자료구조란?

자료구조 공부에 필요한 기초지식을 쌓아봅시다!

데이터를 저장하고 관리하는 방법론.

데이터를 저장하는 컨테이너는 많지만 우리는 우리가 사용할 데이터에 따른, 그리고 필요한 상황에 맞는 올바른 자료구조를 사용해야합니다.

마치 일상생활에서 소고기를 보관하는데 반드시 냉장고를 사용해야 하는것 처럼요.

자료구조에는 우리가 사용하는 모든 데이터 타입을 저장합니다.

그리고 마치 우리가 어지러운 방을 가지런히 정리하는것 처럼 정리하기도 합니다.

자료구조의 기본 작동방식

접근, 탐색, 삽입, 삭제 연산은 자료구조에서 가장 일반적으로 사용되는 기본 작업들입니다.

1. Access (접근):
    - 접근은 자료구조에서 특정 요소에 직접 접근하여 값을 읽거나 수정하는 작업을 말합니다.
    - 예를 들어, 배열에서 특정 인덱스의 요소에 접근하는 것은 해당 인덱스에 저장된 값을 읽거나 수정하는 것을 의미합니다. (배열의 3 인덱스 값에 접근할 경우)
2. Search (탐색):
    - 탐색은 자료구조에서 특정 값을 찾는 작업을 말합니다.
    - 예를 들어, 배열에서 특정 값을 찾는 것이 탐색 작업입니다. (배열에서 5의 값을 가지는 원소를 찾고자 하는 경우)
3. Insertion (삽입):
    - 삽입은 자료구조에 새로운 요소를 추가하는 작업을 말합니다.
    - 예를 들어, 배열에 새로운 요소를 추가하는 것이 삽입 작업입니다.
4. Deletion (삭제):
    - 삭제는 자료구조에서 특정 요소를 제거하는 작업을 말합니다.

- 예를 들어, 배열에서 요소를 삭제하는 것이 삭제 작업입니다.

이러한 작업들을 효율적으로 수행하기 위해 적절한 자료구조를 선택하고 알고리즘을 구현해야 합니다. 각 자료구조는 특정 작업에 대해 뛰어난 성능을 보이는 경우가 있으므로, 문제의 요구사항에 맞는 자료구조를 선택하는 것이 중요합니다.

빅오(Big O) 표기법

https://www.bigocheatsheet.com/

알고리즘은 특정 문제를 해결하기 위한 명확하게 정의된 지침의 집합입니다. 이러한 문제를 해결하기 위해 다양한 방법을 사용할 수 있습니다.

즉, 동일한 해결책에 도달하기 위해 사용하는 방법이 모두 다를 수 있기 때문에 성능과 효율성(알고리즘이 실행되는 데 걸리는 시간 및 소비되는 총 메모리 양)을 평가하는 방법이 있어야 합니다.

여기서 Big O 표기법이 등장합니다. Big O 표기법은 알고리즘의 효율성을 결정하는 지표입니다. 입력 크기의 증가에 따라 코드가 실행되는 데 걸리는 시간과 데이터의 공간이 얼마나 필요하게 되는지를 추정하는 데 사용됩니다.

Big O란 무엇일까?

Big O 표기법은 알고리즘이 **평균적인 경우와 최악의 경우** 얼마나 복잡해질지 나타냅니다.

Big O는 알고리즘의 실행에 필요한 시간을 정의합니다. 입력 크기가 증가함에 따라(얼마나 많은 데이터를 처리하는가) 알고리즘의 성능이 어떻게 변경되는지를 나타냅니다.

Big O 표기법은 시간 및 공간 복잡성을 사용하여 알고리즘의 효율성과 성능을 측정합니다.

