SELF STUDY

THEORIES AND IMPLEMENTATION RELATED TO ALGORITHMS

Advanced Algorithm

Compilation of my study materials

Author: Dr. Md Arafat Hossain Khan

Last updated : May 24, 2022

Contents

Chapter 1

Fundamental Concepts

1.1 Asymptotic Notation / Bachmann–Landau Notation

Big-$\mathcal{O}$ notation is a mathematical notation that describes the limiting behavior of a function when the argument tends towards a particular value or infinity. Big O is a member of a family of notations invented by Paul Bachmann, Edmund Landau and others, collectively called Bachmann-Landau notation or asymptotic notation. The letter $\mathcal{O}$ was chosen by Bachmann to stand for *Ordnung*, meaning the order of approximation [1].

The definitions of the various asymptotic notations are 'closely' (I will address this particular word later) related to the definition of a limit. Let f and g be two functions $f, g : \mathbb{N} \to \mathbb{R}_+$. $\lim_{n\to\infty} f(n)/g(n)$ reveals a lot about the asymptotic relationship between f and g, provided the limit exists. Therefore, skill with limits from elementary calculus can be helpful in working out asymptotic relationships. Thus I will introduce it in both ways.

Definition 1: Big-$\mathcal{O}$

Let f and g be two functions $f, g : \mathbb{N} \to \mathbb{R}_+$. We say that

$$f \in \mathcal{O}(g)$$

if $\exists\ c \in \mathbb{R}_+$ and $n_0 \in \mathbb{N}$ such that for every integer $n \geq n_0$,

$$f(n) \leq cg(n).$$

i.e.

$$\mathcal{O}(g) = \{f : \exists\ c \in \mathbb{R}_+, \exists n_0 \in \mathbb{N} \ni\ \forall n > n_0,\ f(n) \leq cg(n)\}$$

Interpretation:

f grows NO FASTER than g

- f is (asymptotically) less than or equal to g.
- Big-$\mathcal{O}$ gives an asymptotic upper bound g of f.

Limit interpretation:

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} \neq \infty \qquad \text{OR} \qquad \lim_{n\to\infty} \frac{f(n)}{g(n)} \in [0,\infty) \quad \Longrightarrow \quad f \in \mathcal{O}(g)$$

Caution: It's possible that $f = \mathcal{O}(g)$ even though $f(n)/g(n)$ does not converge to a limit, e.g.

$$f(n) = (2 + (-1)^n)\ g(n)$$

So the opposite implication in the limit interpretation is not necessarily true. For the equivalence we need something more general than $\lim_{n\to\infty} \frac{f(n)}{g(n)}$, something that always exists. So the modified relation is -

$$\limsup_{n\to\infty} \frac{f(n)}{g(n)} \neq \infty \qquad \text{OR} \qquad \limsup_{n\to\infty} \frac{f(n)}{g(n)} \in [0,\infty) \quad \Longleftrightarrow \quad f \in \mathcal{O}(g)$$

Definition 2: Big-Θ

Let f and g be two functions $f, g : \mathbb{N} \to \mathbb{R}_+$. We say that

$$f \in \Theta(g)$$

if $\exists\ c_1, c_2 \in \mathbb{R}_+$ and $n_0 \in \mathbb{N}$ such that for every integer $n \geq n_0$,

$$c_1 g(n) \leq f(n) \leq c_2 g(n).$$

i.e.

$$\Theta(g) = \{f : \exists\ c_1, c_2 \in \mathbb{R}_+, \exists n_0 \in \mathbb{N} \ni\ \forall n > n_0,\ c_1 g(n) \leq f(n) \leq c_2 g(n)\}$$

Interpretation:

f grows at the SAME rate as g

- f is (asymptotically) equal to g.
- f is bounded above and below by g.
- Big-Θ gives an asymptotic equivalence.

