Hashing #3 - Linear & Quadratic probing

CptS 223 - Fall 2017 - Aaron Crandall

Today’s Agenda

- Announcements
- Thing of the day
- Hashing - Linear and Quadratic probing

Announcements

- Andrew Kass, VP of product development, platforms and operations for Tableau Software, is scheduled to speak to a general campus audience at 1 p.m., Friday, October 20, Goertzen 21

IoT FTW

- WiFi issues
- Mesh net issues
- IFTTT dev time
- Finally voice controlled
- The English love their tea. No power in the ‘verse will stop them!

English man spends 11 hours trying to make cup of tea with Wi-Fi kettle

Data specialist Mark Rittman spent an entire day attempting to set up his new appliance so that it would boil on command

Mark Rittman set about trying to make a cup of tea at 9am but night had fallen by the time his new Wi-Fi enabled kettle could complete the task. Photograph: Alamy

Review Exam

Continuing Hashing!

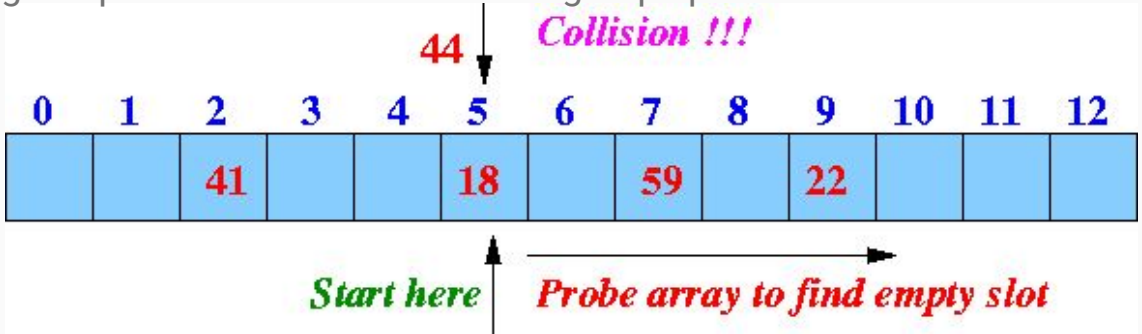
1) Hash table
   a) Normally a vector (array) of elements and indexed by hashed key
   b) Vector is of TableSize size
2) Hash Function
   a) Takes a key and returns the index in the hash table to place the record
   b) Ideally, puts elements uniformly throughout the hash table
3) Collision resolution
   a) What do you do if two elements want the same index? - Normally inf keys, finite indexes
   b) Today: linear and quadratic probing with more depth

Doing an insert - linear probing

- Take data's key
  - Integer, string, or other
- Run through hash function
  - Returns int for that key, hashkey This is a function: h(x)
- Try to Insert data into table
  - table[hashkey] = data;
- If collision (bucket already full) - then probe by adding probe function: f(i)
  - $h_i(x)$ = (hash(x) + f(i)) % TableSize, with f(0) = 0 -> while still collision, do i++ and repeat
  - Resolution strategy is f(i)
  - For linear probing: f(i) = i

Linear probing details

- Because the table holds the data (unlike chaining), table must be bigger
  - λ <= 0.5 -- Recall: chaining used λ ~= 1.0 λ == N / TableSize
- With linear probing, the f(i) algorithm is formally:
  - Collision resolution strategy: f(i) = i
  - This works as long as the table is big enough (see λ <= 0.5)
  - Primary clustering is a problem since numbers will group up in the table



Linear Probing Example: [89, 18, 49, 58, 69]

	Empty Table	After 89	After 18	After 49	After 58	After 69
0				49	49	49
1					58	58
2						69
3						
4						
5						
6						
7						
8			18	18	18	18
9		89	89	89	89	89

*note cluster!

