Data structure

김종규, PhD

2017-03-06

Course overview

- What is it? After learning the concept of abstract data type, use it to simplify complex problems in a disciplined way
- What you can do? Design and analyze software systems with abstract data types
  - Understand the definition of computation
  - Divide complex problems into smaller operations
  - Design a solution for implementing an operation
  - Learn how to express the solution using pseudocode
  - Analyze the efficiency of the solution
  - Compare solutions and choose better one
  - Learn useful theoretical tools for designing abstract data types

Course overview

- Textbook and references
  - Corman, Thomas et al. Introduction to algorithms
  - Brad Miller. Problem solving with algorithms and data structures
  - 이지영. 자바로 배우는 쉬운 자료구조
- Assignments and evaluation
  - Reports, programming assignments
  - A final project

Lecture schedule and guideline

- All lecture slides will be uploaded to the web
- Mon 12:00-13:40 Lecture
- Wed 13:00-13:50 Lecture/Lab
- Submitted materials may be shared during lectures
  - Good examples, bad examples, typical mistakes
  - The identity of authors may be revealed

Evaluation

- Assignments and project: 20
  - Submission within due date is very important
- Midterm: 30
  - Basic concepts of abstract data types
  - How to design abstract data types with simple examples
  - Analyzing the efficiency of the designs
- Final: 40
  - More advanced examples
  - Practical considerations on designing abstract data types
- Class activity: 10
  - Attendance: one day notice
  - Volunteering

자료구조?

- 자료구조를 배우면 뭐가 좋을까?
- 자료구조가 뭐길래 컴퓨터학과의 필수기초일까?
- 자료구조는 어떻게 하는 것이 잘하는 것일까?
- 자료구조를 배우려면 무엇을 잘 알아야야 할까?
- 자료구조는 어디에 사용될까?

뭐가 좋을까?

- 다음 프로그램이 하는 일은?

```
main()
{
   int ch;
   for (ch = getchar(); ch != EOF;ch = getchar()) {
     putchar(ch);
   }
}
```

⟶ 역순으로 출력하시오

뭐가 좋을까?

- 역순으로 출력하시오 $\longrightarrow$ 표준 입력으로 받아들인 문자열을 다음과 같이 변환하는 프로그램을 작성하시오
  - “abcdefz” $\longrightarrow$ “zfedcba”

```
main()
{
  int ch;
  for (ch = getchar(); ch != EOF;ch = getchar()
    push(ch);
  }
  while(!empty()) {
    putchar(pop());
  }
}
```

자료구조란 무엇일까?

- Abstract data type: stack
  - 정의: push, pop, empty $\longrightarrow$ 각각을 어떻게 구현할까?
    $\longrightarrow$ 자료구조
  - 성질 (예: push)
    - 소요되는 시간은?
    - 필요한 메모리량은?
- 자료구조를 공부하는 이유
  - Abstract data type: 복잡한 프로그래밍을 단순화
  - 프로그래머가 한계점을 이해하고 적절하게 사용할 수 있도록 하는 것

$\longrightarrow$ 비전문가와 차별화되는 시점

쉽게 배울 수는 없을까?

- 프로그래밍 절차
  - 생각하기 $\longrightarrow$ 수학
  - 그려보기 $\longrightarrow$ 종이와 연필
  - 코딩하기
  - 컴파일하기
  - 실행하기
  - 디버깅하기 $\longrightarrow$ 프로그래머의 친구

꼭 알아야 하는 것은?

- pointer
- 복잡도의 정의 (The definition of complexity)
- 복잡도 분석 (The analysis of complexity)

어디에 사용하게 될까?

- 알고리즘
- 운영체제
- 데이터베이스
- ...

프로그래밍 언어

- 수업시간에 사용할 언어
  - C/C++
  - Java
  - Python
- 과제에 사용할 언어
  - 세 언어 중 택일

Fibonacci number

- 다음과 같이 정의된 수열에서 f_{10} 을 구하시오.

$$\begin{cases} f_n &= f_{n-1} + f_{n-2} \\ f_1 &= 1 \\ f_0 &= 0 \end{cases}$$

프로그래밍 언어

- C 언어

```
int fib(int n)
{
  if (n <= 2)
    return 1;
  else
    return fib(n - 1) + fib(n - 2);
}
int main()
{
  printf("fib(10) = %d\n", fib(10));
}
```

프로그래밍 언어

- Java

```
class FibSample {
    public static int fib(int n) {
        if (n <= 2)
            return 1;
        else
            return fib(n - 1) + fib(n - 2);
    }
    public static void main(String [] args) {
        System.out.println("fib(10) = " + fib(10));
    }
};
```

프로그래밍 언어

- Python

```
def fib(n):
  if n <= 2:
    return 1
  else:
    return fib(n - 1) + fib(n - 2)
print fib(10);
```

Fibonacci number

- 다음과 같이 정의된 수열에서 f_{45} 를 구하시오.

$$\begin{cases} f_n &= f_{n-1} + f_{n-2} \\ f_1 &= 1 \\ f_0 &= 0 \end{cases}$$

$\longrightarrow$ C 언어로 구현!

f_{45}: C 언어

- ex01.c

```
#include <stdio.h>
int fib(int n)
{
  if (n <= 2)
    return 1;
  else
    return fib(n - 1) + fib(n - 2);
}
int main()
{
  int n = 45;
  printf("fib(%d) = %d\n", n, fib(n));
}
```

f_{45}: C 언어

- ex01.c

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

$\longrightarrow$ 왜 이렇게 오래 걸릴까?

