UNIVERSITY of WASHINGTON

CSE332 Section 5

Hashing

Originally by James Richie & Sulaeman
Revised by Rubee Zhao & Vicky Ye



Announcement

- **Midterm: Monday, Nov 9, 2026 5pm-6pm Smith 120 & 205**
- Content: Everything up to AVL (no hashing and sorting, no graphs)
  - Past exams posted on course website
  - Come to office hours if you have questions about **anything**!
- Review session: **Friday, Nov 6** in lecture



Warm -Up

```
public AVLTree fillUniqueTree(List lst){
    AVLTree avl = AVLTree();
    for(int i = 0; i < lst.size(); i++){
        avl.insert(lst.get(i), lst.get(i));
    }
    return avl;
}
```

What is the running time if the all elements in lst are unique? O(nlogn)

What is the running time if all elements in lst are the same? O(n)



Hashing



Collision Resolution

A collision occurs when two keys map onto the same location in a hash table.

Impossible to eliminate since the number of possible keys exceeds table size.

There are multiple ways to resolve conflicts:

1. Separate Chaining

All elements with keys that map to the same table location are kept in a linked list.

1. Open Addressing

If the slot is occupied, we **probe** the next slot.

On the ith probe, we check the slot with index (h(key) + f(i)) % TableSize.

Linear Probing: f(i) = i

Quadratic Probing: f(i) = i^2

Double Hashing: f(i) = i · g(key)



Problem 1a

- **Linear Probing**
  - **ith probe: (h(key) + i) % TableSize**



Problem 1a: insertion

Insert 7, 9, 48, 8, 37, 57
into an empty table.

Linear Probing

ith probe: (h(key) + i) % TableSize

idx	elem
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	



Problem 1a: insert 7

Insert **7**, 9, 48, 8, 37, 57
into an empty table.

(h(7) + 0) % 10 = 7

Linear Probing

ith probe: (h(key) + i) % TableSize

idx	elem
0	
1	
2	
3	
4	
5	
6	
7	7
8	
9	



Problem 1a: insert 9

Insert 7, **9**, 48, 8, 37, 57
into an empty table.

(h(9) + 0) % 10 = 9

Linear Probing

ith probe: (h(key) + i) % TableSize

idx	elem
0	
1	
2	
3	
4	
5	
6	
7	7
8	
9	9



Problem 1a: insert 48

Insert 7, 9, **48**, 8, 37, 57
into an empty table.

(h(48) + 0) % 10 = 8

Linear Probing

ith probe: (h(key) + i) % TableSize

idx	elem
0	
1	
2	
3	
4	
5	
6	
7	7
8	48
9	9



Problem 1a: insert 8

Insert 7, 9, 48, **8**, 37, 57
into an empty table.

(h(8) + 0) % 10 = 8

(h(8) + 1) % 10 = 9

(h(8) + 2) % 10 = 0

Linear Probing

ith probe: (h(key) + i) % TableSize

idx	elem
0	8
1	
2	
3	
4	
5	
6	
7	7
8	48
9	9



Problem 1a: insert 37

Insert 7, 9, 48, 8, **37**, 57
into an empty table.

(h(37) + 0) % 10 = 7

(h(37) + 1) % 10 = 8

(h(37) + 2) % 10 = 9

(h(37) + 3) % 10 = 0

(h(37) + 4) % 10 = 1

Linear Probing

ith probe: (h(key) + i) % TableSize

idx	elem
0	8
1	37
2	
3	
4	
5	
6	
7	7
8	48
9	9



Problem 1a: insert 57

Insert 7, 9, 48, 8, 37, **57**
into an empty table.

(h(57) + 0) % 10 = 7
(h(57) + 1) % 10 = 8
(h(57) + 2) % 10 = 9
(h(57) + 3) % 10 = 0
(h(57) + 4) % 10 = 1
(h(57) + 5) % 10 = 2

Linear Probing

ith probe: (h(key) + i) % TableSize

idx	elem
0	8
1	37
2	57
3	
4	
5	
6	
7	7
8	48
9	9



Problem 1a: deletion

Delete 37, 7, 57 from the table.

