Lecture 36: Radix vs. Comparison Sorting, Sorting and Data Structures Conclusion

11/20/2020

Intuitive: Radix Sort vs. Comparison Sorting

Merge Sort Runtime

- Merge Sort requires Theta(N log N) compares
- What is Merge Sort's runtime on strings of length W?
    - It depends!
        - Theta(N log N) if each comparison takes constant time
            - Example: Strings are all different in top character
        - Theta(WN log N) if each comparison takes Theta(W) time
            - Example: Strings are all equal

LSD vs. Merge Sort

- The facts
    - Treating alphabet size as constant, LSD Sort has runtime Theta(WN)
    - Merge Sort has runtime between Theta(N log N) and Theta(WN log N)
- Which is better? It depends
    - When might LSD sort be faster
        - Sufficiently large N
        - If strings are very similar to each other
            - Each Merge Sort comparison costs Theta(W) time
    - When might Merge Sort be faster?
        - If strings are highly dissimilar from each other
            - Each Merge Sort comparison is very fast

Cost Model: Radix Sort vs. Comparison Sorting

Alternate Approach: Picking a Cost Model

- An alternate approach is to pick a cost model
    - We'll use number of characters examined
    - By "examined", we mean:
        - Radix Sort: Calling charAt in order to count occurrences of each character
        - Merge Sort: Calling charAt in order to compare two Strings

MSD vs. Mergesort

- Suppose we have 100 strings of 1000 characters each
    - Estimate the total number of characters examined by MSD Radix Sort if **all strings are equal**

        - For MSD Radix Sort, in the worst case (all strings equal), every character is examined exactly once. Thus we have exactly 100,000 total character examinations
    - Estimate the total number of characters examined by Merge Sort **if all strings are equal**
        - Merging 100 items, assuming equal items results in always picking left
            - Total characters examined in a single merge operation: 50 * 2000 = 100,000 (= N/2 * 2000 = 1000N)
        - In total, we must examine approximately 1000Nlog_2(N) total characters
            - 100000 + 50000*2 + 25000*4 + ... = ~660,000

MSD vs. Mergesort Character Examinations

- For N equal strings of length 1000, we found that:
    - MSD radix sort will examine ~1000N characters
    - Mergesort will examine ~1000Nlog_2(N) characters
- If character examination are an appropriate cost model, we'd expect Merge Sort to be slower by a factor of log_2(N)

Empirical Study: Radix Sort vs. Comparison Sorting

Computational Experiment Results

- Computational experiment for W = 100
    - As we expected, Merge sort considers log_2(N) times as many characters
    - But empirically, Mergesort is MUCH faster than MSD sort
        - Our cost model isn't representative of everything that is happening
        - One particularly thorny issue: The "Just In Time" (JIT) Compiler

An Unexpected Factor: The Just-In-Time Compiler

- Java's Just-In-Time COmpiler secretly optimizes your code when it runs
    - The code you write is not necessarily the code that executes!
    - As your code runs, the "interpreter" is watching everything that happens
        - If some segment of code is called many times, the interpreter actually studies and re-implements your code based on what it learned by watching WHILE ITS RUNNING (!!)
            - Example: Performing calculations whose results are unused

Rerunning Our Empirical Study Without JIT

Computational Experiments Results with JIT disabled

- Results with JIT disabled (using the -Xint option)
    - Both sorts are MUCH slower than before
    - Merge sort is slower than MSD (though not by as much we predicted)
    - What this tells us: The JIT was somehow able to massively optimize the compareTo calls
        - Makes some intuitive sense: Comparing "AAA...A" to "AAA...A" over and over is redundant

So Which is Better? MSD or MergeSort?

- We showed that if the JIT is enabled, merge sort is much faster for the case of equal strings, and slower if JIT is disabled
  - Since JIT is usually on, I'd say merge sort is better for this case
- Many other possible cases to consider:
  - Almost equal strings (maybe the trick used by the JIT won't work?)
  - Randomized strings
  - Real world data from some dataset of interest
- In real world applications, you'd profile different implementations of real data and pick one

Bottom Line: Algorithms Can be Hard to Compare

- Comparing algorithms that have the same order of growth is challenging
  - Have to perform computational experiments
  - Experiments can be tricky due to optimizations like the JIT in Java
- Note: There's always the chance that some small optimization to an algorithm can make it significantly faster

Radix Sorting Integers (61C Preview)

Linear Time Sorting

- As we've seen, estimating radix sort vs. comparison sort performance is very hard
  - But in the very large N limit, it's easy. Radix sort is simply faster!
    - Treating alphabet size as constant, LSD Sort has runtime Theta(WN)
    - Comparison sorts have runtime Theta(N log N) in the worst case
- Issue: We don't have a charAt method for integers
  - How would you LSD radix sort an array of integers
    - Convert into a String and treat as a base 10 number. Since the maximum Java int is 2,000,000,000, W is also 10
    - Could modify LSD radix sort to work natively on integers
      - Instead of using charAt, maybe write a helper method like getDthDigit(int D, int d). Example: getDthDigit(15009, 2) = 5

LSD Radix Sort on Integers

- Note: There's no reason to stick with base 10!
  - Could instead treat as a base 16, base 256, base 65536 number
  - Ex: 512,312 in base 16 is a 5 digit number
  - Ex: 512,312 in base 256 is a 3 digit number

Relationship Between Base and Max # Digits

- For Java integers:
  - R=10, treat as a base 10 number. Up to 10 digits
  - R=256, treat as a base 256 number. Up to 4 digits
  - $ = 65335, treat as a base 65536 number. Up to 2 digits
- Interesting fact: Runtime depends on the alphabet size

  - As we saw with the city sorting last time, R = 2147483647 will result in a very slow radix sort (since it's just counting sort)

Another Counting Sort

- Results of a computational experiment:
  - Treating as a base 256 (4 digits), LSD radix sorting integers easily defeats Quicksort

Sorting Summary

Sorting Landscape

- Three basic flavors: Comparison, Alphabet, and Radix based
  - Comparison based algorithms:
    - Selection -> If heapify first -> Heapsort
    - Merge
    - Partition
    - Insertion -> If insert into BST, equiv. to Partition
  - Small-Alphabet (e.g. Integer) algorithms:
    - Counting
  - Radix Sorting algorithms (require a sorting subroutine)
    - LSD and MSD use Counting as a subroutine
- Each can be used in different circumstances, but the important part was the analysis and the deep thought!

Sorting vs. Searching

- We've now concluded our study of the "sort problem"
  - During the data structures part of the class, we studied what we called the "search problem": Retrieve data of interest
  - There are some interesting connections between the two
- Search-By-Key-Identity Data Structures
  - Sets and Maps:
    - 2-3 Tree (Uses compareTo(), Analogous to Comparison-Based)
    - RedBlack Tree (Uses compareTo(), Analogous to Comparison-Based)
    - Separate Chaining (Searches using hashCode() and equals(), Roughly Analogous to Integer Sorting)
    - Tries (searches digit-by-digit Roughly Analogous to Radix Sorting)