1주차-알고리즘과 자료구조의 개요

알고리즘(Algorithm)의 개요

- 알고리즘의 실행 시간 및 기타 자원의 사용량을 분석
    - 기타 자원 : 메모리, 저장장치, 통신 등
- 실행 시간의 분석에 집중해야 되는 이유

점근적 분석

- 알고리즘의 실행 시간에 따른 분류
    - 첫번째 : 입력값의 크기에 따른 알고리즘의 실행 시간
    - 두번째 : 입력값의 크기에 따른 알고리즘의 실행 시간의 성장률

- 점근적 표기법(asymptotic notation)
    - 데이터의 개수 n → ∞ 일 때 수행시간이 증가하는 growth rate로 시간복잡도를 표현하는 기법
    - 점근적 표기법 : 상수 계수와 중요하지 않은 항목을 제거한 것
        - 중요하지 않은 항과 상수 계수를 **제거**하면 이해를 방해하는 불필요한 부분이 없어져서 알고리즘의 실행 시간에서 중요한 부분인 성장률에 집중할 수 있음

- 점근적 표기법의 종류
    - Big-Θ(빅 세타) 표기법 : 실행 시간에 대해 점근적으로 근접한 한계값이 존재하는 표기법
    - **Big-O(빅 오) 표기법** : 점근적 상한선만 제공하고 점근적으로 근접한 한계를 주지 않는 표기법
    - Big-Ω(빅 오메가) 표기법 : 최소한의 표현을 해야할 때, 상한선 없이 점근적 하한선만 존재하는 표기법

- 유일한 분석법도 아니고 가장 좋은 분석법도 아님
    - 다만 **상대적으로** 가장 간단하며 알고리즘의 실행환경에 비의존적임
    - 그래서 가장 광범위하게 사용됨

Big-O 표기법의 상세

- Big-O는 알고리즘의 효율성을 나타내는 대중적인 지표
    - 개선한 알고리즘이 빨라졌는지, 메모리를 많이 사용하지 않는지 등의 알고리즘의 성능을 판단
- 시간에 대한 시간복잡도와 공간에 대한 공간복잡도가 존재

시간복잡도(Time Complexity)

- 시간 복잡도 : 알고리즘의 수행 시간이 얼마인지를 나타냄
  - 수행되는 연산의 수를 가지고 계산하며 알고리즘에서 중요하지 않는 값들은 최대한 무시
  - 알고리즘 실행 시간은 실행 환경에 따라 달라지므로 실행 시간이 아닌 **연산의 실행 횟수를 카운트**

- 수행되는 연산이란 대개 산술, 비교, 대입 등을 말함
  - 연산이 많이 존재하더라도 하나로 취급하여 Big-O값을 구할 수도 있음

- 입력 값(N)에 따라서 실제 소요되는 시간이 Big-O에 의한 결과와 다를 수도 있음
  - 알고리즘의 효율성은 데이터의 입력 값이 얼마나 크냐에 영향을 받으므로 사소한 부분은 무시 가능
  - 최악의 경우 시간복잡도 (worst case)
  - 평균 시간복잡도 (average case)

- 무시하는 항목

상수항 무시	영향력 없는 항 무시
O(2N) → O(N)	**$O(N^2 + N) \rightarrow O(N^2)$**
$O(N^2 + 2) \rightarrow O(N^2)$	**$O(N^2)$** 이 가장 지배적이므로 영향력이 없는 다른 항들은 무시

- Big-O에서 자주 사용되는 복잡도

$O(1) < O(\log n) < O(n) < O(n \log n) < O(n^2) < O(2^n) < O(n!) < O(n^n)$

- 항상 N 변수 하나만 사용되는 것은 아님

시간복잡도의 예제

> 입력 값 N에 대해 N^2을 구하는 프로그램을 작성하고 각각의 시간 복잡도를 비교해보자.

