유니스터디 알고리즘 현장특강

알고리즘 코딩테스트 이해와 대비

UNISTUDY

김동이
dykim@codingmonster.net

강사소개

강사 김동이

- 아주대학교 대학원 컴퓨터공학과 석사과성, 영상처리/딥러닝 전공
- 아주대학교 컴퓨터공학과 졸업
- 안드로이드 및 윈도우 어플리케이션 개발자 경력

- 기업 및 학교 대상 프로그래밍 문제 출제 및 검수
- 학생 및 일반인 대상 프로그래밍 강의 진행
- 직장인 대상 사내 코딩테스트 대비 강의

- 전국 대학생 프로그래밍 경시대회 입상(2015 은상, 2013 동상)
- NAVER CAMPUS HACKDAY 우수 프로젝트 선발
- 팁스타운배 알고리즘 콘테스트 5등
- LG Code Challenger 17등
- …

오늘 할 이야기

아래의 질문을 가지면서 들어주세요

- 알고리즘은 무엇이고 어떻게 받아들여야 할까?
- 기업에서는 왜 코딩테스트를 보는가?
- 어떤 문제들이 나오고 무엇을 요구하는가?
- 어떤 식으로 생각하고 공부해야 하는가?

- **오늘 어떤 질문을 할까?**

알고리즘이란?

일반적인 정의는?

입력과 출력이 명확히 정의 된 문제에 대한 처리 과정을
일반적이고 절차적으로 기술하는 것

입력 값을 통해 출력 값을 만들어 낼 수 있는 방법을
재현(구현)가능한 수준으로 자세하게 기술한 것

계산 vs 알고리즘

단순히 계산을 하는 것과 알고리즘을 설계하는 것의 차이

계산은 입력 값을 알고있을 때 그 입력에 대한 결과만을 찾아내는 것

예〉 [1, 2, 2, 3] 중 가장 많이 등장한 숫자는 무엇인가?

알고리즘은 어떤 입력 값이 들어와도 항상 의도한 결과를 구해낼 수 있는 일반적인 방법이다

예〉 N개의 숫자가 담긴 배열에서 가장 많이 등장한 숫자를 구할 수 있는 방법은?

알고리즘이란?

프로그래밍에서 알고리즘이란,

하나의 함수를 완성하는 과정을 떠올리면 이해하기 쉽다.

```
//n개의 정수가 있는 배열 arr에서 최대값을 찾아 반환하는 함수
int get_maximum(int arr[], int n)
{
    int maximum = 0;
    //알고리즘 시작




    //알고리즘 끝
    return maximum;
}
```

알고리즘이란?

같은 문제를 해결하는 알고리즘은 여러가지가 있을 수 있다.
당연히 알고리즘마다 장점과 단점이 있을 수 있다.