Linear Probing

ith probe: (h(key) + i) % TableSize

idx	elem
0	8
1	37
2	57
3	
4	
5	
6	
7	7
8	48
9	9



Problem 1a: delete 37

Delete **37**, 7, 57 from the table.

(h(37) + 0) % 10 = 7
(h(37) + 1) % 10 = 8
(h(37) + 2) % 10 = 9
(h(37) + 3) % 10 = 0
(h(37) + 4) % 10 = 1

Linear Probing

ith probe: (h(key) + i) % TableSize

idx	elem
0	8
1	DELETED
2	57
3	
4	
5	
6	
7	7
8	48
9	9

Problem 1a: delete 7

Delete 37, 7, 57 from the table.

(h(7) + 0) % 10 = 7

Linear Probing

ith probe: (h(key) + i) % TableSize

idx	elem
0	8
1	DELETED
2	57
3	
4	
5	
6	
7	DELETED
8	48
9	9



Problem 1a: delete 57

Delete 37, 7, **57** from the table.

(h(57) + 0) % 10 = 7
(h(57) + 1) % 10 = 8
(h(57) + 2) % 10 = 9
(h(57) + 3) % 10 = 0
(h(57) + 4) % 10 = 1
(h(57) + 5) % 10 = 2

Linear Probing

ith probe: (h(key) + i) % TableSize

idx	elem
0	8
1	DELETED
2	DELETED
3	
4	
5	
6	
7	DELETED
8	48
9	9



Experiment on 1a

What happens if we now try to remove a non-existent element (e.g. 17) from the table?



Experiment: delete 17

Delete **17** from the table.

(h(17) + 0) % 10 = 7
(h(17) + 1) % 10 = 8
(h(17) + 2) % 10 = 9
(h(17) + 3) % 10 = 0
(h(17) + 4) % 10 = 1
(h(17) + 5) % 10 = 2
(h(17) + 6) % 10 = 3

Linear Probing

ith probe: (h(key) + i) % TableSize

idx	elem
0	8
1	DELETED
2	DELETED
3	
4	
5	
6	
7	DELETED
8	48
9	9

We have reached an empty slot, but have not encountered 17. Therefore, it must not exist.



Problem 1b

- **Quadratic Probing**
  - **ith probe: (h(key) + i^2) % TableSize**



Problem 1b: insertion

Insert 7, 9, 48, 8, 37, 57
into an empty table.

Quadratic Probing

ith probe: (h(key) + i^2) % TableSize

idx	elem
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	



Problem 1b: insert 7

Insert **7**, 9, 48, 8, 37, 57
into an empty table.

$(h(7) + 0^2) \% 10 = 7$

Quadratic Probing

ith probe: $(h(key) + i^2) \% TableSize$

idx	elem
0	
1	
2	
3	
4	
5	
6	
7	7
8	
9	



Problem 1b: insert 9

Insert 7, **9**, 48, 8, 37, 57
into an empty table.

$(h(9) + 0^2) \% 10 = 9$

Quadratic Probing

ith probe: $(h(key) + i^2) \% TableSize$

idx	elem
0	
1	
2	
3	
4	
5	
6	
7	7
8	
9	9



Problem 1b: insert 48

Insert 7, 9, **48**, 8, 37, 57
into an empty table.

$(h(48) + 0^2) \% 10 = 8$

Quadratic Probing

ith probe: $(h(key) + i^2) \% TableSize$

idx	elem
0	
1	
2	
3	
4	
5	
6	
7	7
8	48
9	9



Problem 1b: insert 8

Insert 7, 9, 48, **8**, 37, 57
into an empty table.

$(h(8) + 0^2) \% 10 = 8$

$(h(8) + 1^2) \% 10 = 9$

$(h(8) + 2^2) \% 10 = 2$

Quadratic Probing

ith probe: $(h(key) + i^2) \% TableSize$

idx	elem
0	
1	
2	8
3	
4	
5	
6	
7	7
8	48
9	9

Problem 1b: insert 37

Insert 7, 9, 48, 8, **37**, 57
into an empty table.

