

알고리즘 시작

최백준 choi@startlink.io

알고리즘

알고리즘

Algorithm

3

- In mathematics and computer science, an algorithm (/ˈælgərɪðəm/) is an unambiguous specification of how to solve a class of problems. Algorithms can perform calculation, data processing, and automated reasoning tasks.
- 출처: <https://en.wikipedia.org/wiki/Algorithm>

알고리즘

Algorithm

- 알고리즘(라틴어, 독일어: Algorithmus, 영어: algorithm 알고리즘, IPA: [ælgərɪðm])은 수학과 컴퓨터 과학, 언어학 또는 관련 분야에서 **어떠한 문제를 해결하기 위해 정해진 일련의 절차나 방법을 공식화**한 형태로 표현한 것을 말한다.
- 출처: <https://ko.wikipedia.org/wiki/알고리즘>

알고리즘

Algorithm

5

- 알고리즘(라틴어, 독일어: Algorithmus, 영어: algorithm 알고리즘, IPA: [ælgərɪðm])은 수학과 컴퓨터 과학, 언어학 또는 관련 분야에서 **어떠한 문제를 해결하기 위해 정해진 일련의 절차나 방법을 공식화한 형태로 표현한 것**을 말한다.
- 출처: <https://ko.wikipedia.org/wiki/알고리즘>

알고리즘

Algorithm

- 알고리즘은 음식의 레시피와 비슷하다. (같은 것은 아니다)
- 요리의 재료를 이용해 레시피의 방법으로 요리한 다음, 요리를 완성한다.
- 는
- 입력을 이용해 알고리즘으로 문제를 해결하고, 정답을 출력한다.
- 와 비슷하다.

알고리즘

Algorithm

7

- 다익스트라 (Dijkstra), 프림 (Prim), 크루스칼 (Kruskal)처럼 이름이 붙어있는 것만 알고리즘은 아니다.
- 어떤 문제를 해결하는 방법을 모두 알고리즘이라고 할 수 있다.

알고리즘

Algorithm

- 많은 개발은 어떠한 문제를 해결해야 하는 것이 목적인 경우가 많다.

알고리즘

Algorithm

- 알고리즘 문제는 상대적으로 대부분의 개발보다 크기가 작다.
- 문제도 다른 분야의 특정 문제를 알고리즘 문제 스타일로 변형하는 경우가 많다.
- 코딩 테스트나 인터뷰에서는 문제를 모델링하고 해결하는 능력을 알아보기 위해서 알고리즘 문제를 푼다.

알고리즘

Algorithm

- 게임을 잘하고 싶다. 어떻게 해야 할까?
- 게임을 많이 해야 한다.
- 물론, 게임을 무작정 많이 하면 되는 것은 아니다.

알고리즘

Algorithm

- 코딩을 잘하고 싶다. 어떻게 해야 할까?
- 코딩을 많이 해야 한다.
- 물론, 코딩을 무작정 많이 하면 되는 것은 아니다.

알고리즘

Algorithm

12



- 어떤 알고리즘의 원리와 증명을 이해하는 것은 중요하지만 응용하는 것이 더 중요하다.

알고리즘

Algorithm

- 어떤 문제를 보고 도움 없이 스스로 방법을 떠올려야 할 필요는 없다.
- 예를 들어, 수학 공부를 할 때 피타고라스의 정리, 미분, 적분을 스스로 떠올리는 것을 요구하지 않는다.

알고리즘

Algorithm

14

- 이 문제를 어떻게 푸는지 모르겠으니 어떻게든 스스로 풀어내는 것이 나쁜 생각은 아니다.
- 하지만, 인생은 짧고, 시간은 없고, 할 일은 많다.
- 개발은 모든 것을 스스로 해결하는 것을 요구하지 않는다.
- 도움을 받는 것과 이해하는 것, 검색을 하는 것도 매우 중요한 능력이다.
- 조금 고민해보고 모르겠으면, 답을 보고 이해하자.