Limit interpretation:

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} \neq 0, \infty \qquad \text{OR} \qquad \lim_{n\to\infty} \frac{f(n)}{g(n)} \in (0,\infty) \quad \Longrightarrow \quad f \in \Theta(g)$$

Caution: It's possible that $f(n) = \Theta(g)$ even though $f(n)/g(n)$ does not converge to a limit, e.g.

$$f(n) = (2 + (-1)^n)\ g(n)$$

So the opposite implication in the limit interpretation is not necessarily true. For the equivalence we need something more general than $\lim_{n\to\infty} \frac{f(n)}{g(n)}$, something that always exists. So the modified relation is -

$$\begin{aligned} f \in \Theta(g) &\iff f \in \mathcal{O}(g) \text{ and } f \in \Omega(g) \\ &\iff \limsup_{n\to\infty} \frac{f(n)}{g(n)} \in [0,\infty) \text{ and } \limsup_{n\to\infty} \frac{f(n)}{g(n)} \in (0,\infty] \\ &\iff \limsup_{n\to\infty} \frac{f(n)}{g(n)} \in (0,\infty) \end{aligned}$$

Definition 3: Big-Ω

Let f and g be two functions $f, g : \mathbb{N} \to \mathbb{R}_+$. We say that

$$f \in \Omega(g)$$

if $\exists\ c \in \mathbb{R}_+$ and $n_0 \in \mathbb{N}$ such that for every integer $n \geq n_0$,

$$f(n) \geq cg(n).$$

i.e.

$$\Omega(g) = \{f : \exists\ c \in \mathbb{R}_+, \exists n_0 \in \mathbb{N} \ni\ \forall n > n_0,\ f(n) \geq cg(n)\}$$

Interpretation:

f grows AT LEAST AS FAST AS g

- f is (asymptotically) greater than or equal to g.
- Big-Ω gives an asymptotic lower bound g of f.

Limit interpretation:

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} \neq 0 \quad \text{OR} \quad \lim_{n\to\infty} \frac{f(n)}{g(n)} \in (0,\infty] \quad \implies f \in \Omega(g)$$

Caution: It's possible that $f(n) = \Omega(g)$ even though $f(n)/g(n)$ does not converge to a limit, e.g.

$$f(n) = (2 + (-1)^n)\ g(n)$$

So the opposite implication in the limit interpretation is not necessarily true. For the equivalence we need something more general than $\lim_{n\to\infty} \frac{f(n)}{g(n)}$, something that always exists. So the modified relation is -

$$\liminf_{n\to\infty} \frac{f(n)}{g(n)} \neq 0 \quad \text{OR} \quad \liminf_{n\to\infty} \frac{f(n)}{g(n)} \in (0,\infty] \quad \iff f \in \Omega(g)$$

Whenever $\liminf_{n\to\infty} x_n$ and $\limsup_{n\to\infty} x_n$ both exist, we have

$$\liminf_{n\to\infty} x_n \leq \limsup_{n\to\infty} x_n.$$

Thus the modified definition can be obtained by the following,

$$\limsup_{n\to\infty} \frac{f(n)}{g(n)} \neq 0 \qquad \text{OR} \qquad \limsup_{n\to\infty} \frac{f(n)}{g(n)} \in (0,\infty] \quad \iff f \in \Omega(g)$$

Definition 4: Little-o

Let f and g be two functions $f, g : \mathbb{N} \to \mathbb{R}_+$. We say that

$$f \in o(g)$$

if $\forall\ c \in \mathbb{R}_+$, $\exists n_0 \in \mathbb{N}$ such that for every integer $n \geq n_0$,

$$f(n) < cg(n).$$

i.e.

$$o(g) = \{f : \forall\ c \in \mathbb{R}_+, \exists n_0 \in \mathbb{N} \ni \ \forall n > n_0,\ f(n) < cg(n)\}$$

Interpretation:

f grows SLOWER than g

- f is (asymptotically) strictly less than g.
- Little-o says with respect to the function g asymptotically f becomes insignificant.

