D.cOm

Algorithm- 스터디

#Beginner 편

2강. 수학 시러

2019-11 Ver.

오늘의 학습 목표!

1. 알고리즘의 **복잡도**를 알아봅시다!

2. 다양한 **수학적 원리**들을 알아보고 구현해봅시다.

강의 자료 개요

별표 친 페이지는 반드시 읽어주세요!

이론

이론 내용 입니다. 가벼운 마음으로 읽어주세요!

실습

百聞이 不如一打!
직접 따라해보세요!

읽을거리

읽어두면 배운 것을 심층적으로 이해하는데 있어 도움이 되는 내용입니다! 여유가 있으면 읽어주세요!

알고리즘

Algorithm

어떤 문제를 해결하는 방법

복잡도 (Complexity)

복잡도

내 알고리즘이 얼마나 효율적인지
어떻게 표현할 수 있을까요?

복잡도

알고리즘의 평가 방법에는 **시간 복잡도**와 **공간 복잡도** 두가지가 있습니다!

시간 복잡도 → 알고리즘의 연산의 횟수

공간 복잡도 → 알고리즘의 메모리 사용량

시간 복잡도

시간복잡도는 알고리즘의 연산 횟수를 의미합니다.
그렇다면 알고리즘의 연산의 횟수는 어떻게 구할 수 있을까요?
3가지 경우로 나눌 수 있습니다.

1. 최선의 경우 Best Case
2. 평균적인 경우 Average Case
3. 최악의 경우 Worst Case

평균적인 경우가 가장 이상적이게 보입니다.
하지만 이는 알고리즘이 복잡할 때 구하기 매우 어려워집니다.
<u>따라서 "최악의 경우"로 알고리즘의 연산 횟수를 파악합니다.</u>

복잡도

시간 복잡도

다음 1부터 N까지의 합을 구하는 **3가지 알고리즘**의
연산의 횟수, 즉 **시간복잡도**를 구해봅시다!

```
int sum = 0;
  for (int i=1; i<=N; i++) {
    sum += i;
}
```

```
int sum = 0;
  for (int i=1; i<=N; i++) {
    for (int j=1; j<=N; j++) {
      if (i == j) {
        sum += j;
      }
    }
}
```

```
int sum = 0;
sum = N*(N+1)/2;
```

복잡도

시간 복잡도

반복문이 곧 연산횟수를 결정합니다!
N이 입력 데이터의 양이라면
연산 횟수는 다음과 같습니다!

```
int sum = 0;
  for (int i=1; i<=N; i++) {
    sum += i;
}
```

O(N)

```
int sum = 0;
  for (int i=1; i<=N; i++) {
    for (int j=1; j<=N; j++) {
      if (i == j) {
        sum += j;
      }
    }
}
```

$O(N^2)$

```
int sum = 0;
sum = N*(N+1)/2;
```

O(1)

읽을 거리

시간 복잡도 표기할 때 앞에 O는 무엇인가요?

https://zelord.tistory.com/12

빅오 표기법으로, 영향력이 가장 큰 항만을 기술합니다!
Ex) $2n^2 + n + 5$ => O(n^2)

