CS042

Scalable Data Structure Study

Programmer's guide to the CS

Author:
Seungwoo Schin
Gihyo Jang
Seowon Oh

Typeset by:
Seungwoo Schin

PyDal study fork();

June 29, 2015

Contents

0. Introduction

I. Administrative Details

기본적인 데이터구조들을 여러 언어로 짜보고, 코딩한 결과를 서로 비교해 보는 걸 목적으로 합니다. 다른 스터디에서도 사용할 수 있도록 작성하였습니다. 기본적인 규칙들은 다음과 같습니다.

1. 전반적인 공지나 질문 등은 카톡방을 통해서 공유합니다.
2. 사용하는 언어는 자유이나, file io가 가능해야 합니다. 재귀, loop 등을 사용할 수 있으면 좋습니다.
3. 1주일에 한번씩 매주 토요일 10시부터 12시 반까지 만납니다.
4. 만날 때마다 전 주에 정해진 데이터구조에 대해서 한 명이 공부해서 아래 일들을을 하셔야 합니다.
   (a) 데이터구조에 대한 간략한 설명
   (b) 데이터구조를 txt 파일로 표현하는 방법에 대한 표준 설정
   (c) 해당 데이터구조에 대한 operation의 결과를 txt 파일로 나타내는 방법에 대한 표준 설정
   (d) 테스트용 파일과 솔루션 파일 작성
5. 만난 후 다음번에 만나기 전까지, 해당 스터디에서 커버한 부분에 대한 코드를 작성한 후, 테스트 결과와 같이 올려주세요.
6. 스터디장은 위 코드들과 테스트, 솔루션 파일들을 받아서 github에 업로드해야 하며, 스터디를 진행할 때마다 커버한 부분을 LaTeX으로 작성해서 올려야 합니다.
7. 스터디장은 LaTeX을 할 줄 알아야 합니다.

I.1 사용할 언어들

사용 언어는 자유이며, 아래의 언어는 사용 가능한 언어들의 hello world! 예제입니다. 아래에 없는 언어를 사용하고 싶으시면 스터디장에게 말해서 언어를 추가해 주세요.

1. C

```c
#include <stdio.h>

int main()
{
	printf("Hello World!\n");
}
```

2. Java

```java
public class Foo {
  public static void main(String args[]) {
    System.out.println("for test");
  }
}
```

3. Python

```
print "hello world!"
```

4. JavaScript

```
// Hello in Javascript

// display prompt box that ask for name and
// store result in a variable called who
var who = window.prompt("What is your name");

// display prompt box that ask for favorite color and
// store result in a variable called favcolor
var favcolor = window.prompt("What is your favorite color");

// write "Hello" followed by person' name to browser window
document.write("Hello " + who);

// Change background color to their favorite color
document.bgColor = favcolor;
```

II. Programming Basics

이 항목은 특정 알고리즘이나 데이터구조에 종속된 내용이 아니라 프로그래밍 전반에 관한 기본적인 지식들입니다. [1]

II.1 Time Complexity

어떤 문제를 풀 때 여러 가지 프로그램이 있을 수 있습니다. 이 중 최적의 프로그램을 고르기 위해서는 프로그램 간의 성능차를 정량적으로 비교할 수 있는 기준이 필요합니다. 그런 기준 중 하나가 시간복잡도입니다. 시간복잡도는 알고리즘(혹은 프로그램)이 얼마나 많은 시간을 필요로 하는지를 나타내는 지표입니다. 프로그램이 수행되는데 걸리는 시간을 프로그램의 input의 길이에 대한 함수로 나타낸 것이 시간복잡도의 수학적 정의입니다. 예시를 들기 위해서 다음 코드 예제를 보겠습니다.

```
def count(n):
        for i in range(n):
                print i
```