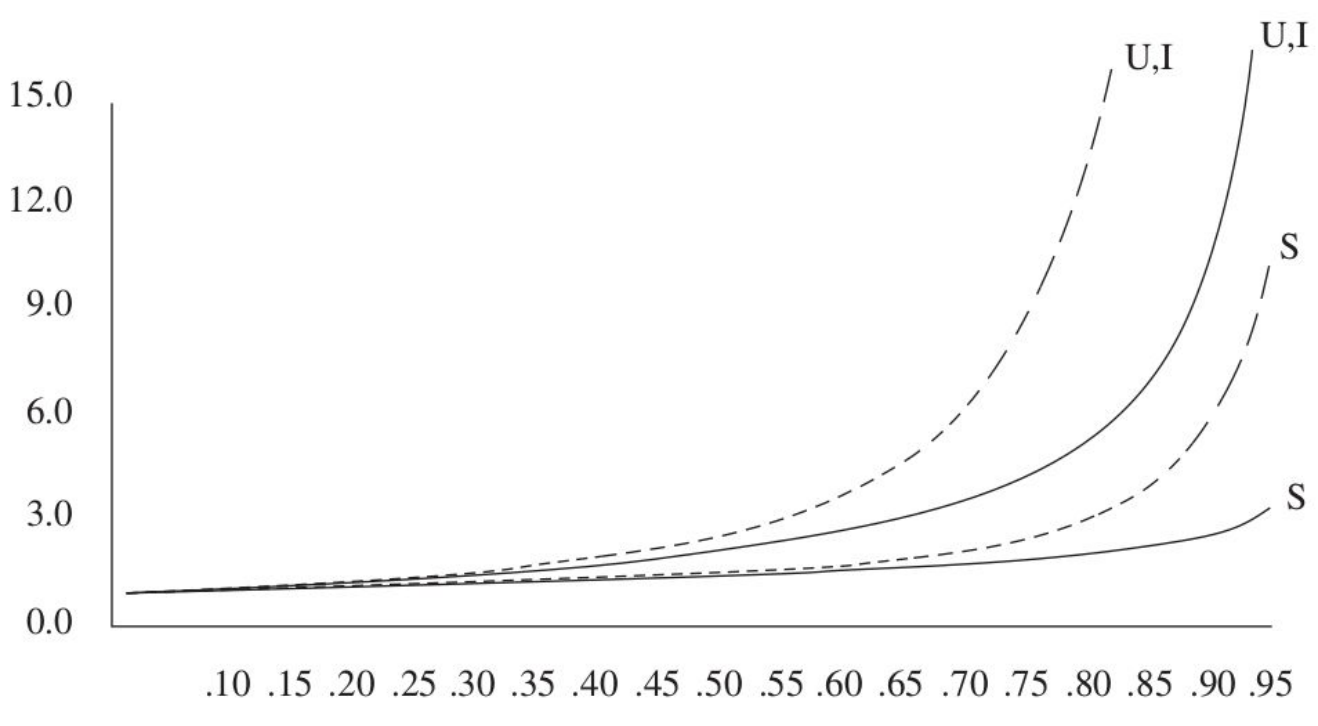
Linear probing expected time cost

- Expected number of probes to insert/search unsuccessfully:
  Probes ~= ½ (1 + 1/(1-λ)^2)
- Expected number of probes to search successfully:
  Probes ~= ½ (1 + 1/(1-λ))
- As the rate of probes changes with λ, it's time for some calculus on inserts in relation to λ, or I(λ):

$$I(\lambda) = \frac{1}{\lambda}\int_0^{\lambda} \frac{1}{1-x}dx = \frac{1}{\lambda}\ln\frac{1}{1-\lambda}$$

Linear probing probe rate vs load factor (λ)

- Why rehash @ λ ~= 0.5?

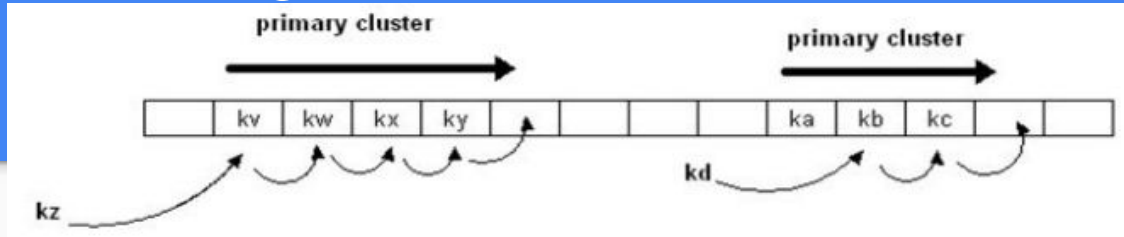


Number of probes plotted against load factor for linear probing (dashed) and random strategy (S is successful search, U is unsuccessful search, and I is insertion)

What about delete? Why can’t remove elements fully with probing? Lazy Delete

- Searching for an element requires that we follow the same series of probes (collisions) as we had then inserting.
- We return “not found” if we come upon an empty bucket before finding the element we’re searching for.
- So… if we actually delete elements, it will leave holes in our probing chains and start to cause searches to fail, as well as duplicate data entries when we go to update old data.
- The solution is lazy deletion: elements are flagged as deleted, not actually removed. The book doesn’t get to this until quadratic probing.