- 소요시간, 코드길이, 사용메모리, 제한사항 …

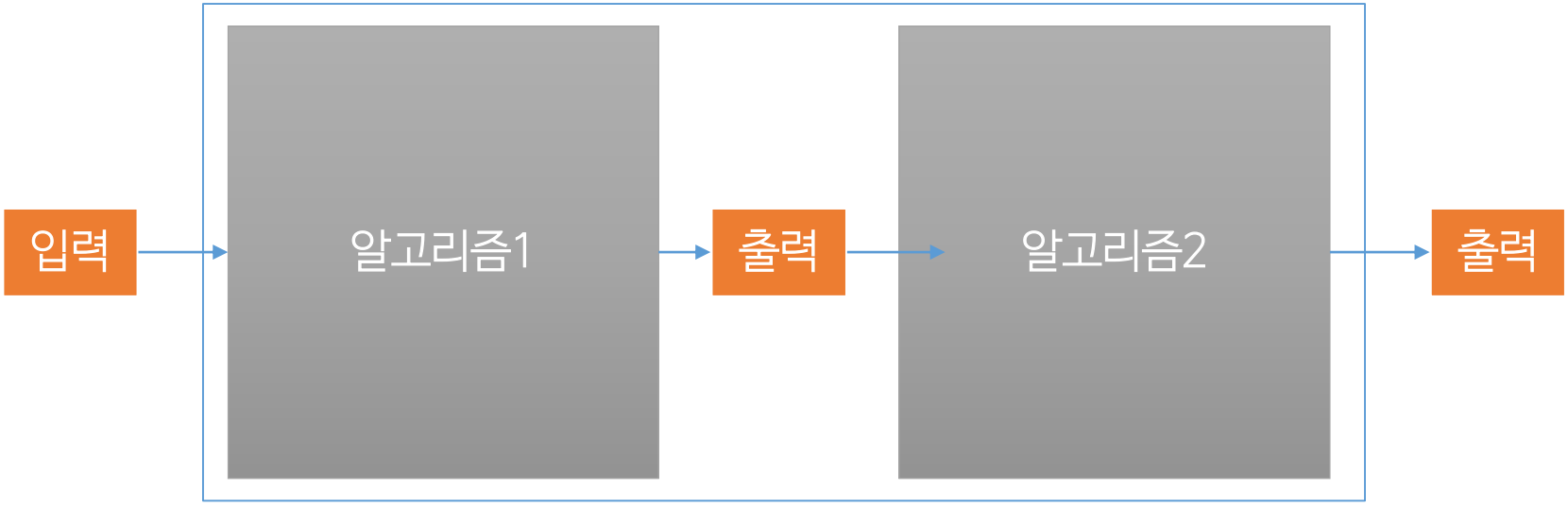
```
//1,000이하의 자연수 n개가 있는 배열 arr에서
//가장 자주 등장한 숫자를 반환하는 함수
int get_frequent_value(int arr[], int n)
{
    int frequent = 0;
    //이 문제를 해결할 수 있는 알고리즘은 어떤 것들이 있을까?

    return frequent;
}
```

알고리즘이란?

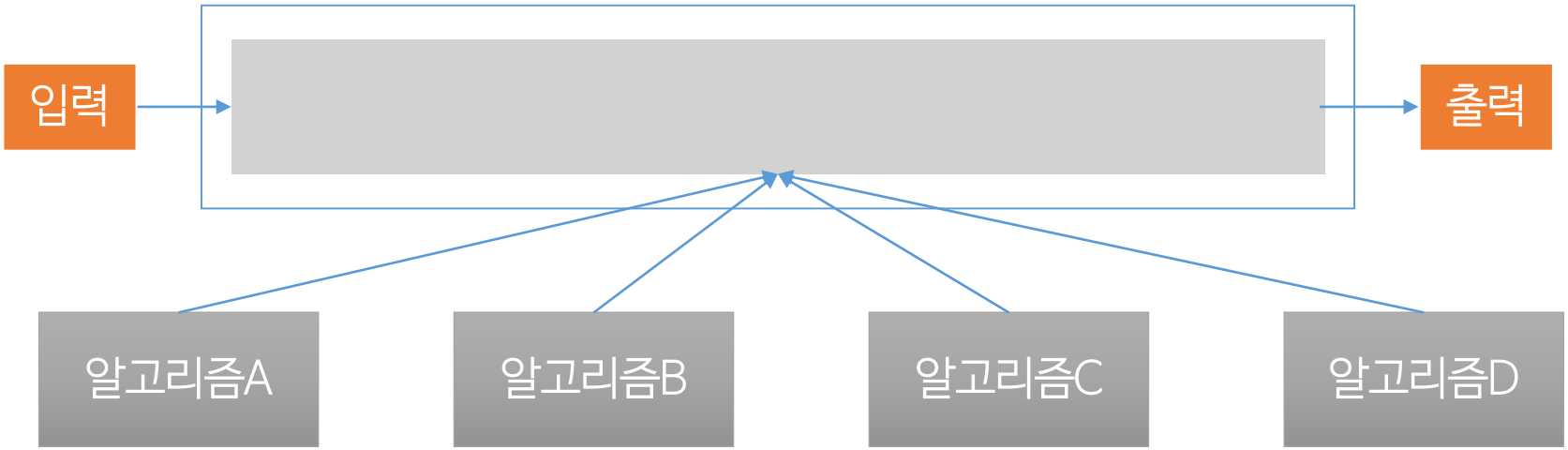
당연히 모든 문제를 한 번에 해결할 수 있는 알고리즘이 있을 수는 없다.

