DATA STRUCTURE

Chapter 9: Hashing

Problem Set 2 (Exam Oriented, with Solutions)

9.1 Hash Functions | 9.2 Deleting Items from Hash Tables

This set contains new problems and more exam-style questions, including worst-case analysis, limitations of probing methods and C programs. Solutions are worked step by step. Slots that are empty in a table 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 17, find the hash addresses of 34, 51, 89, 105, 212 and 300. Comment on the keys that collide.

Solution:

- 34 mod 17 = 0 (17 × 2 = 34)
- 51 mod 17 = 0 (17 × 3 = 51)
- 89 mod 17 = 4 (17 × 5 = 85)
- 105 mod 17 = 3 (17 × 6 = 102)
- 212 mod 17 = 8 (17 × 12 = 204)
- 300 mod 17 = 11 (17 × 17 = 289)

Answer: the addresses are 0, 0, 4, 3, 8 and 11. Keys **34 and 51** collide at address 0. Both are multiples of the table size, so they leave remainder 0. Even a prime m cannot help when many keys are multiples of m, so the key pattern should be considered when m is chosen.

Problem 2. Using the multiplication method with A = 0.6180339887 and m = 16, find the hash addresses of the keys 21, 32 and 47. Why can m be a power of 2 in this method?

Solution:

- k = 21: 21 × A = 12.97871, fractional part = 0.97871, 16 × 0.97871 = 15.659, so h = **15**.
- k = 32: 32 × A = 19.77709, fractional part = 0.77709, 16 × 0.77709 = 12.433, so h = **12**.
- k = 47: 47 × A = 29.04760, fractional part = 0.04760, 16 × 0.04760 = 0.762, so h = **0**.

Answer: the addresses are 15, 12 and 0. In the multiplication method the fractional part of k·A depends on all the bits of the key, and it is then scaled by m. The value of m is therefore **not critical**, and a power of 2 (which makes the scaling a simple bit shift) can be used. In the division method a power of 2 is a poor choice, since only the lowest bits of the key are used.

Problem 3. Find the hash address of the keys 456, 789 and 321 using the mid-square method for a table of size 100 (take the middle two digits of the square).

Solution:

- 456² = 207936. The digits are 2 0 **7 9** 3 6, so h = **79**.
- 789² = 622521. The digits are 6 2 **2 5** 2 1, so h = **25**.
- 321² = 103041. The digits are 1 0 **3 0** 4 1, so h = **30**.

Answer: the addresses are 79, 25 and 30. (Each square has 6 digits, so the 3rd and 4th digits are the middle ones.)

Problem 4. The key 123456789012 is to be hashed into a table of size 10000 using folding with 4-digit parts. Find the address by (a) shift folding and (b) boundary folding.

Solution:
The parts are 1234, 5678 and 9012.

- (a) Shift folding: 1234 + 5678 + 9012 = 15924. Ignoring the carry (keeping 4 digits), h = **5924**.
- (b) Boundary folding: reverse the middle part, 5678 becomes 8765. 1234 + 8765 + 9012 = 19011. Ignoring the carry, h = **9011**.

Problem 5. A table of size 11 uses h(k) = k mod 11 and linear probing. Insert 12, 23, 34, 45 and 56. Show the table, the total probes, and the number of probes needed to search for (a) 56 and (b) 100 (unsuccessful). What does this show?

Solution:
All the keys have h(k) = 1 (12 mod 11 = 1, 23 mod 11 = 1, and so on).

- 12 goes to slot 1 (1 probe).
- 23: slot 1 is occupied, so slot 2 (2 probes).
- 34: slots 1, 2 are occupied, so slot 3 (3 probes).
- 45: slots 1, 2, 3 are occupied, so slot 4 (4 probes).
- 56: slots 1, 2, 3, 4 are occupied, so slot 5 (5 probes).

Index	0	1	2	3	4	5	6	7	8	9	10
Key		12	23	34	45	56					

- **Total probes** = 1 + 2 + 3 + 4 + 5 = **15**.
- **(a) Search 56:** slots 1, 2, 3, 4, 5, so **5 probes**.
- **(b) Search 100:** h(100) = 1. Slots 1 to 5 are occupied and do not match. Slot 6 is EMPTY, so the search stops after **6 probes** and reports not found.

Conclusion: although the load factor is only 5 / 11 ≈ 0.45, all the keys formed one long cluster. This is **primary clustering**, and it shows that the search time of linear probing can degrade to O(n) in the worst case.

