2장. 알고리즘과 성능 분석

2.1 알고리즘과 문제 해결

2.2 알고리즘의 표현

2.3 순환

2.4 프로그램 성능 분석

2.1 알고리즘과 문제 해결

알고리즘과 프로그램

- 알고리즘 (algorithm)
  - 특정 문제를 해결하기 위해 논리적으로 기술한 일련의 명령문

- 프로그램 (program)
  - 알고리즘을 컴퓨터가 이해하고 실행할 수 있는 특정 프로그래밍 언어로 표현한 것
    - Program = algorithm + data structures

알고리즘의 요건

- 완전성과 명확성
  - 수행 단계와 순서가 완전하고 명확하게 명세되어야 함
  - 순수하게 알고리즘이 지시하는 대로 실행하기만 하면 의도한 결과가 얻어져야 함

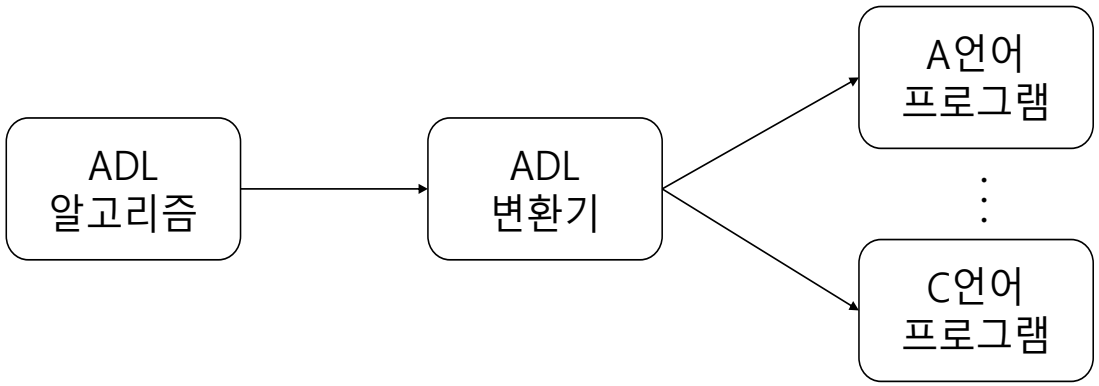
- 입력과 출력
  - 입력 : 알고리즘이 처리해야 할 대상으로 제공되는 데이타
  - 출력 : 입력 데이타를 처리하여 얻은 결과

- 유한성
  - 유한한 단계 뒤에는 반드시 종료

2.2 알고리즘의 표현

알고리즘의 표현

- ADL (Algorithm Description Language)
  - 알고리즘 기술을 위해 정의한 언어
  - 사람이 이해하기 쉽고, 프로그래밍 언어로의 변환이 용이
  - 의사 코드 (pseudo-code) : ADL과 약간의 자연어로 기술한 것
  - ADL 알고리즘에서 프로그램으로의 변환



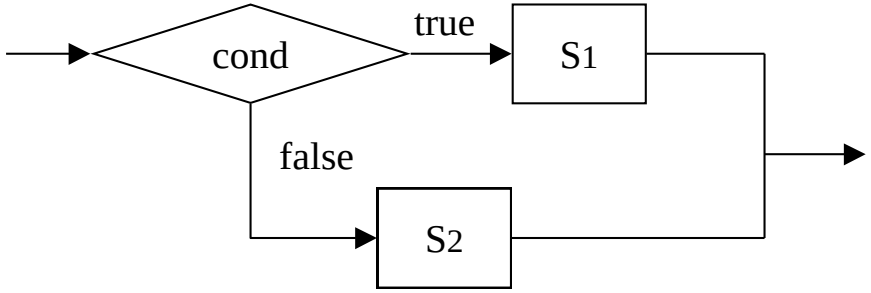
  - ADL의 데이타 : 숫자, 부울(boolean) 값, 문자
  - ADL의 명령문 :
    - 종류 : 지정문, 조건문, 반복문, 함수문, 입력문, 출력문
    - 명령문 끝에는 세미콜론(;)을 사용

