CSE 373: Data Structures and Algorithms

More on Hashing

Autumn 2018

Shrirang (Shri) Mare
shri@cs.washington.edu

Thanks to Kasey Champion, Ben Jones, Adam Blank, Michael Lee, Evan McCarty, Robbie Weber, Whitaker Brand, Zora Fung, Stuart Reges, Justin Hsia, Ruth Anderson, and many others for sample slides and materials ...

Announcements

- Midterm info and practice material (past quarter midterms) are online.
  - https://courses.cs.washington.edu/courses/cse373/18au/exams.html
- Midterm review sessions: 10/31 lecture and Section 05 (11/01)
- Thoughts on in-class feedback collected last Friday

Midterm Topics

For more info visit course website: https://courses.cs.washington.edu/courses/cse373/18au/exams.html

ADT and data structures:

- Difference between them, runtimes, …

Asymptotic analysis:

- Finding c and n0, Modeling runtime, looking at a code or a model and giving Big-O runtimes,
- Definitions of Big-O, Big-Omega, Big-Theta

Trees:

- Doing operations on trees, runtimes, AVL rotations

Hash tables:

- collision strategies, basics of good hash function, doing inserts in a hash table

Design decisions and testing:

- Given a scenario, choose a data structure and explain your choice.
- Pros and cons of different implementation
- How to construct different test cases

Today

- Pros and Cons of different hash collision strategies
- Other applications of hashing
- Average-case analysis

Review: Open Addressing

- Open addressing is a collision resolution strategy where collisions are resolved by storing the colliding key in a different location when the natural choice is full.