위 코드에서의 count 함수는 크기가 n인 정수를 받아서, n번 출력하게 됩니다. 따라서 이 함수가 실행되는데 걸리는 시간은 print함수 한 번 실행되는 시간 * n일 것입니다. 이런 경우 시간복잡도는 1차함수가 됩니다.
일반적으로 편의를 위해서 시간복잡도를 나타낼 때는 그냥 함수 그대로 나타내는 것이 아니라 O notation을 이용합니다. O notation은 어떤 함수의 saymptoticall한 representation으로, 대략적인 함수의 모양을 나타내 줍니다. 예를 들어서 $n^2 + n + 1$은 $O(n^2)$으로 나타내어집니다. O notation의 결과물은 항상 다항식일 필요는 없으며, 로그나 constant, 혹은 지수함수 꼴로 나타내어지는 경우도 있습니다.

[1] 더 필요한 부분이 있다고 생각되면 추가 바랍니다.

II.2 Recursion

어떤 집합이 다음과 같은 두 개의 명제로 정의되면 그 집합을 recursive[2] 하다고 합니다.

- Base cases : recursion 없는 일반적인 정의. $a \in A$ 등.
- 그 외의 경우는 Base case에서 유도됨.

[3] 이런 집합의 예로는 피보나치 수열이 있습니다. 이런 재귀적인 성질을 이용해서 프로그램을 짜는 경우가 많은데, 예시를 몇 가지 들어보겠습니다.

- 하노이의 탑
  함수 Hanoi(n, a, b, c)는 4개의 인자를 받는 함수입니다. 첫 번째 인자는 숫자로, 하노이의 탑에 원판이 몇 개 있는지를 나타냅니다. a,b,c는 각각 기둥의 이름을 나타냅니다. Hanoi 함수를 실행시키면 a에서 c로 원반 n개를 움직일 때 어떻게 움직여야 하는지 알려줍니다. 이러한 함수의 경우 재귀적인 방법을 사용하여 짜면 편리하게 짤 수 있습니다. 파이썬으로 짠 예제는 아래에 있습니다.

```python
# for python 2.x version
def Hanoi(n, a, b, c):
    res = ""
    if n==1:
        res = "move %d from %s to %s"%(n, a, c)
    else:
        res = "%s \nmove %d from %s to %s \n%s" %(Hanoi(n-1, a, c, b), n, a, c,
            Hanoi(n-1, b, a, c))
    return res
```

- Parser

Recursion은 iteration과 매우 흡사한 성질을 가집니다. 사실, iteration과 recursion은 동치로 모든 iteration은 recursion으로 나타내어질 수 있고, 역도 성립합니다.

II.3 Divide and Conquer

definition example

II.4 Dynamic Programming

introcd ../ example vs greedy

[2] https://en.wikipedia.org/wiki/Recursion
[3] 수학적 귀납법과 거의 흡사한 꼴인 것을 알 수 있습니다.

1. List

I. Structures to implement

I.1 Singly linked list

I.2 Doubly linked list

I.3 Circular linked list

II. Operations to implement

II.1 Sorting

II.2 Searching

II.3 Insertion

II.4 Deletion

2. Stack

I. Structures to implement

I.1 Simple stack

II. Operations to implement

II.1 Sorting

II.2 Searching

II.3 Insertion

II.4 Deletion

3. Queue

I. Structures to implement

I.1 Simple queue

I.2 Priority queue

I.3 Dequeue (double ended queue)

II. Operations to implement

II.1 Sorting

II.2 Searching

II.3 Insertion

II.4 Deletion

4. Advanced Structure

I. Structures to implement

I.1 Tree

Simple binary tree

Full binary tree

Perfect binary tree

Complete binary tree

Binary search tree

AVL tree

Red-black tree

Heap tree

B-tree

2-3-4 tree

B+ tree

I.2 Graph

Simple graph

Directed graph

Directed weighted graph

II. Operations to implement

II.1 Elementary algorithms

BFS

DFS

Topological sort

II.2 Minimum spanning tree

Prim

Kruskal

II.3 Shortest path algorithms

Bellman-Ford algorithms

Dijkstra's algorithm

Floyd-Warshall algorithm

Johnson's algorithm