읽을 거리

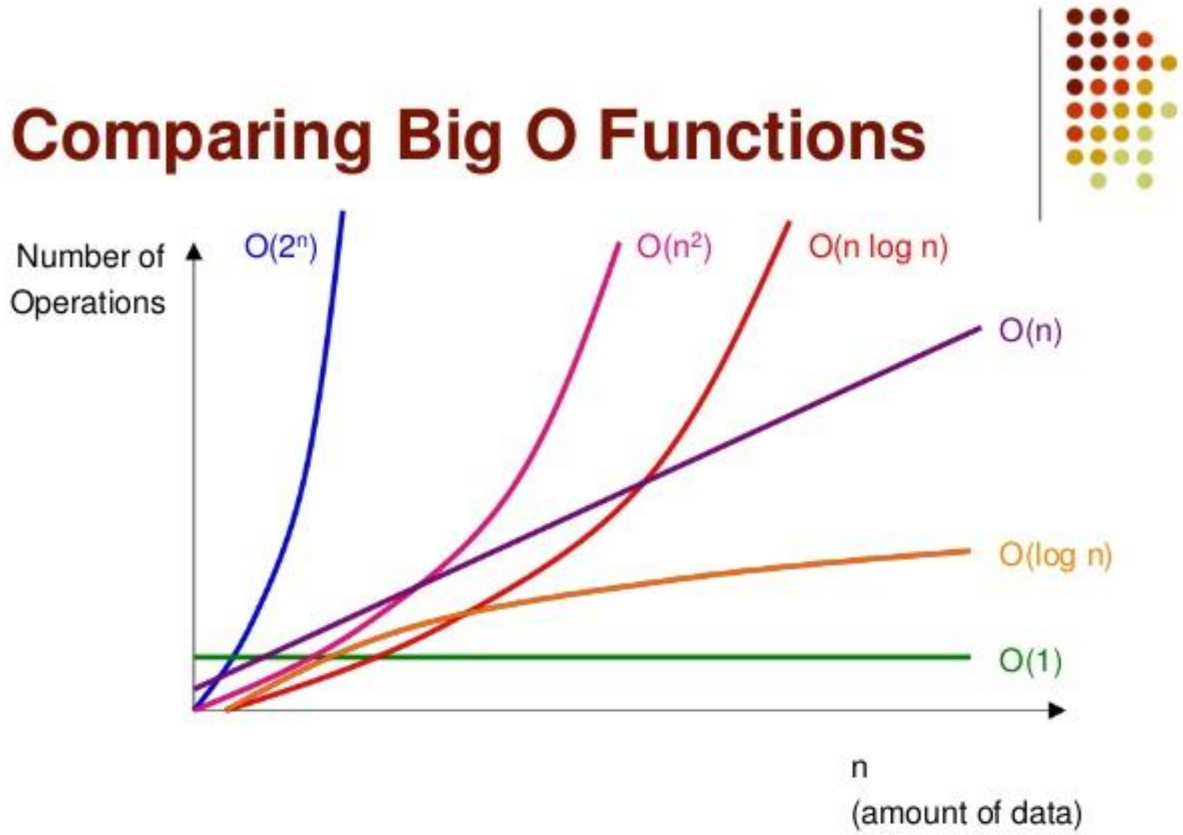
알고리즘의 수행 시간이 아닌 연산의 횟수가
시간 복잡도인 이유는 무엇인가요?

➡ 알고리즘의 수행시간을 기준으로 삼으면 다음과 같은 문제점이 있습니다.

1. 측정을 위해서는 완성된 프로그램이 필요하다!
2. 컴퓨터마다 결과가 달라진다(EX.슈퍼 컴퓨터 VS 노트북)

복잡도

시간 복잡도



빅오 표기법의 종류를 나타낸 그래프입니다.

성능 순서는 다음과 같습니다.
$O(1) < O(logn) < O(n) < O(nlogn) < O(n^2) < O(n^3) < O(2^n)$

특히 **O(nlogn) 이상부터는 데이터의 양에 따라** 연산 횟수가 기하급수적으로 증가하기 때문에 사용에 주의하여야 합니다!

복잡도

시간 복잡도

"대략 1억 번의 연산 횟수는 1초의 실행 시간이 걸립니다."

대부분의 알고리즘 문제는 실행 시간이 주어집니다.
따라서, 알고리즘을 구현하기전 대략적인 시간 복잡도를 생각해 둔다면,
<u>이 문제에 적합한 알고리즘인지 판단할 수 있습니다!</u>

복잡도

공간 복잡도

공간복잡도는
프로그램을 실행 시켜 완료하는데 필요한 메모리의 양입니다.

하지만 보통 알고리즘 문제에서 메모리는 넉넉하기 때문에
크게 걱정 안 해도 됩니다!

복잡도

공간 복잡도

주로 **배열의 크기**를 통해 공간 복잡도를 계산합니다.
배열의 크기 x 자료형의 크기 = 배열이 사용한 공간의 크기

Ex)
int arr[1000] -> 1000*4B = 4000B
Int arr[1000][1000] -> 1000*1000*4B = 4000000B = 4MB

복잡도

“좋은 알고리즘”이란 주어진 상황과 문제에 적합한 시간&공간 복잡도를 가진 알고리즘을 의미합니다.

시간&공간 복잡도가 낮다고 무조건 좋은 알고리즘은 아닙니다!
왜냐하면 대체로 복잡도가 낮은 알고리즘일수록 구현 난이도가 올라갑니다!
즉, 간단한 문제에 굳이 복잡한 알고리즘을 적용할 필요는 없습니다!
“닭 잡는데 어찌 소 잡는 칼을 쓰는고”

수학 (Mathematics)

들어가며..