- 첫번째

```
int sum1(int N){
    return N*N;
}
```

```
// 1. 곱셈 연산 1개이므로 Big-O 표기법 O(1) 이 됨
```

- 두번째

```
int sum2(int N){
    sum = 0;

    for (int i=1; i<=N; i++) {
        sum = sum + N;
    }

    return sum;
}

// 1. 덧셈 연산 1개와 대입 연산 1개가 N만큼 실행
// 2. 2N의 시간이 걸리지만 Big-O 표기법으로는 상수항을 무시하여 O(N) 이 됨
// 3. sum = 0; 또한 대입 연산이지만 상수 값 1 이므로 무시
```

- 세번째

```
int sum3(int N){
    sum = 0;

    for (int i=1; i<=N; i++) {
        for (int j=1; j<=N; j++) {
            sum = sum + 1;
        }
    }

    return sum;
}

// 1. 덧셈 연산, 대입 연산 이 각각 NxN번 실행
// 2. 총 2N²의 시간이 걸리지만 Big-O 표기법으로는 상수항을 무시하여 O(N²) 이 됨
// 3. sum = 0; 또한 대입 연산이지만 상수 값 1 이므로 무시
```

공간복잡도(Space Complexity)

- 공간복잡도 : 알고리즘이 공간을 얼마나 필요로 하는지를 나타냄
  - 시간복잡도 보다 덜 중요하게 취급되지만 신경을 써야 함
  - 크기가 N인 배열을 작성하면 공간 복잡도는 O(N), NxN 배열을 작성하면 $O(N^2)$ 이 됨

- 함수의 재귀적인 호출의 경우 스택 공간도 고려

공간복잡도 예제

> 1부터 N까지의 합을 구하는 프로그램을 재귀 알고리즘으로 작성해보자.

- 첫번째

```
int sum(int N){
    sum = 0;
    if(N<1)
        return 0;
    return N + sum(N-1);
}

// 1. N = 5 일 경우 스택에 쌓이는 최대 메모리는 sum(1) + sum(2) + … + sum(5)
// 2. 따라서 공간복잡도로 나타내면 O(N) 이 됨
// +. N번 호출했다고 해서 공간 복잡도가 항상 O(N)이 되는 것은 아님, 아래 예제 확인
```

- 두번째

```
int mainSum(int N){
    int result = 0;

    for(int i=0; i<N; i++)
        result += sum(i, i+1);

    return result;
}

int sum(int a, int b){
    return a + b;
}

// 1. mainSum() 함수에서 sum() 를 N번 호출
// 2. 하지만 스택에 쌓이는 최대 공간은 mainSum() + sum(), 2 임
// 3. 따라서 공간복잡도로 나타내면 O(1) 이 됨
```

(참조) 점근적 분석의 예: 선형 시간복잡도

- 선형 시간복잡도를 가진다고 말하고 O(n)이라고 표기한다.

```
int sum(int[] A) {
    int n = 0;
    int result = 0;
    for (int i = 0; i < A.length; i++){
        result += A[i];
        n+=1;
    }
    return result;
}
```

가장 자주 실행되는 문장 중 하나이며 실행 횟수는 항상 n번이다.
가장 자주 실행되는 문장이 n번이라면 모든 문장의 실행 횟수의 합은
n에 선형적으로 비례하며, 모든 연산들의 실행횟수의 합도 역시 n에 선형적으로 비례한다.

```
int search(int[] A, int target) {
    for (int i = 0; i < A.length; i++) {
        if (A[i] == target)
            return i;
    }
    return -1;
}
```

가장 자주 실행되는 문장 중 하나이며, 실행 횟수는 최악의 경우 n번이다.
최악의 경우 시간복잡도는 O(n)이다.

```
boolean isDistinct(int[] A) {
    for (int i = 0; i < A.length; i++) {
        for (int j = 0; i + 1 < A.length; j++) {
            if (A[i] == A[j])
                return false;
        }
    }
    return true;
}
```

최악의 경우 배열에 저장된 모든 원소 쌍들을 비교하므로 비교 연산의 횟수는 n(n-1)/2
이다.

따라서 시간복잡도는 O(n2)이다

```
int binarySearch(int n, int[] data, int target) {
    int begin = 0;
    int end = n - 1;
    while (begin <= end) {
        int middle = (begin + end) / 2;
        if (data[middle] == target)
            return middle;
        else if (data[middle] < target)
            begin = middle + 1;
        else
            end = middle - 1;
    }
    return - 1;
}
```

위 알고리즘의 시간복잡도는?

