1 - Basics

Monday, December 30, 2019 2:05 AM

This is an especially large lesson and may take two days.

The primary purpose of this course is to give a language to discuss and think in terms of (I.e. discussion with the self)

Topics for Today:

- Environment Setup
- Difference between ADT and Data Structure
- Algorithmic Thinking
- Models of Computation
- Vectors
- Linked Lists
- Stacks
- Heap
- Queues
- Naïve Binary Trees
- Tree traversal

Work:

- Implement templated vectors, linked lists, binary trees and traversal functions for each
- Easy/low-level programming exercises that I've already done
- Benchmark speed differences with different n

Q What's More Important Than Performance?

- Correctness & Robustness
- Simplicity/Maintainability
- Programmer Time
- User-Friendliness
- Security
- Extensibility
- "Algorithms are at the cutting edge of entrepreneurship" - 2005 Lecturer

ADT = "Abstract Data Type"

- Just the interface for a data-structure (what you would like it to do)
- No algorithms!
- Not a data-structure!
- Atomic types are often defined relative to these rather than some real hardware thing

ADTs

1) Queue (supports enqueueing(q,x) and dequeueing(q))
2) Stack (Push(S,x), Pop(S))
3) Dictionary (Insert(D,key,value), get(D,key)
4) Priority-Queue Insert(S,x), Extract-Max(S)
5) Set (Add(S,E), InSet(S,E))
6) Trivial example: List

7) Exercise: Polygon

Algorithms
Just instructions, so are beholden to the final representation, don't think of a computer

Proving Correctness
Not a huge focus, but we should know how

Loop Invariants
Usually we prove 3 statements:
1) Initialization: Invariant holds on first execution
2) Maintenance: If invariant held on all previous passes through the loop, it holds on current pass
3) Termination: If invariant holds at the end, then some desired property holds (e.g. algorithm is correct)

Models of Computation
Where ADTs lose their shine.
- Ram model
- Just read this and this
- There are also different classes of machines and what they can compute, then we have circuits that deal with "algebra" natively, which simplifies our problem-space, but these are all "weaker than" the ram model

O-Notation
After proving an algorithm works, we want to give its running time
Big-O, little-O, Big-Omega, Little-Omega, Theta

n	Worst AC Algorithm	Comment
$\leq [10..11]$	$O(n!), O(n^6)$	e.g. Enumerating permutations (Section 3.2)
$\leq [15..18]$	$O(2^n \times n^2)$	e.g. DP TSP (Section 3.5.2)
$\leq [18..22]$	$O(2^n \times n)$	e.g. DP with bitmask technique (Section 8.3.1)
≤ 100	$O(n^4)$	e.g. DP with 3 dimensions + $O(n)$ loop, ${}_nC_{k=4}$
≤ 400	$O(n^3)$	e.g. Floyd Warshall's (Section 4.5)
$\leq 2K$	$O(n^2 \log_2 n)$	e.g. 2-nested loops + a tree-related DS (Section 2.3)
$\leq 10K$	$O(n^2)$	e.g. Bubble/Selection/Insertion Sort (Section 2.2)
$\leq 1M$	$O(n \log_2 n)$	e.g. Merge Sort, building Segment Tree (Section 2.3)
$\leq 100M$	$O(n), O(\log_2 n), O(1)$	Most contest problem has $n \leq 1M$ (I/O bottleneck)

Remedial DS's
- Array (technically)
- Linked-List (singly and doubly)*
- Stack*
- Open-Addressing Hash-Table*
- Queue*
- Vector*
- Binary-Tree*

*Make a github repo and implement templated versions of these

Remedial Algo's
- Insertion-sort & Bubble-sort
- Left-to-right arithmetic evaluation
- https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-006-introduction-to-algorithms-fall-2011/lecture-videos/MIT6_006F11_lec03.pdf

Tree Traversal

- Inorder Traversal
- Postorder Traversal
- Preorder Traversal
- https://www.geeksforgeeks.org/tree-traversals-inorder-preorder-and-postorder/

^Implement these guys

Paradigms
https://en.wikipedia.org/wiki/Algorithmic_paradigm
And show hackerrank and a2oj list

Problems
Codeforces:
- 124A
- 459B
- 460B
- 357B
- 327A
- 289B
- 279B
- 260A

Exponential time algorithms = bad
Polynomial time algorithms = good

Things have to sink in over time, man