재귀(Recursion, 순환)

1

재귀 (recursion, 순환)

- 문제의 해를 부분문제(subproblem: 작은 크기의 입력에 대한 동일한 문제)의 해를 이용하여 해결하는 방법
- recursive definition

예 1: 0이상의 정수 n에 대한 n! 정의

n! = 1	if n = 0	// base case
n! = n * (n – 1)!	if n > 0	// recursive case

2

1. n!을 구하는 recursive function (파이썬)

- recursive function: 자기 자신을 호출하는 함수

```
# python
def fact(n):                    → n!
# Precondition(사전조건): n >= 0.
    if n == 0:    #  base case
        return  1
    else:       # recursive case(general case)
        return  n * fact(n - 1)
```

n > 0
n × (n-1)!

수행시간 분석

n! 계산에서 기본연산: 두 정수 곱셈
T(n): fact(n)을 수행할 때 기본연산 수
T(n) = 0 n = 0이면
= 1 + T(n-1) n > 0이면
위의 점화식으로부터 T(n)을 구한다

T(n) = 1 + T(n-1)
= 1 + [1 + T(n-2)]
= 2 + T(n-2)
...
= k + T(n-k)

n-k = 0이면, k = n이다.
T(n) = n + T(1) = n
O(n)

3

fact(n)의 수행과정

```
def fact(n):
    if n == 0:
        return  1
    else:
        return  n * fact(n - 1)

result = fact(4)
```

함수호출 할 때마다 내부적인 일

n
n
n

n에대한 메모리할당함.

stack memory 할당

Recursion depth(level): 4

fact(4) 수행과정

(1) 함수 fact 호출

```
fact(n) : n = 4
   return n*fact(n-1)
```

함수호출이 일어나서 return 못하게됨.

(10) return 24(=4*6)

(2) 함수 fact 호출

```
fact(n) : n = 3
   return n*fact(n-1)
```

(9) return 6(=3*2)

(3) 함수 fact 호출

```
fact(n) : n = 2
   return n*fact(n-1)
```

(8) return 2(=2*1)

(4) 함수 fact 호출

```
fact(n) : n = 1
   return n*fact(n-1)
```

(7) return 1(=1*1)

(5) 함수 fact 호출

```
fact(n) : n = 0
   return 1
```

(6) return 1

4

재귀 수행과제 예 시험문제예시

```
def f(n):
    if n == 0:
        print(n)
    else:
        print(n)
        f(n-1)
        print(n)
```

다음과 같이 주어지고
출력은 어떻게 될까?

f(5) 5 4 3 2 1

5

2. gcd (최대공약수, greatest common divisor)

- 양의 정수 a,b의 최대공약수 gcd(a,b)의 재귀적 정의

gcd(a,b) = b	a가 b로 나누어지면	// base case
= gcd(b,a%b)	그렇지 않으면	// recursive case

a와 b의 최대공약수 = b와 a%b의 최대공약수.

```
# python
def gcd(a, b):
#   if (a%b == 0):
#       return b
    if b == 0:
        return a
    else:
        return gcd(b,a%b)
```

a,b의 양의정수
(40,10)

10 0
gcd를한번더 호출하게됨.

gcd(120,50) — gcd(a,b)
gcd(b, a%b)
gcd(50 , 20)
gcd(20, 10)
gcd(20, 10)
gcd(b, a%b)
(10 , 0)
gcd(a,b)
10 0

6

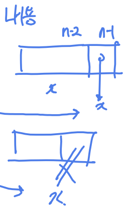
3. 리스트에서 x가 나타나는 횟수

```
# iteration
def count1(L, x):
    cnt = 0
    for item in L:
        if item == x:
            cnt += 1
    return cnt

# list method count
def count2(L, x):
    return L.count(x)
```

count 메소드 사용.

n개

```
# recursion
# 리스트 L[0]부터 L[n-1]까지 원소들 중
# x와 같은 원소의 개수를 구하는 함수
def count3(L, x, n):
    if n == 0:        ⇒ 리스트에 원소 없다는 내용
        return 0
    elif L[n-1] == x:
        return count3(L,x,n-1)+1
    else:
        return count3(L,x,n-1)

print(count(L, x, len(L)))
```



4. x^n 계산

- 0이상 정수 n에 대하여 x^n 계산

```
# python
def exp1(x, n):
    result = 1
    for i in range(1,n+1):
        result *= x;

    return result
```

range(n) 0 ~ n-1 해당
range(1,n+1) 1 ~ n 해당

x^n 계산

- 0이상 정수 n에 대하여 x^n

x^n 의 재귀적 정의 ($n \ge 0$):

$x^n = 1$ if $n = 0$
$= x * x^{n-1}$ if $n > 0$

```
def exp2(x,n):     ← x^n
# n은 0이상 정수
    if n == 0:
        return 1
    else:
        return x*exp2(x,n-1)
```


x^n 계산

x^n 의 재귀적 정의 ($n \ge 0$):

$x^n = 1$ if $n = 0$
$= (x^{n/2})^2$ if n is even 짝수
$= x * x^{n-1}$ if n is odd 홀수

$x^{1024} = (x^{512})^2 = (x^{256})^2 \cdots$ → 2씩 곱해짐.