자료구조(DataStructure)의 개요

> 현실을 프로그래밍적으로 표현하는 것 / 큰 데이터를 효율적으로 관리하는 것

- 책이 한권이면 아무렇게 던져놔도 찾는데 문제가 없으나 100권, 200권씩 그 수가 많아지면 정리를 하지 않는 한 원하는 책을 찾기 어렵다. 정리를 통해 공간을 절약하고 시간을 절약하는 효과를 누린다.

> 정리정돈의 진화 : 종이 → 책 → 책장 → 도서관 → WWW(인터넷 또는 네트워크)

- 컴퓨터에서 중요한 것 : MEMORY, CPU, STORAGE
  - MEMORY : 가격이 비싸고 용량이 적으며 전원을 끄면 데이터도 소실되지만 STORAGE 보다 월등히 빠르게 데이터를 가져올 수 있다.
  - STORAGE : HDD/SDD 로 저장장치이다. 가격이 저렴하고 용량이 크며 전원이 꺼져도 데이터가 저장된다.
  - CPU : 처리 속도 차이가 너무 심해 스토리지에 곧바로 엑세스 하지 않는다. 컴퓨터가 동작하는 데 필요한 모든 계산을 처리한다.

- 처리 순서
  - 메모리가 스토리지에서 데이터를 가져온다.
  - CPU가 메모리에 담긴 데이터를 읽는다.

- 자료구조에 있어 가장 중요한 것은 메모리이다.
  - 자료구조의 미션 : 메모리를 어떻게 효율적으로 사용하느냐?
  - 메모리는 빌딩과 같다. 여러 내부 저장소에 데이터를 넣고, 해당 주소를 통해 데이터를 가져오는 속도가 모든 주소들이 동일하다.

자료구조의 종류

- 자료 간 연결 형태 / 모양에 따른 구분
  - 선형 자료구조 (Linear, 전후 1:1 연결 형태)

```
- 한 원소 뒤에 하나의 원소 만이 존재
    > 자료들이 직선 형태로 나열되어 있는 구조로 원소들 간 순서 고려
    > 전후/인접/선후 원소들 간에 1:1 관계로 나열

- 선형 자료구조의 예
    > 기본 선형 자료구조 : 리스트, 연결 리스트, 배열, 레코드 등
        >> 자료의 삽입 및 삭제가 어느 위치에서도 이루어짐
    > 제한 선형 자료구조 : 스택, 큐, 데크(스택과 큐가 혼합된 형태) 등
        >> 자료의 삽입 및 삭제가 정해진 위치에서만 이루어짐
```

  - 비선형 자료구조 (Nonlinear, 전후 多:多 연결 형태)

```
- 한 원소 뒤에 여러개의 원소들이 존재할 수 있음
    > 인접(전후) 원소들 간에 多:多 관계로 배치됨
    > 계층적 구조(Hierarchical Structure)를 나타내기에 적절
    > 가계도상에서 조상-자손 간의 관계, 직장 상사-부하 간의 관계, 컴퓨터 폴
더 구조 등


- 비선형 자료구조의 예
    > 트리, 그래프 등
```

- 자료 간 연속, 연결 구조에 따른 구분
  - 배열에 기반한 연속 방식 구조 (Continuation) : 리스트 등
  - 포인터 기반의 연결 방식 구조 (Link) : 연결 리스트 등

자료구조의 관점

- 데이터 및 연산을 다루는 관점

    - 자료형(Data Type) : 데이터 중심으로만 고려
        - 자료(변수)가 갖는 값의 종류를 표현
        - 연산은 그 자료형에 맞도록, 별도/부가적/부차적으로 수행되는 관점
    - 추상 자료형(Abstract Data Type) : 데이터와 연산을 함께 고려
        - 자료 및 연산을 모두 하나로 묶어 하나의 단위로 표현
        - 자료 저장 및 처리 보다는 문제 해결 지향적인 자료형

```
- 추상적 자료구조(Abstract Data Structure) : 구현 방법을 명시하지 않
음
    > 리스트
        >> 연결 리스트 (Linked List)
    > 스택 (Stack)
    > 큐 (Queue)
    > 트리 (Tree)
        >> 트라이 (Trie)
    > 그래프 (Graph)
    > 딕셔너리 자료구조 (Dictionaries)
        >> 연관 배열 (Associative array.) Map이라고 칭하기도 한다.
        >> 연관 리스트
        >> 해시 테이블
```

