Algorithm

동양미래대학교 강은영

문제) 1이 될 때까지

- 어떠한 수 N이 1이 될 때까지 다음의 2 과정중 하나를 반복적으로 선택하여 수행하려고 합니다. 단, 두번째 연산은 N이 K로 나누어 떨어질 때만 선택할 수 있습니다.
  1. N에서 1을 뺍니다.
  2. N을 K로 나눕니다.
- 예를 들어 N이 17, K가 4라고 가정합시다. 이때 1번 과정을 한 번 수행하면 N은 16이 됩니다. 이후에 2번 과정을 두 번 수행하면 N은 1이 됩니다. 결과적으로 이 경우 전체 과정을 실행한 횟수는 3이 됩니다. 이는 N을 1로 만드는 최소 횟수입니다.
- N과 K가 주어질 때 N이 1이 될 때까지 1번 혹은 2번의 과정을 수행해야 하는 최소 횟수를 구하는 프로그램을 작성하시오.

문제에 대한 이해 – 그리디 알고리즘

1이 될 때까지

입력	첫째 줄에 N(1<=N<=100000) 과 K(2 <=K<=100000) 가 공백을 기준으로 하여 각각 자연수로 주어진다.	
출력	N이 1이 될 때까지 1번 혹은 2번의 과정을 수행해야 하는 횟수의 최소값을 출력한다.	
입출력 예	입력 25 5	출력 2

1이 될 때까지

Think-1

- 주어진 N에 대하여 최대한 많이 나누기를 수행하면 됩니다.
- N의 값을 줄일 때 2 이상의 수로 나누는 작업이 1을 빼는 작업보다 수를 훨씬 많이 줄일 수 있습니다.
- 예를 들어 N = 25, K = 3일 때는 다음과 같습니다.

단계	연산 과정	N의 값
0단계(초기 단계)		N = 25
1단계	N에서 1 빼기	N = 24
2단계	N을 K로 나누기	N = 8
3단계	N에서 1 빼기	N = 7
4단계	N에서 1 빼기	N = 6
5단계	N을 K로 나누기	N = 2
6단계	N에서 1 빼기	N = 1

1이 될 때까지 (Java)

```java
import java.util.*;

public class Main {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        // N, K를 공백을 기준으로 구분하여 입력 받기
        int n = sc.nextInt();
        int k = sc.nextInt();
        int result = 0;

        while (true) {
            // N이 K로 나누어 떨어지는 수가 될 때까지 빼기
            int target = (n / k) * k;
            result += (n - target);
            n = target;
            // N이 K보다 작을 때 (더 이상 나눌 수 없을 때) 반복문 탈출
            if (n < k) break;
            // K로 나누기
            result += 1;
            n /= k;
        }
        // 마지막으로 남은 수에 대하여 1씩 빼기
        result += (n - 1);
        System.out.println(result);
    }
}
```

단계	연산 과정	N의 값
0단계(초기 단계)		N = 25
1단계	N에서 1 빼기	N = 24
2단계	N을 K로 나누기	N = 8
3단계	N에서 1 빼기	N = 7
4단계	N에서 1 빼기	N = 6
5단계	N을 K로 나누기	N = 2
6단계	N에서 1 빼기	N = 1

1이 될때까지 (C) – 예시1

```c
#include <stdio.h>
int n, k;
int result;
int main(void) {
    scanf("%d %d", &n, &k );
    while (1) {
        // N이 K로 나누어 떨어지는 수가 될 때까지 빼기
        int target = (n / k) * k;
        result += (n - target);
        n = target;
        // N이 K보다 작을 때 (더 이상 나눌 수 없을 때) 반복문 탈출
        if (n < k) break;
        // K로 나누기
        result++;
        n /= k;
    }
    // 마지막으로 남은 수에 대하여 1씩 빼기
    result += (n - 1);
    printf("%d", result);
    return 0;
}
```