Limit interpretation:

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0 \quad \iff f \in o(g)$$

Caution: There is NO caution here in the definition of little-o, since if the limit is equal 0 it must exit.

Definition 5: Little-ω

Let f and g be two functions $f, g : \mathbb{N} \to \mathbb{R}_+$. We say that

$$f \in \Omega(g)$$

if $\forall\ c \in \mathbb{R}_+$, $\exists n_0 \in \mathbb{N}$ such that for every integer $n \geq n_0$,

$$f(n) > cg(n).$$

i.e.

$$\Omega(g) = \{f : \forall\ c \in \mathbb{R}_+, \exists n_0 \in \mathbb{N} \ni \ \forall n > n_0,\ f(n) > cg(n)\}$$

Interpretation:

f grows FASTER than g

- f is (asymptotically) strictly greater than g.
- Little-ω says that asymptotically f makes g insignificant.

Limit interpretation:

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty \quad \iff f \in \Omega(g)$$

Caution: There is NO caution here in the definition of little-ω, since if the limit will always be ∞ and f will asymptotically dominate g.

1.2 Search Algorithms

1.2.1 Binary Search

Algorithm

Given an array A of n elements with values or records $A_0, A_1, A_2, \dots, A_{n-1}$ sorted such that $A_0 \leq A_1 \leq A_2 \leq \cdots \leq A_{n-1}$, and target value T, the following subroutine uses binary search to find the index of T in A.

- Set L to 0 and R to $n-1$.
- If $L > R$, the search terminates as unsuccessful.
- Set m (the position of the middle element) to the 'floor' of $\frac{L+R}{2}$, which is the greatest integer less than or equal to $\frac{L+R}{2}$.
- If $A_m < T$, set L to $m+1$ and go to step 2.
- If $A_m > T$, set R to $m-1$ and go to step 2.
- Now $A_m = T$, the search is done; return m.

Code

```python
# Array A, search value T,
# A is sorted in ascending order
def binary_search(A, T):
    L = 0
    R = len(A)-1

    while L <= R:
        m = (L+R) // 2
        if A[m] < T:
            L = m+1
        elif A[m] > T:
            R = m-1
        else:
            return m
    return -1
```

Time & Space Complexity

Best Case Time Complexity

$\mathcal{O}(1)$. The element to be search is in the middle of the list.

Average Case Time Complexity

$\mathcal{O}(\log N)$. Total number of comparisons =

$$\begin{aligned}&1 \times (\text{Elements requiring 1 comparison}) + 2 \times (\text{Elements requiring 2 comparison}s)\\ &+ ... + \log N \times (\text{Elements requiring } \log N \text{ comparison}s)\\ =&1 \times 1 + 2 \times 2 + 3 \times 4 + ... + \log N \times 2^{\log N - 1}\\ =&N \times (\log N - 1) + 1\end{aligned}$$

Total number of cases $= N + 1$.

Worst Case Time Complexity

$\mathcal{O}(N^2)$. The element is to search is in the first index or last index.

Space Complexity

$\mathcal{O}(1)$ for iterative, $\mathcal{O}(\log N)$ for recursive.

1.3 Sorting Algorithm

1.3.1 Bubble Sort

Algorithm

Given an array A of n elements following is the algorithm.

- Compare $A[0]$ and $A[1]$. If $A[0]$ is bigger than $A[1]$, swap the elements.
- Move to the next element, $A[1]$ (which might now contain the result of a swap from the previous step), and compare it with $A[2]$. If $A[1]$ is bigger than $A[2]$, swap the elements. Do this for every pair of elements until the end of the list.
- Do steps 1 and 2 n times.