ADL (1) - 지정문

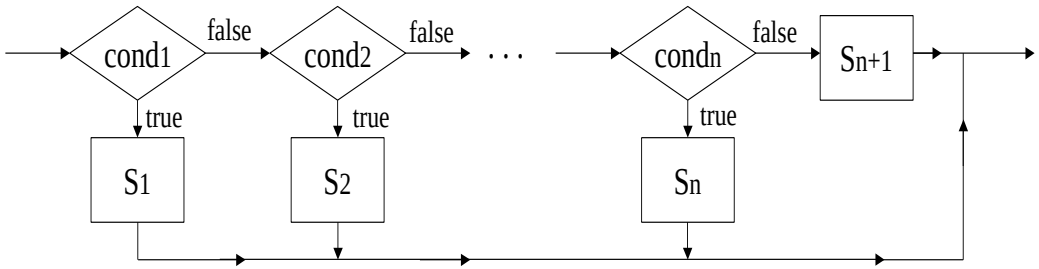
- 지정문 (assignment)
  - 형식 : 변수 ← 식;
  - 식 (expression)
    - 산술식 (arithmatic exp)
    - 부울식 (boolean exp)
      - 결과 : 참(true) 또는 거짓 (false)
      - 표현
        - 논리 연산자(and, or, not)
        - 관계 연산자(<, ≤, =, ≠, ≥, >)
    - 문자식 (character exp)
  - 제어 구조 : 순차적

ADL (2) – 조건문

- 조건문
  - 제어 구조 : 선택적
  - 종류 : if문과 case문
- if문
  - **if** (*cond*) **then** S_1
    else S_2



- case문
  - **case** {
    cond1 : *S*1
    cond2 : *S*2
    . . .
    condn : *S*n
    else : Sn+1
    }



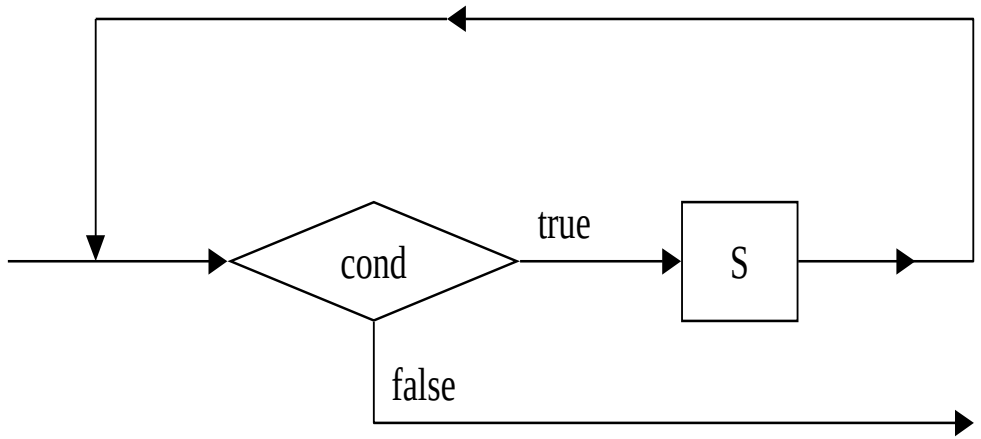
ADL (3) – 반복문 (1/3)

- 반복문 (repeat)
  - 제어 구조 : 일정한 명령문 그룹을 반복해서 수행하는 루프(loop) 형태
  - 종류 : while문, for문, do-while문

- while문
  - 형식
    - **while** (*cond*) **do**

      S
  - 무한 루프
    - **while** (**true**) **do**

      S



ADL (3) – 반복문 (2/3)

- for문
  - 형식
    - **for** (*initialization*; *cond*; *increment*) **do**

      S
  - 동등한 while문
    - initialization

      while (*cond*) **do** {

      S

      increment;