알고리즘


Algorithm

- 고민의 시간은 개인마다 차이가 있지만
- 처음 공부할 때는 2-3일 고민해보는 것이 좋다.
- 어느 정도 익숙해진 이후에는 2-3시간 고민해보는 것이 좋고,
- 많이 익숙해진 이후에는 20-30분 고민해보는 것이 좋다.

알고리즘

Algorithm

16

- 나는 BFS도 알고, 브루트 포스도 알고, 다이나믹 프로그래밍도 안다. 그런데 문제는 못 풀겠다.
- 그러니까 **이 문제를 푸는 알고리즘이 무엇인지는 어떻게 알아내는 것인가?**
- **이것이 요구하는 능력이다.**
- 따로 법칙은 없기 때문에, 각각의 알고리즘의 특징을 알고  **왜 그 알고리즘으로 다른 문제를 풀 수 있었는지를 위주로 기억해서 문제에 적용해봐야 한다.**

알고리즘

Algorithm

- **먼저**, 각각의 알고리즘 문제를 풀어보면서, **알고리즘을 익히는 것이 좋다.**

알고리즘

Algorithm

- 알고리즘 공부에 가장 효과적인 것은
- 문제 풀이이다.

효율성

효율성

Algorithm

20

- 알고리즘 문제를 해결하는 어떤 코드를 작성했을 때, 이 프로그램이 얼마나 효율적인지 알고 싶다.
- 다음 중 무엇이 가장 중요할까?
 1. 수행 시간
 2. 사용한 메모리
 3. 코드의 길이

효율성

Algorithm

21

- 수행 시간이 가장 중요하다.

효율성

Algorithm

22

- 어떤 프로그램을 작성했는데, 시간이 30일이 걸리면 정말로 30일동안 실행시켜야 한다.
- 어떤 프로그램을 작성했는데, 메모리가 64GB 필요한데, 메모리가 부족하면 램을 구매하면 된다.
- 이런 이유 때문에 문제를 해결할 때는 시간이 중요하다.

효율성

Algorithm

- 앞에서 얘기한 것 보다 더 많은 항목이 있고, 상황에 따라 무엇이 중요한지는 다르다
- 앞 페이지에서 얘기한 것은 알고리즘 문제 해결에 한정지은 것이다.

문제의 크기

문제의 크기

Algorithm

- 개발 상황에서 접하게 되는 상황은 문제를 해결하는 것이고
- 항상 **문제의 크기가 존재**한다.
- 예시 1)쇼핑몰 장바구니 **물건의 개수**
- 예시 2)**게임 동시 접속자의 수**
- 이러한 문제의 크기를 보통 N이라고 하고, **문제의 크기 N에 따라 걸리는 시간**이 다르다.

문제의 크기

Algorithm

- 웹 사이트를 만드는 경우에
- 10명이 동시 접속하는 사이트를 만드는 것과 10만명이 동시 접속하는 사이트를 만드는 방법은
- 매우 큰 차이가 있다. 또, 10만명이 동시 접속하는 사이트를 만드는 방법이 더 어렵다.
- 문제를 해결할 때 ~~또~~ **문제의 크기에 따라 알맞은 방법을 선택하는 것이 좋다.**
- 대부분의 문제는 가장 빠른 방법이 정해져 있지만, 가장 빠른 방법이 너무 어려운 경우일 수도 있어, 그 방법보다는 상대적으로 느린 (하지만 문제는 해결할 수 있음) 방법을 이용하기도 한다.

문제의 크기

Algorithm

27

- 이러한 이유 때문에 문제를 해결할 때 문제의 크기를 먼저 보고 방법을 생각해야 한다.

