CSC161 2010F *Imperative Problem Solving*

Class 44: Hash Tables

Held: Wednesday, 17 November 2010

Summary: We consider one of the more important ADTs, the dictionary, and an equally important implementation of dictionaries, the hash table.

Related Pages:

- EBoard.
- Reading: K&R 6.5 and 6.6.
- Due: Assignment 8.

Notes:

- There is no reading for Friday!
- EC for tomorrow's Thursday Extra: Experimental Algorithmics.
- EC for some part of the Rosenfield Corn Symposium.
- I will be unavailable after 1:45 today.

Overview:

- ADTs and Data Structures.
- The Dictionary ADT.
- Implementing Dictionaries with Association Lists.
- Hash Tables.

ADTs and Data Structures

- As you've started to see, in our algorithm design, different ways of organizing data give you different advantages and disadvantages.
- For example, in CSC151 you learned about arrays and lists. You've just learned about files.
    - Both provide a way of grouping data.
    - Lists are easy to extend and shrink.
    - Arrays are mutable.
    - Arrays provide fast access to individual elements.
    - Files provide persistent storage.
- It's time to start looking at other mechanisms for organizing data.
- In designing structures, we may look at three related issues:
    - The high-level overview of what operations the structure provides. We often refer to this as the *abstract data type* or ADT.
    - The high-level details of the implementation of the ADT. We often refer to this as the *data structure*. One ADT may have many corresponding data structures.

    - The low-level details of the implementation of the data structure. (E.g., do we implement it with an array, or a bunch of pairs, or ....)
- Note: While we will often start with the ADT and work down to the implementing data structure, the history of CS suggests the reverse: People designed data structures and then later realized that they should generalize.

The Dictionary ADT

- The *Dictionary* is one of everyone's favorite ADT's. A dictionary stores key/value pairs and typically provides the following basic operations.
    - Add a key/value pair to the dictionary.
    - Given a key, look up the corresponding value in the dictionary.
- Sometimes we add additional operations
    - Delete a key/value pair
    - Given a value, find one (all) of the corresponding keys.
    - Do something for each key/value pair
    - Determine whether a key appears in the dictionary
    - ...
- In a typed language like C, we often limit the type of the key and value (e.g., to strings).
- Dictionaries have many other names: I have seen them referred to as Maps, Tables, Hashes, and Associative Arrays

Association Lists

- You've already seen one implementation of dictionaries: Association lists.
- Association lists are easy to implement and use.
- But they are slow.

Hash Tables

- Hash tables are one of the most common mechanisms for implementing dictionaries.
- Hash tables are fast (Expected time: A constant number of steps.)
- Key idea: Arrays are fast. So turn the key into a number that you can use to index the array.
- We use a *hash function* to turn keys into integers. We then mod the integer by the size of the table.
- A hash function must always give the same integer for the same key.
- A hash function should give different integers for different keys. (Counting suggests that this is not always possible.)
- What do we do about collisions? Ah, that's the rub.
    - We can repeatedly add some constant and look elsewhere in the table until we find a free spot.
    - We can store a list/array of values at each integer.