      }
  - 무한 루프
    - **for** (; ;) **do**

      S

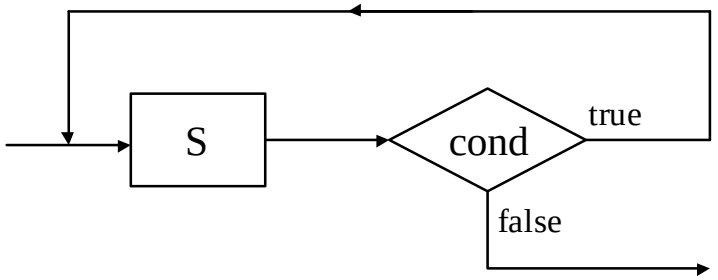
      (cond의 기정 값이 true이므로)

ADL (3) – 반복문 (3/3)

- do-while문
  - 형식
    - **do**

      S

      while (*cond*);
  - 특징
    - *S*가 최소한 한 번은 실행됨



- 루프 명령문
  - **goto** 명령문 : 루프에서 바로 빠져나갈 때 사용
  - **exit**문 : 자신을 둘러싸고 있는 가장 가까운 루프 밖의 명령문으로 제어를 이동시킴

ADL (4) – 함수문

- 함수문
  - 형식
    - function-name(parameter_list)

      S

      end
  - 호출 함수로의 복귀
    - return expr;
    - 여기서 expr은 함수의 실행 결과
  - 함수 호출
    - function-name(argument-list)
  - 인자와 매개변수와의 연관
    - 값 호출 (call by value) 규칙
    - 각 인자의 실제 값이 호출된 함수로 전달
    - 인자의 값이 주소(참조)가 되면 매개 변수에 주소 값이 전달되어 값은 데이타 지시

ADL (5) – 입출력문

- 입력 함수 : read (argument_list);
- 출력 함수 : print (argument_list);
- 인자 : 변수나 인용부호가 있는 문자열

ADL (6) – 기타

- 기타 명령문
  - stop : 현재 진행 중인 함수의 실행을 정지
  - 코멘트 : //는 그 행 끝까지, /*과 */은 코멘트의 시작과 끝 표시
  - 배열 : a[n], a[n_1, n_2], a[n_1, n_2, , n_n]

- ADL 기술 규칙
  - 함수의 입출력 변수를 명확히 명세
  - 변수의 의미를 알 수 있게 정의
  - 알고리즘의 제어 흐름은 되도록 순차적
  - 시각적 구분을 위해 인덴테이션(indentation) 이용
  - 코멘트는 짧으면서 의미는 명확히
  - 함수를 적절히 사용

2.3 순환 (recursion)

순환

- 정의하려는 개념 자체를 정의 속에 포함하여 이용
  - 자연수 정의 예
    1. 0은 자연수이다.
    2. n이 자연수이면 n+1은 자연수이다.
    3. 자연수에는 이 이외의 수는없다.

- 종류
  - 직접 순환 : 함수가 직접 자신을 호출
  - 간접 순환 : 다른 제 3의 함수를 호출하고 그 함수가 다시 자신을 호출
- 순환 방식의 적용
  - 분할 정복(divide and conquer)의 특성을 가진 문제에 적합
  - 어떤 복잡한 문제를 직접 간단하게 풀 수 있는 작은 문제로 분할하여 해결하려는 방법.
  - 분할한 문제는 원래의 문제와 그 성질이 같기 때문에 푸는 방법도 동일
- 순환 함수의 명령문 골격

  if (simplest case) **then** solve directly

  else {make a recursive call to a simpler case};

순환 함수의 예 – Factorial

- $a_n = n!$
  - $a_0 = 1$
  - $a_n = n \cdot (n - 1)! = n \cdot a_{n-1}$

- factorial(n) = n * factorial(n-1)

- factorial(5) = 5 · factorial(4)
  = 5 · (4 · factorial(3))
  = 5 · (4 · (3 · factorial(2)))
  = 5 · (4 · (3 · (2 · factorial(1))))
  = 5 · (4 · (3 · (2 · 1)))
  = 5 · (4 · (3 · 2))
  = 5 · (4 · 6)
  = 5 · 24
  = 120