시간 복잡도

시간 복잡도

Time Complexity

- 시간 복잡도를 이용하면 작성한 코드가 시간이 대략 얼마나 걸릴지 예상할 수 있다.
- 표기법으로 대문자 O를 사용한다. (다양한 시간 복잡도가 많지만, 보통 Big-O만 사용한다)
- 영어로는 Big O Notation
- ☆ 입력의 크기 N에 대해서 시간이 얼마나 걸릴지 나타내는 방법
- 즉, 최악의 경우에 시간이 얼마나 걸릴지 알 수 있다.

시간 복잡도

Time Complexity

- 총 N 명의 사람이 식당에 방문했다.
- 식당에는 메뉴가 M 개 있고, 메뉴판이 1개 있다.
- 사람 1명이 메뉴판을 읽는데 걸리는 시간은 $O(M)$ 이다.
- 주문한 모든 메뉴는 동시에 나왔고, 각 사람 i 가 식사를 하는데 걸리는 시간은 A_i 이다.
- 각 사람이 계산을 하는데 걸리는 시간은 $O(P)$ 이다.
- 각 사람이 메뉴판에 있는 모든 메뉴를 읽는 시간 복잡도 = $O(NM)$
- 모든 사람이 식사를 마치는데 걸리는 시간 = $O(\max(A_i))$
- 식사를 모두 마친 다음 한 줄로 서서 각자 계산을 하는 시간 복잡도 = $O(NP)$

시간 복잡도

Time Complexity

- 시간 복잡도는 소스를 보고 계산할 수도 있고, 소스를 작성하기 전에 먼저 계산해볼 수 있다.
- 문제를 풀기 전에 먼저 생각한 방법의 시간 복잡도를 계산해보고 이 계산 시간 안에 수행될 것 같은 경우에만 구현하는 것이 좋다.

시간 복잡도

Time Complexity

- 아래 소스는 1부터 N까지 합을 계산하는 소스이다.

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


시간 복잡도

Time Complexity

- 아래 소스는 1부터 N까지 합을 계산하는 소스이다.

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

- 시간 복잡도: $O(N)$

시간 복잡도

Time Complexity

- 아래 소스는 1부터 N까지 합을 계산하는 소스이다.

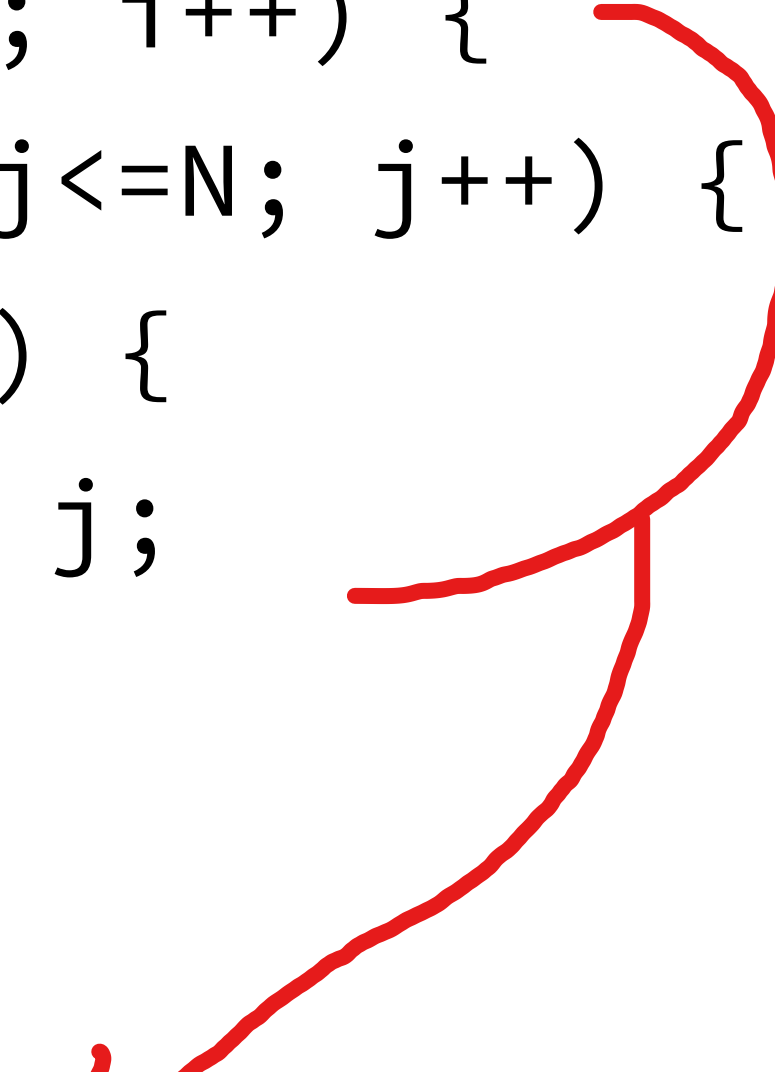
```
int sum = 0;
for (int i=1; i<=N; i++) {
    for (int j=1; j<=N; j++) {
        if (i == j) {
            sum += j;
        }
    }
}
```