이번 단원에서는 알고리즘에 쓰이는 기본적인 수학 내용을 알아봅니다!
딱딱하고 어려운 수학이 아니니 편한 마음으로 배워봅시다!

-> 최대공약수, 최소공배수

-> 소수

들어가며..

"이번 단원에서 수학을 다루는 이유는 무엇인가요?"

본격적으로 알고리즘을 구현하기 전,
수학적 명제를 코드로 옮기며 연습해보는 단계입니다.

이번 단원은 통해 자신이 생각한 알고리즘을
코드로 작성하는 "**구현력**"을 향상시키는 것이 목적입니다

최대공약수 최소공배수(GCD, LCM)

최대공약수

"사탕 75개, 초콜릿 102개, 풍선껌 153개를 수학 반 학생들에게 똑같이 나누어 주었더니
사탕이 3개, 초콜릿이 6개, 풍선 껌이 9개가 남았다. 가능한 수학 반 학생수를 모두 구하여라."

#놀랍게도 중1 수학입니다.

"가능한 많이 똑같이 나누려면"
어떻게 할까요?

최대공약수

"사탕 75개, 초콜릿 102개, 풍선껌 153개를 수학 반 학생들에게 똑같이 나누어 주었더니
사탕이 3개, 초콜릿이 6개, 풍선 껌이 9개가 남았다. 가능한 수학 반 학생수를 모두 구하여라."

#놀랍게도 중1 수학입니다.

"가능한 많이 똑같이 나누려면"
어떻게 할까요?

➡ 최대공약수(GCD)

최대공약수

최대공약수의 정의

“0이 아닌 두 정수나 다항식의 공통되는 약수 중에서 가장 큰 수를 말한다.”

최대공약수

최대공약수의 정의

“0이 아닌 두 정수나 다항식의 공통되는 약수 중에서 가장 큰 수를 말한다.”

최대공약수

최대공약수의 정의

“0이 아닌 두 정수나 다항식의 공통되는 약수 중에서 가장 큰 수를 말한다.”

어떻게 구현할 수 있을까요?

최대공약수

최대공약수의 정의

“0이 아닌 두 정수나 다항식의 공통되는 약수 중에서 가장 큰 수를 말한다.”

⇒ [1, min(a,b)] 범위에서 두 수
모두의 약수가 되는 값들의 최댓값을 구하자!

최대공약수

[redacted]

```
int g = 1;
int min = min(a, b);
for (int i = 2; i <= min; i++)
{
    if (a % i == 0 && b % i == 0)
    {
        g = i;
    }
}
```

코드를 다음과 같이 작성할 수 있습니다!
다음 코드의 시간 복잡도는 어떻게 될까요?

최대공약수

```
int g = 1;
int min = min(a, b);
for (int i = 2; i <= min; i++)
{
    if (a % i == 0 && b % i == 0)
    {
        g = i;
    }
}
```

최악의 경우
Min(a,b)까지 수를 순회하게 됩니다.
즉, O(min(a,b))만큼의 시간 복잡도를 가집니다.

최대공약수

```
int g = 1;
int min = min(a, b);
for (int i = 2; i <= min; i++)
{
    if (a % i == 0 && b % i == 0)
    {
        g = i;
    }
}
```

최악의 경우는?

최대공약수

```
int g = 1;
int min = min(a, b);
for (int i = 2; i <= min; i++)
{
    if (a % i == 0 && b % i == 0)
    {
        g = i;
    }
}
```

최악의 경우는?

→ **“두 수가 서로소일 때”**

최대공약수

더욱 효율적인
다른 방법은 없을까요?

최대공약수

“유클리드 호제법”

Euclidean Algorithm

#인류 최초의 알고리즘

최대공약수

유클리드 호제법

“2개의 자연수 a,b에 대해서

a를 b로 나눈 나머지를 r이라 하면 (단, a>b)

a와 b의 최대공약수는 b와 r의 최대공약수와 같다.”

최대공약수

좀 더 쉽게 풀면?

Gcd(a,b) = Gcd(b,r)
r을 구하는 과정을 반복하여 r이 0이 되었을 때

그 때의 b가 최대 공약수이다.