Code

```python
def bubble_sort(A):
# We go through the list as many times as there are elements
for i in range(len(A)):
    # We want the last pair of adjacent elements to be (n-2, n-1)
    for j in range(len(A) - 1):
        if A[j] > A[j+1]:
            # Swap
            A[j], A[j+1] = A[j+1], A[j]
return A
```

Time & Space Complexity

Best Case Time Complexity

$\mathcal{O}(n)$. This case occurs when the given array is already sorted.

Average Case Time Complexity

$\mathcal{O}(N^2)$

- If an element is in index i_1 but it should be in index i_2, then it will take a minimum of $i_2 - i_1$ swaps to bring the element to the correct position.
- An element a_{j_k} will be at a distance of j_k from its position in sorted array. Maximum value of j_k will be $N-1$ for the edge elements and it will be $N/2$ for the elements at the middle.
- The sum of maximum difference in position across all elements will be:

$$
\begin{aligned}
&(N-1)+(N-3)+(N-5)+...+0+...+(N-3)+(N-1)\\
=&N \times N - 2 \times (1+3+5+...+N/2)\\
=&N^2 - 2 \times N^2/4\\
=&N^2/2
\end{aligned}
$$

Worst Case Time Complexity

$\mathcal{O}(\log N)$. This is the case when the array is reversely sorted.

Space Complexity

$\mathcal{O}(1)$ for iterative, $\mathcal{O}(\log N)$ for recursive.

1.4 Trees

1.4.1 Binary Tree

Definition 6: Binary Tree and Binary Search Tree

Binary Tree is a specialized form of tree with two child (left child and right Child). It is simply representation of data in Tree structure.

Definition 7: Binary Search Tree

Binary Search Tree (BST) is a special type of Binary Tree that follows following condition:

- left child node is smaller than its parent Node.
- right child node is greater than its parent Node.

A Binary Tree which is not a BST

```
        5
      /   \
     /     \
    9       2
  /  \     / \
15   17  19   21
```

A Binary Search Tree which is also a Binary Tree

```
      50
    /    \
   /      \
  25       75
 /  \     /  \
20   30  70   80
```

Chapter 2

Advanced Algorithms

2.1 Kosaraju-Sharir's Algorithm

Definition 8: Strongly Connected Component

A directed graph is strongly connected if there is a path between all pairs of vertices. A strongly connected component (SCC) of a directed graph is a maximal strongly connected subgraph

In computer science, Kosaraju-Sharir's algorithm (also known as Kosaraju's algorithm) is a linear time algorithm to find the strongly connected components of a directed graph. Aho, Hopcroft and Ullman credit it to S. Rao Kosaraju and Micha Sharir. Kosaraju suggested it in 1978 but did not publish it, while Sharir independently discovered it and published it in 1981. It makes use of the fact that the transpose graph (the same graph with the direction of every edge reversed) has exactly the same strongly connected components as the original graph [2].

Proof of correctness

2.2 Aho–Corasick Algorithm

In computer science, the Aho–Corasick algorithm is a string-searching algorithm invented by Alfred V. Aho and Margaret J. Corasick in 1975 [3].

Proof of correctness

Bibliography

[1] Wikipedia contributors. *Big O notation — Wikipedia, The Free Encyclopedia.* [Online; accessed 24-May-2022]. 2022. URL: https://en.wikipedia.org/w/index.php?title=Big_O_notation&oldid=1089214730 (page - 1).

[2] Wikipedia contributors. *Kosaraju's algorithm — Wikipedia, The Free Encyclopedia.* [Online; accessed 24-May-2022]. 2022. URL: https://en.wikipedia.org/w/index.php?title=Kosaraju%27s_algorithm&oldid=1068143838 (page - 9).

[3] Wikipedia contributors. *Aho–Corasick algorithm — Wikipedia, The Free Encyclopedia.* [Online; accessed 24-May-2022]. 2022. URL: https://en.wikipedia.org/w/index.php?title=Aho%E2%80%93Corasick_algorithm&oldid=1078139607 (page - 9).