하나의 문제를 해결하기 위하여 여러 알고리즘과 자료구조가 종합적으로 적용되는 경우가 일반적이다.



알고리즘이란?

마찬가지로, 하나의 문제를 해결 할 수 있는
여러가지 알고리즘이 있을 수도 있다.



문제해결 by 프로그래밍

기업들이 프로그래머 요구하는 **문제해결능력**이란?

- 일상/업무적인 표현으로 기술된 문제적 상황을 이해
- 잘 정의된 입력/출력을 요구하는 프로그래밍적 문제로 추상화
- 이를 해결/개선하는 알고리즘들을 설계하고
- 그 중 현 상황에 가장 적합한 방법들 선택한 후
- 이를 실제로 구현하는 것

어려워 보인다 …

문제해결 by 프로그래밍

문제해결능력을 기르는 건 쉽지가 않다…
그 과정에서 요구되는 지식들이 종합적이기 때문

- 일상/업무적인 표현으로 기술된 문제적 상황을 이해
  - 커뮤니케이션 능력, 어휘와 언어에 대한 지식
- 잘 정의된 입력/출력을 요구하는 프로그래밍적 문제로 추상화
  - 비슷한 사례나 관련 분야에 대한 경험 혹은 지식
- 이를 해결/개선하는 알고리즘들을 설계하고
  - 알고리즘, 자료구조들에 대한 지식
- 그 중 현 상황에 가장 적합한 방법들 선택한 후
  - 시스템에 대한 이해, 효율성에 대한 분석과 평가를 할 수 있는 지식
- 이를 실제로 정확히 구현하는 것
  - 알고리즘을 실제로 정확히 구현할 수 있는 프로그래밍적 지식

문제해결능력을 기르는 방법

물론 모든 걸 경험하고 공부하는게 가장 좋지만…

현실적으로 너무 힘들고 시간이 오래 걸린다

문제해결능력을 기르는 방법

알고리즘 문제를 풀어보며 필요한 내용들을 찾아가며 공부하면 자연스럽게 아래 과정을 반복하게 된다.

- 문제를 읽고 프로그래밍적으로 이해하기
- 알고리즘 설계하기
- 시간/메모리 제한에 대해 고민하기
- 새로운 알고리즘도 고려하기
- 올바르게 구현하기

많은 문제를 풀고 필요한 내용을 공부하는 과정에서 많은 훈련이 된다.

문제해결능력을 기르는 방법

어떻게 공부를 시작하면 좋을까?

- 기본적으로 사용할 언어를 하나 정하고 문법을 공부한다
- 처음부터 어려운 알고리즘에 도전하기 보다는, 기초적인 구현부터 연습한다
  - 결국 어려운 알고리즘도 기본적인 알고리즘들이 모여서 만들어진다.
- 막히는 문법, 구현법 위주로 공부해 나간다
- 구현이 익숙해지면 알고리즘 공부를 병행한다'

- 대부분의 책들은 대회 대비용 상급자 대상이기 때문에 바로 보기에는 어렵다!

- http://edu.goorm.io/index 에서 '알고리즘 문제해결기법 입문' 수강!
  - (메모)

Warming Up. A-합 구하기

아래와 같은 함수를 완성하는 문제라고 생각해보자

```
//n개의 정수가 있는 배열 arr의 모든 원소의 합을 반환하는 함수
int get_sum(int arr[], int n)
{
    int sum = 0;
    //알고리즘 시작



    //알고리즘 끝
    return sum;
}
```

Warming Up. A-합 구하기

배열의 모든 원소의 합을 S이라고 하자

$$S = arr[0] + arr[1] + \ \ldots + arr[n-1]$$

$$\therefore S = \sum_{0}^{n-1} arr[i]$$

즉 모든 원소를 '한 번씩' 더하면 전체의 합을 계산할 수 있다.