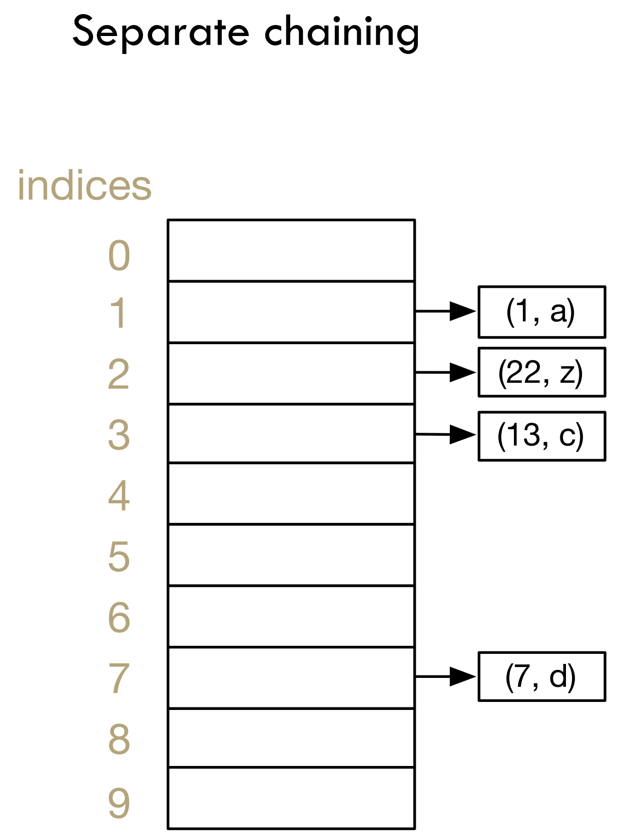
```
put(21, value21)
```

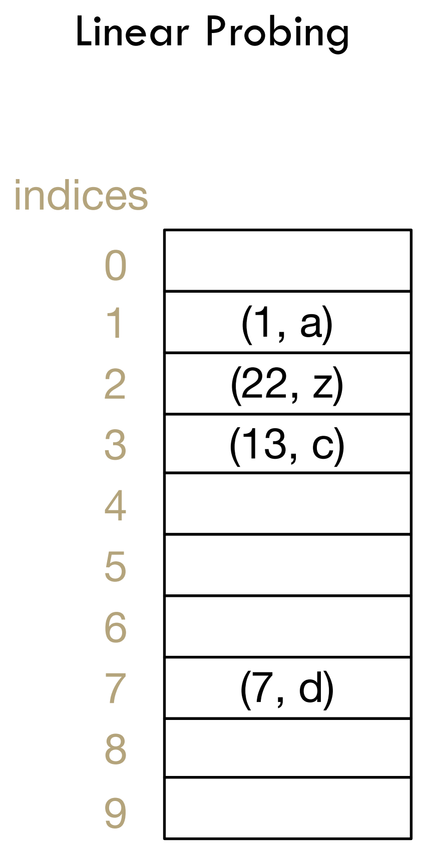
indices	
0	
1	1
2	22
3	13
4	
5	
6	
7	7
8	
9	

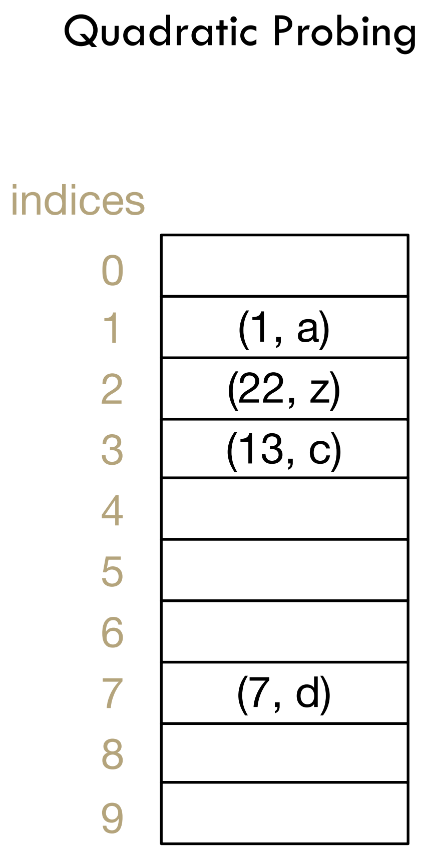
Note: For simplicity, the table shows only keys, but in each slot both, key and value, are stored.