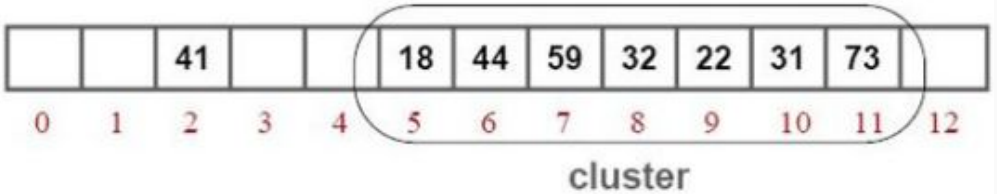
What is primary clustering?



- Linear probing can cause primary clustering, where elements tend to cluster around table locations they hash to.
- These clusters can combine into larger clusters, which create long probes

Example of a primary cluster: Insert keys: **18, 41, 22, 44, 59, 32, 31, 73**, in this order, in an originally empty hash table of size **13**, using the hash function **h(key) = key % 13** and **c(i) = i**:

h(18) = 5
h(41) = 2
h(22) = 9
h(44) = 5+1
h(59) = 7
h(32) = 6+1+1
h(31) = 5+1+1+1+1+1
h(73) = 8+1+1+1



Solution for primary clustering? Quadratics!

- Instead of f(i) = i, use f(i) = i^2
  - Ensures you skip around table
  - Reduces primary clustering
- But! No guarantee of finding an empty place if λ > 0.5
  - Even earlier if the TableSize is not prime
  - Go through theorem 5.1 to see why TableSize needs to be prime
- In 5.4.2 they *finally* mention that standard deletion doesn't work in probing hash tables.
  - They introduce lazy deletion, but don't call it that.

Same simple example, but with f(i) = i^2

- Note: less Clustering than with linear probing.

	Empty Table	After 89	After 18	After 49	After 58	After 69
0				49	49	49
1						
2					58	58
3						69
4						
5						
6						
7						
8			18	18	18	18
9		89	89	89	89	89

Double hashing - VERY quickly

- Use a second hash function for resolution:
  - f(i) = i * hash_2(x)
  - Apply second hash function to x, add to result of first hash function
  - Probes at a distance of hash_2(x), then 2*hash_2(x), 3*hash_2(x),....
- Poor choice of hash_2(x) would be disastrous
  - Must make sure all cells can be probed, TS should be prime!
  - Can never evaluate to zero (or we don't move to other buckets)
- Example: hash_2(x) = R - (x mod R), with R being any prime less than TS
- All of this isn't needed if you do quadratic probing, though
- More interesting from a CS research perspective, not for daily use

What is rehashing?

- Essentially making more room in our hash table (or shrinking it, I suppose)
- If the load factor gets too high (> 1.0 for separate chaining, > 0.5 for probing), then we rehash:
  - 1) Make new Table. TableSize = FindNextPrime(TableSize * 2)
  - 2) Take every element from the old table and hash it into new table
  - 3) Destroy old table itself

Rehashing Example (linear probe)

0	6
1	15
2	23
3	24
4	
5	
6	13

1. Double TS
2. Move TS to next prime
3. Rehash all elements to new table
4. Destroy old table

0	
1	
2	
3	
4	
5	
6	6
7	23
8	24
9	
10	
11	
12	
13	13
14	
15	15
16	

Onto some code!

- Quadratic probing implementation
- Note:
  - How lazy deletion is implemented
  - Insert code
  - Search (contains) code
  - Why did they note not to swap lines 9 & 10 in contains routine?

Final hashing questions?

- Read through Chapter 5 again (especially if this is the first time)
- There's a *ton* more hash algorithms, but for this course we're focused on the key ones:
    - separate chaining
    - linear/quadratic probing
- Wednesday will be Chapter 6 - heaps