3134 Data Structures in Java

Lecture 13
Mar 7 2007
Shlomo Hershkop
Announcements

- Done grading midterms

- Reading:
  - Chapter hashtables, sorting
Outline

- Hash DS
  - Overview
  - Collisions
  - Ds
  - applications

- sorting
  - Basics
  - complicated
Hash Table DS

- This data structure is for organizing an unordered set of items

- Have the following runtimes:
  - find
  - insert
  - delete
Comparison of average runtime

- **Best Tree:**
  - AVL
    - find
    - insert
    - delete

- **Hash Table**
  - find
  - insert
  - delete
Hash Function
- mapping function between items and locations in the hashtable DS

Examples
Issues

- What hash function to use?
- What do you do about collisions?
Example

- Lets say you need a dictionary
- For each word insert in hash table
  - runtime ?
- when I need to look up a word call find on hash table
  - runtime ?
hash functions

- The truth is that hash functions should be based on the data

- Let's step through some examples
Option 1: integral keys

- items are numbers
- can use them directly to compute hash

\[ \text{Hash(key)} = \text{key} \% \text{Tablesize} \]

- Example

- Question: why not use randomness to make sure to avoid collisions?
Option 2: String key

- Hash(key) = sum of ascii values

- Hash(abc) = 97 + 98 + 99

- any idea if this will work?
Counter example:
- dictionary
- tablesize 40,000
- what is the maximum word size
- what would be the max value returned by the hash ??
Option 3: power

- lets add some spread to the summation

- $\text{Hash(ley)} = \text{key}[1] \times 26^0 + \text{key}[1] \times 26^1 \times \ldots \text{key}[i] \times 26^i$
issues

- non uniform distribution of characters in the english language
- only 28% of your table will actually be reached
- collisions!
Option 4: Adjusted power

- Hash(ley) = (key[1]*37^0 + key[1]*37^1 * ..key[i]*37^i) % tablesise

- need to make sure it will be positive
- java uses 31^i
- performs well on general strings
ok so now we know how to get things into the table

what do you do when 2 things map to same array location ??
Option 1: Separate Chaining

- At each array location have a linked list
  - how would the insert in the LL work?

- how do you perform a find on the hash table?
Option 2: open addressing

- if collision occurs, will try to find alternate cell in the array to store item

- let's see how this works
strategy

- first try hash(x)
- if full
  - try Hash(x) + f(i) % tableszie to locate

  - f is used to move around the array to find a location to use

  - different options, any ideas?
Linear probing

- $f(i) = i$

- Example

- can you think of any issues?
clustering

- linear probing suffers from a problem called clustering
- domino affect
Quadratic probing

- \( f(i) = i^2 \)

- how will this affect clusters?
Theorem

- if quadratic probing is used and table size is prime, and table is at least half empty then we will always find a spot for a new element
Option 3: Double Hashing

- Apply a second hash function $H_2$ and probe at distance $i \times \text{hash}_2(x)$

- $f(i) = \text{rehash}(i)$

- $\text{hash}(x) + i \times f_i(x)$

- **Note:**
  1. can’t return 0
  2. entire table must be addressable
Load factor

- number of element
- divided by
- table size
growing

- So how do you resize a hash ??
deletion

- how would deletion work

- any issues?
Extendible Hashing

- setup similar to B+ tree
- hashing routine which has growth built in
- use partial bits for keys
- when need to grow will use more bits
question

from the data structures we have covered which is the most space efficient ??
Wrapping up

- Say you want to add a new operation to heaps

- DecreasePriority \((p,d)\)
  - want to subtract \(d\) from priority \(p\)

- any ideas on run time ??
Switching gears
When we come back from break, we will be doing much more programming background etc

- Inheritance
- Class relationships
- Viruses
- Virus checking program
Application

- anyone know how Google works from a data structure point of view

- runtime ??
Search engine technology

- generally search engines work in the following way:
- collect documents e.g. webpages
- index information
- wait for search
  - understand query
  - search and match
  - scoring system
Any ideas how to design a search engine so that you can quickly find results?
hash table of search words

inverted index table
Vector Model

- Each document is a vector in an n-dimensional vector space of search terms.
- Take query and find closest points.
- Sparse (very).
- If one word tokens, order will be ignored.
First we generate a master word list

can strip out stop words

Stemming: can also calculate related words i.e. runs and run worry and worrying
master word list

- cat
- dog
- fine
- good
- got
- hat
- make
- pet

# A cat is a fine pet
$vec = [ 1, 0, 1, 0, 0, 0, 1 ];
many ways of calculating similarity between search term and documents

- cosine
- can generate relevance scoring
General issues

- Better parsing
- Non-English Collections
  - stemming
  - stop words
- Similarity Search
  - can combine a few docs to find similarity
- Term Weighting
- Incorporating Metadata
- Exact Phrase Matching
More DS

- Searching
Simple

- So its straightforward to sort in $O(N^2)$ time
  - Insertion sort
  - Selection sort
  - Bubble sort
More complicated

- **Shell Sort**
  - This is an $O(N^{1.5})$ algorithm that is simple and efficient in practice
  - originally presented as an $O(N^2)$ algorithm
  - complicated to analyze
  - took many years to get better bounds
More Complex

- $O(N \log N)$ algorithms
  - merge sort
  - heapsort
Quicksort

- worst case $O(n^2)$
- average case $O(N \log N)$

will learn how to make the worst case occur with such low probability that we will end up dealing with average case
Selection sort

- anyone remember how this one works??

- 2 arrays, sorted and unsorted
- keep choosing min from the unsorted list and append to sorted
Bubble Sort

- Anyone ??

- iterate and swap out of ordered elements
Insertion sort

- this is the quickest of the $O(N^2)$ algorithms for small sets
Insertion

- sort 1^{st} element
- sort first 2
- sort first 3
- etc