Review: Collision avoidance strategies

`put(42, k)`





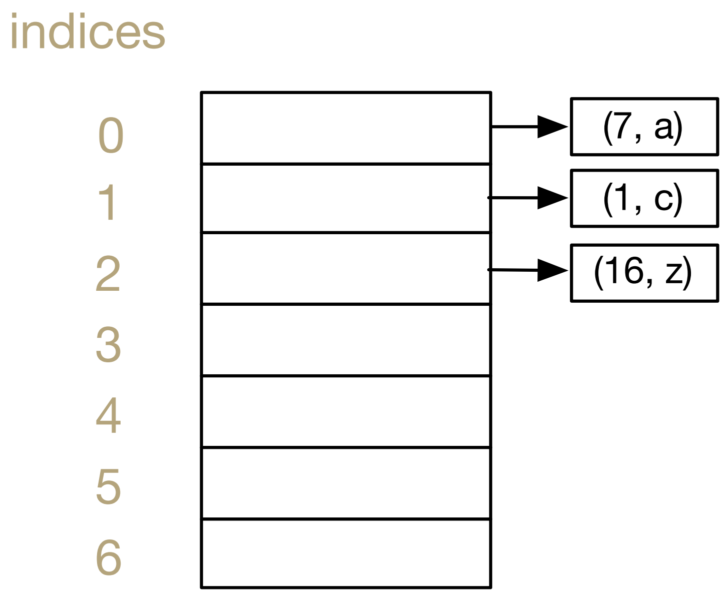


Worksheet Q1-Q4

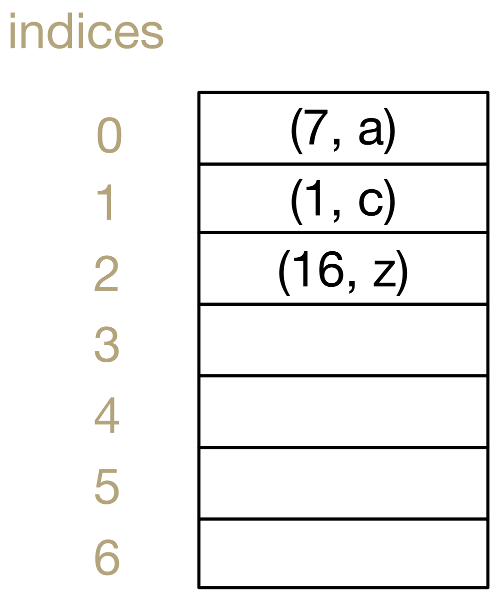
Do worksheet questions Q1-Q4

Worksheet Q1

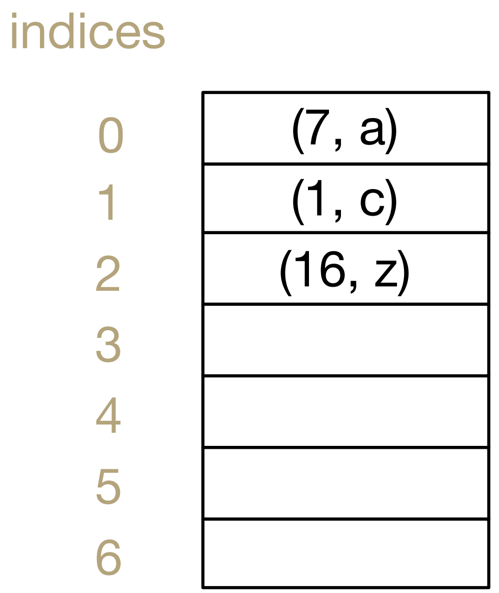
Separate chaining



Linear Probing



Quadratic Probing



Worksheet Q2

```
16124187,alice12,Alice,Smith,AG,JUNIOR,Electrical Engineering,alice12@uw.edu
```

Worksheet Q3

A. 10 – all keys hash to either 0 or 5 (collisions!)

B. 15 – all keys hash to 0 (lots of collisions!!)

C. 7 – Fewer collisions but wasted table size

D. 9 – Fewer collisions, but one wasted space

E. 11 – Fewer collisions, no wasted space. **(answer)**

Worksheet Q4

Do mod by TableSize

$$\text{array index} = h(k, i) \ \% \ \text{array.length}$$

Hash function with collision resolution

We can express our hash function as $h(k, i) = h(k) + f(k, i)$

where
h is the hash function,
i is the attempt to find a slot, and
f is the resolution function.

- For separate chaining: $f(k, i) = 0$
- For linear probing: $f(k, i) = i$
- For quadratic probing: $f(k, i) = i^2$

Hash function with collision resolution

We can express our hash function as $h(k, i) = h(k) + f(k, i)$

where
h is the hash function,
i is the attempt to find a slot, and
f is the resolution function.

- For separate chaining: $f(k, i) = 0$
- For linear probing: $f(k, i) = i$
- For quadratic probing: $f(k, i) = i^2$
- For double hashing: $f(k, i) = i * g(k)$ where g is some new hash function

