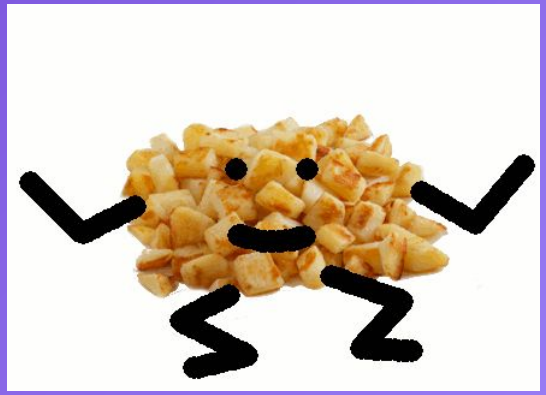
Hashing

CSE 332 – Section 5

Slides by James Richie Sulaeman



Announcements

- Midterm Next Week! Wednesday April 30th, 6:00-7:20 PM in BAG 154 and 131 (we will let you know which room to go to)
  - Covers everything up to and including Hash Tables
  - Check the bottom of the *Exams* section of the course website for past exams
  - Come to office hours if you have questions about **anything!**

Hashing

Collision Resolution

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

- This is impossible to eliminate since the number of possible keys exceeds table size.

There are multiple ways to resolve conflicts:

- Separate Chaining
  - *All elements with keys that map to the same table location are kept in a linked list.*
- Open Addressing
  - *If the slot is full, we **probe** the next slot.*
  - *On the i^{th} probe, we check the slot with index* $(h(\text{key}) + f(i))\ \%\ \text{TableSize}$.
    - Linear Probing: $f(i) = i$
    - Quadratic Probing: $f(i) = i^2$
    - Double Hashing: $f(i) = i \cdot g(\text{key})$

Problem 1a

Linear Probing

i^{th} probe: $(h(\text{key}) + i)\ \%\ \text{TableSize}$

Problem 1a

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

Linear Probing

i^{th} probe: $(h(\text{key}) + i)$ % TableSize

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

Problem 1a

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

- $(h(7) + 0) \% 10 = 7$

Linear Probing

i^{th} probe: $(h(\text{key}) + i) \% \text{TableSize}$

0	
1	
2	
3	
4	
5	
6	
7	7
8	
9	

Problem 1a

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

- $(h(9) + 0)\ \%\ 10 = 9$

Linear Probing

i^{th} probe: $(h(\text{key}) + i)\ \%\ \text{TableSize}$

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

Problem 1a

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

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

Linear Probing

i^{th} probe: $(h(\text{key}) + i) \% \text{TableSize}$

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

Problem 1a

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

i^{th} probe: $(h(\text{key}) + i)\ \%\ \text{TableSize}$

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

Problem 1a

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

i^{th} probe: $(h(\text{key}) + i)\ \%\ \text{TableSize}$

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

Problem 1a

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

i^{th} probe: $(h(\text{key}) + i) \% \text{TableSize}$

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

Problem 1a

Delete 37, 7, 57 from the table.

Linear Probing

i^{th} probe: $(h(\text{key}) + i)$ % TableSize

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

Problem 1a

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

i^{th} probe: $(h(\text{key}) + i) \% \text{TableSize}$

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

Problem 1a

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

- $(h(7) + 0)\ \%\ 10 = 7$

Linear Probing

i^{th} probe: $(h(\text{key}) + i)\ \%$ TableSize

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

Problem 1a

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

i^{th} probe: $(h(\text{key}) + i)\ \%\ \text{TableSize}$

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

Experiment

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

Experiment

Delete 17 from the table.