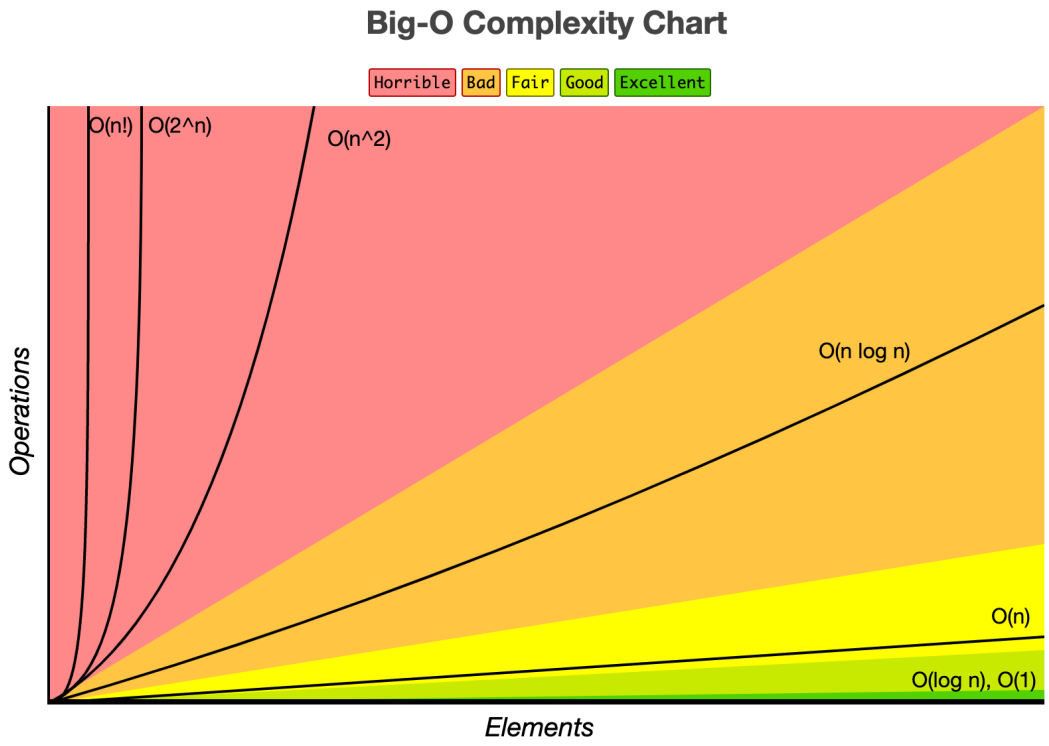
시간 및 공간 복잡성은 무엇인가?

프로그램의 성능과 효율성에 영향을 주는 주요한 요인은 하드웨어(CPU, 주 기억장치, 보조 기억장치) 및 운영 체제의 성능입니다.

그러나 알고리즘의 성능을 분석할 때는 이러한 요소를 고려하지 않습니다. 사용자마다 다른 하드웨어 스팩과 운영체제를 가지기 때문에 알고리즘의 성능을 특정할 수 없기 때문입니다. 대신, 입력 크기에 따라 표현되는 시간 및 공간 복잡성이 중요합니다.

알고리즘의 시간 복잡성은 입력 크기에 따라 알고리즘이 실행되는 데 걸리는 시간을 의미합니다. 마찬가지로, 알고리즘의 공간 복잡성은 알고리즘이 실행되는 데 필요한 메모리 양을 말합니다.

오늘날의 컴퓨터 하드웨어는 예전에 비해 성능이 아주 향상되었기 때문에 한정된 하드웨어를 사용하는 특수한 경우가 아니라면 개발할때는 주로 시간 복잡성에 초점을 맞추는 편입니다.



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)

시간 및 공간 복잡성의 주요 유형

Big O 표기법에서는 시간 및 공간 복잡성의 여섯 가지 주요 유형이 있습니다:

1. 상수 시간(Constant): O(1)

- 상수 시간 복잡도는 입력 크기에 관계없이 실행 시간이 일정한 알고리즘을 나타냅니다.
- 즉, 입력 크기에 관계없이 실행 시간이 일정하므로 가장 효율적인 알고리즘입니다.
- 예를 들어, 배열에서 첫 번째 요소를 반환하는 경우와 같이 단일 작업을 수행하는 알고리즘이 이에 해당합니다.