최대공약수

Ex) $78696 = 19332 \times 4 + 1368$

$19332 = 1368 \times 14 + 180$

$1368 = 180 \times 7 + 108$

$180 = 108 \times 1 + 72$

$108 = 72 \times 1 + 36$

$72 = 36 \times 2$

최대공약수

Gcd(a,b) = Gcd(b,r)

r을 구하는 과정을 반복하여 r이 0이 되었을 때

①　　　　　　　　　　②

그 때의 b가 최대 공약수이다.

③

최대공약수

Gcd(a,b) = Gcd(b,r)

r을 구하는 과정을 반복하여 r이 0이 되었을 때

① “반복” ② “종료조건”

그 때의 b가 최대 공약수이다.

③ “결과값”

최대공약수

다음 단서들을 바탕으로 알고리즘을 구현해봅시다!

최대공약수

#초기조건, 종료조건, 구하고자 하는 결과, 필요한 코드가 무엇이 있나요?

```
int gcd(int a, int b)
{
    if (b == 0)
    {
        return a;
    }
    else
    {
        return gcd(b, a % b);
    }
}
```

1. 재귀 사용

최대공약수

#초기조건, 종료조건, 구하고자 하는 결과, 필요한 코드가 무엇이 있나요?

```c
int gcd(int a, int b)
{
    while (b != 0)
    {
        int r = a % b;
        a = b;
        b = r;
    }
    return a;
}
```

2. 반복문 사용

최소공배수

그렇다면 **최소 공배수**는?
(LCM)

최소공배수

$$\mathrm{LCM}(A, B) = \frac{A \ * B}{GCD(A, B)}$$

최대공약수를 이용하여
간편하게 최소공배수를 구할 수 있습니다!

연습문제

https://www.acmicpc.net/problem/2609

https://www.acmicpc.net/problem/1934

읽을 거리

"유클리드 호제법의 시간 복잡도는 어떻게 되나요?"
"정말 유클리드 호제법이 더욱 빠른가요?"

https://www.weeklyps.com/entry/%EC%9C%A0%ED%81%B4%EB%A6%AC%EB%93%9C-%ED%98%B8%EC%A0%9C%EB%B2%95-%EC%B5%9C%EB%8C%80%EA%B3%B5%EC%95%BD%EC%88%98-%EA%B5%AC%ED%95%98%EA%B8%B0

읽을 거리

재귀 VS 반복문

세련됨 vs 성능

https://newstars.tistory.com/17

읽을 거리

"최소공배수 공식은 어떻게 나온거죠?"

https://mathbang.net/206

소수 (Prime Number)

소수

소수

약수로 1과 자기 자신만을 가지는 수.

소수

소수

약수로 1과 자기 자신만을 가지는 수.

⬇

N이 소수가 되려면,
2 이상, N-1 이하의 자연수로 나누어 떨어지면 안된다!

소수

"N이 소수가 되려면,
2 이상, N-1 이하의 자연수로 나누어 떨어지면 안된다!"

위를 바탕으로 소수를 판단하는
알고리즘을 구현해봅시다!

소수

#시간 복잡도는 무엇일까요?

```
bool prime(int n)
{
    if (n < 2)
    {
        return false;
    }
    for (int i = 2; i <= n - 1; i++)
    {
        if (n % i == 0)
        {
            return false;
        }
    }
    return true;
}
```

소수

더 좋은 방법이 있을까요?

합성수 *N*의 약수들을 살펴봅시다!

32 〉1 2 4 8 16 32

36 〉1 2 3 4 6 9 12 18 36

39 〉1 3 13 39

소수

"자기 자신을 제외한 약수들은 N/2를 초과하지 않습니다!"

32 〉1 2 4 8 16 32

36 〉1 2 3 4 6 9 12 18 36

39 〉1 3 13 39

소수

Why?

N = A X B로 나타낼 수 있습니다. (A <= B)
이때, 가능한 A 중에 가장 작은 값은 2입니다.
따라서, B는 N/2를 초과할 수 없습니다!

32 〉1 2 4 8 $\boxed{16}$ 32

36 〉1 2 3 4 6 9 12 $\boxed{18}$ 36

39 〉1 3 $\boxed{13}$ 39

소수

즉, 소수를 판별하기 위해선 N/2까지만 약수인지 판별하면 됩니다!

소수

#시간 복잡도는 무엇일까요?

```
bool prime(int n)
{
    if (n < 2)
    {
        return false;
    }
    for (int i = 2; i <= n / 2; i++)
    {
        if (n % i == 0)
        {
            return false;
        }
    }
    return true;
}
```

한 끗 차이지만 연산 횟수는 두배로 줄었습니다!