1이 될때까지 (C) – 예시2

```
#include <stdio.h>
int n, k;
int result;
int main(void) {
    scanf("%d %d", &n, &k );
    while (1) {
        // N이 K로 나누어 떨어지는 수가 될 때까지 빼기
        if ((n % k) == 0 ) {
            n /= k;
        } else {
            n = n -1;
        }
        result++;
        if (n < k) break;
    }
    // 마지막으로 남은 수에 대하여 1씩 빼기
    result += (n - 1);
    printf("%d", result);
    return 0;
}
```

단계	연산 과정	N의 값
0단계(초기 단계)		N = 25
1단계	N에서 1 빼기	N = 24
2단계	N을 K로 나누기	N = 8
3단계	N에서 1 빼기	N = 7
4단계	N에서 1 빼기	N = 6
5단계	N을 K로 나누기	N = 2
6단계	N에서 1 빼기	N = 1

1이 될때까지 (Python)

```python
# N, K을 공백을 기준으로 구분하여 입력 받기
n, k = map(int, input().split())

result = 0

while True:
    # N이 K로 나누어 떨어지는 수가 될 때까지 빼기
    target = (n // k) * k
    result += (n - target)
    n = target
    # N이 K보다 작을 때 (더 이상 나눌 수 없을 때) 반복문 탈출
    if n < k:
        break
    # K로 나누기
    result += 1
    n //= k

# 마지막으로 남은 수에 대하여 1씩 빼기
result += (n - 1)
print(result)
```

1이 될 때까지(C++)

```cpp
#include <bits/stdc++.h>

using namespace std;

int n, k;
int result;

int main(void) {
    cin >> n >> k;
    while (true) {
        // N이 K로 나누어 떨어지는 수가 될 때까지 빼기
        int target = (n / k) * k;
        result += (n - target);
        n = target;
        // N이 K보다 작을 때 (더 이상 나눌 수 없을 때) 반복문 탈출
        if (n < k) break;
        // K로 나누기
        result++;
        n /= k;
    }
    // 마지막으로 남은 수에 대하여 1씩 빼기
    result += (n - 1);
    cout << result << '\n';
}
```

1이 될 때까지 – 정당성 분석

- 가능하면 최대한 많이 나누는 작업이 최적의 해를 항상 보장할 수 있을까요?
- N이 아무리 큰 수여도, K로 계속 나눈다면 기하급수적으로 빠르게 줄일 수 있습니다.
- 다시 말해 K가 2 이상이기만 하면, K로 나누는 것이 1을 빼는 것보다 훨씬 빠릅니다.
- 또한 N은 항상 1에 도달하게 됩니다. (최적의 해 성립)

1이 될 때까지

- 주어진 $\boldsymbol{n}$에 대해서 소스코드의 시간 복잡도는 $\boldsymbol{O(logn)}$입니다.
- 이 알고리즘의 시간 복잡도는 k로 나누므로

문제) 곱하기 혹은 더하기

- 각 자리가 숫자(0부터 9)로만 이루어진 문자열 S가 주어졌을 때, 왼쪽부터 오른쪽으로 하나씩 모든 숫자를 확인하며 숫자 사이에 ‘×’ 혹은 ‘+’ 연산자를 넣어 결과적으로 만들어질 수 있는 가장 큰 수를 구하는 프로그램을 작성하세요. 단, +보다 ×를 먼저 계산하는 일반적인 방식과는 달리, 모든 연산은 왼쪽에서부터 순서대로 이루어진다고 가정합니다.
- 예를 들어 02984라는 문자열로 만들 수 있는 가장 큰 수는 ((((0 + 2) × 9) × 8) × 4) = 576입니다. 또한 만들어질 수 있는 가장 큰 수는 항상 20억 이하의 정수가 되도록 입력이 주어집니다.