Clustering: Disadvantage of open addressing

Linear Probing

indices	
0	(7, a)
1	(1, c)
2	(16, z)
3	(8, a)
4	(14, b)
5	
6	

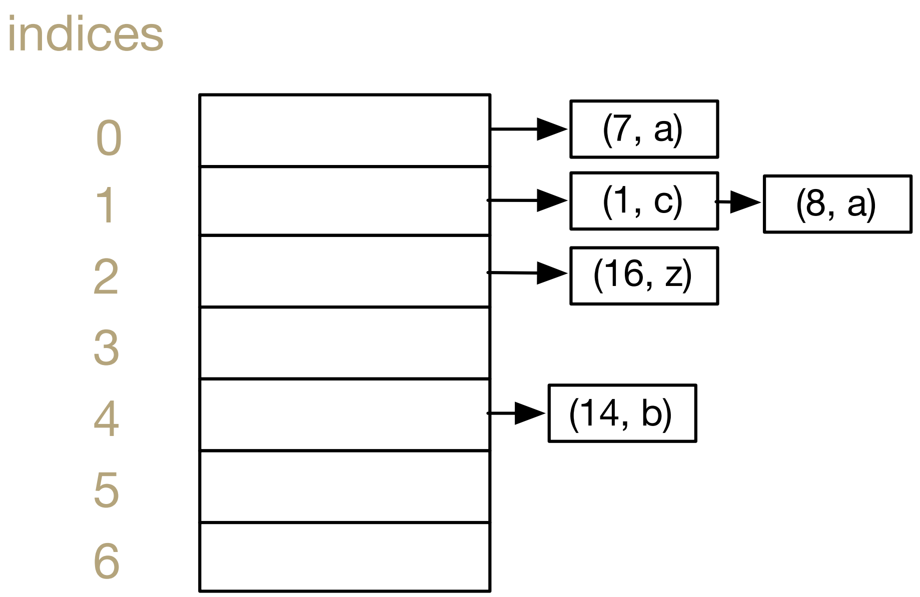
Quadratic Probing

indices	
0	(7, a)
1	(1, c)
2	(16, z)
3	
4	(14, b)
5	(8, a)
6	

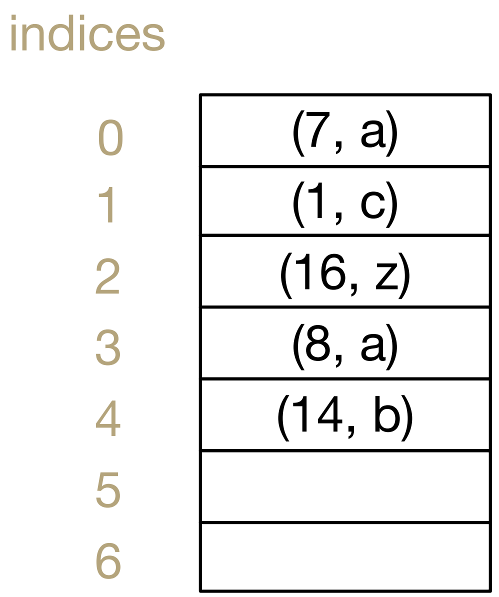
Linear probing leads to primary clustering

