judge.koreatech.ac.kr 사용법

1. 개요

- 이 사이트는 프로그래밍 경시대회 문제 풀이 및 대회 개최 사이트입니다.
- 여기서 말하는 프로그래밍 경시대회는 ACM ICPC에 준하는 대회를 말합니다. ACM은 국제 컴퓨터공학 관련 학회이며, 이 학회는 매년 전 세계 대학생들이 참가할 수 있는 국제 프로그래밍 경시대회를 개최하고 있습니다. 주어진 시간에 주어진 문제를 가장 빨리 정확하게 해결한 팀이 우승을 합니다.
- 프로그래밍 경시대회는 주어진 입력에 대해 문제에서 요구하는 출력을 주는 알고리즘을 구현해야 합니다. 따라서 기출 문제를 많이 풀어보면 알고리즘 설계 및 구현 능력, 자료구조 활용 능력 등을 배양할 수 있습니다.

2. 문제와 제출 코드 검증 절차

- 문제의 구성
  - 문제는 문제 설명, 입력 설명, 출력 설명, 입력 예시, 출력 예시, 도움말로 구성되어 있습니다.
  - 입력 설명에서는 각 데이터의 범위를 잘 파악하여야 합니다. 이를 통해 알고리즘에서 각 데이터를 입력받을 때 사용할 타입을 결정해야 하며, 알고리즘에서 연산을 수행할 때 오버플로우를 고려해야 하는지 등도 판단할 수 있습니다. 또 필요한 알고리즘의 시간복잡도를 예측할 수 있습니다. 예를 들어 테스트 케이스 수가 T이고 $O(n^2)$의 알고리즘을 제출한 경우 최대 n에 대해 주먹구구에 의해 $T \times n^2 < 10^8$이면 통과할 수 있다고 생각하면 됩니다. 정확한 것은 아니며, 시간 복잡도에서 생략된 계수도 영향을 주며, 입력 데이터와 출력 데이터를 처리하는 시간도 고려해야 합니다.
- 테스트 데이터
  - 코드를 작성하여 제출하면 입력 예시에 있는 데이터뿐만 아니라 준비된 테스트 데이터를 이용하여 코드의 정확성을 검증합니다.
  - 보통 하나의 테스트 데이터는 두 개의 파일(*.in, *.out)로 구성되며, 이와 같은 테스트 데이터가 다수 준비되어 있습니다.
  - 확장자가 .in인 파일이 입력 데이터이고, 이 입력 데이터에 대해 제출한 알고리즘의 수행 결과를 .out 파일과 비교하게 됩니다. 물론 special judge를 이용할 경우에는 단순 비교를 하지 않을 수 있습니다.
  - 시간 제한은 모든 테스트 데이터 파일을 통과하는 시간을 검사하는 것이 아니라 각 데스트 데이터 파일을 통과하는 시간을 측정합니다. 이 시간은 실제 수행시간을 측정하기 때문에 동일 코드를 여러 번 제출하면 수행 시간은 매번 조금씩 달라질 수 있습니다.
  - 테스트 데이터의 입력 데이터는 다양하게 구성할 수 있지만 전형적인 형태는 첫 줄에 테스트 케이스가 주어지고, 그다음 줄부터 각 테스트 케이스에 해당하는 입력 데이터가 주어집니다.
- 다양한 테스트 데이터를 이용하여 검증하기 때문에 소스를 제출하기 전에 실제 다양한 테스트데이터를 이용하여 검증하여야 합니다. 보통 코너 케이스를 잘 찾아야 한다고 말합니다. 코너 케이스는 예외적인 경우나 일반적인 경우와 조금 다르게 동작한 경우를 말한다.
  - 방법 1. 테스트해보아야 하는 입력 데이터의 유형을 생각하여 각 유형에 대해 테스트한다. 예를 들어 입력 범위가 32비트 정수이면, **INT_MIN**, 음수, 0, 양수, **INT_MAX**에 대해 올바르게 동작하는지 검사해 볼 수 있습니다.
  - 방법 2. 시간 복잡도 때문에 테스트를 통과할 수 없는 코드이지만 정확한 코드(전수조사 코드)이면 작은 범위에 대해 랜덤하게 테스트 데이터를 만들어 내 알고리즘의 결과와 느리지만 정확한 코드의 결과와 비교해 봅니다.