문제에 대한 이해 – 그리디 알고리즘

곱하기 혹은 더하기

입력	첫째 줄에 여러 개의 숫자로 구성된 하나의 문자열 S가 주어집니다. (1<=S의 길이<=20)	
출력	만들어질 수 있는 가장 큰 수를 출력한다.	
입출력 예	입력 02984	출력 576
	576	210

곱하기 혹은 더하기

Think-1

- 대부분의 경우 '+'보다는 '×'가 더 값을 크게 만듭니다.
  - 예를 들어 5 + 6 = 11이고, 5 × 6 = 30입니다.
- 다만 두 수 중에서 하나라도 '0' 혹은 '1'인 경우, 곱하기보다는 더하기를 수행하는 것이 효율적입니다.
- 따라서 두 수에 대하여 연산을 수행할 때, 두 수 중에서 하나라도 1 이하인 경우에는 더하며, 두 수가 모두 2 이상인 경우에는 곱하면 정답입니다.

곱하기 혹은 더하기 (Java)

```java
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String str = sc.next();
        // 첫 번째 문자를 숫자로 변경한 값을 대입
        long result = str.charAt(0) - '0';
        for (int i = 1; i < str.length(); i++) {
            // 두 수 중에서 하나라도 '0' 혹은 '1'인 경우, 곱
하기보다는 더하기 수행
            int num = str.charAt(i) - '0';
            if (num <= 1 || result <= 1) {
                result += num;
            }
            else {
                result *= num;
            }
        }
        System.out.println(result);
    }
}
```

곱하기 혹은 더하기(C)

```c
#include <stdio.h>
#include <string.h>

int main(void) {

    char str[20];
    scanf("%s", str);
    //첫번째 문자를 숫자로 변경한 값을 대입
    int result = str[0] - '0';
    for (int i = 1; i< strlen(str); i++) {
        //두 수 중에서 하나라도 '0' 혹은 '1'인 경우, 곱하기보다는 더하기 수행
        int num = str[i] - '0';
        if (num <= 1 || result <= 1) result += num;
        else result *= num;
    }
    printf("%d ", result );
}
```

곱하기 혹은 더하기 (Python)

```python
data = input()

# 첫 번째 문자를 숫자로 변경하여 대입
result = int(data[0])

for i in range(1, len(data)):
    # 두 수 중에서 하나라도 '0' 혹은 '1'인 경우, 곱하기보다는 더하기 수행
    num = int(data[i])
    if num <= 1 or result <= 1:
        result += num
    else:
        result *= num

print(result)
```

곱하기 혹은 더하기 (C++)

```cpp
#include <bits/stdc++.h>
using namespace std;

string str;

int main(void) {
    cin >> str;

    // 첫 번째 문자를 숫자로 변경한 값을 대입
    long long result = str[0] - '0';

    for (int i = 1; i < str.size(); i++) {
        // 두 수 중에서 하나라도 '0' 혹은 '1'인 경우, 곱하기보다는 더하기 수행
        int num = str[i] - '0';
        if (num <= 1 || result <= 1) result += num;
        else result *= num;
    }

    cout << result << '\n';
}
```

곱하기 혹은 더하기

- 주어진 $\boldsymbol{n}$에 대해서 소스코드의 시간 복잡도는 $\boldsymbol{O(n)}$입니다.

문제) ATM

출처) https://www.acmicpc.net/problem/11399

인하은행에는 ATM이 1대밖에 없다. 지금 이 ATM앞에 N명의 사람들이 줄을 서있다. 사람은 1번부터 N번까지 번호가 매겨져 있으며, i번 사람이 돈을 인출하는데 걸리는 시간은 P_i분이다.

사람들이 줄을 서는 순서에 따라서, 돈을 인출하는데 필요한 시간의 합이 달라지게 된다. 예를 들어, 총 5명이 있고, $P_1 = 3$, $P_2 = 1$, $P_3 = 4$, $P_4 = 3$, $P_5 = 2$ 인 경우를 생각해보자. [1, 2, 3, 4, 5] 순서로 줄을 선다면, 1번 사람은 3분만에 돈을 뽑을 수 있다. 2번 사람은 1번 사람이 돈을 뽑을 때 까지 기다려야 하기 때문에, 3+1 = 4분이 걸리게 된다. 3번 사람은 1번, 2번 사람이 돈을 뽑을 때까지 기다려야 하기 때문에, 총 3+1+4 = 8분이 필요하게 된다. 4번 사람은 3+1+4+3 = 11분, 5번 사람은 3+1+4+3+2 = 13분이 걸리게 된다. 이 경우에 각 사람이 돈을 인출하는데 필요한 시간의 합은 3+4+8+11+13 = 39분이 된다.