Problem 6. A table of size 8 uses quadratic probing, h(k, i) = (k mod 8 + i^2) mod 8. Insert 8, 16, 24 and 32. What happens? Explain the reason and show how a prime table size (m = 11, keys 11, 22, 33, 44) removes the problem.

Solution:
All four keys have h(k) = 0. The probe offsets i^2 mod 8 for i = 0, 1, 2, 3, 4, 5, 6, 7 are 0, 1, 4, 1, 0, 1, 4, 1. So the slots that can ever be probed are **0, 1 and 4 only**.

- 8 goes to slot 0.
- 16: slot 0 is occupied, i = 1 gives slot 1. Placed at slot 1.
- 24: slots 0 and 1 are occupied, i = 2 gives slot 4. Placed at slot 4.

- 32: slots 0, 1 and 4 are all occupied, and all the other values of i repeat the same three slots. **32 cannot be inserted**, although slots 2, 3, 5, 6 and 7 are empty.

Index	0	1	2	3	4	5	6	7
Key	8	16			24			

With m = 11: the keys 11, 22, 33 and 44 all have h(k) = 0. The offsets i^2 mod 11 are 0, 1, 4, 9, 5, 3, ... so 11 goes to slot 0, 22 to slot 1, 33 to slot 4 and 44 to slot 9. All four keys are inserted.

Conclusion: for quadratic probing the table size should be a **prime**, and the table should be **at most half full**. Then an empty slot is always found.

Problem 7. A table of size 13 uses double hashing with h1(k) = k mod 13 and h2(k) = 1 + (k mod 11). Insert 18, 41, 22, 44, 59 and 31 in this order.

Solution:
The probe sequence is (h1(k) + i × h2(k)) mod 13.

- 18: h1 = 5. Placed at slot **5**.
- 41: h1 = 41 mod 13 = 2. Placed at slot **2**.
- 22: h1 = 22 mod 13 = 9. Placed at slot **9**.
- 44: h1 = 44 mod 13 = 5, occupied. h2 = 1 + (44 mod 11) = 1 + 0 = 1. i = 1: (5 + 1) = 6, free. Placed at slot **6**.
- 59: h1 = 59 mod 13 = 7. Placed at slot **7**.
- 31: h1 = 31 mod 13 = 5, occupied. h2 = 1 + (31 mod 11) = 1 + 9 = 10. i = 1: (5 + 10) mod 13 = 2, occupied. i = 2: (5 + 20) mod 13 = 12, free. Placed at slot **12** (3 probes).

Index	0	1	2	3	4	5	6	7	8	9	10	11	12
Key			41			18	44	59		22			31

Note: h2(k) = 1 + (k mod 11) is always between 1 and 11, so it is never 0. With a prime table size 13, the step size is always relatively prime to m, so the probe sequence can visit every slot.

Problem 8. Insert the keys 15, 11, 27, 8, 23, 35 and 41 into a table of size 6 using separate chaining with h(k) = k mod 6 (new keys are added at the end of the chain). Find the load factor, the number of empty slots, the longest chain, the average number of comparisons for a successful search, and the comparisons for an unsuccessful search of 17.

Solution:

- 15 mod 6 = 3; 11 mod 6 = 5; 27 mod 6 = 3; 8 mod 6 = 2; 23 mod 6 = 5; 35 mod 6 = 5; 41 mod 6 = 5.

Index	Chain
0	(empty)
1	(empty)
2	8
3	15 → 27
4	(empty)
5	11 → 23 → 35 → 41

- **Load factor** α = 7 / 6 ≈ **1.17**.
- **Empty slots:** 3 (slots 0, 1 and 4).
- **Longest chain:** 4 (slot 5).
- **Successful search:** comparisons for each key are 8 → 1; 15 → 1, 27 → 2; 11 → 1, 23 → 2, 35 → 3, 41 → 4. Total = 1 + 3 + 10 = 14. Average = 14 / 7 = **2**.
- **Unsuccessful search of 17:** h(17) = 17 mod 6 = 5. The whole chain 5 has to be compared (4 keys), so **4 comparisons**.

Problem 9. Write C functions to insert and search in a hash table of size 11 using the division method and linear probing. Show how they work by giving their time complexity.