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

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

Linear Probing

i^{th} probe: $(h(\text{key}) + i) \% \text{TableSize}$

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

Problem 1b

Quadratic Probing

i^{th} probe: $(h(\text{key}) + i^2)$ % TableSize

Problem 1b

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

Quadratic Probing

i^{th} probe: $(h(\text{key}) + i^2)$ % TableSize

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

Problem 1b

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

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

Quadratic Probing

i^{th} probe: $(h(\text{key}) + i^2)\ \%\ \text{TableSize}$

0	
1	
2	
3	
4	
5	
6	
7	7
8	
9	

Problem 1b

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

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

Quadratic Probing

i^{th} probe: $(h(\text{key}) + i^2)\ \%\ \text{TableSize}$

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

Problem 1b

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

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

Quadratic Probing

i^{th} probe: $(h(\text{key}) + i^2)\ \%\ \text{TableSize}$

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

Problem 1b

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

i^{th} probe: $(h(\text{key}) + i^2)\ \%\ \text{TableSize}$

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

Problem 1b

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

i^{th} probe: $(h(\text{key}) + i^2)\ \%\ \text{TableSize}$

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

Problem 1b

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

i^{th} probe: $(h(\text{key}) + i^2)\ \%\ \text{TableSize}$

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

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

Separate Chaining

Use a linked list for each slot

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

Problem 1c

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

- $h(7) \% 10 = 7$

Separate Chaining

Use a linked list for each slot

0	
1	
2	
3	
4	
5	
6	
7	7
8	
9	

Problem 1c

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

- $h(9) \,\%\, 10 = 9$

Separate Chaining

Use a linked list for each slot

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

Problem 1c

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

- $h(48) \% 10 = 8$

Separate Chaining

Use a linked list for each slot

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

Problem 1c

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

- $h(8) \% 10 = 8$

Separate Chaining

Use a linked list for each slot

Problem 1c

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

- $h(37) \% 10 = 7$

Separate Chaining

Use a linked list for each slot

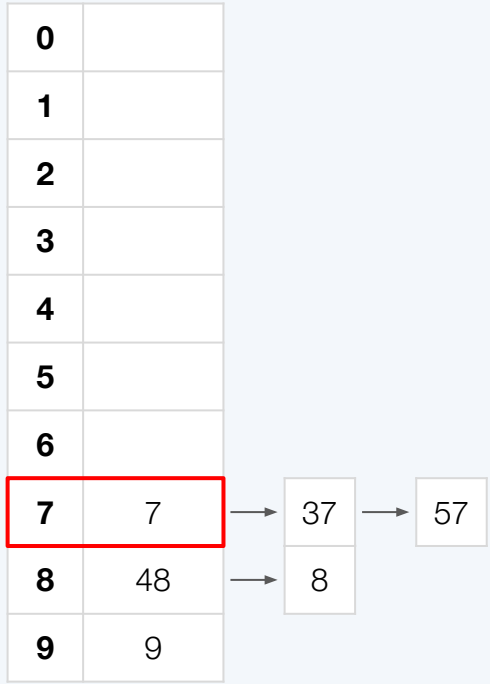
Problem 1c

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

- $h(57) \% 10 = 7$

Separate Chaining

Use a linked list for each slot



Problem 2

Problem 2a

Describe double hashing.

- On the i^{th} probe, we check the slot with index $(h(\text{key}) + i \cdot g(\text{key}))$ % 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. $1 \cdot g(\text{key})$, $2 \cdot g(\text{key})$, ... distance away from the initial location).

Problem 2b

List two disadvantages of quadratic probing.

1. 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.
2. 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.

Open Addressing	Separate Chaining
Handles collisions by searching for an open slot within the table itself.	Handles collisions by adding elements to a chain at the corresponding index.
Can use less memory since all elements are stored within the table itself.	Uses more memory since additional data structures are needed. Worse memory locality.
Linear probing suffers from primary clustering, but is guaranteed to find an open slot.	Average runtime: $\mathcal{O}(1 + \lambda)$
Quadratic probing suffers from secondary clustering, and is only guaranteed to find an empty slot when $\lambda < 0.5$.	Best-case runtime: $\mathcal{O}(1)$
Double hashing does not suffer from clustering, but requires an additional hash function (computationally expensive).	Worst-case runtime: $\mathcal{O}(n)$

Problem 3

Problem 3

For each of the following questions, choose a collision resolution between **Linear Probing, Quadratic Probing, Double Hashing, and Separate Chaining**

a) You are implementing a hash table on hardware with low memory and low computational power. Thankfully, your hash function almost always spreads keys out evenly and the items you are hashing take up a small amount of memory (e.g. integers or shorts). Which collision resolution is best?

b) Now you are working on creating a hash table specifically for Strings. The issue is these strings are all extremely long! The Strings are so long that the process of hashing them (which includes iterating through every char) affects the runtime. Which collision resolution is best?

c) You are designing a hash table with your CSE 332 TA. However, the initial hash function that they designed causes items to cluster and they are adamant about keeping it. Which collision resolution is best?

Problem 3a

a) You are implementing a hash table on hardware with **low memory** and **low computational power**. Thankfully, your hash function almost always **spreads keys out evenly** and the items you are hashing take up **a small amount of memory** (e.g. integers or shorts). Which collision resolution is best?

Double Hashing –

- Requires a second hash function bad for low computational power environment

Quadratic Probing –

- Leaves holes to resolve collisions less efficient space use
- Load factor must be < 0.5
  more frequent resizes
  bad for low computational power

Separate Chaining –

- keys are small and using low memory device memory overhead from using LinkedList is bad.

Therefore Linear Probing is best

- Tightly packs data together tightly -> saves space
- Primary clustering isn't a big issue since our hash function almost always spreads keys out evenly.

Problem 3b

b) Now you are working on creating a hash table specifically for Strings. The issue is these **strings are all extremely long**! The Strings are so long that the process of **hashing them** (which includes iterating through every char) **affects the runtime**. Which collision resolution is best?

Double Hashing –

- Requires a second hash on collision □ Want to avoid rehashing so not good

Quadratic Probing –

- Load factor must be < 0.5 □ more frequent resizes □ must rehash all elements more frequently not good

Linear Probing –

- Load Factor must be < 1.0 □ more frequent resizes □ must rehash elements more frequently so not good

Therefore, Separate Chaining is best

- Load factor can be > 1.0, without breaking the hash table □ less resizing □ minimize rehashing
- Long strings □ unlikely to hash to the same index □ good distribution □ more even chain length
- Long strings = high memory □ overhead from LinkedList is negligible.

Problem 3c

c) You are designing a hash table with your CSE 332 TA. However, the initial hash function that they designed **causes items to cluster** and they are adamant about keeping it. Which collision resolution is best?

Quadratic Probing –
- Simple Hash Function □ more collisions □ Secondary clustering

Linear Probing –
- Simple Hash Function □ more collisions □ Primary clustering

Separate Chaining –
- Simple Hash Function □ more collisions □ long chains around cluster areas.

Therefore, Double Hashing is best
- Provides the most even distribution of data on collision □ less clustering issues

Thank You!