시간 복잡도

Time Complexity

- 아래 소스는 1부터 N까지 합을 계산하는 소스이다.

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



- 시간 복잡도: $O(N^2)$

시간 복잡도

Time Complexity

- 아래 소스는 1부터 N까지 합을 계산하는 소스이다.

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

시간 복잡도

Time Complexity

- 아래 소스는 1부터 N까지 합을 계산하는 소스이다.

```
int sum = 0;
```

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

- 시간 복잡도: $O(1)$

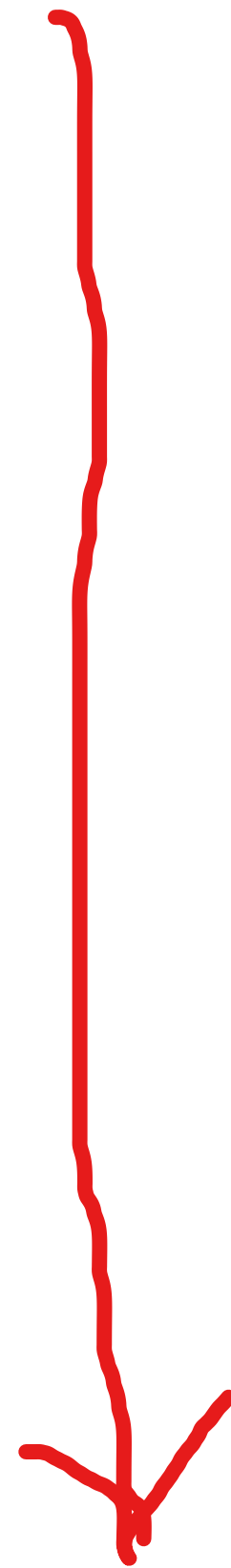


시간 복잡도

Time Complexity

- 대표적인 시간 복잡도는 아래와 같다.

- $O(1)$
- $O(\lg N)$
- $O(N)$
- $O(N \lg N)$
- $O(N^2)$
- $O(N^3)$
- $O(2^N)$
- $O(N!)$



