Go, Lua, Scala를 이용하여 H-Index 계산, 정렬 구현- 프로그래밍 언어

고려대학교 정보대학 컴퓨터학과 2017320124 송동욱

목차

- 각 언어의 특징
  - Go
  - Lua
  - Scala
- 프로그램 설명
  - H-Index
    - 정의
    - 계산 알고리즘
  - Random Quick Sort
  - 프로그램 설명
    - Input
    - Output
      - Console Output
      - File Output
- 프로그램 비교
  - 출력문의 비교
  - 조건문의 비교
  - 반복문의 비교
  - 파일 입출력의 비교
    - 파일 입력
    - 파일 출력
  - 그 외 특징
    - Go 의 goroutine을 Quick Sort에 적용하기
    - Go 의 arrayToString function
    - Lua 의 slice function
    - IDE(or editor), 개발 버전
- 결과 스크린샷(console)
  - Go
  - Lua
  - Scala
- Github repository
- 참고 문헌

각 언어의 특징

Go

Go 프로그래밍 언어는 프로그래머의 생산성 향상을 위하여 만들어진 오픈 소스 프로젝트입니다. Go는 expressive하고, 간결하며 깨끗하며 효율적입니다. Cocurrency mechanism은 멀티 코어와 멀티 네트워크를 활용하는 프로그램을 쉽게 짤 수 있도록 도와주며, Novel type system은 유연함과 모듈 프로그램 구성을 가능하게 합니다. Go는 기계 코드로 빠르게 컴파일되지만 garbage collection의 편의성과 run-time reflection기능을 제공합니다. 동적으로 interpret되는 언어처럼 느껴지겠지만, 빠르게 동작하는 정적 타입의 컴파일 언어입니다.

이 언어는 2009년에 구글이 만든 프로그래밍 언어입니다. Robert griesemer, Rob Pike, Ken Thompson에 의해 디자인되었습니다. 위의 특징은 Go Document에 명시된 Golang의 특징입니다. 현재 구글, 드롭박스, 클라우드플레어, 넷플릭스 등에서 사용하고 있는 언어입니다.

Lua

Lua는 강력하고 효율적이며 가볍고 내장 가능한 스크립팅 언어입니다. Procedure programming, object-oriented programming, functional programming, data-driven programming 그리고 data description을 지원합니다. Lua는 간단한 Procedural syntax 그리고 associative array와 extensible semantic을 기반으로 하는 data description 구조를 결합합니다. Lua는 동적으로 타이핑되고, 레지스터 기반의 가상 머신을 기반으로 바이트코드를 해석하여 실행됩니다. Incremental garbage collection을 이용하여 자동 메모리 관리를 합니다.

Scala

스칼라는 객체지향 프로그래밍 언어와 함수형 프로그래밍의 요소가 결합된 다중패러다임 프로그래밍 언어이다. 기존의 Java언어가 너무 복잡하다는 단점을 극복하기 위하여 2004년 Martin Odersky가 처음 개발하여 배포하였다. 간결한 소스 코드를 사용하여 Java에서 구현할 수 있는 대부분의 기능을 구현할 수 있다. Scala는 자바 바이트코드를 자용하기 때문에 자바 가상 머신에서 실행할 수 있고, Java언어와 호환되어 대부분의 자바 API를 그대로 사용할 수 있다.

프로그램 설명

먼저 구현할 큰 키워드인 H-Index와 Quick Sort의 정의에 대해 살펴보고, 프로그램의 Input과 Output에

대해서 설명한다.

H-Index

정의

h-index는 과학자의 생산성과 영향력을 알아보기 위한 지표이다. 과학자 이외에도 과학자 집단, 대학, 혹은 집단에게도 적용 가능하다. 정의는 다음과 같다. 어느 과학자의 h개 논문이 h번 이상 인용되고 나머지 논문들은 h번 미만 인용되었다면, 그 과학자의 h-index는 h이다.

계산 알고리즘

h-index 계산의 input과 output은 다음과 같다.

- input - 논문 피인용수가 기록된 리스트
- output - h-index

그리고 h-index를 계산하는 방법은 다음과 같다. 1. 인용 횟수를 저장한 배열에 대해 내림차순으로 정렬한다. 2. 인용 횟수가 인용된 논문의 수보다 작아지는 순간이 H-Index보다 1 커지는 순간이며, 고로 그 전의 index를 return 하면 답을 도출할 수 있다.

