Searching and Sorting

Lecture 16 (7/31/2006)

Useless Fact of the Day

• “E” is the most common letter used in English words. About 11.2% of words in the dictionary have an “e” in them.

• In 1939, Ernest Vincent Wright wrote an entire 50,000-word book without using the letter “e” (even avoiding abbreviations like “Mr.” or “Mrs.,” where the full word has an “e” in it).

• http://www.spinelessbooks.com/gadsby/

Search

• Also known as “retrieval,” or “looking something up”

• One of the most fundamental operations in computing and in normal life

• Without a computer:
  • Indexes
  • Table of Contents
  • Reference books
  • Finding a room in a building

Linear Search

• A.k.a. “sequential” search

• Go through the material one item at a time

• Always works!

• Can be extremely slow

• With a completely unsorted list of stuff, it’s the only way to search it!

• If no rooms in a building had numbers and there were no signs, no method of searching for your classroom would be faster than looking in every room
Binary Search

- Way, way faster than linear search!
- Requires a sorted list of stuff to search in
  - For example, a phone book
- The algorithm:
  1. Pick the middle thing
  2. Is this what you're looking for, or is it before or after this?
  3. Discard half of the list
  4. Repeat until you pick the thing you're looking for

Hashing

- Allows us to get directly at information, instead of searching for it...if we store the information in a hash table instead of a list
- A hash table assigns a number (usually) to each thing in the table. The number is calculated based on some properties of the thing
  - Example: We could make a hash table for storing names up to 10 letters long. We create the hash code for a name by adding the name's letters' positions in the alphabet (for example, Bob would be $2 + 15 + 2 = 19$). Our hash table will have 260 bins (because what if someone's name is Zzzzzzzzzz?).
  - If we wanted to find “Bob,” we’d calculate the hash code for “Bob” and get 19, and we immediately know “Bob” is stored in bin number 19.
  - The problem comes when we have collisions -- people's names that have the same hash code (like “Jim” -- $10 + 9 + 13 = 32$, and “Zabba” -- $26 + 1 + 2 + 2 + 1 = 32$). We have to solve this by making sub-lists, for multiple people in each bin, but then we can't get directly at everyone anymore...

Search Performance

- $O(N)$ (e.g. “order N,” or $\sim N$, or linear time)
  - Example: linear search
  - The amount of time we need is directly proportional to the amount of stuff we're searching through
- $O(\log(N))$ (or $\sim \log(N)$)
  - Example: binary search
  - The amount of time we need is proportional to the log (usually base-2) of the amount of stuff
- $O(1)$ (or constant time)
  - Example: hashing (without too many collisions...)
  - The amount of time we need doesn't change, no matter how much stuff we're searching through

Sorting

- A fundamental part of many algorithms and procedures
- Often you must sort your stuff before you can do other things with it
  - Example: binary search requires a sorted list
- Also, a requirement for your data to be readable by humans! (could you stand reading a randomly-shuffled phone book?)
Selection Sort

- Very simple (and slow) sorting algorithm
- 1. Find the smallest thing in the non-sorted part of the list
- 2. Swap it with the first thing in the non-sorted part of the list
- 3. Ignore the already-sorted part of the list, and repeat until you're done

Selection Sort Performance

- \(O(N^2)\)
- The amount of time we need is proportional to the square of the number of things we’re sorting
- On each pass we have to make \(N/2\) comparisons on average (to search for the smallest thing), and we have to make \(N-1\) passes (to find all \(N-1\) small things)
- This is a total of \((N-1) \times N/2\) comparisons, which is “order \(N\) squared,” or \(O(N^2)\)

Selection Sort: Example

<table>
<thead>
<tr>
<th></th>
<th>2</th>
<th>4</th>
<th>7</th>
<th>3</th>
<th>1</th>
<th>8</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>4</td>
<td>7</td>
<td>3</td>
<td>2</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>7</td>
<td>3</td>
<td>4</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>7</td>
<td>1</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>7</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>8</td>
<td>7</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>

Sort Performance

- \(O(N^2)\) -- Simple but slow
  - Selection sort, insertion sort, bubble sort
- \(O(N \log(N))\) -- Fastest possible in general
  - QuickSort, Merge Sort
- \(O(N)\) -- Fastest, but only used in special cases
  - Bucket Sort
Sorting Java Collections

• Static methods in the “Arrays” class and the “Collections” class (which need to be imported)

• For arrays:
  • Arrays.sort(theArray) -- returns void
  • Arrays.binarySearch(theArray, theThing) -- returns an int, which is the index of “theThing” in “theArray”

• For ArrayLists and other collections:
  • Collections.sort(theList) -- returns void
  • Collections.binarySearch(theList, theThing) -- returns an int, which is the index of “theThing” in “theList”