소수

더 좋은 방법이 있을까요?

#잘못했어요.. 살려주세요..

소수

더 좋은 방법이 있을까요?

"주어진 자연수 N이 소수이기 위한
필요충분조건은
N이 N의 제곱근보다 크지 않은 어떤 소수로도 나눠지지 않는다"

왜 그럴까요? 이번 문제는 <u>직접 생각</u>해봅시다!

소수

[redacted]

#시간 복잡도는 무엇일까요?

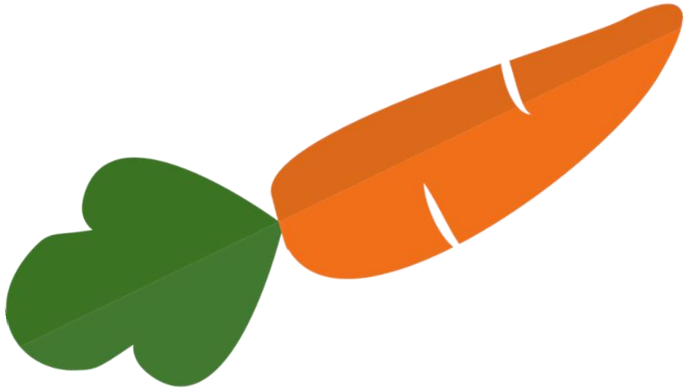
```
bool prime(int n)
{
    if (n < 2)
    {
        return false;
    }
    for (int i = 2; i * i <= n; i++)
    {
        if (n % i == 0)
        {
            return false;
        }
    }
    return true;
}
```

연습문제

https://www.acmicpc.net/problem/1978

소수

이제 2부터 N까지의 소수를 구하는
알고리즘을 구현해봅시다!



#그만..

소수

2부터 N까지의 소수를 구해보자!

앞서 만든 소수 판정 알고리즘을 2~N까지 돌려볼 수 있을까요?

만약 1부터 1000000까지의 소수를 구해봅시다!
각각의 수에 대해 소수 판정 알고리즘을 적용하면
1000000 x 1000 번 즉, 10억 번의 연산횟수가 필요합니다!
따라서 주어진 범위 내의 소수를 찾기에는
비효율적!입니다

소수

2부터 N까지의 소수를 구해보자!

<u>어떤 점을 개선할 수 있을까요?</u>

-> 소수의 배수는 그 소수를 제외하고 소수가 아닙니다!

소수

2부터 N까지의 소수를 구해보자!

어떤 점을 개선할 수 있을까요?

-> 소수의 배수는 그 소수를 제외하고 소수가 아닙니다!

⬇

에라토스테네스 체의 기본 아이디어!

소수

에라토스테네스의 체에 대해 알아봅시다!

1~N까지의 모든 소수를 구하려면?

1. 2부터 N까지의 모든 수를 써 놓는다.
2. 아직 지워지지 않은 수에서 가장 작은 수를 찾는다.
3. 그 수는 소수이다.
4. 그 수의 배수를 제거한다.

읽을 거리

에라토스테네스의 체!
영상을 통해서도 이해해 봅시다!

https://ko.khanacademy.org/computing/computer-science/cryptography/comp-number-theory/v/sieve-of-eratosthenes-prime-adventure-part-4

소수

```cpp
int prime[100];  // 소수 저장
int pn = 0;      // 소수의 개수
bool check[101]; // 소수가 아니면 true
int n = 100;     // 100까지 소수
for (int i = 2; i <= n; i++)
{
    if (check[i] == false)
    {
        prime[pn++] = i;
        for (int j = i * i; j <= n; j += i)
        {
            check[j] = true;
        }
    }
}
```

연습문제

https://www.acmicpc.net/problem/1929

읽을 거리

소수는 실생활 뿐만 아니라
전산 및 암호학에서 매우 유용하게 사용됩니다!
컴퓨터공학과에서 앞으로도 볼 일이 많습니다!

https://www.gereports.kr/how-prime-numbers-are-used-to-keep-your-data-safe/

다음시간!

기본적인 자료구조 스택, 큐, 덱 등의
원리와 실생활 속 활용 사례를 알아보고,
이를 바탕으로 문제를 풀어봅시다!

The end.

“네 시작은 미약하였으나 네 나중은 심히 창대하리라”
-욥기 8장 7절