04강

알고리즘과 자료구조

축소 정복, 분할 정복

서울과학기술대학교 신일훈 교수

한국방송통신대학교 프리임칼리지

학습목표

1. 축소정복, 분할정복의 개념을 이해한다.
2. 재귀함수의 개념을 이해한다.
3. 팩토리얼을 구하는 알고리즘을 설계할 수 있다.
4. 피보나치 수열을 구하는 알고리즘을 설계할 수 있다.
5. 제곱근을 구하는 알고리즘을 설계할 수 있다.

01

축소정복, 분할정복 개념

1. 축소정복 , 분할정복 개념

❑ 축소정복 (decrease & conquer) 전략 개념

- 원래 문제를 더 작은 문제로 축소해가며, 간단하게 풀 수 있는 작은 문제를 해결함으로써 원래 문제를 해결.
  - n에 대한 문제를 n-1에 대한 문제로 축소
    - 1~N의 합계 => 1~(N-1)의 합계로 축소 => 1~(N-2)의 합계로 축소 => … => 1~1의 합계로 축소
    - N! => (N-1)!로 축소 => (N-2)!로 축소 => … => 1! 로 축소
  - 솔루션이 존재하는 범위를 축소
    - 제곱근 구하기
    - 1~100 숫자 맞추기
  - 재귀(recursion) 또는 반복을 활용

1. 축소정복 , 분할정복 개념

❑ 분할정복 (divide & conquer) 전략 개념

- 원래 문제를 더 작은 부분 문제들로 분할하여, 부분 문제들을 해결함으로써 원래 문제를 해결.
  - 피보나치 수열
  - 병합(merge) 정렬
  - 재귀(recursion) 또는 반복을 활용

02

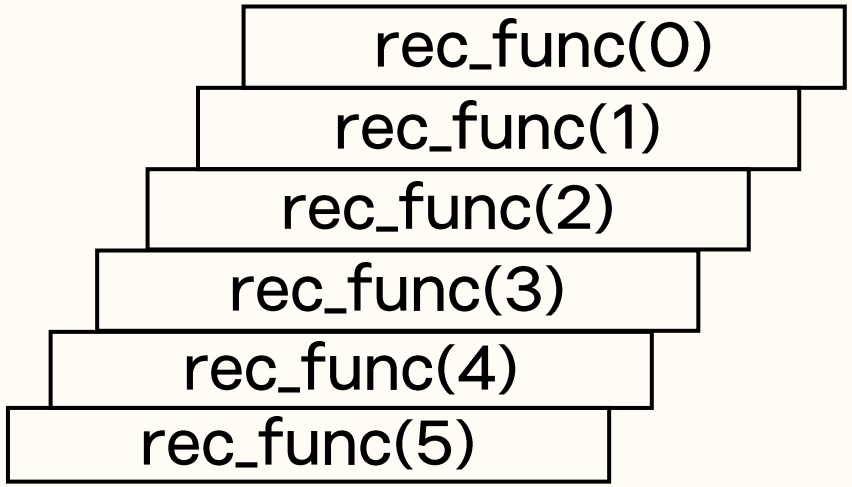
재귀

2. 재귀 개념

재귀함수

- 자기 자신을 호출하는 함수를 재귀 함수라고 함
- 재귀 함수의 예

```
def rec_func(call_count) :
    if (call_count == 0) :
        return
    print(call_count)
    call_count -= 1
    rec_func(call_count)
rec_func(5)
```



2. 재귀 개념

❑ 재귀를 활용한 축소 정복

- 원리
  - f(n)의 해를 f(n-1)을 이용해 구하라.
- 예시
  - 팩토리얼 구하기
    - f(n) = n * f(n-1) and f(1) = 1
  - 1~n까지 합계
    - f(n) = n + f(n-1) and f(1) = 1

2. 재귀 개념

재귀를 활용한 분할 정복

- 예시
  - 피보나치 수열
    - f(n) = f(n-1) + f(n-2)
    - f(0) = 0, f(1) = 1

2. 재귀 개념

장점

- 직관적이고 이해하기 쉬운 문제 해결 기법

단점

- 과도한 함수 호출로 인한 stack overflow 가능성
- 과도한 함수 호출로 인한 낮은 성능 (반복에 비해)

03

팩토리얼 계산

3. 팩토리얼 계산

- N! (팩토리얼)을 계산하시오.

아이디어

N! = 1 * 2 * … (N-1) * N

N에 대한 문제를 어떻게 N-1의 문제로 축소할까?

3. 팩토리얼 계산

- N! (팩토리얼)을 계산하시오.

아이디어

$N! = 1 * 2 * \cdots (N-1) * N$

$factorial(N) = N * factorial(N-1)$

3. 팩토리얼 계산

- N! (팩토리얼)을 계산하시오.