Warming Up. A-합 구하기

아래와 같이 구현하여 답을 구할 수 있다.

```
//n개의 정수가 있는 배열 arr의 모든 원소의 합을 반환하는 함수
int get_sum(int arr[], int n)
{
    int sum = 0;
    //알고리즘 시작
    for(int i = 0 ; i < n ; i++)
    {   //모든 배열의 원소 i에 대하여
        //배열의 원소 arr[i]가 한번 씩만 지나간다!!

        //변수 sum에 모든 arr[i]를 한 번씩 더한다
        sum = sum + arr[i];
    }
    //알고리즘 끝
    return sum;
}
```

Warming Up. B-배열의 최대값

아래와 같은 함수를 완성하는 문제라고 생각해보자

```
//n개의 정수가 있는 배열 arr에서 최대값을 찾아 반환하는 함수
int get_maximum(int arr[], int n)
{
    int maximum = arr[0];
    //알고리즘 시작




    //알고리즘 끝
    return maximum;
}
```

Warming Up. B-배열의 최대값

배열의 최대값 K는 아래와 같은 특징을 가진다

- 배열의 원소 중 하나이다
- 배열의 모든 원소는 K이하의 값을 가진다.

즉 K배열의 모든 값 중 자신보다 큰 값이 없는 숫자이다

Warming Up. B-배열의 최대값

아래와 같은 함수를 완성하는 문제라고 생각해보자

```
int get_maximum(int arr[], int n)
{
    int maximum = arr[0];
    for(int i = 0 ; i < n ; i ++)
    { //모든 arr[i]에 대해
        if(maximum < arr[i])
        {   // arr[i]이 maximum보다 크다면
            //maximum은 arr[i]로 덮어쓴다
            maximum = arr[i];
        }
    }
    //maximum에는 배열의 원소 중 최대값이 저장되어 있다
    return maximum;
}
```

Warming Up. C-배열 탐색하기

아래와 같은 함수를 완성하는 문제라고 생각해보자

```
int find_index(int arr[], int n, int k)
{   //배열 arr에서 정수 k가 존재한다면 그 인덱스를 반환하는 함수
    //존재하지 않는다면 -1을 반환한다
    //단, 모든 정수는 서로 다르다
    int index = -1;

     //some algorithm


    return index;
}
```

Warming Up. C-배열 탐색하기

어떤 조건에 맞는 데이터가 존재하는지, 어디에 있는지를 판단하는 것을 '탐색'이라고 한다.

탐색에 대해서는 항상 아래 두 가지에 유의한다.

- 조건에 맞는 데이터가 여러 개라면?
- 조건에 맞는 데이터가 존재하지 않는다면?

Warming Up. C-배열 탐색하기

아래와 같이 구현할 수 있다.

```
int find_index(int arr[], int n, int k)
{ //배열 arr에서 정수 k가 존재한다면 그 인덱스를 반환하는 함수
    //존재하지 않는다면 -1을 반환한다
    int index = -1;

    for(int i = 0 ; i < n ; i ++)
    { //모든 arr[i]에 대해 비교해본다

        if(arr[i] == k )
        { //arr[i]가 k와 일치하는 i만 index 변수에 저장한다
            index = i;
        }

    }
    //문제의 조건에 부합하는가?
    return index;
}
```

Warming Up. D-카운팅 하기

아래와 같은 함수를 완성하는 문제라고 생각해보자

```
int get_count(int arr[], int n, int k)
{   //n개의 데이터가 존재하는 배열 arr에서
    //k가 등장하는 횟수를 반환하는 함수
    int cnt = 0;

    //some algorithm...

    return cnt;
}
```

Warming Up. D-카운팅 하기

다음의 함수가 항상 성립한다고 보장할 수 있는가?

```
int get_count(int arr[], int n, int k)
{ //n개의 데이터가 존재하는 배열 arr에서
    //k가 등장하는 횟수를 반환하는 함수
    int cnt = 0;

    for(int i = 0 ; i < n ; i ++)
    { //모든 arr[i]에 대해서 수행

        if(arr[i] == k)
        { //arr[i]가 k와 일치하는 i에 대해서만
            cnt ++;
        }
    }

    return cnt;
}
```

Practice. A-놀이공원

전체 N명의 손님들 중 몸무게가 P이하인 사람들이 모두 함께 놀이기구를 탈 수 있는지? 를 판단하는 문제

놀이기구를 탈 수 있다?