$(h(37) + 0^2) \% 10 = 7$

$(h(37) + 1^2) \% 10 = 8$

$(h(37) + 2^2) \% 10 = 1$

Quadratic Probing

ith probe: $(h(key) + i^2) \% TableSize$

idx	elem
0	
1	37
2	8
3	
4	
5	
6	
7	7
8	48
9	9



Problem 1b: insert 57

Insert 7, 9, 48, 8, 37, **57**
into an empty table.

$(h(57) + 0^2) \% 10 = 7$
$(h(57) + 1^2) \% 10 = 8$
$(h(57) + 2^2) \% 10 = 1$
$(h(57) + 3^2) \% 10 = 6$

Quadratic Probing

ith probe: $(h(key) + i^2) \% TableSize$

idx	elem
0	
1	37
2	8
3	
4	
5	
6	57
7	7
8	48
9	9

Problem 1c

- **Separate Chaining**
  - **Use a linked list for each slot**



Problem 1c: insertion

Insert 7, 9, 48, 8, 37, 57
into an empty table.

idx	elem
0	/
1	/
2	/
3	/
4	/
5	/
6	/
7	/
8	/
9	/

Separate Chaining

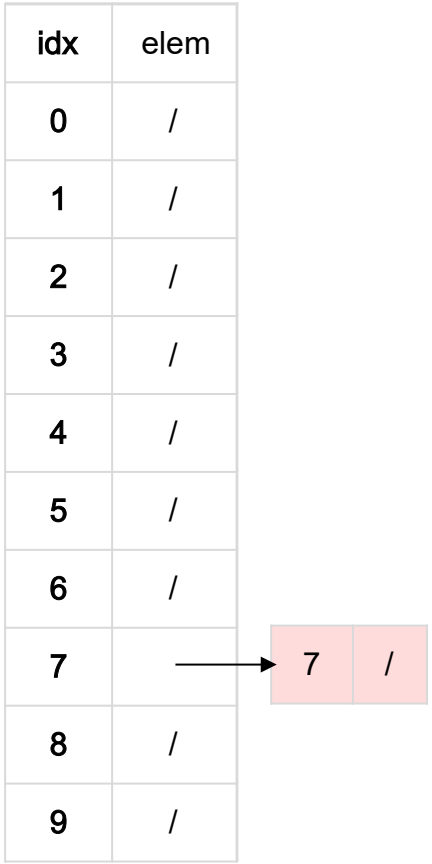
Use a linked list for each slot



Problem 1c: insert 7

Insert 7, 9, 48, 8, 37, 57 into an empty table.

(h(7) + 0) % 10 = 7



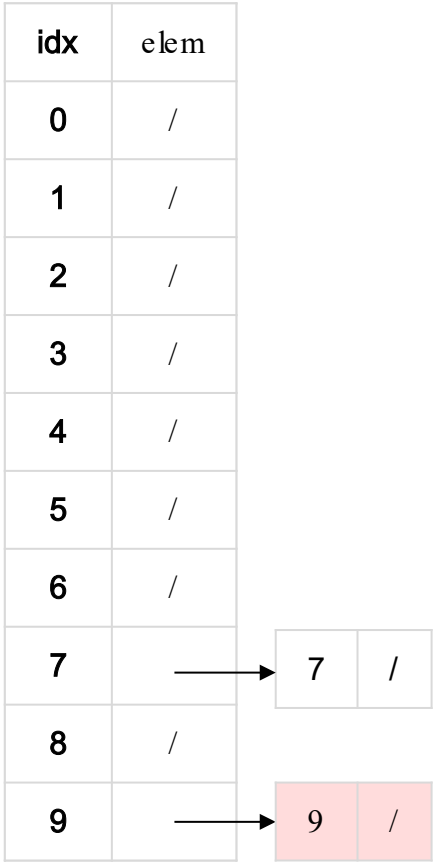
Separate Chaining

Use a linked list for each slot

Problem 1c: insert 9

Insert 7, **9**, 48, 8, 37, 57 into an empty table.