순환 함수의 예 – Factorial

- 순환 함수의 표현

```
public static int factorial(int n) {
    if (n <= 1) return 1;
    else return n * factorial(n - 1);
}
```

- 비순환 함수로의 표현

```
public static int nFactorial(int n) {
    if (n <= 1) return 1;

    int fact = 1;
    for(int i = 2; i <= n; i++) {
            fact = fact * i;
    }

    return fact;
}
```

순환 함수의 예 – 이원 탐색

- 이원 탐색 (binary search)
  - 주어진 탐색키 key가 저장된 배열a[]의 위치(인덱스)를 찾아내는 방법
    - mid ← (left + right) / 2
    - key = a[mid] : 탐색 성공, return mid
    - key < a[mid] : a[mid]의 왼편에 대해 이원탐색
    - key > a[mid] : a[mid]의 오른편에 대해 이원탐색
  - 순환 함수의 표현

```
public static int binsearch(int[] a, int key, int left, int right) {
    if (left <= right) {
        int mid = (left + right) / 2;
        if (key == a[mid])
            return mid;
        else if (key < a[mid])
            return binsearch(a, key, left, mid - 1);
        else
            return binsearch(a, key, mid + 1, right);
    }
    return -1;
}
```

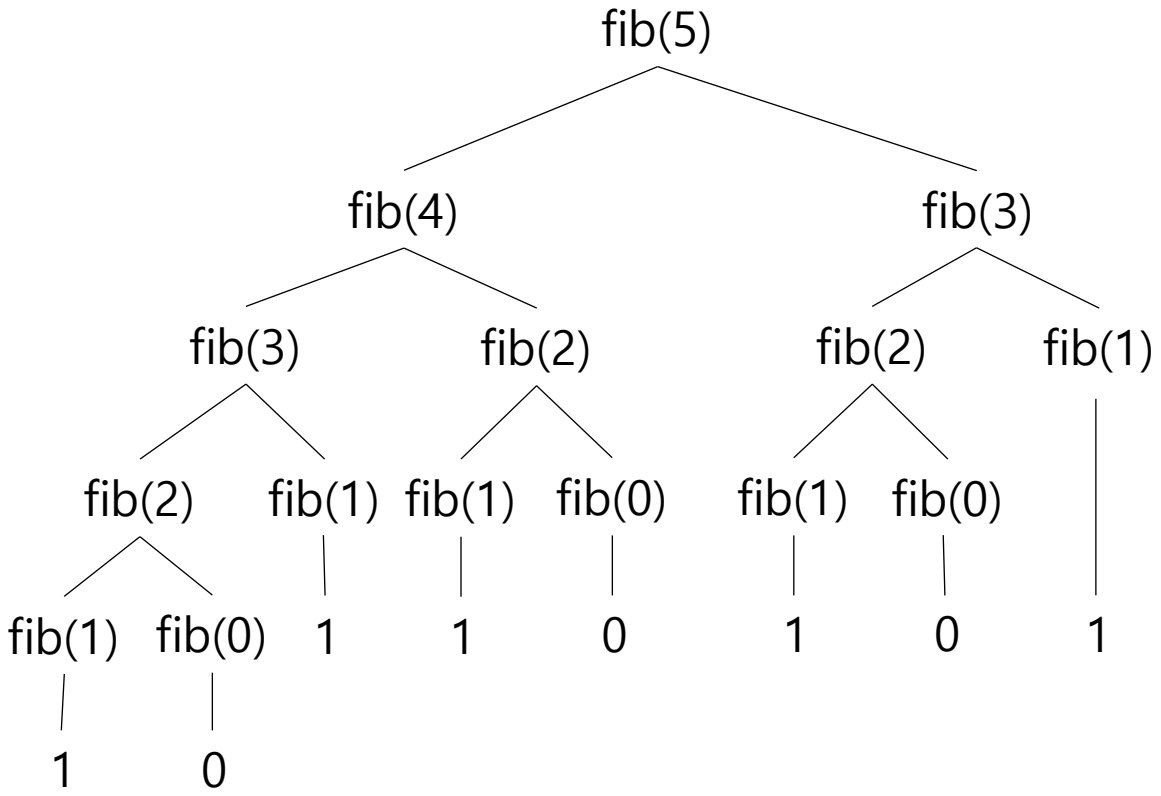
순환 함수의 예 – 피보나치 수열

- 피보나치 수열 (Fibonacci sequence) - 0, 1, 1, 2, 3, 5, 8, 13, …
  - 순환 정의
    - n=0 : $f_0 = 0$
    - n=1 : $f_1 = 1$
    - $n \geq 2 : f_n = f_{n-1} + f_{n-2}$
  - 순환 함수의 표현

```
public static int fib(int n) {
    if (n <= 0) return 0;
    if (n == 1) return 1;
    return fib(n - 1) + fib(n - 2);
}
```

    - 이 fib() 함수는 순환 호출 횟수가 급증하여 실행시간을 기준으로 볼 때 반복 함수보다 비효율적임
      - n=10이면 177번, n=25이면 242785번 함수 호출