How find works

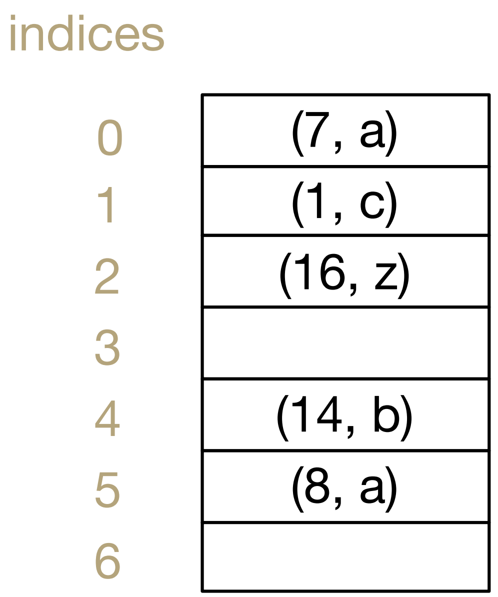
Separate chaining



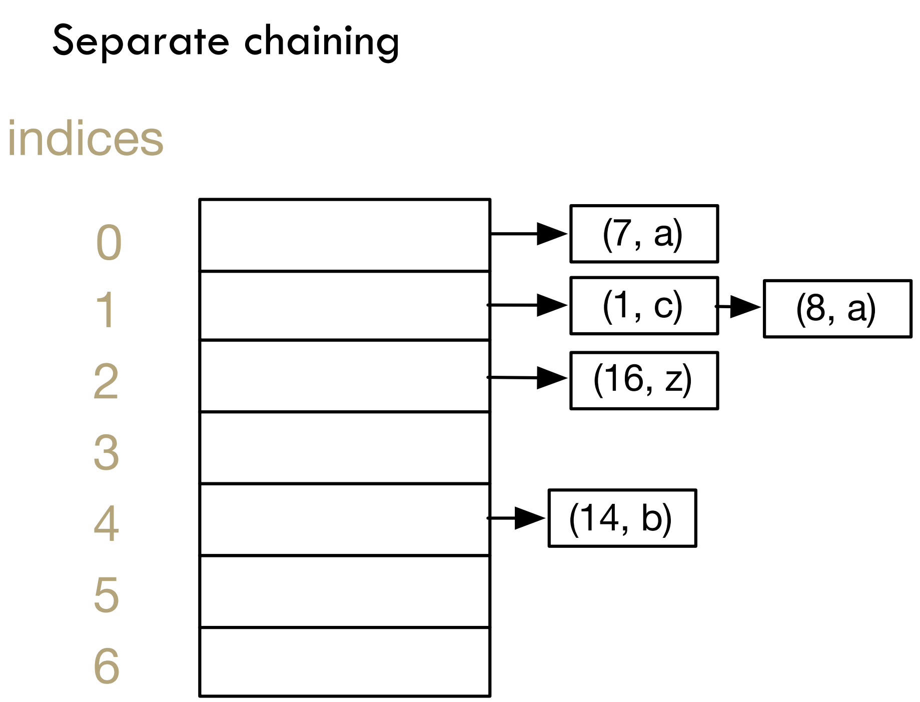
Linear Probing

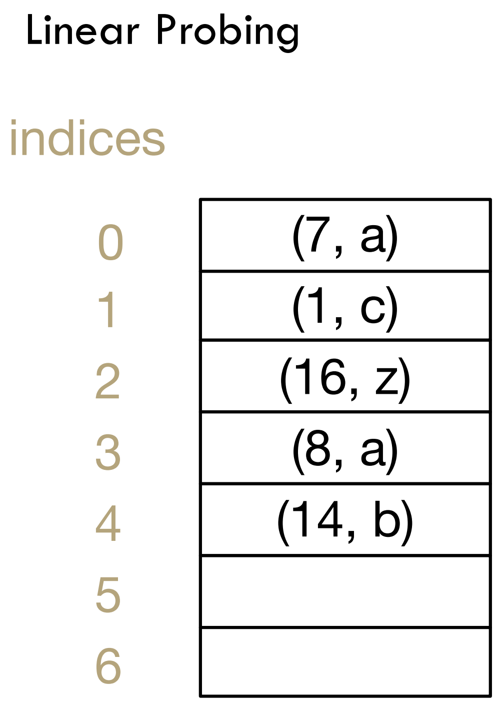


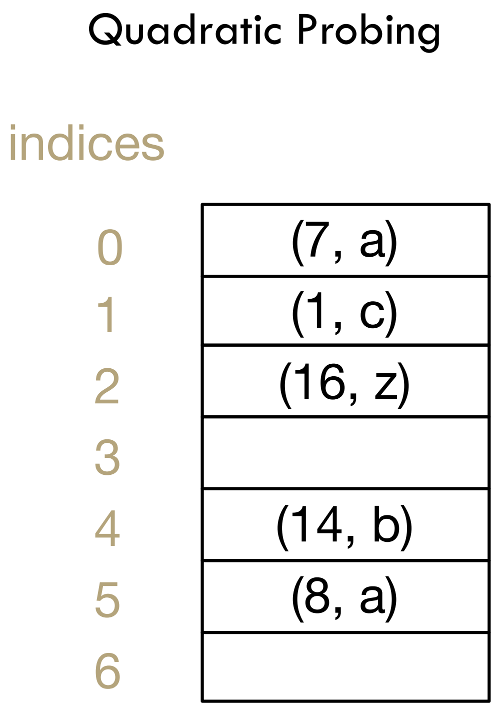
Quadratic Probing



How delete works







Resizing

How do we resize?