3. 코드의 구성

- 사용하는 각 언어마다 코드를 구성하는 방식이 다릅니다. 사이트 자주묻는질문을 참고하여 주세요.
- 어떤 언어를 사용하던 모든 테스트 케이스를 입력받은 다음에 처리하는 형태가 아니라 각 테스트 케이스를 입력받고 출력하는 형태로 해결해야 합니다. 모든 테스트 케이스를 다 입력받아 유지하면 메모리 공간이 너무 많이 필요하고, 코딩하기도 불편해집니다. 출력 데이터를 모두 모아 출력하는 것도 마찬가지입니다. 특히, 메모리 사용 초과로 자동 실패할 수도 있습니다.
  - C++

```cpp
int T;
std::cin >> T;
for(int t=0; t<T; ++t){
    // read testcase
    // solve testcase
    // print solution
}
```

  - Java

```java
Scanner in = new Scanner(System.in);
int T = in.nextInt();
for(int t=0; t<T; ++t){
    // read testcase
    // solve testcase
    // print solution
}
```

  - Python3

```python
T = int(stdin.readline())
for _ in range(T):
    # read testcase
    # solve testcase
    # print solution
```

- 입력을 처리할 때 줄단위로 입력받아야 하는 경우가 있습니다. 이 경우 테스트 케이스를 입력받은 후 남아 있는 줄바꿈 문자를 주의해야 합니다. 특히, 정수를 입력받다가 줄단위 문자열 입력을 받는 경우에는 버퍼에 있는 줄바꿈 문자를 제거해 주어야 합니다.
  - C++

```cpp
int T;
std::cin >> T;
// 버퍼 비우기
std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
for(int t=0; t<T; ++t){
    std::string input;
    std::getline(std::cin, input);
    // solve testcase
    // print solution
}
```

  - Java

```java
Scanner in = new Scanner(System.in);
int T = in.nextInt();
in.nextLine(); // 버퍼 비우기
for(int t=0; t<T; ++t){
    String line = in.nextLine();
    // solve testcase
    // print solution
}
```

  - Python은 줄단위로 대부분 입력을 처리하기 때문에 이 부분에 대해 특별히 다르게 처리할 필요는 없습니다.

- 원칙은 출력의 공백의 수, 줄바꿈 위치가 중요합니다. 현재 judge는 정확하게 일치하지 않아도 통과되지만 다른 사이트는 출력 형태가 정확하게 요구사항을 만족하지 않으면 답이 틀린 것으로 판단합니다. 배열에 있는 일련의 정수를 출력해야 하면 다음과 같이 출력하는 것이 올바른 방법입니다.

  - C++

```cpp
std::cout << nums[0];
for(int i=1; i<size; ++i){
    std::cout << ' ' << nums[i];
}
std::cout << '\n';
```

  - Java

```java
System.out.println(IntStream.of(nums).mapToObj(String::valueOf)
    .collect(Collectors.joining(" ")));
```

  - Python3

```python
print(' '.join(map(str, nums)))
```

  물론 크기가 0일 수 있으면 위와 같이 처리할 수 없습니다. 또 이 경우에 마지막에 줄바꿈이 없으면 다음 출력 테스트케이스 출력과 이전 테스트케이스 마지막 출력이 붙어서 출력되기 때문에 답을 틀린 것으로 판단하게 됩니다. 자바와 파이썬은 입출력이 매우 느리기 때문에 제시한 것처럼 출력에 해당하는 문자열을 만든 후 출력을 한번만 하는 것이 속도 향상에 효과적입니다.

- 테스트 케이스마다 입력의 크기가 다를 수 있기 때문에 동적 할당을 이용합니다.

  - C++

```cpp
for(int t=0; t<T; ++t){
    int N;
    std::cin >> N;
    int* nums{new int[N]};
    for(int i=0; i<N; ++i) std::cin >> nums[i];
    // solve testcase
    // print solution
    delete [] nums;
}
```

    C++의 경우에는 일반 배열 대신에 std::vector를 이용하면 직접 반납할 필요가 없기 때문에 편리합니다.

```cpp
for(int t=0; t<T; ++t){
    int N;
    std::cin >> N;
    std::vector<int> nums(N);
    for(int i{0}; i<N; ++i) std::cin >> nums[i];
    // solve testcase
    // print solution
}
```

  - Java

```java
for(int t=0; t<T; ++t){
    int N = in.nextInt();
    int[] nums = new int[N];
    for(int i=0; i<N; ++i) nums[i] = in.nextInt();
    // solve testcase
    // print solution
}
```

- Python3

```
for _ in range(T):
    N = int(stdin.readline())
    nums = list(map(int, stdin.readline().split()))
    # solve testcase
    # print solution
```

파이썬은 N을 활용할 필요가 없습니다.