Solution:

```
#define M 11
#define EMPTY -1
int table[M];   /* all slots set to EMPTY at the start */

int hash(int k) { return k % M; }

int insert(int k) {
    int i, idx;
    for (i = 0; i < M; i++) {
        idx = (hash(k) + i) % M;      /* i-th probe */
        if (table[idx] == EMPTY) {
            table[idx] = k;
            return idx;               /* inserted */
        }
    }
    return -1;                        /* table is full */
}

int search(int k) {
    int i, idx;
    for (i = 0; i < M; i++) {
        idx = (hash(k) + i) % M;
        if (table[idx] == EMPTY) return -1;  /* not found */
        if (table[idx] == k)     return idx; /* found */
    }
    return -1;
}
```

- The probe index is (hash(k) + i) % M, which wraps around the end of the table.
- The search stops at the first EMPTY slot, because the key cannot be beyond it.

- **Time complexity:** average O(1) when the load factor is low; worst case O(M) when the keys form one long cluster.

Section B: Problems on 9.2 Deleting Items from Hash Tables

Problem 10. From the chained hash table of Problem 8, delete the keys 8, 35 and 15 in this order. State the comparisons needed for each deletion, show the table afterwards, and give the new load factor.

Solution:

- **Delete 8:** h = 2. Chain 2 holds only 8. It is found in **1 comparison**, and chain 2 becomes empty.
- **Delete 35:** h = 5. Chain 5 is 11 → 23 → 35 → 41. Compare with 11, 23, 35 (**3 comparisons**), then link 23 directly to 41.
- **Delete 15:** h = 3. Chain 3 is 15 → 27. 15 is the first node (**1 comparison**). The head of the chain is changed to point to 27.

Index	Chain
0	(empty)
1	(empty)
2	(empty)
3	27
4	(empty)
5	11 → 23 → 41

Load factor now = 4 / 6 ≈ **0.67**. Total comparisons for the three deletions = 1 + 3 + 1 = 5. Deleting the first node of a chain requires changing the head pointer of the slot, while deleting any other node requires changing the next pointer of its predecessor.

Problem 11. Take the table of Problem 5 (m = 11, keys 12, 23, 34, 45, 56 in slots 1 to 5) with linear probing and tombstones. Perform: delete 23; delete 45; search 56; insert 67; search 78. Show the table after each step where it changes and the probes required. Finally rehash the live keys into a table of size 13.

Solution:

Delete 23: h = 1. Slot 1 (12, no), slot 2 (23, found). Mark slot 2 as D (2 probes).

Delete 45: h = 1. Slot 1 (12), slot 2 (D, continue), slot 3 (34), slot 4 (45, found). Mark slot 4 as D (4 probes).

Index	0	1	2	3	4	5	6	7	8	9	10
Key		12	D	34	D	56					

Search 56: slot 1, slot 2 (D, continue), slot 3, slot 4 (D, continue), slot 5 (found). **5 probes.**

Insert 67: h(67) = 1. Slot 1 (12), slot 2 (D, note it as reusable and continue), slot 3 (34), slot 4 (D), slot 5 (56), slot 6 is EMPTY. So 67 is not already present, and it is inserted in the first D slot, **slot 2**.

Index	0	1	2	3	4	5	6	7	8	9	10
Key		12	67	34	D	56					

Search 78 (unsuccessful): h(78) = 1. Slots 1, 2, 3 do not match. Slot 4 (D, continue), slot 5 (56, no), slot 6 is EMPTY. **6 probes**, not found. The tombstone at slot 4 has cost an extra probe.

Rehashing the live keys 12, 67, 34, 56 into a table of size 13 (h(k) = k mod 13): 12 → 12, 67 → 2, 34 → 8, 56 → 4.

Index	0	1	2	3	4	5	6	7	8	9	10	11	12
Key			67		56				34				12

All tombstones are gone and there is no collision. Searching 78 now needs only 1 probe (78 mod 13 = 0, and slot 0 is empty).

Problem 12. A table of size 7 uses quadratic probing, h(k, i) = (k mod 7 + i^2) mod 7, with tombstones. Insert 14, 21, 28 and 35, delete 21, and then search for 35. Show why the slot of 21 must not be made EMPTY.

Solution:
All keys have h(k) = 0.

- 14 goes to slot 0.
- 21: slot 0 is occupied. i = 1: slot 1. Placed at slot 1.
- 28: slots 0, 1 are occupied. i = 2: slot 4. Placed at slot 4.
- 35: slots 0, 1, 4 are occupied. i = 3: (0 + 9) mod 7 = 2. Placed at slot 2.