시간 복잡도

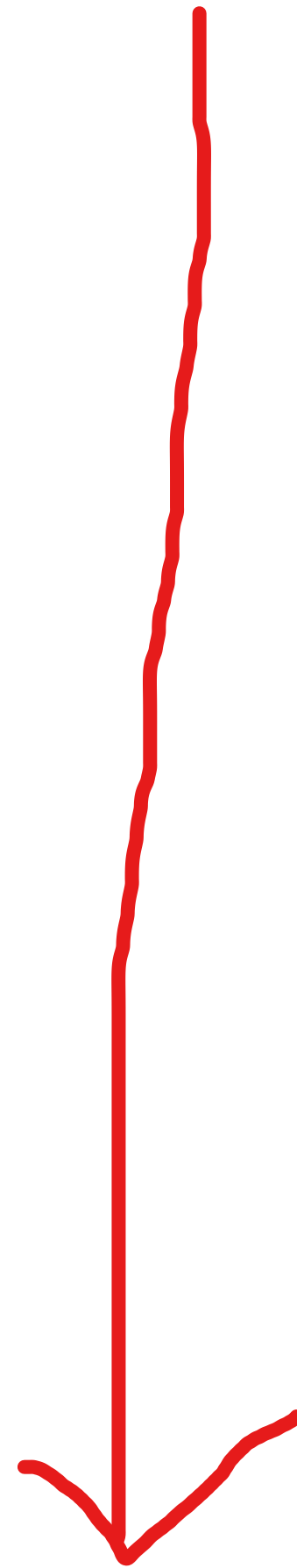
Time Complexity

- 시간 복잡도 안에 가장 큰 입력 범위를 넣었을 때, 1억이 1초정도이다.
- 이 값은 대략적인 값으로, 실제로 구현해보면 1억을 조금 넘어도 1초 이내에 수행이 가능하다.

시간 복잡도

Time Complexity

- 1초가 걸리는 입력의 크기 N
- $O(1)$
- $O(\lg N)$
- $O(N)$: 1억 ☆
- $O(N \lg N)$: 5백만 ☆
- $O(N^2)$: 1만
- $O(N^3)$: 500
- $O(2^N)$: 20
- $O(N!)$: 10



시간 복잡도 계산

Time Complexity

- Big O Notation 에서 상수는 버린다.
- $O(3N^2) = O(N^2)$
- $O(1/2 N^2) = O(N^2)$
- $O(5) = O(1)$
- 두 가지 항이 있을 때, 변수가 같으면 큰 것만 빼고 다 버린다.
- $O(N^2 + N) = O(N^2)$
- $O(N^2 + N \lg N) = O(N^2)$
- 두가지 항이 있는데 변수가 다르면 놔둔다.
- $O(N^2 + M)$

메모리

사용한 메모리

Memory

- ~~메모리~~ 제한은 보통 넉넉하기 때문에, 걱정할 필요가 없다.
- 대략적으로 얼마나 공간을 사용할지 예상할 수는 있다.

사용한 메모리

Memory

- 보통 가장 많은 공간을 사용하는 것은 보통 배열이다.
- 배열이 사용한 공간: 배열의 크기 \times 자료형의 크기 B
- `int a[10000];`
- `int a[100000];`
- `int a[1000000];`
- `int a[1000][1000];`
- `int a[10000][10000];`
- `int a[100000][100000];`

사용한 메모리

Memory

- 보통 가장 많은 공간을 사용하는 것은 보통 배열이다.
- 배열이 사용한 공간: 배열의 크기 \times 자료형의 크기 B
- `int a[10000];` $\rightarrow 10000 \times 4B$
- `int a[100000];` $\rightarrow 100000 \times 4B$
- `int a[1000000];` $\rightarrow 1000000 \times 4B$
- `int a[1000][1000];` $\rightarrow 1000 \times 1000 \times 4B$
- `int a[10000][10000];` $\rightarrow 10000 \times 10000 \times 4B$
- `int a[100000][100000];` $\rightarrow 100000 \times 100000 \times 4B$

사용한 메모리

Memory

- 보통 가장 많은 공간을 사용하는 것은 보통 배열이다.
- 배열이 사용한 공간: 배열의 크기 \times 자료형의 크기 B
- `int a[10000];` $\rightarrow 10000 \times 4B = 40,000B$
- `int a[100000];` $\rightarrow 100000 \times 4B = 400,000B$
- `int a[1000000];` $\rightarrow 1000000 \times 4B = 4,000,000B$
- `int a[1000][1000];` $\rightarrow 1000 \times 1000 \times 4B = 4,000,000B$
- `int a[10000][10000];` $\rightarrow 10000 \times 10000 \times 4B = 400,000,000B$
- `int a[100000][100000];` $\rightarrow 100000 \times 100000 \times 4B = 40,000,000,000B$