- 데이터의 표현/구현/설계 관점
    - 사전 정의형(프로그래밍 언어에서 기본 내장 제공)
    - 사용자 정의형(프로그래머가 응용에 따라 직접 구현)

(참조) 자료구조의 사용처

- ★(필수) List : 모든 곳
- Set : 집합, 교집합 / 합집합 / 차집합 등
- ★(필수) Tree : 조직도, 디렉토리 등
- Graph : 지도 어플리케이션 등, 최단거리 이동

1주차 - 문제풀기(배열)

- 1번 - 코딩인터뷰

```
- "중복이 없는가"
  * > 문자열이 주어졌을 때, 이 문자열에 같은 문자가 중복되어 등장하는지 확인하는
```

```
알고리즘을 작성하라.
  * > 자료구조를 추가로 사용하지 않고 풀 수 있는 알고리즘 또한 고민하라.
```

```
boolean isUniqueChars(String str){

    // 코드 작성

    return true;
}
```

- 2번 - 코딩인터뷰

```
- "URI변환"
  * > 문자열에 들어 있는 모든 공백을 '%20'으로 바꿔 주는 메서드를 작성하라.
  * > 최종적으로 모든 문자를 다 담을 수 있을 만큼 충분한 공간이 이미 확보되어 있다.
  * > 문자열의 최종 길이가 함께 주어진다고 가정해도 된다.
  * > (자바로 구현한다면 배열 안에서 작업할 수 있도록 문자 배열(character array)를 이용하길 바란다)
```

```
char[] replaceSpaces(String str, int trueLength){
    char[] array = {};

    // 코드 작성

    return array;
}
```

- 3번 - 프로그래머스

```
배열 array의 i번째 숫자부터 j번째 숫자까지 자르고 정렬했을 때, k번째에 있는 수를 구하려 합니다.

예를 들어 array가 [1, 5, 2, 6, 3, 7, 4], i = 2, j = 5, k = 3이라면
array의 2번째부터 5번째까지 자르면 [5, 2, 6, 3]입니다.
1에서 나온 배열을 정렬하면 [2, 3, 5, 6]입니다.
2에서 나온 배열의 3번째 숫자는 5입니다.

배열 array, [i, j, k]를 원소로 가진 2차원 배열 commands가 매개변수로 주어질 때,
commands의 모든 원소에 대해
앞서 설명한 연산을 적용했을 때 나온 결과를 배열에 담아 return 하도록 solution 함수를 작성해주세요.

[제한사항]
array의 길이는 1 이상 100 이하입니다.
```

```
array의 각 원소는 1 이상 100 이하입니다.
commands의 길이는 1 이상 50 이하입니다.
commands의 각 원소는 길이가 3입니다.
```

```
public int[] solution(int[] array, int[][] commands) {
    int[] answer = {};

    // 코드 작성

    return answer;
}
```

- 4번 - 백준알고리즘

```
동혁이는 나무 조각을 5개 가지고 있다. 나무 조각에는 1부터 5까지 숫자 중 하나가 쓰여져 있다.
또, 모든 숫자는 다섯 조각 중 하나에만 쓰여 있다.

동혁이는 나무 조각을 다음과 같은 과정을 거쳐서 1, 2, 3, 4, 5 순서로 만들려고 한다.

첫 번째 조각의 수가 두 번째 수보다 크다면, 둘의 위치를 서로 바꾼다.
두 번째 조각의 수가 세 번째 수보다 크다면, 둘의 위치를 서로 바꾼다.
세 번째 조각의 수가 네 번째 수보다 크다면, 둘의 위치를 서로 바꾼다.
네 번째 조각의 수가 다섯 번째 수보다 크다면, 둘의 위치를 서로 바꾼다.

만약 순서가 1, 2, 3, 4, 5 순서가 아니라면 1 단계로 다시 간다.

처음 조각의 순서가 주어졌을 때, 위치를 바꿀 때 마다 조각의 순서를 출력하는 프로그램을 작성하시오.
```

뒤로가기