- 탑승하려는 사람들의 몸무게의 합이 Q이하이다.

Practice. A-놀이공원

전체 N명의 손님들 중 몸무게가 P이하인 사람들이 모두 함께 놀이기구를 탈 수 있는지? 를 판단하는 문제

- 몸무게가 P이하인 사람들의 수를 C, 그 몸무게 합을 S라 하자.
- 문제를 풀기 위해서는 S와 C를 알아야 한다.
- S와 C는 간단히 계산할 수 있다.
- S와 Q를 if문으로 비교하는 문제가 된다.
- 비교 결과에 따라 적당히 출력해준다.

Practice. A-합구하기2

결국 S와 C라는 값을 계산할 수 있다고 가정하면 쉬운 문제

```
//문제의 정답을 출력하는 함수
void print_answer(int arr[], int n, int p, int q)
{
    int s, c;
    s = get_sum(arr, n, p); //arr에서 p이하인 값들의 합
    c = get_cnt(arr, n, p); //arr에서 p이하인 값들의 수

    printf("%d %d\n", c, s);
    if( s <= q ){
        printf("YES\n");
    }else{
        printf("NO\n");
    }
    return 0;
}
```

Practice. B-오름차순인가?

주어진 N개의 숫자들의 순서가 오름차순인지 확인하는 문제

- 전체가 옳은 지 확인하는 것 보다는
- 하나라도 옳지 않은 지 확인하는 것이 더 효율적인 경우가 많다

Practice. B-오름차순인가?

오름차순을 위반하는 숫자의 수를 세면 쉽게 판단 가능하다

```
//문제의 정답을 출력하는 함수
void print_answer(int arr[], int n)
{
    int revcnt = 0;
    //자기 오른쪽 원소보다 큰 값을 가지는 원소의 수를 계산하는 함수
    revcnt = get_rev(arr, n);

    if( revcnt == 0 ){
        printf("YES\n");
    }else{
        printf("NO\n");
    }
    return 0;
}
```

Practice. B-오름차순인가?

특정 조건을 만족하는 횟수를 계산하는 문제로 바뀐다

```
//자기 오른쪽 원소보다 큰 값을 가지는 원소의 수를 계산하는 함수
int get_rev(int arr[], int n)
{
    int cnt = 0;
    for(int i = 0 ; i < n-1; i ++)
    { //마지막 원소는 오른쪽 원소가 없으므로 제외

        if(arr[i] > arr[i+1])
        {
            cnt ++;
            //여기서 break 하면 어떻게 될까?
        }
    }
    return cnt;
}
```

Practice. C-전화번호

N개의 전화번호 뒷자리들 중 가장 많이 등장한 것들 찾는 문제

- 0000~9999의 전화번호는 0~9999의 숫자로 보아도 된다.
- 즉, 그냥 배열에 가장 많이 등장한 정수를 찾는 문제

- 각 번호가 등장한 횟수를 세는 것 자체는 쉽다!
- 하지만 가장 많이 등장한 번호를 바로 찾기는 쉽지 않다.

Practice. C-전화번호

앞의 내용을 응용해 아래와 같이 접근해볼 수 있다.

```
//n개의 정수가 있는 배열 arr에서 가장 여러 번 등장한 숫자를 반환하는 함수
int get_frequent_number(int arr[], int n)
{
    int answer = arr[0];
    int frequency = 0;
    for(int number = 0 ; number <=9999; number ++)
    {
        //배열 arr에서 number가 등장한 횟수를 반환하는 함수라 하자
        int count = get_count(arr, n, number);

        //count가 가장 큰 number가 이 함수의 정답이 된다
        if(count > frequency)
        {
            answer = number;
            frequency = count;
        }
    }
    return answer;
}
```