- 생각, 그림, ...

f_{45}: 더 빠른 구현 (C 언어)

- ex02.c

```
int fib(int n)
{
  int a = 0,b = 1;
  int k, tmp;
  for (k = 0; k < n; k++) {
    tmp = b;
    b = a + b;
    a = tmp;
  }
  return a;
}
```

f_{45}: 더 빠른 구현 (Python)

```
def fib(n):
  a,b = 0,1
  for k in range(n):
    a,b=b,a+b
  return a
print fib(45);
```

생각하는데 필요한 것들

- 변수와 데이터 타입 + pointer $\longrightarrow$ 변수 개념, 배열 개념, 포인터 개념
- 조건문과 연산 $\longrightarrow$ 조건문 개념, 연산 개념
- 반복문의 문법과 의미 $\longrightarrow$ 반복문 개념
- 입출력 절차 $\longrightarrow$ 거의 필요 없지만 디버깅에 유용함
- 함수의 정의와 활용 $\longrightarrow$ 프로그래밍 함수의 개념
- 구조체와 추상화 $\longrightarrow$ 매우 중요!

개념의 개념은?

- 변수: 값을 넣어두는 메모리 공간
  - 이름을 지정할 수 있고
  - 값을 넣을 수 있고
  - 값을 잃어올 수 있는 대상
- 예: C 언어에서 `int n;` 으로 선언
- 질문: 3a 라는 이름으로 변수를 선언하면 안되나?

$\longrightarrow$ 개념상 안될 이유는 없으나

$\longrightarrow$ C 컴파일러는 거부!

개념과 프로그래밍의 차이

- 다음 함수가 하는 일은?

```
int findmax(int ary[], int num)
{
  int i;
  int x = ary[0];
  int res = 0;
  for (i = 1; i < num; i++) {
    if (x > ary[i]) {
      x = ary[i];
      res = i;
    }
  }
  return res;
}
```

프로그래밍 언어의 불편함

- 다음 C 프로그램의 오류를 찾으시오.

```
int findmax(int ary[], int num)
{
   ...
}
double findmax(double ary[], int num)
{
   ...
}
```

프로그래밍 언어의 불편함

- 개념적으로 동일한 일이기 때문에 분기문, 반복문의 구조는 같다
- 그러나 입력 타입이 다르기 때문에 각각에 대해서 함수를 정의하여야 한다
- 이 경우 개념적으로 동일한 일이라 같은 이름을 사용하고 싶은데 그럴 수 없다

$\longrightarrow$ 프로그래밍 언어에 따른 인위적인 제약

생각의 도구

- 개념: 주어진 배열에서 최대값을 찾는 함수

```
function findmax(ary[])
  x = -infty          # negative infinity
  for i in [1..n]     # n is the size of array
    if (ary[i] > x)
      ndx = i
    end if
  end for
  return ndx
end function
```

$\longrightarrow$ Pseudocode

알고리즘 분석

- 다음과 같이 정의된 수열에서 f_{45} 를 구하시오.

$$\begin{cases} f_n &= f_{n-1} + f_{n-2} \\ f_1 &= 1 \\ f_0 &= 0 \end{cases}$$

알고리즘 분석 f_{45}

- + 연산은 몇 번 수행될까?

```
#include <stdio.h>
int fib(int n)
{
  if (n <= 2)
    return 1;
  else
    return fib(n - 1) + fib(n - 2);
}
int main()
{
  int n = 45;
  printf("fib(%d) = %d\n", n, fib(n));
}
```

알고리즘 분석 f_{45}

- + 연산은 몇 번 수행될까?
- = 연산은 몇 번 수행될까?

```
int fib_fast(int n)
{
  int a = 0,b = 1;
  int k;
  int tmp;
  for (k = 0; k < n; k++) {
    tmp = b;
    b = a + b;
    a = tmp;
  }
  return a;
}
```

알고리즘 분석의 개념

- 주요 연산을 파악
- 주요 연산이 몇 번 수행되는지
- `fib(1)` 을 계산하면?
  - `+` 연산을 한 번도 수행하지 않는다
- `fib_fast(1)` 를 계산하면?
  - `=` 을 여섯 번 수행한다
  - `+` 을 한 번 수행한다
- 어떤 것이 더 좋은 방식일까?

$\longrightarrow$ 새로운 개념 소개: $O(g(n))$

Reading assignment

- 이번 시간: Introduction to algorithms, Chapter 1, 2
- 다음 시간: Introduction to algorithms, Chapter 3

Wrap-up (1/2)

- Pseudocode
  - Though computer programming is essential in learning data structure,
  - Real programming langauges such as C are not convenient to write down abstract concepts, e.g., findmax()
  - Therefore, throughout this lecture, we will use pseudocode to denote algorithms

Wrap-up (2/2)

- Algorithm comparison
  - Although we should compare algorithms regarding the execution time,
  - actual execution time is not suitable for comparing algorithms
  - We will learn new concepts called growth function for algorithm comparison
  - $O(g(n))$ is one of them and the most widely used one