CSE 373

OCTOBER 20TH – HASHING CONCLUSION + MEMORY ANALYSIS

ASSORTED MINUTIAE

- **Office hour change**
  - 9 am Friday office hours *moved* to Monday at 9 am in CSE 220

PROJECT 2 CLARIFICATIONS

- **ChainingHashTables**
  - The hashtable must be an array of dictionaries, what the dictionary is determines the chain
  - Your code is an interface between the supported functions of ChainingHash and the dictionaries you've written

PROJECT 2 CLARIFICATIONS

- **ChainingHashTables**
  - The hashtable must be an array of dictionaries, what the dictionary is determines the chain
  - Your code is an interface between the supported functions of ChainingHash and the dictionaries you've written
- **Dictionary vs. Set**

PROJECT 2 CLARIFICATIONS

- **ChainingHashTables**
  - The hashtable must be an array of dictionaries, what the dictionary is determines the chain
  - Your code is an interface between the supported functions of ChainingHash and the dictionaries you've written
- **Dictionary vs. Set**
  - A dictionary stores key, value pairs
  - A set contains only keys—membership is the important thing to track.

EXAMPLES

HASHTABLE ANALYSIS

- If our table is properly maintained, we expect O(1) runtimes

HASHTABLE ANALYSIS

- **If our table is properly maintained, we expect O(1) runtimes**
  - What is the worst-case?

HASHTABLE ANALYSIS

- **If our table is properly maintained, we expect O(1) runtimes**
  - What is the worst-case? *Resizing*

HASHTABLE ANALYSIS

- **If our table is properly maintained, we expect O(1) runtimes**
  - What is the worst-case? *Resizing*
  - This resize can be amortized to O(1) the same way array resizing did

HASHTABLE ANALYSIS

- **If our table is properly maintained, we expect O(1) runtimes**
  - What is the worst-case? *Resizing*
  - This resize can be amortized to O(1) the same way array resizing did
  - On average, however, we expect O(1) find so long as the table is well-maintained

HASHTABLE ANALYSIS

- **If our table is properly maintained, we expect O(1) runtimes**
  - What is the worst-case? *Resizing*
  - This resize can be amortized to O(1) the same way array resizing did
  - On average, however, we expect O(1) find so long as the table is well-maintained
  - Performance and memory are direct tradeoffs here.

HASHTABLE IMPLEMENTATION

- **Hashtables implement dictionaries**

HASHTABLE IMPLEMENTATION

- **Hashtables implement dictionaries**
  - <Key, Value> pairs

HASHTABLE IMPLEMENTATION

- **Hashtables implement dictionaries**
  - <Key, Value> pairs
  - Don’t allow duplicate keys

HASHTABLE IMPLEMENTATION

- **Hashtables implement dictionaries**
  - <Key, Value> pairs
  - Don’t allow duplicate keys
  - Keys with the same “value” must have the same hash code

HASHTABLE IMPLEMENTATION

- **Hashtables implement dictionaries**
  - <Key, Value> pairs
  - Don't allow duplicate keys
  - Keys with the same "value" must have the same hash code
  - For open addressing, stored either as an array of <key,value> class objects, or as two parallel arrays, one of keys and the other of values

HASHTABLE IMPLEMENTATION

- **Resizing**

HASHTABLE IMPLEMENTATION

- **Resizing**
  - Only get O(1) operations if the table is well-maintained

HASHTABLE IMPLEMENTATION

- **Resizing**
  - Only get O(1) operations if the table is well-maintained
  - Easy to get good runtimes, if you don't consider memory

HASHTABLE IMPLEMENTATION

- **Resizing**
  - Only get O(1) operations if the table is well-maintained
  - Easy to get good runtimes, if you don't consider memory
  - bigO analysis can apply to memory consumption in the same way it applies to clock cycles

HASHTABLE IMPLEMENTATION

- **Resizing**
  - Only get O(1) operations if the table is well-maintained
  - Easy to get good runtimes, if you don't consider memory
  - bigO analysis can apply to memory consumption in the same way it applies to clock cycles
  - Resizing takes O(n) extra memory, because you need to maintain the original hash table while you build the second.

HASHTABLE IMPLEMENTATION

- **Resizing**
  - Iterate through the table (these are not in any meaningful order)

HASHTABLE IMPLEMENTATION

- **Resizing**
  - Iterate through the table (these are not in any meaningful order)
  - Insert each of the <k,v> pairs into the new hashtable (which may be larger or smaller)
  - Move pointers to new hash table

HASHTABLE IMPLEMENTATION

- Assorted things

HASHTABLE IMPLEMENTATION

- **Assorted things**
  - Prime table sizes
    - Usually keep an array of all the primes that roughly double in size precalculated

HASHTABLE IMPLEMENTATION

- **Assorted things**
  - Prime table sizes
    - Usually keep an array of all the primes that roughly double in size precalculated
    - Finding primes is actually very computationally difficult,

HASHTABLE IMPLEMENTATION

- **Assorted things**
  - Prime table sizes
    - Usually keep an array of all the primes that roughly double in size precalculated
    - Finding primes is actually very computationally difficult,
    - If n is large enough, finding the new prime can be the most consuming portion of the resize

HASHTABLE IMPLEMENTATION

- **Assorted things**
  - Prime table sizes
    - Usually keep an array of all the primes that roughly double in size precalculated
    - Finding primes is actually very computationally difficult,
    - If n is large enough, finding the new prime can be the most consuming portion of the resize
  - If a.equals(b) then a.hashcode() == b.hashcode()

HASHTABLE IMPLEMENTATION

- **Assorted things**
  - Prime table sizes
    - Usually keep an array of all the primes that roughly double in size precalculated
    - Finding primes is actually very computationally difficult,
    - If n is large enough, finding the new prime can be the most consuming portion of the resize
  - If a.equals(b) then a.hashcode() == b.hashcode()
  - Hardware constraints, even if you have lots of memory, over allocating fails to take advantage of spatial locality and can be problematic

HASH TABLES

- **Hash tables are a good overall data structure**

HASH TABLES

- **Hash tables are a good overall data structure**
  - Can provide O(1) access times

HASH TABLES

- **Hash tables are a good overall data structure**
  - Can provide O(1) access times
  - Can be memory inefficient

HASH TABLES

- **Hash tables are a good overall data structure**
  - Can provide O(1) access times
  - Can be memory inefficient
  - Probing can fail, and delete with probing mechanisms is difficult

HASH TABLES

- **Hash tables are a good overall data structure**
  - Can provide O(1) access times
  - Can be memory inefficient
  - Probing can fail, and delete with probing mechanisms is difficult
  - Chaining can be a good balance, but there is a lot of overhead maintaining all those data structures