Radix Sorts

I) Strings in Java

String = sequence of characters

- C char data type = Typically an 8-bit integer
  - supports 7-bit ASCII
  - can represent only 256 characters
- Java char data type = 16-bit unsigned integer
  - supports original 16-bit Unicode
  - supports 21-bit Unicode 3.0 (awkwardly)

String data type in Java

- sequence of characters (immutable)
- length = no. of characters
- indexing = get the i^{th} character
- substring extraction = get a contiguous subsequence of characters
- string concatenation = append one

character to end of another string.

* length(), charAt(), substring() = in constant time → O(1)

* concat → O(N)

Memory = 40 + 2N bytes for a virgin string of length N

String Builder = sequence of characters (mutable)

* substring() takes O(N)

* concat() → O(1)* amortized

• underlying implementation = resizing char[] array and length

! How to efficiently reverse a string?

=> String → O(N²)

=> StringBuilder → O(N)

Alphabets

Digital key = sequence of digits over fixed alphabet

Radix = number of digits R in alphabet

$R \longrightarrow \log_2 R$ = bytes to represent

Key Indexed Counting

Lower bound $\sim N \lg N$ compares required by any compare-based algorithm

* Can we do better? Yes! if we don't depend on key compares

Assumption: keys are integers between 0 and $R-1$ => Can use key as an array index

! Remark: keys may have associated data => can't just count up no. of keys of each value.

Goal: Sort an array a[] of N integers between 0 and R-1

* **Key-indexed counting** uses ~ 11 N + 4R array accesses to sort N items whose keys are integers between 0 and R-1
* Key-indexed counting uses extra space proportional to N + R
* We move things in the order that we see them ⇒ STABLE

LSD Radix sort

Least-Significant-Digit-first string sort

LSD string (radix) sort

- consider characters from right to left
- stably sort using d^{th} character as the key (using key-indexed counting)

* Fixed length keys → LSD runs in $O(2wx1)$

w = length of keys small constant

Problem => sort one million 32 bit integers

MSD Radix Sort

MSD string (radix) sort

- partition array into R pieces according to first character (use key-index counting)
- recursively sort all strings that start with each character (key-indexed counts delineate subarrays to sort)

* variable-length strings -> treat strings as if they had an extra char at the end (smaller than any char)

* you have to have a count array with the size of the alphabet

Observation 1 Much too slow for small subarrays

Observation 2 Huge no. of small sub arrays because of recursion

MSD performance - no. of characters examined

- MSD examines just enough to sort the keys
- $\boxed{\sim N \log_R N}$ random

MSD = accesses memory "randomly" (cache inefficient)

- inner loop has a lot of instructions
- extra space for $\begin{cases} \text{count}[\,] \\ \text{aux}[\,] \end{cases}$

Quicksort - linearithmic number of string compares (not linear)

- has to rescan many characters in keys with long prefix matches

3-Way radix Quicksort

- do 3-way partitioning on the d-th character
- pick partitioning item
- use first character to partition into — less / equal / greater subarrays
- recursively sort subarrays excluding first character for middle subarray

Standard QS - uses $\sim 2N \ln N$ string comp.

3-way string - uses $\sim 2N \ln N$ character comparisons

Suffix Arrays

- Keyword in context search
  - Suffix sorting
    - take input string and form suffixes
    - sort suffixes to bring repeated substrings together
      - * preprocess: suffix sort the text
      - * query: binary search for query => scan until mismatch

Longest repeated substring

given a string of N characters, find the longest repeated substring

Brute force:

- try all indices i, j for start of possible match
- compute LCP (longest common prefix) for each pair

- form suffixes
- sort suffixes to bring repeated substrings together
- compute longest prefix between adjacent suffixes

Suffix sorting challenge

Solution: Manber-Myers MSD algorithm