Index	0	1	2	3	4	5	6
Key	14	21	35		28		

Delete 21: i = 0: slot 0 (14, no). i = 1: slot 1 (21, found). Mark slot 1 as D.

Index	0	1	2	3	4	5	6
Key	14	D	35		28		

Search 35 with the tombstone: i = 0: slot 0 (14, no). i = 1: slot 1 (D, continue). i = 2: slot 4 (28, no). i = 3: slot 2 (35, found). **Found in 4 probes.**

If slot 1 had been made EMPTY: the search for 35 would stop at i = 1 and wrongly report "not found", while 35 is actually in slot 2. So the tombstone is necessary. Since the probe sequence of quadratic probing jumps between slots (0, 1, 4, 2), the backward shift method is not easy to use, so tombstones are preferred for quadratic probing and double hashing.

Problem 13. A table of size 10 uses h(k) = k mod 10 with linear probing. (a) Insert 20, 30, 21, 41 and 42, then delete 30 using backward shift deletion. (b) In another table the keys 30, 31 and 12 are inserted and 30 is deleted. Show the results.

Solution:

Rule used: after removing a record, examine the next slots up to the first EMPTY slot. A record can be moved into the hole only if its **home address** (h(k)) is at or before the hole. Otherwise it is left in place.

(a) Insertion: 20 goes to slot 0. 30 (home 0) goes to slot 1. 21 (home 1) finds slot 1 occupied and goes to slot 2. 41 (home 1) goes to slot 3. 42 (home 2) finds slots 2 and 3 occupied and goes to slot 4.

Index	0	1	2	3	4	5	6	7	8	9
Key	20	30	21	41	42					

Delete 30 (slot 1). The hole is at slot 1.

- Slot 2 holds 21 (home 1). Home 1 is at the hole, so move 21 to slot 1. The hole is now at slot 2.
- Slot 3 holds 41 (home 1). Home 1 is before the hole, so move 41 to slot 2. The hole is now at slot 3.
- Slot 4 holds 42 (home 2). Home 2 is before the hole, so move 42 to slot 3. The hole is now at slot 4.
- Slot 5 is EMPTY, so stop.

Index	0	1	2	3	4	5	6	7	8	9
Key	20	21	41	42						

Check: search 42: h = 2, slot 2 (41, no), slot 3 (42, found).

(b) Keys 30, 31 and 12 go to slots 0, 1 and 2 (their home addresses). Delete 30 (slot 0), so the hole is at slot 0.

- Slot 1 holds 31 with home 1. The home is **after** the hole, so it must **not** be moved (it would then be before its home address and the search for 31 would fail).
- Slot 2 holds 12 with home 2, which is also after the hole, so it stays.
- Slot 3 is EMPTY, so stop. Only slot 0 becomes empty, and 31 and 12 remain in slots 1 and 2.

Problem 14. Write C functions for deletion and for insertion (with reuse of deleted slots) in an open addressing table using linear probing and tombstones. Trace them on a table of size 5 for: insert 5, 10, 15; delete 10; search 15.

Solution:

```
#define M 11
#define EMPTY   -1
#define DELETED -2
int table[M];      /* initially all EMPTY */
```

```
int search(int k) {
    int i, idx;
    for (i = 0; i < M; i++) {
        idx = (k % M + i) % M;
        if (table[idx] == EMPTY) return -1; /* stop only here */
        if (table[idx] == k)     return idx;
        /* DELETED slot: just continue probing */
    }
    return -1;
}

int delete_key(int k) {
    int idx = search(k);
    if (idx == -1) return 0;          /* key not present */
    table[idx] = DELETED;             /* put a tombstone */
    return 1;
}

int insert(int k) {
    int i, idx;
    if (search(k) != -1) return -1;  /* already present */
    for (i = 0; i < M; i++) {
        idx = (k % M + i) % M;
        if (table[idx] == EMPTY || table[idx] == DELETED) {
            table[idx] = k;           /* reuse tombstone */
            return idx;
        }
    }
    return -1;                        /* table is full */
}
```

Trace (M = 5): insert 5, 10 and 15 (all with home 0) go to slots 0, 1 and 2. Delete 10: search finds it at slot 1, which is marked DELETED. Now the table is [5, D, 15, EMPTY, EMPTY]. Search 15: slot 0 (5, no), slot 1 (DELETED, continue), slot 2 (15, found). The function returns 2 after 3 probes.