사용한 메모리

Memory

47

☆ 1억까지는 여유 (int)

- 보통 가장 많은 공간을 사용하는 것은 보통 배열이다.
- 배열이 사용한 공간: 배열의 크기 \times 자료형의 크기 B
- `int a[10000];` $\rightarrow 10000 \times 4B = 40,000B = 39.06KB$
- `int a[100000];` $\rightarrow 100000 \times 4B = 400,000B = 390.62KB$
- `int a[1000000];` $\rightarrow 1000000 \times 4B = 4,000,000B = 3.814MB$
- `int a[1000][1000];` $\rightarrow 1000 \times 1000 \times 4B = 4,000,000B = 3.814MB$
- `int a[10000][10000];` $\rightarrow 10000 \times 10000 \times 4B = 400,000,000B = 381.469MB$
- `int a[100000][100000];` $\rightarrow 100000 \times 100000 \times 4B = 40,000,000,000B = 37.25GB$

사용한 메모리

Memory

48

- 보통 배열의 크기가 크면 시간 초과를 받는 경우가 많다.
- 불필요한 공간이 없다면, 대부분 메모리 제한은 알아서 지켜진다.

입/출력

C 입출력

C

- C의 경우에는 scanf/printf를 사용할 수 있다.

C++ 입출력

C++

- C++의 경우에는 scanf/printf, cin/cout을 사용할 수 있다.
- cin/cout은 scanf/printf보다 느리기 때문에, 입/출력이 많은 문제의 경우에는 scanf/printf를 사용하는 것이 좋다.
- cin/cout의 경우 아래 세 줄을 추가하면 scanf/printf만큼 빨라진다.

ios_base::sync_with_stdio(false);

cin.tie(NULL);

~~cout.tie(NULL);~~

- 이 경우에는 cin/cout와 scanf/printf를 섞어 쓰면 안된다. 즉, cin/cout만 써야 한다.

Java 입출력

Java

- Java는 입력은 Scanner, 출력은 System.out을 사용한다.

```
Scanner sc = new Scanner(System.in);
```

- 입력이 많은 경우에는 속도가 느리기 때문에, BufferedReader를 사용한다.

```
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
```

- 출력이 많은 경우에는 StringBuilder를 사용해서 한 문자열로 만들어서 출력을 한 번만 사용하거나
- BufferedWriter를 사용한다.

입력 속도 비교

Input

- 10,000이하의 자연수 10,000,000개가 적힌 파일을 입력받는데 걸리는 시간.
- C++17 (sync_with_stdio, cin.tie 사용): 0.5938초
- Java (BufferedReader): 0.6585초
- C11 (scanf): 0.9206초
- C++17 (cin): 2.1742초
- Java (Scanner): 4.8448초
- 출처: <https://www.acmicpc.net/blog/view/56>

출력 속도 비교

Output

- 1부터 10,000,000까지 자연수를 한 줄에 하나씩 출력하는 시간

- C++17 (cout, '\n' 사용): 0.827초

- C11 (printf): 0.9118초

- Java (BufferedWriter): 0.9581초

- Java (StringBuilder 사용): 1.1881초

- C++17 (cout, endl 사용): 11.5322초

- Java (System.out.println): 30.013초

- 출처: <https://www.acmicpc.net/blog/view/57>

→ endl → 느리다.