$O(\log_2 n)$

$2\log_2 n$

```
def exp3(x, n):# n은 0이상 정수     O(log2 n)
    global count
    if n == 0:
        return 1
    elif (n % 2 == 0):
        temp = exp3(x,n/2)     count += 1     T(n/2)
        return temp*temp
    else:
        count += 1
        return x*exp3(x,n-1)     T(n-1) → 짝수
```

$x^{100} = (x^{50})^2$

$x^{101} = x * x^{100}$
$= x * (x^{50})^2$

$T(\frac{n-1}{2})+1$
→ 곱하기 한번 더 하므로 $T(\frac{n-1}{2})+2$

print (exp3(2, 20)) → 1048576
print (count) → 6

print (exp3(2,21))
print (count) → 7

$T(n) = 0$ $n = 0$
$\le T(\frac{n}{2})+2$ $n > 0$

$T(65) \le T(32) + 2$
$\le T(16) + 2$
$+2$
$T(0) +2$

최대의 곱셈수 $2\log_2 n$

5. 이진탐색 (binary search) 생략

- 정렬되어 있는 리스트 A에서 item과 같은 원소의 위치를 찾아라.
- 탐색에서 다룰 예정

```
def binarySearch(A, item, left, right):
    if left <= right:
        mid = (left + right)//2
        if item == A[mid]:
            return mid
        elif item < A[mid]:
            return binarySearch(A, item, left, mid-1)
        else:
            return binarySearch(A, item, mid+1, right)
    else:
        return -1
```

binarySearch(A,item,0,len(A)-1)을 호출
수행시간: O(log n) where n = len(A)

6. 하노이 탑(Hanoi Tower) 문제

- 세 개의 막대기 1, 2, 3이 있고 서로 다른 크기의 n개의 원반들이 막대기 1에 크기순서(위에서부터 아래로 크기가 증가하는 순서)대로 놓여 있다. 다음 규칙을 지키면서 막대기 1에 있는 모든 원반들을 막대기 3으로 옮겨라.

  규칙 1: 한번에 막대기의 맨 위에 있는 한 장의 원반만을 다른 막대기 위로 옮길 수 있다.

  규칙 2: 큰 원반은 절대로 작은 원반 위에 놓여질 수 없다.

- 알고리즘

단계 1) 막대기 1의 가장 큰 원반(가장 아래에 있는 원반)을 제외한 나머지 n –1 개의 원반을 막대기 2로 옮긴다 (막대기 3을 이용).

단계 2) 막대기 1의 원반(가장 큰 원반)을 막대기 3으로 옮긴다.

단계 3) 막대기 2에 놓여 있는 n-1개의 원반을 막대기 3으로 옮긴다.

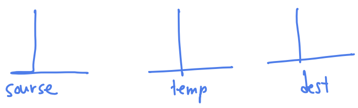
하노이 탑(Hanoi Tower) 문제

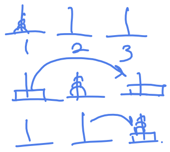
- 프로그램

base 막대기 목적지 1→2 옮길때 임시막대는 3이므로 temp 매개변수사용

```
def hanoiTower(n, source, dest, temp):
    if (n == 1):
        print( "Move a disk from peg %d to peg  %d" % (source, dest))
#       print( "Move a disk from peg {0} to peg  {1}".format(source,  dest))

    else:
        hanoiTower(n-1, source, temp, dest)
        print( "Move a disk from peg %d to peg  %d" % (source, dest))
        hanoiTower(n-1, temp, dest, source)
```

임시막대이 사용안하고 바로 옮길 때





→ 3개옮길때 7번

1에서 3으로 바로 1번

→ 3개옮길때 7번

15번

하노이탑 알고리즘 수행시간 분석

- **수행시간 분석**

T(n) : n장의 원반을 옮기는데 필요한 move 횟수

n = 1 인 경우 ⇒ $T(n) = 1$

n > 1 인 경우 ⇒ $T(n) = 2*T(n-1) + 1$

$T(n) = 2T(n-1) + 1 = 2[2T(n-2) + 1] + 1$

$= 2^2T(n-2) + 2 + 1 = 2^2[2T(n-3) + 1] + 2 + 1$

$= 2^3T(n-3) + 2^2 + 2 + 1$

...

$= 2^kT(n-k) + 2^{k-1} + \dots + 2^2 + 2 + 1 = 2^kT(n-k) + 2^k-1$

$n-k = 1$, 즉 $k = n-1$일때,

$= 2^{n-1}T(1) + 2^{n-1} -1$

$= 2^n - 1$

- 시간복잡도 :

$T(n) : O(2^n)$