Practice. C-전화번호

하지만! 반복문을 $10000 \times N$번 정도 수행하게 되므로 느리다

```
//n개의 정수가 있는 배열 arr에서 가장 여러 번 등장한 숫자를 반환하는 함수
int get_frequent_number(int arr[], int n)
{
    int answer = arr[0];
    int frequency = 0;
    for(int number = 0 ; number <=9999; number ++)
    {
        //배열 arr에서 number가 등장한 횟수를 반환하는 함수라 하자
        int count = get_count(arr, n, number);

        //count가 가장 큰 number가 이 함수의 정답이 된다
        if(count > frequency)
        {
            answer = number;
            frequency = count;
        }
    }
    return answer;
}
```

Practice. C-전화번호

아래와 같은 표가 있다고 가정하면 문제를 풀 수 있을까?

0000	…	…	…	4444	4445	4446	…	…	9999
5				10	12	7			9

〈각 번호가 배열 arr에서 등장한 횟수가 기록된 표 'table'〉

Practice. C-전화번호

앞서 정의한 표가 존재한다면, 더 빠르게 계산할 수 있다.

```
//n개의 정수가 있는 배열 arr에서 가장 여러번 등장한 숫자를 반환하는 함수
int get_frequent_number(int arr[], int n)
{
    int answer = arr[0];
    int frequency = 0;
    int table[10000]; //table[i] := 번호 i가 arr에서 등장한 횟수
    make_table(arr, n, table); //빈도 배열을 만들자 O(N)
    for(int number =0; number <= 9999; number ++)
        {
        //table[i] := 번호 i가 arr에서 등장한 횟수
        int count = table[number];
        if(count > frequency)
        {
            answer = number;
            frequency = count;
        }
    }
    return answer;
}
```

Practice. C-전화번호

앞서 정의한 표가 존재한다면, 더 빠르게 계산할 수 있다.

```
void make_table(int arr[], int n, int table[])
{    //각 번호에 대한 빈도수 표인 table을 만드는 함수
     for(int number = 0; number <= 9999; number ++)
     { //table배열을 0으로 모두 초기화한다
          table[number] = 0;
     }

     for(int i = 0 ; i < n; i++)
     { //배열 arr에 등장한 각 번호 arr[i]들에 대해
          int number = arr[i];
          //table배열에서 그 번호에 해당하는 칸의 수를 1 증가시킨다
          table[number] ++;

          //결국 arr에 있는 각 원소가 한 번씩 해당 칸의 빈도를 1 증가 시키므로
          //table[number] := arr에서 number라는 원소가 등장한 횟수가 된다.
     }
}
```

Practice. C-전화번호

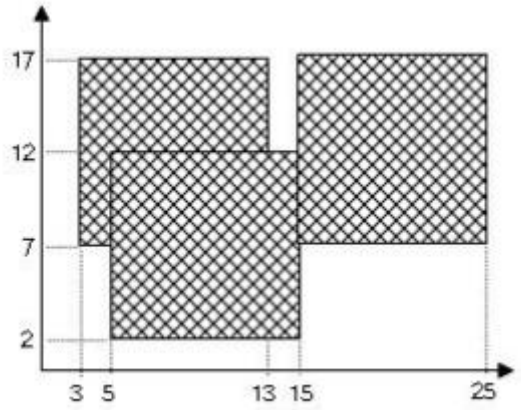
총 걸리는 시간은 이 전의 방법보다 줄어든다.

- table배열을 만드는 데에 $10000 + N$번 정도의 계산
- table배열을 이용해 최빈 값 탐색에 $10000 + N$번 정도의 계산
- 총 $20000 + 2N$번 정도의 연산

- 앞서의 $10000 \times N$번과 비교하면 상당한 차이가 난다

Practice. D-색 종이

수식으로 계산하기는 어려워 보인다..



세 개이상의 사각형이 동시에 겹치기 시작한다면?

Practice. D-색 종이

아래의 특징을 발견하자.

- 항상 축에 수직/수평한 사각형
- 좌표의 범위가 작다
- 모든 좌표는 자연수다!

