

# 주요 알고리즘

## (1) 그리디 알고리즘

양유진

# Contents

01 그리디 알고리즘 정의

02 예제 문제 (1)

03 예제 문제 (2)

04 예제 문제 (3)



# 그리디 알고리즘(Greedy Algorithm)이란?

- ‘탐욕적(greedy)’이라는 표현은 현재 상황에서 지금 당장 좋은 것만 고르는 방법을 의미한다.
- 단순하지만 강력한 문제 해결 방법으로, 어떠한 문제가 있을 때 현재의 선택이 나중에 미칠 영향은 고려하지 않고 단순 무식하게 문제를 푸는 알고리즘이다.
- “탐욕법”이라고 번역되어 쓰이기도 한다.
- 보통 이런 유형의 문제들은 사전에 외우고 있지 않아도 풀 수 있을 가능성이 높은 문제 유형이다.

## 예제 문제 (1) 거스름돈

손님에게 거슬러 줘야 할 돈이 N원일 때 거슬러 줘야 할 **동전의 최소 개수**를 구하는 프로그램을 작성하여라.

(단, N은 항상 10의 배수이다.)

[가정]카운터에는 거스름돈으로 사용할 동전(500원, 100원, 50원, 10원)이 무한히 존재한다.

### POINT

“가장 **큰** 화폐 단위부터 돈을 거슬러 주는 것”

# 예제 문제 (1) 거스름돈 (python)

#거스름돈

```
n = int(input("거스름돈을 입력하세요>>"))
```

```
N = n
```

#동전 개수

```
count = 0
```

```
list = [500, 100, 50, 10]
```

```
for coin in list:
```

```
    count += N//coin
```

```
    N %= coin
```

```
print("거스름돈 {}원의 동전개수는 {}개이다.".format(n, count))
```

```
Console Shell
거스름돈>>1260
거스름돈 1260원의 동전개수는 6개이다.
>
```

# 예제 문제 (1) 거스름돈 (c 언어)

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

```
int main(void) {  
    int N, n, count = 0;  
    int money[] = {500, 100, 50, 10};  
  
    while(N<1){  
        printf("거스름돈을 입력하세요>>");  
        scanf("%d", &N);  
    }  
  
    n=N;  
    for(int i=0; i<sizeof(money)/sizeof(money[0]); i++){  
        if(n<money[i]){  
            i++;  
            continue;  
        }  
        count += n/money[i];  
        n %= money[i];  
    }  
    printf("거스름돈 %d원의 동전개수는 %d개이다. \n", N, count);  
    return 0;  
}
```

Console

Shell

```
> clang-7 -pthread -lm -o main main.c  
> ./main  
거스름돈을 입력하세요>>-1  
거스름돈을 입력하세요>>0  
거스름돈을 입력하세요>>760  
거스름돈 760원의 동전개수는 5개이다.  
> □
```

## 예제 문제 (2) 큰 수의 법칙

큰 수의 법칙은 다양한 수로 이루어진 배열이 있을 때 주어진 수들을 M번 더하여 가장 큰 수를 만드는 법칙이다.

(단, 배열의 특정 인덱스에 해당하는 수가 연속해서 K번을 초과하여 더해질 수 없다.)

배열의 크기 N, 숫자가 더해지는 횟수 M, 그리고 K가 주어질 때 **큰 수의 법칙에 따른 결과**를 구하는 프로그램을 작성하여라.

[가정] 이때, 서로 다른 인덱스에 해당하는 수가 같은 경우에는 서로 다른 것으로 간주한다.

예를 들어, 순서대로 2, 4, 5, 4, 6으로 이루어진 배열이 있을 때, M이 8이고 K가 3이라고 가정하면  $6+6+6+5+6+6+6+5=46$ 으로 결과는 46이다.

### POINT

“가장 큰 수를 K번 더하고  
두 번째로 큰 수를 한 번 더하는 연산 반복”

## 예제 문제 (2) 큰 수의 법칙 (python) – 시간초과판정

```
#N, M, K를 공백으로 구분하여 입력받기
print("n, m, k 순서대로 입력하십시오")
n, m, k = map(int, input().split())

#N개의 수를 공백으로 구분하여 입력받기
print("공백을 구분하여 자연수 {}개 입력하십시오".format(n))
data = list(map(int, input().split()))

#입력받은 수 오름차순으로 정렬하기
data.sort()
first = data[n-1] #가장 큰 수
second = data[n-2] #두번째로 큰 수
result = 0
while True:
    for i in range(k):
        if m == 0:
            break
        result += first
        m -= 1
    if m == 0:
        break
    result += second
    m -= 1
print("N={}, M={}, K={}일 때, 결과는 {}이다.".format(n, m, k, result))
```

### map() 함수

- 리스트의 요소를 지정된 함수로 처리해주는 함수
- 복수의 데이터를 일괄적으로 형 변환하여 입력 받을 때 사용됨.

