1 # CS3134 #18

11/6/03

Janak J Parekh

2 # Administrivia

- HW4 due on Tuesday!
  - Anyone not understand what the HW is saying?
- HW2 tester to be posted…
  - And HW3 solutions
  - Give me a few days
- I have to cancel today's office hours
  - Come see me for a make-up time

3 # Agenda

- Hashing

4 # Hash Table

- Believe it or not, we can build a data structure that has O(1) performance for insert, search, remove
- Several disadvantages
  - Array-based, so sometimes difficult to expand
  - Performance can suffer based on various parameters
  - Can't visit items in order

5 # Dictionary/Map Model

- First, explain how hash tables are frequently used
- Many applications keep a tuple of data
  - (key, data), i.e., key *maps* to data
- For example,
  - (Dictionary, definition) – this is why it's called a "dictionary" structure
  - (SSN, Employee Record)
- Not only for hash tables
- Alternative: *set* data model
  - Does it exist, or does it not?
  - What you're doing for HW4

6 # Keys?

- In general, we want to make lookup by keys very fast
- In an array, the *index number* is the key
  - Not useful as a "real" key, as this number may change
  - But numbers are very fast.
- OK, so how do we use a "word" as a key?
  - We convert it to a number somehow

7 # Here's a simple one…

- Take the numeric value of all the letters
  - a = 1, b = 2, … , z = 26
  - Add them together

  - Put the word in that cell
    - cats == 43
- How well would this work?
  - What's the minimum value?
  - What's the maximum value for a 10-letter word?
  - How many words could be in between?

8 A bit more sophisticated

- For each character, multiply it by 26 to the position
  - Always produces unique number for each word
- cats == $3 * 26^3 + 1 * 26^2 + 20 * 26^1 + 19 * 26^0$
- What's the minimum value?
- What's the maximum value for a 10-letter word?
- Why is this so inefficient?
- Need to *hash* this large value into a smaller one
  - How about % arraySize?
  - This is one of the simplest hash functions

9 Collisions

- All of this would be good if we could come up with a *perfect hash* function: one that maps every possible entry into a different cell
- Guess what? We usually can't, unless we know precisely what data we'll be inputting
- Several different methodologies to deal with this

10 Collision handling: separate chaining

- Make each hash cell a "bucket" for multiple entries
- Use a linked list or array or similar construct to store the entries
- Must make sure lists don't get too long: good hash function
  - But much less sensitive to load factors than open addressing

11 Next time

- Finish hashing
  - Open addressing
  - Good hash functions?
  - Hashing efficiency
- Begin heaps