잘못된 의사코드

```
def my_fact(n) :
    return (n * my_fact(n-1))
```

3. 팩토리얼 계산

- N! (팩토리얼)을 계산하시오.

잘못된 의사코드

```
def my_fact(n) :
    return (n * my_fact(n-1))
```

** 재귀함수가 종료될 수 있도록, base case를 잊지 말자 **

3. 팩토리얼 계산

- N! (팩토리얼)을 계산하시오.

의사코드

```
def my_fact(n) :
    if (n == 1) :
        return 1
    return (n * my_fact(n-1))
```

3. 팩토리얼 계산

- N! (팩토리얼)을 계산하시오.

의사코드

```
def my_fact(n) :
    if (n == 1) :
        return 1
    return (n * my_fact(n-1))
```

최악 시간복잡도 ?

3. 팩토리얼 계산

- N! (팩토리얼)을 계산하시오.

의사코드

```
def my_fact(n) :
    if (n == 1) :
        return 1
    return (n * my_fact(n-1))
```

최악 시간복잡도 : O(N)

3. 팩토리얼 계산

- N! (팩토리얼)을 계산하시오.

파이썬 코드

```
def my_fact(num) :
    if (num == 1) :
        return 1
    return (num * my_fact(num-1))
print(my_fact(5))
```

3. 팩토리얼 계산

- N! (팩토리얼)을 계산하시오.

파이썬 코드 실행

```
def my_fact(num) :
    if (num == 1) :
        return 1
    return (num * my_fact(num-1))
print(my_fact(5))
```

```
In [32]: runfile('D:/data/lecture/
파이썬으로배우는자료구조와알고리즘/code/알고리즘/
untitled1.py', wdir='D:/data/lecture/
파이썬으로배우는자료구조와알고리즘/code/알고리즘')
120
```

04

피보나치 수열

4. 피보나치 수열

- 피보나치 수열 fibo(N)을 구하시오.

아이디어

$$fibo(n) = fibo(n-1) + fibo(n-2)$$

$$fibo(0) = 0$$

$$fibo(1) = 1$$

4. 피보나치 수열

- 피보나치 수열 fibo(N)을 구하시오.

의사코드

```
def fibo(n) :
    if (n == 0 or n == 1) :
        return n
    return (fibo(n-1) + fibo(n-2))
```

4. 피보나치 수열

- 피보나치 수열 fibo(N)을 구하시오.

의사코드

```
def fibo(n) :
    if (n == 0 or n == 1) :
        return n
    return (fibo(n-1) + fibo(n-2))
```

최악 시간복잡도 ?

4. 피보나치 수열

- 피보나치 수열 fibo(N)을 구하시오.

의사코드

```
def fibo(n) :
    if (n == 0 or n == 1) :
        return n
    return (fibo(n-1) + fibo(n-2))
```

최악 시간복잡도 : $O(2^N)$

4. 피보나치 수열

- 피보나치 수열 fibo(N)을 구하시오.

파이썬 코드

```
def fibo(n) :
    if (n == 0 or n == 1) :
        return n
    return (fibo(n-1) + fibo(n-2))
print(fibo(10))
```

4. 피보나치 수열

- 피보나치 수열 fibo(N)을 구하시오.

파이썬 코드 실행

```
def fibo(n):
    if (n == 0 or n == 1):
        return n
    return (fibo(n-1) + fibo(n-2))
print(fibo(10))
```

```
In [34]: runfile('D:/data/lecture/
파이썬으로배우는자료구조와알고리즘/code/알고리즘/
untitled1.py', wdir='D:/data/lecture/
파이썬으로배우는자료구조와알고리즘/code/알고리즘')
55
```

05

제곱근 구하기

5. 제곱근 구하기

- 양의 정수 N의 제곱근을 구하시오.

아이디어

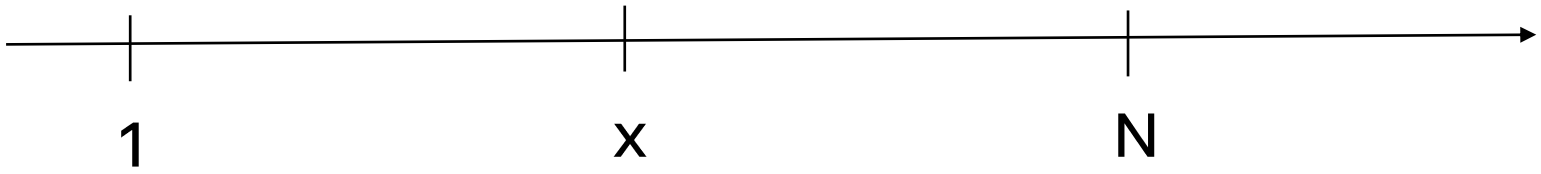
- 정확한 해를 구하지 못할 수 있음. => 근사값을 구해야 함.
- 근사값과 해의 차이가 충분히 작다면 (가령 제곱의 결과 차이가 0.00010하), 이 값을 해로 간주

5. 제곱근 구하기

- 양의 정수 N의 제곱근을 구하시오.