### split() 함수

- 입력 받은 값을 공백을 기준으로 분리하여 변수에 차례대로 저장해줌.

### sort() 함수

- 리스트를 오름차순으로 정렬해줌.



## 예제 문제 (2) 큰 수의 법칙 (python)

```
#N, M, K를 공백으로 구분하여 입력받기
print("n, m, k 순서대로 입력하십시오")
n, m, k = map(int, input().split())
```

```
#N개의 수를 공백으로 구분하여 입력받기
print("공백을 구분하여 자연수 {}개 입력하십시오".format(n))
data = list(map(int, input().split()))
```

```
#입력받은 수 오름차순으로 정렬하기
data.sort()
first = data[n-1] #가장 큰 수
second = data[n-2] #두번째로 큰 수
result = 0
```

```
#가장 큰 수가 더해지는 횟수 계산
count = int(m/(k+1)) * k
count += m % (k+1)

result += (count) * first
result += (m-count) * second
```

```
print("N={}, M={}, K={}일 때, 결과는 {}이다.".format(n, m, k, result))
```

$$\underbrace{6 + 6 + 6 + 5}_{(1)} + \underbrace{6 + 6 + 6 + 5}_{(2)} + \boxed{6} = 52$$

$$\frac{m}{\boxed{k+1}} \times k + \underline{m \% (k+1)}$$

두 번째로 큰 수

count는 가장 큰 수가 더해지는 횟수

## 예제 문제 (2) 큰 수의 법칙 (python)

```
#N, M, K를 공백으로 구분하여 입력받기
print("n, m, k 순서대로 입력하십시오")
n, m, k = map(int, input().split())
```

```
#N개의 수를 공백으로 구분하여 입력받기
print("공백을 구분하여 자연수 {}개 입력하십시오".format(n))
data = list(map(int, input().split()))
```

```
#입력받은 수 오름차순으로 정렬하기
data.sort()
first = data[n-1] #가장 큰 수
second = data[n-2] #두번째로 큰 수
result = 0
```

```
#가장 큰 수가 더해지는 횟수 계산
count = int(m/(k+1)) * k
count += m % (k+1)
```

```
result += (count) * first
result += (m-count) * second
```

```
print("N={}, M={}, K={}일 때, 결과는 {}이다.".format(n, m, k, result))
```

Console

Shell

```
n, m, k 순서대로 입력하십시오
5 8 3
공백을 구분하여 자연수 5개 입력하십시오
2 4 5 4 6
N=5, M=8, K=3일 때, 결과는 46이다.
>
```

Console

Shell

```
n, m, k 순서대로 입력하십시오
5 7 2
공백을 구분하여 자연수 5개 입력하십시오
3 4 3 4 3
N=5, M=7, K=2일 때, 결과는 28이다.
>
```

$4 + 4 + 4 + 4 + 4 + 4 + 4 = 28$

## 예제 문제 (2) 큰 수의 법칙 (c 언어)

```
#include <stdio.h>
#include <stdlib.h>

int compare(const void *a , const void *b){
    if( *(int*)a > *(int*)b )
        return 1;
    else if( *(int*)a < *(int*)b )
        return -1;
    else
        return 0;
}
```

※내림차순으로 사용할 경우 등호를 바꾸어 주면 됨

### qsort() 함수

- 숫자와 문자열을 정렬해주는 함수.

void qsort(배열의 포인터, 배열 크기, 원소 하나의 크기,  
비교 수행 함수 포인터)

```
int main(void) {
    int n, m, k;
    int first, second, result, count=0;

    printf("n, m, k 순서대로 입력하시오>>");
    scanf("%d %d %d", &n, &m, &k);
    int seq[n];
    printf("공백을 구분하여 자연수를 %d개 입력하시오>>", n);
    for(int i=0; i<n; i++)
    {
        scanf("%d", &seq[i]);
    }

    qsort(seq, n, sizeof(seq[0]), compare);
    first = seq[n-1]; second = seq[n-2];

    count += ( (m/(k+1)) * k + m%(k+1) );
    result += ( (count) * first + (m-count) * second );

    printf("N=%d, M=%d, K=%d일 때, 결과는 %d이다.\n", n, m, k, result);
    return 0;
}
```