    - 순환적 정의가 순환적 알고리즘으로 문제를 해결하는 데 최적의 방법이 아닐 수도 있다는 것을 보여주는 예

순환 함수의 예 – fib(5)의 실행과정



2.4 프로그램 성능 분석

프로그램 평가

- 프로그램의 평가 기준
  - 원하는 결과의 생성 여부
  - 시스템 명세에 따른 올바른 실행 여부
  - 프로그램의 성능
  - 사용법과 작동법에 대한 설명 여부
  - 유지 보수의 용이성
  - 프로그램의 판독 용이

- 프로그램의 성능 평가
  - 성능 분석 (performance analysis)
    - 프로그램을 실행하는데 필요한 시간과 공간의 추정
  - 성능 측정 (performance measurement)
    - 컴퓨터가 실제로 프로그램을 실행하는데 걸리는 시간 측정

공간 복잡도와 시간 복잡도

- 공간 복잡도 (space complexity)
  - 프로그램을 실행시켜 완료하는데 소요되는 총 저장 공간
  - $S_p = S_c + S_e$
    - S_c : 고정 공간
      - 명령어 공간, 단순 변수, 복합 데이터 구조와 변수, 상수
    - S_e : 가변 공간
      - 크기가 변하는 데이터 구조와 변수들이 필요로 하는 저장 공간
      - 런타임 스택(runtime stack)을 위한 저장 공간

- 시간 복잡도 (time complexity)
  - 프로그램을 실행시켜 완료하는데 걸리는 시간
  - $T_p = T_c + T_e$
    - T_c : 컴파일 시간
    - T_e : 실행 시간
    - 컴파일 시간은 정적인 반면 실행 시간은 가변이기 때문에 이 실행시간을 추정하여 성능평가로 대체

실행 시간 추정 요소

- 단위 명령문 하나를 실행하는 데 걸리는 시간
  - 단위 시간으로써 일정하다고 가정

- 단위 명령문 실행 빈도수(frequency count)
  - 단위 명령문을 프로그램 단계(program step)로 간주

→ 성능은 프로그램 **단계 실행 빈도수**로 추정

즉 큰 자리 수(order of magnitude): 1, 10, 100

피보나치수의 예 (1)

- n번째 항을 계산하는 반복식 프로그램

```
        fib_i(n)
1           if (n < 0)
2               then stop;   // error 발생
3           if (n ≤ 1)
4               then return n;
5           fn2 ← 0;
6           fn1 ← 1;
7           for (i ← 2; i <= n; i ← i + 1) do {
8               fn ← fn1 + fn2;
9               fn2 ← fn1;
10              fn1 ← fn;
11          }
12          return fn;
13      end fib_i()
```

피보나치수의 예 (2)

- f_n 계산을 위한 실행 빈도수 (n > 1)

