Hash Tables (finish)
Review

CS206

March 9

Open Addressing
Probing

- Store only <K,V> at each location in array
- If key is different and location is in use then go to next
  - repeat until free location found

Lab

- Show the final contents of the hashtable using linear probing assuming
  - table size is 13
  - h(t) = t % 13
- Data: <0,a> <32,b> <39,c> <12,d> <14,e> <35,f> <27,g> <13,h> <15,i> <5,j> <12,k> <13,l> <4,m> <0,n> <35,o>
- What is the most number of steps you needed to take to find a free location?
  - during put == 4
  - during a future contains key
    - Any key that hashed to 12 (other than 12) will take 11 steps to determine that it is not in the hashtable.

Hash Value	
0	<0,~~a~~ n>(hv=0)
1	<39,c>(hv=0)
2	<14,e>(hv=1)
3	<27,g>(hv=1)
4	<13,~~h~~ l>(hv=0)
5	<15,i>(hv=2)
6	<32,b>(hv=6)
7	<5,j>(hv=5)
8	<4,m>(hv=4)
9	<35,~~f~~ o>(hv=9)
10	
11	
12	<12,~~d~~ k>(hv=12)

Probing Distance

- Given a hash value $h(x)$, linear probing generates $h(x),\ h(x)+1, h(x)+2, \ldots$
  - so for a key of 2, h(2) = 2 you would probe at 2,3,4,5,6,7,…
- Quadratic probing – $h(x),\ h(x)+1, h(x)+4,\ h(x)+9, \ldots$
  - So for a key of 0, h(0)=0, you probe at 0, 1, 4, 9, 3, 12, …
  - Quadratic probing leads to secondary clustering, not as dramatic, but still systematic
- Double hashing
  - Use a second hash function to determine jumps
  - for instance in lab h(x) = x%13
  - suppose double hashing function h2(x) = x%7 + 1
    - why +1 in h2(x)?
  - so for a key of 27, h(27)=1, h2(27)=7, so you probe at 1,8,2,9, …
  - for a key of 23 h(23)=10, h2(23)=3, so you probe at 10, 0, 3,6,9, …

Performance Analysis for probing

- In the worst case, searches, insertions and removals take $O(n)$ time
  - when all the keys collide
- The load factor α affects the performance of a hash table
  - expected number of probes for an insertion with open addressing is $\frac{1}{1-\alpha}$
  - So, good rule of thumb, keep load factor below 0.5.
- Expected time of all operations is $O(1)$ provided α is not close to 1
    - NOTE: cheating here O() is about true worst case

Probing and Tombstones

- Probing has issues with deletions.
  - For instance suppose we remove 14.
  - Then would incorrectly say that 27, 13 and 15 were not in the table.
  - Solution: Tombstone
    - A special k,v pair.
      - get or containsKey: the tombstone is treated as there but will not match anything
      - put: tombstone is treated as not there and replaced.

Hash Value	
0	<0,~~a~~ n>(hv=0)
1	<39,c>(hv=0)
2	<14,e>(hv=1)
3	<27,g>(hv=1)
4	<13,~~h~~ l>(hv=0)
5	<15,i>(hv=2)
6	<32,b>(hv=6)
7	<5,j>(hv=5)
8	<4,m>(hv=4)
9	<35,~~f~~ o>(hv=9)
10	
11	
12	<12,~~d~~ k>(hv=12)

Growing the Hashtable

- When load factor exceeds threshold
  - Create a new array `newArr` that is double the size of `backingArray`
  - for each pair in `backingArray` compute new hash value and put pair in `newArr`
  - replace `backingArray` with `newArr`
- O() ???

Java

- Classes and Inheritance
  - Overloading and Overriding of methods
- Exceptions and their handling

- Generics

- Inner classes

Data Structures

- Arrays
- ArrayList
- Maps
  - key-value pairs
- Hashtables

Theory

- Complexity Analysis — Big-O
  - drop constants
  - focus on dominant term
  - always look at worst case

- Modularity, Abstraction and **Encapsulation**

No Lab

Test Questions: Live!