(1,100)	…	…	…	(100,100)
(1,99)				…
…				…
(1,2)				…
(1,1)	(2,1)	…	(99,1)	(100,1)

즉…

도화지의 각 좌표선이 만나는 격자들을 모아 2차원 배열로 표현하기 쉬워진다

Practice. D-색 종이

도화지의 각 격자 칸을 덮는 색종이의 수를 알 때
결과적으로 덮여진 넓이를 알 수 있을까?

```
//도화지를 덮고있는 색종이의 넓이를 계산하는 함수
//board[x][y] := 격자 (x,y)를 덮고 있는 색종이의 수
int get_area(int board[][101])
{
    int area = 0;

    //some algorithm

    return area;
}
```

Practice. D-색 종이

하나 '이상의' 색종이가 덮고 있는 격자의 수가
곧 덮여진 색종이가 덮고 있는 도화지의 넓이가 된다.

```
//도화지를 덮고있는 색종이의 넓이를 계산하는 함수
//board[x][y] := 격자 (x,y)를 덮고 있는 색종이의 수
int get_area(int board[][101]){
    int area = 0;
    for(int x = 1; x<=100; x++){
        for(int y = 1; y<=100; y++){
            //모든 격자 (x,y)에 대해
            if(board[x][y] >= 1)
            { //색종이가 덮혀져 있는 격자에 대해
                area ++;
            }
        }
    }
    return area;
}
```

문제 해결하기

결과적으로 아래와 같은 사실들을 알 수 있다.

풀기 어려운 문제도 단계적으로 나누어 생각해보면,
간단한 알고리즘들이 모여서 구성된다.

같은 기능이라도 구현법이 다양하며, 각 방법마다 장단점이 있다.

해당 문제의 입력 범위, 제한 시간, 메모리 제한에 따라 유연하게
내가 사용할 알고리즘을 결정해야 한다.

전처리(Pre-processing)

데이터를 내가 처리하기 쉽도록 사전에 가공하는 과정

매번 계산하기 힘든 값들을 미리 모두 계산 해두거나,

임의대로 주어진 데이터들을
내 알고리즘 상에서 처리하기 쉬운 형태로 미리 가공해두는 등의 행위

대표적으로 정렬이 있다.

Practice. E-중복인가?

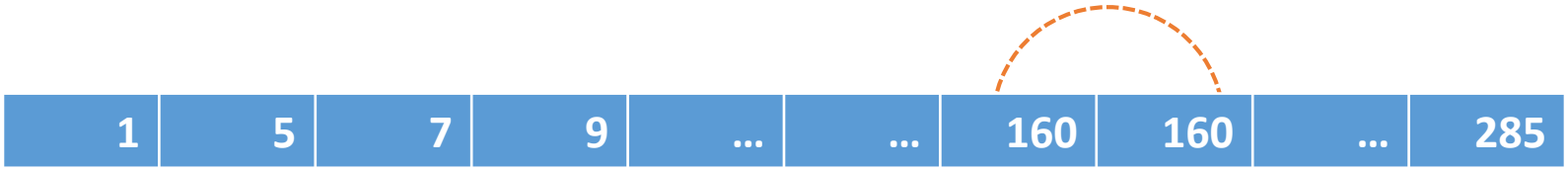
N개의 숫자들 중 중복 값이 존재하는지 확인하는 문제

1. 데이터가 많다. (최대 20만)
2. 값들의 범위가 넓어 배열에 빈도수를 모두 저장할 수 없다.
3. 데이터들에 규칙성이 없다.

Practice. E-중복인가?

숫자들이 만약 오름차순이면 쉽게 구할 수 있을 텐데….

1. 오름차순이라면 중복 숫자는 항상 붙어있다!
2. 배열의 원소 중 하나라도 옆 칸과 일치하면? 중복이 존재한다



하지만 데이터는 정렬되어 주어지지 않는다….

그렇다면 내가 정렬을 하면 된다.

Practice. E-중복인가?

언어별 내장 정렬 기능을 적극 활용하자

- $O(N \log_2 N)$번 만에 모든 원소를 정렬해준다

```
arr : 배열의 이름
n : 정렬 할 원소의 수

//[C++]
//#include<algorithm>
//using namespace std;
sort(arr, arr+n);


//[Java]
Arrays.sort(arr);
```

Practice. E-중복인가?