- Remake the table
- Evaluate the hash function over again.
- Re-insert.

When to resize?

- Depending on our load factor λ
- Heuristic:
  - for separate chaining λ between 1 and 3 is a good time to resize.
  - For open addressing λ between 0.5 and 1 is a good time to resize.

Separate chaining: Running Times

What are the running times for:

`insert`

- **Best:** $O(1)$
- **Worst:** $O(n)$ (if insertions are always at the end of the linked list)

`find`

- **Best:** $O(1)$
- **Worst:** $O(n)$

`delete`

- **Best:** $O(1)$
- **Worst:** $O(n)$

Separate chaining: Average Case

What about on average?
Let's **assume** that the keys are randomly distributed

What is the average running time if the size of the table $TableSize$ and we've inserted n keys?

`insert`

`find`

`delete`

Separate chaining: Average Case

What about on average?
Let's **assume** that the keys are randomly distributed

What is the average running time if the size of the table $TableSize$ and we've inserted n keys?

`insert` $O(1)$

`find` $O\left(1 + \frac{n}{TableSize}\right)$

`delete` $O\left(1 + \frac{n}{TableSize}\right)$

Separate chaining: Average Case

What about on average?
Let's **assume** that the keys are uniformly distributed

What is the average running time if the size of the table $TableSize$ and we've inserted n keys?

`insert` $O(1)$

`find` $O(1+\lambda)$

`delete` $O(1+\lambda)$

$\frac{n}{TableSize}$ is λ load factor

Linear probing: Average-case insert

If $\lambda < 1$ we'll find a spot eventually.

What's the average running time?

Uniform Hashing Assumption

for any pair of elements x, y

the probability that h(x) = h(y) is $\frac{1}{TableSize}$

If find is unsuccessful: $\frac{1}{2}\left(1 + \frac{1}{(1-\lambda)^2}\right)$

If find is successful: $\frac{1}{2}\left(1 + \frac{1}{(1-\lambda)}\right)$

We won't ask you to prove these

Summary

Separate Chaining

- Easy to implement
- Running times $O(1 + \lambda)$

Open Addressing

- Uses less memory.
- Various schemes:
- Linear Probing – easiest, but need to resize most frequently
- Quadratic Probing – middle ground
- Double Hashing – need a whole new hash function, but low chance of clustering.

Which you use depends on your application and what you’re worried about.

Other applications of hashing

- Cryptographic hash functions: Hash functions with some additional properties
  - Commonly used in practice: SHA-1, SHA-265
  - To verify file integrity. When you share a large file with someone, how do you know that the other person got the exact same file? Just compare hash of the file on both ends. Used by file sharing services (Google Drive, Dropbox)
  - For password verification: Storing passwords in plaintext is insecure. So your passwords are stored as a hash.
  - For Digital signature
  - Lots of other crypto applications
- Finding similar records: Records with similar but not identical keys
  - Spelling suggestion/corrector applications
  - Audio/video fingerprinting
  - Clustering
- Finding similar substrings in a large collection of strings
  - Genomic databases
  - Detecting plagiarism
- Geometric hashing: Widely used in computer graphics and computational geometry

Wrap Up

Hash Tables:

- Efficient find, insert, delete **on average, under some assumptions**
- Items not in sorted order
- Tons of real world uses
- ...and really popular in tech interview questions.

Need to pick a good hash function.

- Have someone else do this if possible.
- Balance getting a good distribution and speed of calculation.

Resizing:

- Always make the table size a prime number.
- λ determines when to resize, but depends on collision resolution strategy.