줄을 [2, 5, 1, 4, 3] 순서로 줄을 서면, 2번 사람은 1분만에, 5번 사람은 1+2 = 3분, 1번 사람은 1+2+3 = 6분, 4번 사람은 1+2+3+3 = 9분, 3번 사람은 1+2+3+3+4 = 13분이 걸리게 된다. 각 사람이 돈을 인출하는데 필요한 시간의 합은 1+3+6+9+13 = 32분이다. 이 방법보다 더 필요한 시간의 합을 최소로 만들 수는 없다.

줄을 서 있는 사람의 수 N과 각 사람이 돈을 인출하는데 걸리는 시간 P_i가 주어졌을 때, 각 사람이 돈을 인출하는데 필요한 시간의 합의 최솟값을 구하는 프로그램을 작성하시오.

문제에 대한 이해 – 그리디 알고리즘

ATM

입력	첫째 줄에 사람의 수 N(1 ≤ N ≤ 1,000)이 주어진다. 둘째 줄에는 각 사람이 돈을 인출하는데 걸리는 시간 P_i가 주어진다. (1 ≤ P_i ≤ 1,000)	
출력	첫째 줄에 각 사람이 돈을 인출하는데 필요한 시간의 합의 최솟값을 출력한다.	
입출력 예	입력 5 3 1 4 3 2	출력 32

ATM

Think-1

- 인하은행에는 ATM이 1대밖에 없다.
- 지금 이 ATM앞에 N명의 사람들이 줄을 서있다.
- 사람은 1번부터 N번까지 번호가 매겨져 있으며, i번 사람이 돈을 인출하는데 걸리는 시간은 P_i분이다.

ATM

Think-1

- 사람들이 줄을 서는 순서에 따라 돈을 인출하는데 필요한 시간의 합이 달라지게 된다.
- 예를 들어, 총 5명이 있고 P1=3, P2=1, P3=4, P4=3, P5=2
- [1,2,3,4,5] 순서로 줄을 선다면, 1번 사람은 3분
- 2번 사람은 3+1=4
- 3번 사람은 3+1+4=8분
- 4번 사람은 3+1+4+3=11분
- 5번 사람은 3+1+4+3+2=13분
- 각 사람이 돈을 인출하는데 필요한 시간의 합은
- 3 + (3+1) + (3+1+4) + (3+1+4+3) + (3+1+4+3+2) = 39분
- 3 + 4 + 8 + 11 + 13

ATM

Think-1

- 줄을 서 있는 사람의 수 N과 각 사람이 돈을 인출하는데 걸리는 시간 P_i가 주어졌을 때, 각 사람이 돈을 인출하는데 필요한 시간의 합의 최소값을 구하는 문제
- 기다리는 시간이 짧은 사람부터 ATM을 인출하는 것이 좋다.
- P_i 를 오름차순으로 정렬
- 1 + (1+2) + (1+2+3) + (1+2+3+3) + (1+2+3+3+4) = 32분
- 1 + 3 + 6 + 9 + 13

ATM

3	1	4	3	2
1	3	4	3	2
1	3	4	3	2
1	3	3	4	2
1	2	3	3	4

삽입정렬 $O(n^2)$

A[]

0	1	2	3	4
1	2	3	3	4

s[]

0	1	2	3	4
1	3	6	9	13

1+3+6+9+13=32

소스코드 관리

- 문제1(1이 될 때까지), 문제2(곱하기 혹은 더하기)를 풀어봅니다.

- ATM(https://www.acmicpc.net/problem/11399)를 풀고 소스코드를 JAVA, C, C++, Python 중 2가지를 택하여 작성하고
- 과제(LMS(eclass.dongyang.ac.kr)에 제출해 주세요.