Faster Searching Methods

Corky Cartwright

Department of Computer Science

Rice University

Hashing: Motivation

- Consider the problem of counting the numbers of each kind of char in a file.
  - If chars are represented by bytes (perspective of pre-Java langages), then we can index a 256 entry table by character code. Idea; convert chars to unique table indices.
  - If chars are represented by 16-bit Unicode (UTF-16), we can still use this approach; only requires an array of counters of size 64K (2**16).
  - But what it we want to generalize our application to process characters encoded using 32-bit unicode (UTF-32). No longer practical to use direct mapping of a char to its binary representation for a table index.

How can we handle UTF-32?.

Hashing: Motivation cont.

- Consider a similar more interesting problem of counting the number of occurrences of each word in a huge text file. We can easily parse the input stream in words (assuming we can agree on the set of delimiter characters). But how we represent and manage the table recording the words we have seen.
- Key idea: using a "scrambling" (*hash*) function to map large chars (UTF-32) or strings (words) to indices in the rnge [0, N-1] where N is approximately equal to the size of the longest file (measured in the items we are counting) we expect to process.

Hashing Functions

- Standard practice: hash functions yield an unsigned binary number in same format as machine addresses (formerly 32-bit binary but morphing to 64-bit binary).
- Address-sized hash codes are easily mapped to indices in the range [0, N-1] by the remainder operation (a by-product of machine division). If N is a power of 2, then remainders can by computed by shift operations (which are extremely fast), but there are theoretical advantages to using a prime number for N. (Perhaps not worth it in practice.)

Hashing Functions

- Devising good hash functions is an art (lots of pages on the web and *some* of them are technically sound)
- Rules of thumb:
  - The hash code for an object must be consistent with equality. (Equal objects *never* hash to different codes.)
  - Hashing mutable objects is insane unless you are using object identity as the definition of object equality. (Default **`hashCode()`** in Java has this property. Can also use **`IdentityHashMap`**.
  - The hash code for an object should depend on all of the fields of the object.
  - Exclusive-or is a good way to combine hash codes because it directly depends on all bits, yet is very cheap.
  - Computation of hash code should be cheap, although some extra expense is justifiable if the code is cached with the object.
  - Achilles heel of hash functions: aliasing (unequal objects mapped to same code).

Two basic approaches to hashing

- Open addressing: all counters are stored directly in table. Collisions force reprobing which must be deterministic. Simple scheme is linear probing. But in practice, forget this approach. No significant advantage over direct chaining except in unusual situations.
- Direct chaining ("bucket hashing") Table consists of a block of linked list headers. There is a linked list for each hash code value. Actual hash entries are stored as separate objects in an auxiliary areas (usually the heap). Only significant weakness is less locality because object addresses are scattered across the heap. (Can be overcome by allocating objects within array blocks stored in heap, but this is a big book-keeping hassle.)

Sample Hash Table Code

- The MyHashMap code base implements exactly the same MapI interface as OOTreeMap.
- It is straightforward but ugly; it is classic procedural code encapsulated as a Java class to hide the procedural details. From the client's perspective, there is no way to detect that the implementation is procedural. The linked list Node class is a private nested class.
- Why did I use procedural coding? For a simple data structure like MyHashMap, the procedural code is tractable and signficantly more efficient that OO code (which is important in a library). In Java software development, almost nobody writes hash table implementations anymore! Everybody uses HashSet, HashMap, ConcurrentHashMap, and IdentityHashMap. (HashTable is obsolescent). If procedural code is easily encapsulated, significantly more efficient, and important to an applications overall efficiency, then I have no objection to writing procedural code. But note that the conjunction of these criteria doesn't arise very often.
- Each bucket is a singly linked list. Within a bucket, linear searching is necessary.
- Optimization trick in cases where load factor is high: move last referenced item to front of list on each access. (I did not bother with this optimization, because it only makes sense when buckets get large, which this implementation prevents.)
- Large load factors should be avoided if possible. MyHashMap never lets it get above 1.0. In an application written in a high-level language (not C/C++), it is almost always possible. Why?
- When the table gets full, double the table size and rehash! MyHashMap does this and it only takes about 10 lines of code. The asymptotic cost is zero! (Why? The sum of 2^k, k = 0, … N-1 = 2^N-1.)

Exam preparation

- Read the notes on OO Design up through end of Ch. 2.
- Emphasis on how to write clean OO code using design patterns. The functional subset is important. Given a simple Scheme program manipulating inductively defined data, you should be able to convert it to a corresponding Java program (same recursion pattern) defined on a corresponding composite class hierarchy. Then perform tail recursion optimization. Then convert it to a loop. More precisely
    - Convert the data definition to OO form (composite with optional singleton).
    - Convert the Scheme function to a method defined over the composite using the interpreter pattern.
    - Convert method to tail recursive form by introducing a help method.

Exam preparation cont.

- Convert tail recursive method with help function to a loop (with no help function). Loop iteration corresponds to a call on help function.
- Convert interpreter definition of method to visitor form.
- Understanding generics helps.

For Next Class

- Exam over OO material will be distributed on Friday, April 10, and due by 11:59PM on Friday, April 17 in my office.
- Tic-tac-toe homework due Friday. We are demoting the Alpha-Beta pruning part to extra credit. Like the last assignment, you only have to write a modest part of the total application. Have fun.
- Please, please read my notes on OO Design.
- Friday’s lecture will discuss HW 12.