- 문제에서 요구하는 출력만 출력해야 합니다. 일반적으로 프로그래밍할 때처럼 prompt를 출력하면 그것과 답을 비교하기 때문에 틀린 답으로 판정합니다.

```
std::cout >> "Testcase: ";   // 이렇게 하면 안 됩니다.
std::cin >> T;
```

4. 언어마다 수행 속도의 차이

- 당연한 것이지만 실행 수행시간을 측정하다보니 사용하는 언어에 따라 동일 알고리즘이더라도 수행 시간에 큰 차이가 있습니다. 이 때문에 기본적으로 수행시간을 1초로 제한하지만 자바와 파이썬은 그보다 긴 3초로 제한하고 있습니다.
- 또 입력 데이터가 많거나 출력해야 할 데이터가 많으면 그만큼 수행속도는 더 느려집니다.
  - C++의 경우에는 cout, cin 대신에 printf와 scanf를 사용하는 경우도 있고, 다음을 통해 cout, cin를 사용하더라도 입출력 속도를 빠르게 하는 경우도 있습니다.

```
std::ios_base::sync_with_stdio(false);
std::cin.tie(nullptr);
std::cout.tie(nullptr);
```

  - python3은 T = int(input()) 대신에 다음을 이용할 수 있습니다.

```
from sys import stdin
T = int(stdin.readline())
```

readline을 사용할 경우 줄바꿈 문자를 포함하므로 문자열을 처리할 경우 다음과 같이 사용해야 합니다.

```
line = stdin.readline().rstrip()
```

  - 자바는 한 줄에 있는 여러 개 정수를 입력받을 경우 다음과 같이 할 수 있습니다.

```
int[] nums = Arrays.stream(in.nextLine().split("\\s"))
    .mapToInt(Integer::parseInt).toArray();
```

이 예에서 알 수 있듯이 매우 많은 정수(N개)를 입력받아야 하면 in.nextInt()를 N번 반복하는 것보다 in.nextLine()을 이용하여 줄단위로 한번 읽은 후 각 N개의 문자열을 정수로 변환하는 것이 속도가 더 빠릅니다. 또 Scanner를 이용하는 것보다 다음과 같이 BufferedReader를 이용하면 조금 더 속도를 향상할 수 있습니다.

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

이 경우 in.readLine()를 통해 한 줄을 문자열로 입력받을 수 있습니다. Scanner와 BufferedReader 모두 버퍼 기반 입력이지만 후자는 버퍼의 크기가 훨씬 큽니다. BufferedReader를 이용할 경우 StringTokenizer를 활용하여 줄단위 입력받은 것을 공백 기준으로 나누어 처리하기도 합니다.

  - 하지만 문제 출제가 보통 입출력 속도를 고려하여 문제를 출제하기 때문에 처음부터 입출력 속도를 빠르게 하실 필요는 없습니다. 수행 결과를 보고 입출력 속도만 빠르게 하면 통과할 수 있을 경우에만 속도를 향상하기 위한 방법을 사용하시는 것이 좋습니다. 어차피 문제에서 요구하는 시간 복잡도가 아니면 입출력 속도를 빠르게 한다고 통과하지 못합니다.

5. 입력 데이터의 표현

- 문제에 따라 입력 데이터가 단순 숫자가 아니라 어떤 의미를 지닐 수 있습니다. 이 경우 C++는 구조체, 자바는 내부 **static** 클래스, 파이썬은 클래스를 정의하여 사용하는 것이 코드 가독성 측면에서 효과적일 수 있습니다.
- 예를 들어 x, y 두 개의 값으로 구성된 평면 좌표 데이터를 처리해야 하는 문제이면 C++는 **std::pair**를 활용할 수 있지만 구조체를 정의하여 사용하면 **first**, **second** 대신에 x, y 필드 이름을 통해 프로그래밍을 할 수 있습니다. 파이썬은 간단하게 튜플을 통해 나타낼 수 있지만 파이썬도 클래스를 정의하여 사용하면 가독성에 좋습니다.
  - C++

```cpp
struct Point{
    int x;
    int y;
};

for(int t=0; t<T; ++t){
    int N;
    std::cin >> N;
    std::vector<Point> points(N);
    for(int i{0}; i<N; ++i) std::cin >> points[i].x >> points[i].y;
    // solve testcase
    // print solution
}
```