2. 선형 시간(Linear time): O(n)
    - 선형 시간 복잡도는 입력 크기에 비례하여 실행 시간이 증가하는 알고리즘을 나타냅니다.
    - 예를 들어, 배열의 모든 요소를 한 번씩 방문하면서 처리하는 경우, 배열의 요소가 늘어날 수록 처리해야 하는 시간도 비례해서 늘어나는것이 선형 시간 알고리즘입니다.
3. 로그 시간(Logarithmic time): O(n log n)
    - 로그 시간 복잡도는 입력 크기에 비례하여 실행 시간이 증가하는것으로 선형시간과 비슷하지만, 선형 시간보다는 느립니다.
    - 주로 분할 정복(divide and conquer) 알고리즘에서 나타닙니다.
4. 이차 시간(Quadratic time): O(n^2)
    - 이차 시간 복잡도는 입력 크기의 제곱에 비례하여 실행 시간이 증가하는 알고리즘을 나타냅니다.
    - 즉, 입력 크기가 증가할수록 실행 시간이 기하급수적으로 증가합니다.
    - 예를 들어, 이중 반복문을 사용하여 배열의 모든 요소 쌍을 확인하는 경우 이차 시간 알고리즘입니다.
5. 지수 시간(Exponential time): O(2^n)
6. 팩토리얼 시간(Factorial time): O(n!)

가장 기본이 되는 자료구조

1. Queue

    FIFO(파이포) : first in first out

    데이터를 저장할 때 먼저 저장된 데이터를 먼저 처리하는 방식

놀이공원에서 줄을 설 때 먼저 온 사람이 먼저 들어가는 방식으로 생각하면 편합니다.

큐에 새롭게 데이터를 추가하는것을 인큐(enqueue), 데이터가 처리되어 제거되는것을 디큐(dequeue) 라고 합니다.

2. Stack

   LIFO(라이포) : last in first out

   큐와 반대되는 자료구조입니다. 먼저 저장된 데이터를 가장 나중에 처리하고 나중에 들어온 데이터를 먼저 처리합니다. 스택에 쌓인 데이터를 표현할 때 프레임이라고 부릅니다.

   스택의 처리 순서는 총알을 장전하는 원리와 같다고 보시면 됩니다. 총알을 장전할 때 가장 먼저 장전한 총알이 가장 나중에 발사되고, 가장 나중에 장전한 총알이 먼저 발사됩니다.

   스택에 데이터를 추가하는것을 푸쉬, 제거하는 것을 팝이라고 합니다.

3. Heap

   힙은 이진 트리의 일종으로, 최댓값이나 최솟값을 빠르게 찾을 수 있도록 설계된 자료구조입니다. 이진 트리는 각 노드가 최대 두 개의 자식 노드를 가지는 트리 자료 구조입니다. 가장 대표적인 힙으로는 노드가 자식 노드보다 같거나 큰 값을 가지는, 즉 루트 노드가 트리에서 가장 큰값을 가지는 Max Heap, 노드가 자식 노드보다 같거나 작은 값을 가지는, 즉 루트 노드가 가장 작은 값을 가지는 Min Heap 이 있습니다.

4. Array

   데이터를 순서에 따라 저장하는 자료구조. 다른말로 리스트라고 부르기도 합니다. 가장 폭넓게 기본적으로 사용되는 자료구조입니다.

5. hashTable

   키와 값의 쌍을 저장하는 자료 구조입니다. 해시 테이블은 해시 함수를 사용하여 키를 해시 값으로 변환한 다음, 해시값을 사용하여 키와 값의 쌍을 저장합니다.