Hello World

<https://www.acmicpc.net/problem/2557>

- Hello World!를 출력하는 문제

A+B

A+B

- 두 수를 입력받고 A+B를 출력하는 문제
- <https://www.acmicpc.net/problem/1000>
- <https://www.acmicpc.net/problem/2558>
- <https://www.acmicpc.net/problem/10950>
- <https://www.acmicpc.net/problem/10951>
- <https://www.acmicpc.net/problem/10952>
- <https://www.acmicpc.net/problem/10953>
- <https://www.acmicpc.net/problem/11021>
- <https://www.acmicpc.net/problem/11022>

A+B

<https://www.acmicpc.net/problem/1000>

- 소스: <http://codeplus.codes/a63ff228d9c64303a98b2cb26fe118e1>

A+B - 3

58

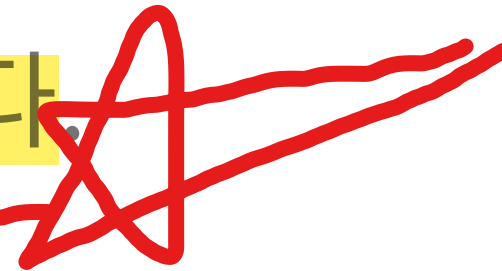
<https://www.acmicpc.net/problem/10950>

- 소스: <http://codeplus.codes/ed44a4f395a945bc9bb2c88d1b4e5fb2>

A+B - 3

<https://www.acmicpc.net/problem/10950>

- 테스트 케이스 형식으로 주어지는 경우에는
- 각각을 독립적인 문제로 생각하고 풀면 된다.
- 전체 테스트 케이스를 입력 받은 다음에, 풀지 않아도 된다.



A+B - 3

<https://www.acmicpc.net/problem/10950>

```
int t;
int a[100], b[100];
scanf("%d", &t);
for (int i=0; i<t; i++) {
    scanf("%d %d", &a[i], &b[i]);
}
```

```
for (int i=0; i<t; i++) {
    printf("%d\n", a[i]+b[i]);
}
```

- 이렇게 입력을 다 받고, 모아서 하나씩 출력하지 않고

A+B - 3

<https://www.acmicpc.net/problem/10950>

```
int t;
int a,b;
scanf("%d",&t);
while (t--) {
    scanf("%d %d",&a,&b);
    printf("%d\n",a+b);
}
```

- 이렇게 **하나 하나 입력받고 풀면 된다.**
- 앞 페이지의 방법 처럼 구현하면, T개수를 모를때 배열의 크기를 정하기 어렵다.
- 또, 전체 입력이 매우 큰 경우에 매우 큰 크기의 배열을 필요하게 된다.

A+B - 4

<https://www.acmicpc.net/problem/10951>

개수 반환 기능

- 이 문제 처럼 입력이 몇 개인지 주어지지 않은 경우에는

- 입력을 EOF까지 받으면 된다.

- C: while (scanf("%d %d", &a, &b) == 2)

- C++: while (cin >> a >> b)

- Java: while (sc.hasNextInt())

T, F 반환

- scanf의 리턴값은 성공적으로 입력받은 변수의 개수다.

A+B - 4

<https://www.acmicpc.net/problem/10951>

- 소스: <http://codeplus.codes/3c1537433cd14c3c9405d4da1a4048bb>

끝

코드 플러스

<https://code.plus>

- 슬라이드에 포함된 소스 코드를 보려면 "정보 수정 > 백준 온라인 저지 연동"을 통해 연동한 다음, "백준 온라인 저지"에 로그인해야 합니다.
- 강의 내용에 대한 질문은 코드 플러스의 "질문 게시판"에서 할 수 있습니다.
- 문제와 소스 코드는 슬라이드에 첨부된 링크를 통해서 볼 수 있으며, "백준 온라인 저지"에서 서비스됩니다.
- 슬라이드와 동영상 강의는 코드 플러스 사이트를 통해서만 볼 수 있으며, 동영상 강의의 녹화와 다운로드, 배포와 유통은 저작권법에 의해서 금지되어 있습니다.
- 다른 경로로 이 슬라이드나 동영상 강의를 본 경우에는 codeplus@startlink.io 로 이메일 보내주세요.
- 강의 내용, 동영상 강의, 슬라이드, 첨부되어 있는 소스 코드의 저작권은 스타트링크와 최백준에게 있습니다.