(h(9) + 0) % 10 = 9



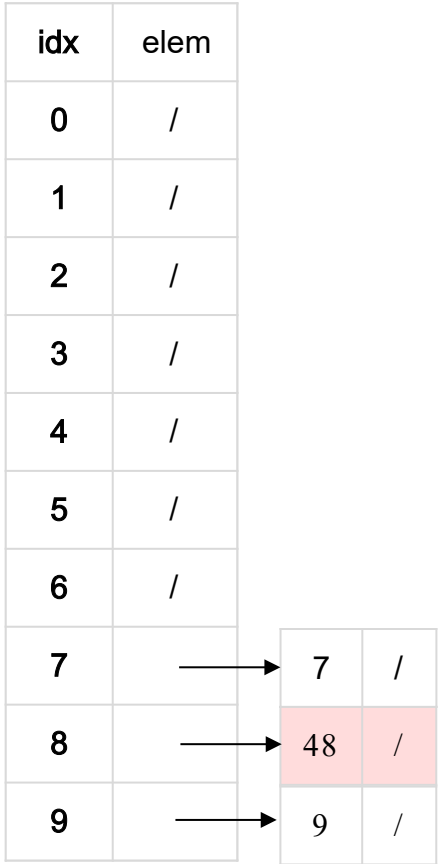
Separate Chaining

Use a linked list for each slot

Problem 1c: insert 48

Insert 7, 9, **48**, 8, 37, 57
into an empty table.

(h(48) + 0) % 10 = 8



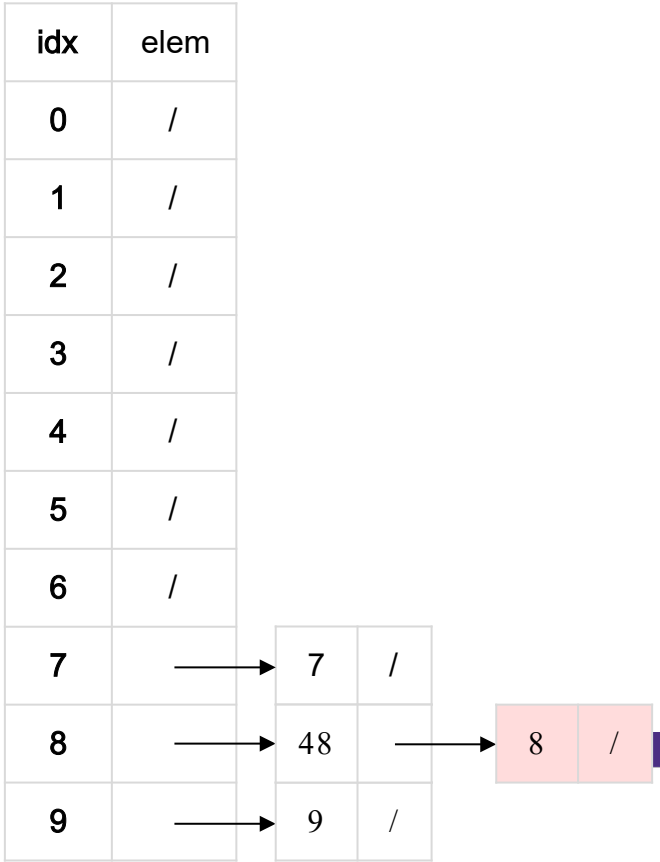
Separate Chaining

Use a linked list for each slot

Problem 1c: insert 8

Insert 7, 9, 48, **8**, 37, 57
into an empty table.