  - Python3

```python
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

for _ in range(T):
    N = int(stdin.readline())
    data = list(map(int, stdin.readline().split()))
    points = [Point(x, y) for x, y in zip(data[0::2], data[1::2])]
    # solve testcase
    # print solution
```

6. 데이터를 정렬해야 하는 경우

- 정렬은 $O(n \log n)$에 할 수 있기 때문에 정렬이 문제 해결에 도움이 되면 언제든지 활용할 필요가 있습니다. 즉, 구현하는 알고리즘의 복잡도가 $O(n \log n)$보다 나쁘면 필요한 데이터를 정렬하는 것을 주저할 이유가 없습니다.
- 정렬할 때 주의할 점은 구조체 데이터를 정렬할 경우 첫 번째 기준뿐만 아니라 첫 번째 기준 값이 같을 경우 어떻게 정렬해야 하는지 추가 고민이 필요합니다. 예를 들어 좌표를 정렬하는데, x 좌표를 기준으로 정렬하였을 때 x 좌표가 같은 경우 어떤 순서로 정렬해야 하는지 추가로 지정하지 않으면 테스트데이터를 생성한 기준과 다른 기준으로 정렬하여 답이 틀릴 수 있습니다. 언어마다 사용하는 정렬이 순서를 유지(기준이 같은 경우 원래 순서에 있으면 앞에 있으면 앞에 오도록 해주는 경우)할 수 있고, 그렇지 않을 수 있기 때문입니다.
- C++의 경우 복합 타입 배열(예: 구조체)이면 값 배열 대신에 포인터 배열을 사용하는 것이 효과적입니다. 포인터 배열이면 아무리 무거운 복합 타입도 주소만 swap하면 됩니다.

7. 언어의 라이브러리를 이용할 때

- 주어진 문제에 대한 알고리즘을 구현하기 위해 입력 데이터를 문제 해결에 적합한 자료구조로 표현해야 할 수 있습니다. 당연히 이때 필요한 자료구조가 표준 라이브러리에서 제공하면 직접 구현하는 것보다 이를 사용하는 것이 강건성, 코드 가독성과 간결성, 개발 속도 측면에서 모두 유리합니다.
- 하지만 라이브러리에 있는 각종 클래스나 함수를 사용할 때 그것의 시간 복잡도를 잘 이해하고 사용해야 합니다.
  - 예1) C++의 **std::vector**나 자바의 **ArrayList**는 내부적으로 동적 배열 기법을 사용합니다. 따라서 중간 노드의 삭제의 시간 복잡도는 $O(n)$입니다. 또 보통 초기용량이 0인 상태에서 출발하여 용량이 부족하면 2배씩 용량을 확장하는 방식을 사용하고 있습니다. 따라서 저장해야 할 데이터의 개수를 사전에 알고 있으면 필요한만큼 확보하여 사용하는 것이 훨씬 효과적입니다.
  - 예2) 집합이나 맵 자료구조를 활용해야 할 때가 있습니다. 집합이나 맵 자료구조는 보통 내부적으로 해싱이나 균형이진트리를 이용합니다. 예를 들어 C++에서 **std::set**은 균형이진트리를 사용하는 구현이고, **std::unordered_set**은 해싱을 이용한 구현입니다. 균형이진트리로 구현된 집합이나 맵 자료구조의 각종 연산의 시간복잡도가 $O(\log n)$이고, 해싱을 이용하는 버전의 각종 연산의 시간복잡도는 $O(1)$입니다. 그런데 해싱은 데이터를 해시하는 비용이 항상 소요되며, 충분한 용량을 사전에 확보하여 사용하지 않으면 충돌이 많이 발생할 수 있고, 용량을 확장하는 과정에서 비용이 많이 소요될 수 있습니다.

이 문서에 오류가 있거나 추가되면 학습하는 분들께 도움이 될 수 있는 내용이 있으면 *sangjin@koreatech.ac.kr*로 연락주세요.