정렬 된 데이터의 특징을 활용하면 빠르게 풀 수 있다.

```
bool duplicated(int arr[], int n)
{ //배열에 중복된 원소가 존재하면 true,
//그렇지 않으면 false를 반환

    sort(arr, arr+n); //오름차순으로 정렬하자
    for(int i = 0 ; i < n-1; i++)
    { //마지막 원소를 제외하고
        if(arr[i] == arr[i+1])
        {   //오른쪽 원소가 같은 값을 가지는
            //원소가 존재한다면 true를 반환
            return true;
        }
    }
    //그렇지 않으면 false
    return false;
}
```

Practice. F-두 카드

자연수 S를 두 자연수의 합으로 나타낼 수 있는지 판단

$S = x + y$가 되는 x와 y가 배열에 존재하는가?

어떻게 판단할 수 있을까?

=〉 두 숫자를 정하고 더해보고 비교한다.

Practice. F-두 카드

아래의 방법은 정답을 구할 수 있을까?

```
bool canMake(int arr[], int n, int s)
{ //arr의 두 원소의 합으로 s를 만들 수 있으면 true 반환
    for(int i = 0 ; i < n ; i++){
        for(int j = 0 ; j < n ; j++){
            if ( arr[i] + arr[j] == s){
                return true;
            }
        }
    }
    return false;
}
```

입력의 크기가 커서 힘들 것이다.

Practice. F-두 카드

일일이 두 숫자를 모두 다 정해볼 필요가 있을까?

```
bool canMake(int arr[], int n, int s)
{ //두 원소의 합으로 s를 만들 수 있으면 true 반환
    for(int i = 0 ; i < n ; i++){
        int x = arr[i];
        int y = s - z; // s와 x가 정해지면 y의 값도 결정된다

        //some algorithm

    }
    return false;
}
```

배열 arr에서 y가 존재하는지 빠르게 찾을 방법이 필요하다

Practice. F-두 카드

Q〉 숫자들이 많이 저장된 배열에서 특정 원소가
존재하는지 빠르게 찾을 수 있는 방법을 좀 알려주세요!!

Practice. F-두 카드

Q〉 숫자들이 많이 저장된 배열에서 특정 원소가
존재하는지 빠르게 찾을 수 있는 방법을 좀 알려주세요!!

A〉 배열이 정렬되어 있으면 바이너리 서치로 $O(\log_2 N)$만에 찾을 수 있습니다.
구현도 간단하고, 보통은 언어에 함수가 내장되어 있습니다.

정렬이 되어 있어야만 사용할 수 있는 기능이다….

그렇다면 내가 정렬을 하면 된다.

Practice. F-두 카드

Q〉 숫자들이 많이 저장된 배열에서 특정 원소가
존재하는지 빠르게 찾을 수 있는 방법을 좀 알려주세요!!

```
//[C++]
//원소가 존재하면 true, 존재하지 않으면 false
bool result = binary_search(arr, arr+n, k);


//[Java]
//원소가 존재하면 0이상의 인덱스, 존재하지 않으면 음수
int result = Arrays.binarySearch(arr, k);
```

Practice. F-두 카드

바로 적용해보자

```
bool canMake(int arr[], int n, int s)
{ //두 원소의 합으로 s를 만들 수 있으면 true 반환
    //정렬은 이미 되어있다고 가정하자
    for(int i = 0 ; i < n ; i++){
        int x = arr[i];
        int y = s - z; // s와 x가 정해지면 y의 값도 결정된다

        if(binary_search(arr, arr+n, y))
        {   //y가 arr에 존재하면?
            //s=x+y인 x와 y가 존재한다!
            return true;
        }
    }
    return false;
}
```

QnA

궁금한 점이 있다면 편하게 질문해주세요 ☺

강사 김동이
메일주소: dykim@codingmonster.net
페이스북 페이지 : https://fb.com/CodingMonster (코딩몬스터)