Algorithms: Design & Practice

Deepak Kumar
Bryn Mawr College
Spring 2017
Course Essentials

• Algorithms – Design & Practice
  • How to design
  • Learn some good ones
  • How to implement – practical considerations

• How to write about algorithms
• How to present algorithms
• How to discuss algorithms
Five Key Elements of Problem Solving

• What to compute – problem-spec
• How to compute – Known solutions
• How to implement – what language and implementation
• Is it correct? – Correctness, and testing
• How to measure performance? – Efficiency, empirical measurements
An Algorithm for Algorithm Development

def algorithm-development (problem-spec):

    correct = false
    while not correct or not fastEnough(runningTime)
        algorithm = deviseAlgorithm(problem-spec)
        correct = analyzeCorrectness(algorithm)
        runningTime = analyzeEfficiency(algorithm)

    return algorithm
### Topics & Algorithms

<table>
<thead>
<tr>
<th>Correctness</th>
<th>Complexity</th>
<th>C/C++</th>
<th>Java</th>
<th>Python</th>
</tr>
</thead>
<tbody>
<tr>
<td>Runtime</td>
<td>Recursion</td>
<td>Fisher-Yates</td>
<td>Linear Search</td>
<td>Binary Search</td>
</tr>
<tr>
<td>Selection Sort</td>
<td>Insertion Sort</td>
<td>Merge Sort</td>
<td>Heap Sort</td>
<td>Quick Sort</td>
</tr>
<tr>
<td>Hybrid Quicksort</td>
<td>Dual Pivot QS</td>
<td>Anagrams</td>
<td>Hash Tables</td>
<td>Hash Functions</td>
</tr>
<tr>
<td>Graphs</td>
<td>Directed Acyclic Graphs</td>
<td>Breadth-First Search</td>
<td>Depth-First Search</td>
<td>The W-O-M-A-N Puzzle</td>
</tr>
<tr>
<td>Critical Path</td>
<td>Topological Sort</td>
<td>Shortest Paths</td>
<td>Dijkstra’s</td>
<td>Bellman-Ford Floyd-Warshall</td>
</tr>
<tr>
<td>Soundex</td>
<td>Metaphone</td>
<td>Regular Expressions</td>
<td>Python Library</td>
<td>Java Library</td>
</tr>
<tr>
<td>K-Nearest Neighbors</td>
<td>Decision Trees</td>
<td>Neural Networks</td>
<td>Data Compression</td>
<td>Huffman Encoding</td>
</tr>
<tr>
<td>LZ Compression</td>
<td>LZW Compression</td>
<td>Databases</td>
<td>Digital Signatures</td>
<td>Cryptographic Hash Functions</td>
</tr>
</tbody>
</table>
What we didn’t do?
What we didn’t do? Strings!!

• Longest Common Subsequence

• Matching – Knuth-Morris Pratt, Boyer Moore, etc.

• String Transformation (Dynamic Programming)
What you **Liked**, **Liked to Program**, and **Hated**.
Programming Languages

- Python: 45%
- Java: 39%
- C: 15%
- C++: 1%
Core Foundations – Primitive Types

• Machine representations of int, char, float, double, etc.

• What operations are available on them

<table>
<thead>
<tr>
<th>Type</th>
<th>Contains</th>
<th>Default</th>
<th>Size</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean</td>
<td>true or false</td>
<td>false</td>
<td>1 bit</td>
<td>NA</td>
</tr>
<tr>
<td>char</td>
<td>Unicode character</td>
<td>\u0000</td>
<td>16 bits</td>
<td>\u0000 to \uFFFF</td>
</tr>
<tr>
<td>byte</td>
<td>Signed integer</td>
<td>0</td>
<td>8 bits</td>
<td>-128 to 127</td>
</tr>
<tr>
<td>short</td>
<td>Signed integer</td>
<td>0</td>
<td>16 bits</td>
<td>-32768 to 32767</td>
</tr>
<tr>
<td>int</td>
<td>Signed integer</td>
<td>0</td>
<td>32 bits</td>
<td>-2147483648 to 2147483647</td>
</tr>
<tr>
<td>long</td>
<td>Signed integer</td>
<td>0</td>
<td>64 bits</td>
<td>-9223372036854775808 to 9223372036854775807</td>
</tr>
<tr>
<td>float</td>
<td>IEEE 754 floating point</td>
<td>0.0</td>
<td>32 bits</td>
<td>±1.4E-45 to ±3.4028235E+38</td>
</tr>
<tr>
<td>double</td>
<td>IEEE 754 floating point</td>
<td>0.0</td>
<td>64 bits</td>
<td>±4.9E-324 to ±1.7976931348623157E+308</td>
</tr>
</tbody>
</table>
Core Foundations – Arrays

- Machine representations: 1-D, 2-D, etc.
- What operations are available on them
- Know how to iterate, resize, partition, merge, insert, delete, shuffle, initialize
- Library facilities available
Core Foundations – Strings

• Machine representations

• What operations are available on them

• How to create, compare, copy, match, join, split, etc.

• Algorithms for string matching, transformation, etc.

String Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>str.length()</td>
<td>Get length of string</td>
</tr>
<tr>
<td>str.toLowerCase()</td>
<td>Convert to lower case</td>
</tr>
<tr>
<td>str.toUpperCase()</td>
<td>Convert to upper case</td>
</tr>
<tr>
<td>str.charAt(i)</td>
<td>What is at character i?</td>
</tr>
<tr>
<td>str.contains(...)</td>
<td>String contains another string?</td>
</tr>
<tr>
<td>str.startsWith(...)</td>
<td>String starts with some prefix?</td>
</tr>
<tr>
<td>str.indexOf(...)</td>
<td>What is the position of a character?</td>
</tr>
</tbody>
</table>

Advantage of String class: many built-in methods for String manipulation
Core Data Structures - Lists

• Arrays versus linked representations (single-, doubly-linked)

• How to define, create, iterate, insert, delete, search, etc.

• How dynamic allocation works?
Core Data Structures – Stacks & Queues

• Arrays versus linked representations (single-, doubly-linked)

• How to define, create, iterate, insert, delete, search, etc.

• How dynamic allocation works?

• Last-in First out, First-in First out. Dequeues

• Provided in library?
Core Data Structures – Heaps

• Machine representations (arrays)

• What operations are available on them

• How to create, insert, delete.

• $O(1)$ lookup (max, min), $O(\log n)$ insert/delete

• Priority Queues
Core Data Structures – Hash Tables

- Hash Functions, collisions, collision resolution (linear, quadratic probing, lists, etc.)
- Library?
- $O(1)$ insertion/delete/lookup (mostly!)
- Load factor affects performance
Core Data Structures – Trees

• Binary, Binary Search, Quad, etc.

• Properties: full, complete, balanced, etc.

• How algorithms are impacted (e.g. $O(\log n)$ insert, delete, search in BSTs)
Core Data Structures – Graphs

- Kinds of graphs (undirected, directed, weighted)

- Graph representations (adjacency matrix, array of linked lists, etc.)

- Graph algorithms (Topological sort, Dijkstra’s, Bellman Ford, Floyd Warshall, Minimum Spanning Tree, etc.)
...and

• Lots of other stuff!
Finally...

RTFM, please.
I'd like to thank Stack Overflow for this degree.
PLEASE,
Don’t be this person!!!
Thank you.
GOOD ALGORITHMS ARE YUGE!