- The search skips DELETED slots and stops only at EMPTY. DELETED (−2) never equals a valid key, so no special test is needed.
- insert first calls search, so that a key is not stored twice when a tombstone occurs before its existing copy.

Problem 15. A table of size 1000 (linear probing) holds 400 live keys and 300 tombstones. Find the actual and the effective load factor, and estimate the expected probes for successful and unsuccessful searches in both cases. What is the effect of rehashing into a table of size 1000 and of size 2000?

Solution:

For linear probing: successful ≈ ½ (1 + 1 / (1 − α)); unsuccessful ≈ ½ (1 + 1 / (1 − α)²). A search cannot stop at a tombstone, so tombstones behave approximately like occupied slots.

- **Actual load factor** = 400 / 1000 = **0.4**.
- **Effective load factor** = (400 + 300) / 1000 = **0.7**.

- **With α = 0.4 (no tombstones):** successful = ½ (1 + 1.667) ≈ **1.33**; unsuccessful = ½ (1 + 2.778) ≈ **1.89**.
- **With α = 0.7 (tombstones present):** successful = ½ (1 + 3.333) ≈ **2.17**; unsuccessful = ½ (1 + 11.11) ≈ **6.06**.

Rehashing: in a table of the same size 1000, all 300 tombstones are cleared and the load factor becomes 0.4, so the unsuccessful search drops from about 6.06 to 1.89 probes. In a table of size 2000 the load factor is 0.2 and the unsuccessful search needs about ½ (1 + 1 / 0.64) ≈ 1.28 probes. **Conclusion:** many deletions make unsuccessful searches about three times slower, so the table should be rehashed when tombstones accumulate.

Section C: Practice Problems (with Answers)

Solve these yourself and then compare with the answers.

Problem 16. Find h(k) = k mod 19 for 38, 57, 100 and 250.

> **Answer:** 0, 0, 5 and 3 (38 = 19 × 2; 57 = 19 × 3; 100 = 19 × 5 + 5; 250 = 19 × 13 + 3). Keys 38 and 57 collide.

Problem 17. Using the multiplication method with A = 0.6180339887 and m = 100, find h(1000).

> **Answer:** 1000 × A = 618.0339887; fractional part = 0.0339887; 100 × 0.0339887 = 3.39, so h = 3.

Problem 18. Find the mid-square address of the key 57 for a table of size 100 (middle two digits of the square).

> **Answer:** $57^2 = 3249$; the middle two digits are 24, so the address is 24.

Problem 19. Key 5551234 is split into 2-digit parts from the left (55, 51, 23, 4). Find the address for a table of size 100 using shift folding and boundary folding.

> **Answer:** Shift: 55 + 51 + 23 + 4 = 133, ignoring the carry gives 33. Boundary: 55 + 15 + 23 + 4 = 97 (2nd and 4th parts reversed), so 97.

Problem 20. Insert 16, 24 and 32 into a table of size 8 with h(k) = k mod 8 using linear probing.

> **Answer:** All three hash to 0, so they go to slots 0, 1 and 2. (m = 8 is a poor choice, since all the keys are multiples of 8.)

Problem 21. A table of size 7 uses double hashing with h1 = k mod 7 and h2 = 5 − (k mod 5). Insert 12 and then 19.

> **Answer:** 12 goes to slot 5. 19: h1 = 5 (occupied), h2 = 5 − 4 = 1, so (5 + 1) mod 7 = 6. 19 goes to slot 6.

Problem 22. Insert 4, 8, 12, 5 and 9 into a table of size 4 using separate chaining, h(k) = k mod 4. Then delete 8. How many comparisons are needed to delete 8?

> **Answer:** Chain 0: 4 → 8 → 12; chain 1: 5 → 9. Deleting 8 needs 2 comparisons (4, then 8). Chain 0 becomes 4 → 12.

Problem 23. Table size 5, h(k) = k mod 5, linear probing with tombstones. Insert 6, 11, 16; delete 11; search 16. How many probes does the search take?

> **Answer:** 6, 11, 16 go to slots 1, 2, 3. After deleting 11, slot 2 is DELETED. Searching 16: slot 1 (6), slot 2 (D, continue), slot 3 (16, found), so 3 probes.