(h(8) + 0) % 10 = 8



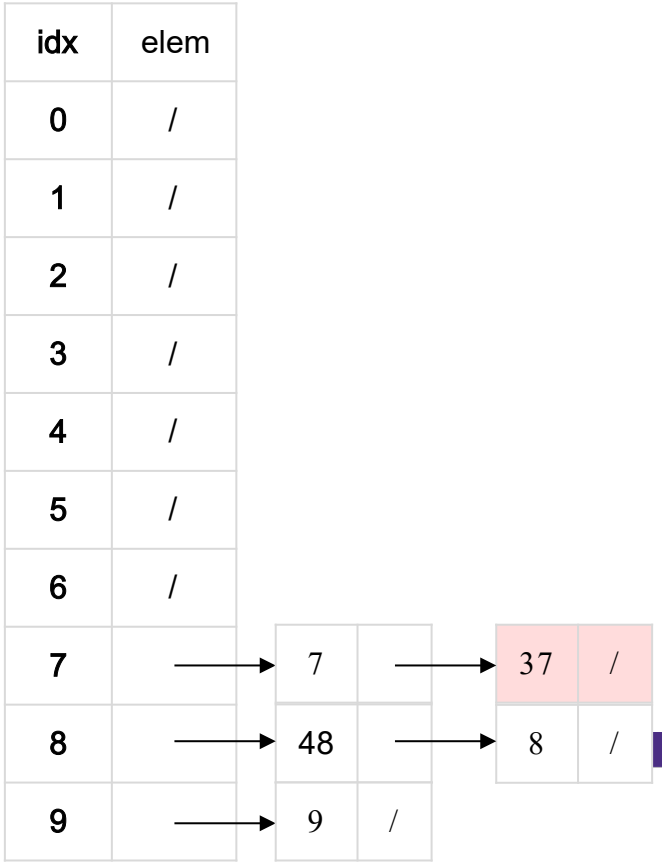
Separate Chaining

Use a linked list for each slot

Problem 1c: insert 37

Insert 7, 9, 48, 8, **37**, 57
into an empty table.

(h(37) + 0) % 10 = 7



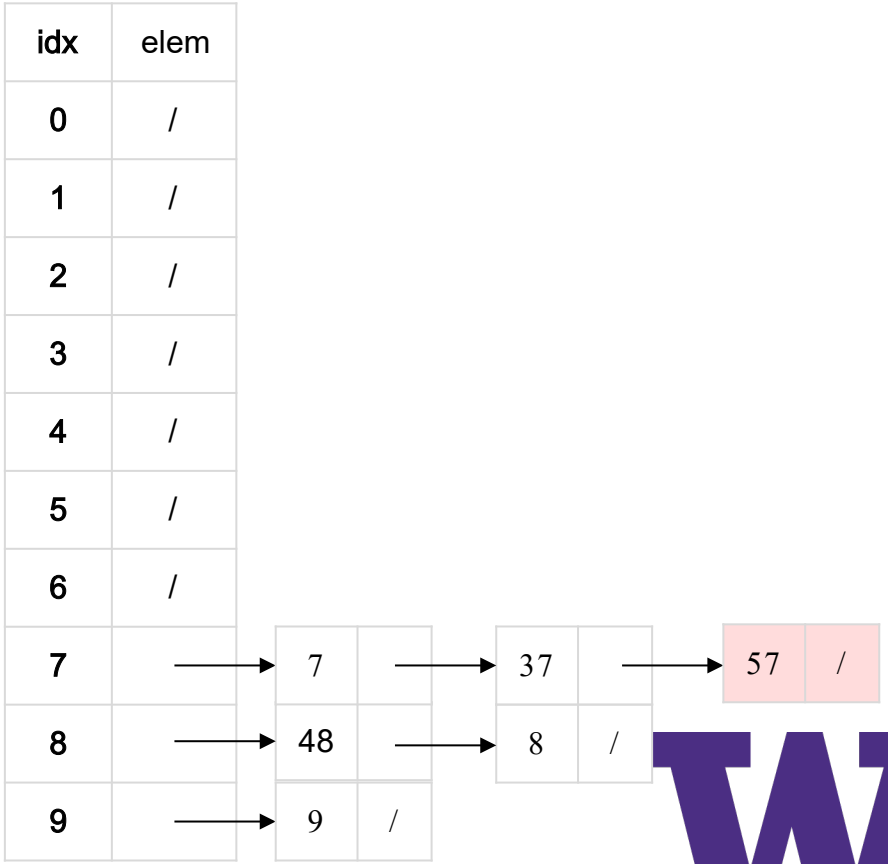
Separate Chaining

Use a linked list for each slot

Problem 1c: insert 57

Insert 7, 9, 48, 8, 37, **57**
into an empty table.

(h(57) + 0) % 10 = 7



Separate Chaining

Use a linked list for each slot

Problem 2



Problem 2a

Describe double hashing.

