; search 20; then insert 25. State the probes for the search and the slot of 25.

> **Answer:** The keys go to slots 0, 1, 2, 3. After deleting 5, slot 0 is D. Searching 20: slots 0 (D), 1, 2, 3 (found), so 4 probes. For 25 (home 0): slot 0 is D (reusable), slots 1, 2, 3 are occupied and slot 4 is EMPTY, so 25 is absent and it is placed at slot 0.