아이디어

- 해(x)가 위치하는 구간



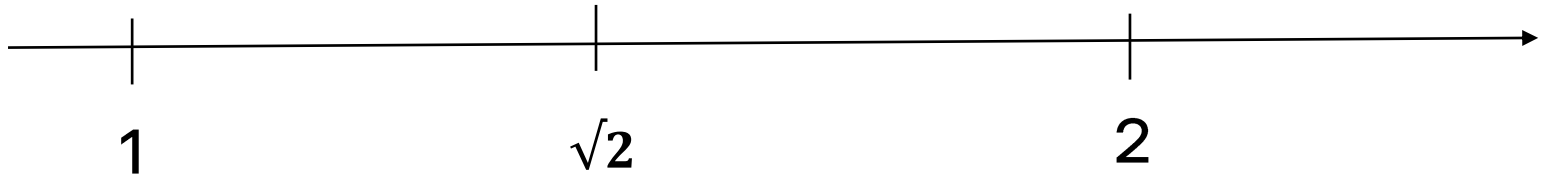
- 해(x)가 위치하는 구간을 어떻게 좁혀갈 것인가?
  => 중간값을 해로 간주하여, 근사 해를 찾을 때까지 반복

5. 제곱근 구하기

- 양의 정수 N의 제곱근을 구하시오.

아이디어

- $\sqrt{2}$
  1. 1과 2의 중간값 1.5를 해로 간주하여, 오차 계산
  2. 오차가 + 이므로 1과 1.5의 중간값 1.25를 해로 간주하여, 오차 계산
  3. 오차가 −이므로 1.25와 1.5의 중간값 1.375를 해로 간주하여, 오차 계산
  4. …



5. 제곱근 구하기

- 양의 정수 N의 제곱근을 구하시오.

의사코드

```
def my_sqrt(num):
    if (num >= 1):
        sol = my_sqrt_rec(1, num, num)
    elif (num > 0):
        #생략
    else:
        sol = -1
    return sol
```

5. 제곱근 구하기

- 양의 정수 N의 제곱근을 구하시오.

의사코드

```
def my_sqrt(num):
    if (num >= 1):
        sol = my_sqrt_rec(1, num, num)
    elif (num > 0):
        #생략
    else:
        sol = -1
    return sol
```

```
def my_sqrt_rec(low, high, num):
    ans = (low + high) / 2
    diff = ans*ans - num
    if (abs(diff) < 0.0001):
        return ans
    elif (diff > 0):
        return my_sqrt_rec(low, ans, num)
    else:
        return my_sqrt_rec(ans, high, num)
```

5. 제곱근 구하기

- 양의 정수 N의 제곱근을 구하시오.

의사코드

```
def my_sqrt(num):
    if (num >= 1):
        sol = my_sqrt_rec(1, num, num)
    elif (num > 0):
        #생략
    else:
        sol = -1
    return sol
```

최악 시간복잡도 ?

5. 제곱근 구하기

- 양의 정수 N의 제곱근을 구하시오.

파이썬 코드

```
def my_sqrt(num):
    if (num >= 1):
        sol = my_sqrt_rec(1, num, num)
    elif (num > 0):
        #생략
    else:
        sol = -1
    return sol
```

```
def my_sqrt_rec(low, high, num):
    ans = (low + high) / 2
    diff = ans*ans - num
    if (abs(diff) < 0.0001):
        return ans
    elif (diff > 0):
        return my_sqrt_rec(low, ans, num)
    else:
        return my_sqrt_rec(ans, high, num)
```

5. 제곱근 구하기

- 양의 정수 N의 제곱근을 구하시오.

파이썬 코드 실행

```
print(my_sqrt(4))
print(my_sqrt(1))
print(my_sqrt(2))
```

```
In [49]: runfile('D:/data/lecture/
파이썬으로배우는자료구조와알고리즘/code/알고리즘/
untitled1.py', wdir='D:/data/lecture/
파이썬으로배우는자료구조와알고리즘/code/알고리즘')
2.000001907348633
1.0
1.414215087890625
```

정리하기

- 축소정복, 분할정복, 재귀의 개념
- 팩토리얼을 구하는 알고리즘
- 피보나치 수열을 구하는 알고리즘
- 제곱근을 구하는 알고리즘

05강

다음 시간 안내 ▶▶▶

분할 정복과 동적프로그래밍