## 예제 문제 (2) 큰 수의 법칙 (c 언어)

```
#include <stdio.h>
#include <stdlib.h>
```

```
int compare(const void *a, const void *b) {
    if( *(int*)a > *(int*)b)
        return 1;
    else if( *(int*)a < *(int*)b)
        return -1;
    else
        return 0;
}
```

```
int main(void) {
```

Console

Shell

```
> clang-7 -pthread -lm -o main main.c
> ./main
n, m, k 순서대로 입력하시오>>5 8 3
공백을 구분하여 자연수를 5개 입력하시오>>2 4 5 4 6
N=5, M=8, K=3일 때, 결과는 46이다.
> 
```

Console

Shell

```
> clang-7 -pthread -lm -o main main.c
> ./main
n, m, k 순서대로 입력하시오>>5 7 2
공백을 구분하여 자연수를 5개 입력하시오>>3 4 3 4 3
N=5, M=7, K=2일 때, 결과는 28이다.
> 
```

```
count=0;
```

```
하시오>>");
```

```
;
```

```
를 %d개 입력하시오>>", n);
```

```
{
```

```
compare);
```

```
{n-2];
```

```
(k+1) );
```

```
(m-count) * second );
```

```
printf("N=%d, M=%d, K=%d일 때, 결과는 %d이다.\n", n, m, k, result);
return 0;
```

```
}
```

## 예제 문제 (3) 1이 될 때까지

어떠한 수  $N$ 이 1이 될 때까지 다음 두 과정 중 하나를 반복적으로 선택하여 수행하려고 한다.

1)  $N$ 에서 1을 뺀다.

2)  $N$ 을  $K$ 로 나눈다.

(단, 두 번째 연산은  $N$ 이  $K$ 로 나누어 떨어질 때만 선택할 수 있다.)

$N$ 과  $K$ 가 주어질 때  $N$ 이 1이 될 때까지 1번 혹은 2번의 과정을 수행해야 하는 **최소 횟수**를 구하는 프로그램을 작성하시오.




예를 들어  $N$ 이 17,  $K$ 가 4일 때, 1번 연산 수행해서 16으로 만든 후 2번 연산을 2번 수행하면 전체 과정을 실행한 횟수가 3이 된다. 이는  $N$ 을 1로 만드는 최소 횟수이다.


### POINT

“1번 과정보다 **2번 과정**을 최대한 많이 수행하기”

## 예제 문제 (3) 1이 될 때까지 (python)

```
#N, K를 공백으로 구분하여 입력받기
print("n, k 순서대로 입력하십시오")
n, k = map(int, input().split())
N=n
result = 0
```

```
while n >= k:  나눌 수 있는 경우
    while n%k != 0:  나머지 값이 존재하는 경우 (2번 수행할 수 없는 환경)
        n -= 1
        result += 1
    n //= k  2번 수행
    result += 1
```

```
while n > 1:  더 나눌 수 없는 경우 (1번 수행)
    n -= 1
    result += 1
```

```
print("N={}, K={}일 때, 결과는 {}이다.".format(N, k, result))
```

# 예제 문제 (3) 1이 될 때까지 (python) 최적화 코드

```
#N, K를 공백으로 구분하여 입력받기
print("n, k 순서대로 입력하십시오")
n, k = map(int, input().split())
N=n
result = 0
```

Console

Shell

```
n, k 순서대로 입력하십시오
25 5
N=25, K=5일 때, 결과는 2이다.
```

```
while True:
    target = (n//k) * k
    result += (n - target)
    n = target
    if n < k:
        break

    result += 1
    n //= k
```

— N이 K의 배수가 되도록 한 번에 빼줌. (1번 수행)

— 2번 수행

```
result += (n - 1)
print("N={}, K={}일 때, 결과는 {}이다.".format(N, k, result))
```

— 남은 수에 대하여 1씩 빼기

## 예제 문제 (3) 1이 될 때까지 (c 언어)

```
#include <stdio.h>

int main(void) {
    int n, k, target;
    long result = 0;

    printf("n, k 순서대로 입력하시오>>");
    scanf("%d %d", &n, &k);
    int N = n;

    while(1){
        target = (n/k) * k;
        result += (n-target);
        n = target;

        if (n<k){break;}
        result += 1;
        n /= k;
    }

    result += (n-1);
    printf("N=%d, K=%d일 때, 결과는 %ld이다.\n", N, k, result);
    return 0;
}
```

Console

Shell

```
> clang-7 -pthread -lm -o main main.c
> ./main
n, k 순서대로 입력하시오>>25 5
N=25, K=5일 때, 결과는 20이다.
> □
```



감사합니다