알고리즘

개요

이영석

yslee@cs-cnu.org

학습목표

- 알고리즘 문제 해결 전략 익히기
  - 복잡도
  - 샘플 문제들
    - 완전탐색(Exhaustive search or brute-force)
    - 대안
      - 정렬
      - 자료구조
      - (재귀)

코딩 테스트 문제 해결의 이해

- 정확한 입출력
- 시간제한
  - 예) 복잡도 이해: $O(n)$, $O(n \log n)$, $O(n^2)$, $O(n^3)$, $O(2^n)$, $O(n!)$
  - 컴퓨터의 CPU 클럭 1 GHz
    - 대략 1초에 1,000,000,000번 명령 수행 가능
    - $O(n)$이면 입력 크기 최대 1,000,000,000
    - $O(n^2)$이면 입력 크기 최대 30,000
    - $O(n^3)$이면 입력 크기 최대 1,000

Notation	Name	Example
$O(1)$	constant	Determining if a binary number is even or odd; Calculating $(-1)^n$; Using a constant-size lookup table
$O(\log \log n)$	double logarithmic	Number of comparisons spent finding an item using interpolation search in a sorted array of uniformly distributed values
$O(\log n)$	logarithmic	Finding an item in a sorted array with a binary search or a balanced search tree as well as all operations in a Binomial heap
$O((\log n)^c)$ $c>1$	polylogarithmic	Matrix chain ordering can be solved in polylogarithmic time on a parallel random-access machine.
$O(n^c)$ $0<c<1$	fractional power	Searching in a k-d tree
$O(n)$	linear	Finding an item in an unsorted list or in an unsorted array; adding two n-bit integers by ripple carry
$O(n \log^* n)$	n log-star n	Performing triangulation of a simple polygon using Seidel's algorithm, or the union–find algorithm. Note that $\log^*(n) = \begin{cases} 0, & \text{if } n \leq 1 \\ 1 + \log^*(\log n), & \text{if } n > 1 \end{cases}$
$O(n \log n) = O(\log n!)$	linearithmic, loglinear, quasilinear, or "n log n"	Performing a fast Fourier transform; Fastest possible comparison sort; heapsort and merge sort
$O(n^2)$	quadratic	Multiplying two n-digit numbers by a simple algorithm; simple sorting algorithms, such as bubble sort, selection sort and insertion sort; (worst case) bound on some usually faster sorting algorithms such as quicksort, Shellsort, and tree sort
$O(n^c)$	polynomial or algebraic	Tree-adjoining grammar parsing; maximum matching for bipartite graphs; finding the determinant with LU decomposition
$L_n[\alpha, c] = e^{(c+o(1))(\ln n)^\alpha (\ln \ln n)^{1-\alpha}}$ $0<\alpha<1$	L-notation or sub-exponential	Factoring a number using the quadratic sieve or number field sieve
$O(c^n)$ $c>1$	exponential	Finding the (exact) solution to the travelling salesman problem using dynamic programming; determining if two logical statements are equivalent using brute-force search
$O(n!)$	factorial	Solving the travelling salesman problem via brute-force search; generating all unrestricted permutations of a poset; finding the determinant with Laplace expansion; enumerating all partitions of a set

Common Data Structure Operations

Data Structure	Time Complexity								Space Complexity
	Average				Worst				Worst
	Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion	
Array	Θ(1)	Θ(n)	Θ(n)	Θ(n)	O(1)	O(n)	O(n)	O(n)	O(n)
Stack	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Queue	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Singly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Doubly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Skip List	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n log(n))
Hash Table	N/A	Θ(1)	Θ(1)	Θ(1)	N/A	O(n)	O(n)	O(n)	O(n)
Binary Search Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)
Cartesian Tree	N/A	Θ(log(n))	Θ(log(n))	Θ(log(n))	N/A	O(n)	O(n)	O(n)	O(n)
B-Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Red-Black Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Splay Tree	N/A	Θ(log(n))	Θ(log(n))	Θ(log(n))	N/A	O(log(n))	O(log(n))	O(log(n))	O(n)
AVL Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
KD Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)

https://www.bigocheatsheet.com/

Array Sorting Algorithms

Algorithm	Time Complexity			Space Complexity
	Best	Average	Worst	Worst
Quicksort	Ω(n log(n))	Θ(n log(n))	O(n^2)	O(log(n))
Mergesort	Ω(n log(n))	Θ(n log(n))	O(n log(n))	O(n)
Timsort	Ω(n)	Θ(n log(n))	O(n log(n))	O(n)
Heapsort	Ω(n log(n))	Θ(n log(n))	O(n log(n))	O(1)
Bubble Sort	Ω(n)	Θ(n^2)	O(n^2)	O(1)
Insertion Sort	Ω(n)	Θ(n^2)	O(n^2)	O(1)
Selection Sort	Ω(n^2)	Θ(n^2)	O(n^2)	O(1)
Tree Sort	Ω(n log(n))	Θ(n log(n))	O(n^2)	O(n)
Shell Sort	Ω(n log(n))	Θ(n(log(n))^2)	O(n(log(n))^2)	O(1)
Bucket Sort	Ω(n+k)	Θ(n+k)	O(n^2)	O(n)
Radix Sort	Ω(nk)	Θ(nk)	O(nk)	O(n+k)
Counting Sort	Ω(n+k)	Θ(n+k)	O(n+k)	O(k)
Cubesort	Ω(n)	Θ(n log(n))	O(n log(n))	O(n)

https://www.bigocheatsheet.com/

입력 크기와 실전 복잡도

Table 3.1 Estimating time complexity from input size

Input size	Expected time complexity
$n \leq 10$	$O(n!)$
$n \leq 20$	$O(2^n)$
$n \leq 500$	$O(n^3)$
$n \leq 5000$	$O(n^2)$
$n \leq 10^6$	$O(n \log n)$ or $O(n)$
n is large	$O(1)$ or $O(\log n)$

Guide to Competitive Programming, Springer, 2017

정리

- 기본적인 복잡도
- 문제해결방법의 이해
  - 완전탐색