명령문(행)	실행 빈도수	명령문(행)	실행 빈도수
1	1	8	n-1
2	0	9	n-1
3	1	10	n-1
4	0	11	0
5	1	12	1
6	1	13	0
7	n		

- 실행 시간
  - 4n+2 : O(n) – 전체 값은 n의 크기에 달려있다는 의미

- "함수 fib_i()의 시간 복잡도는 O(n)이다"라고 말함

점근식 표기법 (1)

- 점근식 표기법(Asymptotic notation)
  - Big-Oh (O)
  - Big-Omega (Ω)
  - Big-Theta (Θ)

- Big-Oh (O)
  - f, g가 양의 정수를 갖는 함수일 때, 두 양의 상수 a, b가 존재하고, 모든 n≥b에 대해 $f(n) \leq a \cdot g(n)$ 이면, $f(n) = O(g(n))$
  - **상한(upper bound)**

  - 예
    - $f(n) = 3n + 2$: $f(n) = O(n)$ (a=4, b=2)
    - $f(n) = 1000n^2 + 100n - 6$: $f(n) = O(n^2)$ (a=1001, b=100)
    - $f(n) = 6 \bullet 2^n + n^2$: $f(n) = O(2^n)$ (a=7, b=4)
    - $f(n) = 100$: $f(n) = O(1)$ (a= 100, b=1)

점근식 표기법 (2)

- 연산 시간 그룹
  - 상수시간 : O(1)
  - 로그시간 : O(logn)
  - 선형시간 : O(n)
  - n로그시간 : O(nlogn)
  - 평방시간 : O(n^2)
  - 입방시간 : O(n^3)
  - 지수시간 : O(2^n)
  - 계승시간 : O(n!)

- 연산 시간의 크기 순서

  $O(1) < O(log n) < O(n) < O(n log n) < O(n^2) < O(n^3) < O(2^n) < O(n!)$

- $O(n^k)$: polynomial time

점근식 표기법 (3)

- Big-Omega (Ω)
  - f, g가 양의 정수를 갖는 함수일 때, 두 양의 상수 a, b가 존재하고
    모든 $n \geq b$에 대해 $f(n) \geq a \cdot g(n)$ 이면,
    $f(n) = \Omega(g(n))$
  - **하한(lower bound)**
  - 예
    - $f(n) = 3n + 2$: $f(n) = \Omega(n)$ (a=3, b=1)
    - $f(n) = 1000n^2 + 100n - 6$: $f(n) = \Omega(n^2)$ (a=1000, b=1)
    - $f(n) = 6 \cdot 2^n + n^2$: $f(n) = \Omega(2^n)$ (a=6, b=1)

점근식 표기법 (4)

- Big-Theta(Θ)
  - f, g가 양의 정수를 갖는 함수일 때,
    세 양의 상수 a, b, c가 존재하고, 모든 n ≥ c에 대해
    $a \cdot g(n) \le f(n) \le b \cdot g(n)$ 이면, $f(n) = \Theta(g(n))$
  - **상한과 하한으로 한정**

  - 예
    - $f(n) = 3n + 2$: $f(n) = \Theta(n)$ (a=3, b=4, c=2)
    - $f(n) = 1000n^2 + 100n - 6$: $f(n) = \Theta(n^2)$ (a=1000, b=10001, c=100)
    - $f(n) = 6 \bullet 2^n + n^2$: $f(n) = \Theta(2^n)$ (a=6, b=7, c=4)

fib_i()의 예

- 점근적 복잡도의 계산

세그먼트	점근적 복잡도
1~6	O(1)
7~11	O(n)
12~13	O(1)

$T(Fib\_i) = O(1) + O(n) + O(1) = O(n)$

  - 예
    - $O(1) + O(1) = O(1)$
    - $O(1) + O(n) = O(n)$
    - $O(n) + O(n) = O(n)$
    - $O(n) + O(n^2) = O(n^2)$
    - $O(1) + O(n) + O(n^2) = O(n^2)$

- 실행 환경
  - 최선의 경우 (best case)
  - 최악의 경우 (worst case)