- On the ith probe, we check the slot with index (h(key) + i · g(key)) % TableSize.
- The first hash function h determines the location where we initially try to place the item.
- If there is a collision, then the second hash function g determines the probing step size (i.e. 1 · g(key), 2 · g(key), … distance away from the initial location).



Problem 2b

List two disadvantages of quadratic probing.

- If the table is more than half full (i.e. load factor > 0.5), then we are not guaranteed to find a location to insert an item.
- Suffers from secondary clustering since items that initially hash to the same location resolve the collision identically.

Describe how double hashing fixes one of these disadvantages.

- A good second hash function prevents secondary clustering since items that initially hash to the same location will likely resolve the collision differently.
- Items that have the same value for the first hash function f, will likely have different values for the second hash function g, leading to different probing step sizes.



Problem 2c

Compare open addressing with separate chaining.

Attributes	Open Addressing	Separate Chaining
Collision Handling	Handles collisions by searching for an open slot within the table itself.	Handles collisions by adding elements to a chain at the corresponding index.
Memory Access	Array is laid out contiguously in memory → better memory locality.	Non-contiguous memory → worse memory locality.
Runtime	Linear probing suffers from primary clustering, but is guaranteed to find an open slot. Quadratic probing suffers from secondary clustering, and is only guaranteed to find an empty slot when $\lambda < 0.5$. Double hashing does not suffer from clustering, but requires an additional hash function (computationally expensive).	Average runtime: $O(1 + \lambda)$ Best-case runtime: $O(1)$ Worst-case runtime: $O(n)$

Problem 2 (Double Hashing Example)

- **Double Hashing**
  - **ith probe: (h(key) + i · g(key)) % TableSize**



Problem 2: insertion

Insert 7, 9, 48, 8, 37, 57
into an empty table.

h(k) = k % 10
g(k) = 1 + (k % 9)

Double Hashing

ith probe: (h(key) + i * g(key)) % TableSize

idx	elem
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	



Problem 2: insert 7

Insert 7, 9, 48, 8, 37, 57
into an empty table.

h(k) = k % 10
g(k) = 1 + (k % 9)

Double Hashing

ith probe: (h(key) + i * g(key)) % TableSize

idx	elem
0	
1	
2	
3	
4	
5	
6	
7	7
8	
9	



Problem 2: insert 9

Insert 7, **9**, 48, 8, 37, 57
into an empty table.

h(k) = k % 10
g(k) = 1 + (k % 9)

Double Hashing

ith probe: (h(key) + i * g(key)) % TableSize

idx	elem
0	
1	
2	
3	
4	
5	
6	
7	7
8	
9	9



Problem 2: insert 48

Insert 7, 9, **48**, 8, 37, 57
into an empty table.

h(k) = k % 10
g(k) = 1 + (k % 9)

Double Hashing

ith probe: (h(key) + i * g(key)) % TableSize

idx	elem
0	
1	
2	
3	
4	
5	
6	
7	7
8	48
9	9



Problem 2: insert 8

Insert 7, 9, 48, **8**, 37, 57
into an empty table.

h(k) = k % 10
g(k) = 1 + (k % 9)

Double Hashing

ith probe: (h(key) + i * g(key)) % TableSize

idx	elem
0	
1	
2	
3	
4	
5	
6	8
7	7
8	48
9	9



Problem 2: insert 37

Insert 7, 9, 48, 8, **37**, 57
into an empty table.

h(k) = k % 10
g(k) = 1 + (k % 9)

Double Hashing

ith probe: (h(key) + i * g(key)) % TableSize

idx	elem
0	
1	37
2	
3	
4	
5	
6	8
7	7
8	48
9	9



Problem 2: insert 57

Insert 7, 9, 48, 8, 37, **57**
into an empty table.

h(k) = k % 10
g(k) = 1 + (k % 9)

Double Hashing

ith probe: (h(key) + i * g(key)) % TableSize

idx	elem
0	
1	37
2	
3	
4	
5	57
6	8
7	7
8	48
9	9



UNIVERSITY of WASHINGTON

Thank You!