그것을 pseudo code로 작성하면 다음과 같다.

```
sortedArray = descentSort(citationList)
for(i=0 to sortedArray.size, i++){
  if(sortedArray[i]<i+1){
    return i;
  }
}
return sortedArray.size
```

H-Index를 구하는 데 있어 배열을 내림차순으로 정렬하는 과정이 포함되어 있다. 각각 언어들의 소팅 메소드가 존재하지만, 프로그램의 특징을 자세히 보여주기 위하여 정렬을 구현하였다. 아래는 구현한 퀵 정렬에 대한 설명이다.

Random Quick Sort

퀵 소트는 정렬 알고리즘의 하나로, 다른 원소와의 비교만으로 정렬을 수행하는 비교 정렬이다. 퀵 정렬은 n개의 데이터를 정렬할 때, 최악의 경우에는 O(n^2)번의 비교를 수행하고, 평균적으로 O(nlogn)번의 비교를 수행한다. 최악의 경우를 피하기 위해 피봇 값을 랜덤 값으로 설정해주는 것이 Random Quick Sort이다.

1. 리스트 가운데서 하나의 원소를 고른다. 이렇게 고른원소를 피봇이라고 한다.
2. 피봇 앞에는 피봇보다 작은 모든 원소들이 오고, 피봇 뒤에는 피봇보다 값이 큰 모든 원소들이 오도록 피봇을 기준으로 리스트를 둘로 나눈다. 이렇게 리스트를 둘로 나누는 것을 분할이라고 한다. 분할을 마친 뒤에 피봇은 더이상 움직이지 않는다.
3. 분할된 두 개의 작은 리스트에 대해 재귀적으로 이 과정을 반복한다. 재귀는 리스트의 크기가 0이나 1이 될 때까지 반복한다.

프로그램 설명

프로그램은 유저들의 배열을 받아 각 유저들의 h-index를 계산하고, 그 h-index를 기반으로 유저들을 정렬한다. 프로그램의 input과 output은 아래와 같다.

Input

프로그램은 Json(JavaScript Object Notation, attribute-value pairs and array data types(or any other serializable value)또는 또는 키-값 쌍으로 이루어진 문자 기반의 표준 포맷) 파일을 받는다. (Scala의 경우 객체를 생성한다.)

Json의 속성-값 쌍 예시는 다음과 같다. 기본적으로 Json은 User List를 담고 있다. 그에 따른 User의 속성-값 쌍을 가진다.

```
"users": [
    {
      "name": "Alice",
      "age": 26,
      "citationList": [3, 0, 6, 1, 5]
    },
```

users 배열에서 각각의 원소들은 name, age, citationList의 값들을 가진다. name은 String형이며, 이름을 의미한다. age는 Int형이며, 나이를 의미한다. citationList은 Int형의 배열이며, User의 논문 당 인용된 횟수의 배열을 의미한다. Go와 Lua의 경우 json형식을 그대로 파일으로 받아 파싱하였으며, Scala의 경우에는 바로 User객체를 생성하였다. 그 대신 이름과 학번이 담긴 userInfo.txt를 읽고 이를 출력하였다.

Output

Console Output

콘솔에서 프로그램은 json파일을 파싱하여 나온 유저들의 정보(name, age, citationList)와, 프로그램의 목적인 h-index를 출력한다.
출력 예시는 다음과 같다.

> User Name : Oliver
> User Age : 20
> User Citation List
> [17 12 11 10 9 4 4 1]
> Oliver 의 h-index는 5

File Output

세 언어로 쓰인 각각의 프로그램 모두, .txt파일을 output으로 제공한다. 이 파일에서는, 유저들의 h-index를 기준으로 유저를 정렬하여 출력한다. 파일의 예시는 다음과 같다.

> 1 번째 : Eva이고, H-Index는 10
> 그의 논문 피인용수는 19,17,17,17,16,15,14,13,13,11,10,9,9,7,2
>
> 2 번째 : Elin이고, H-Index는 9
> 그의 논문 피인용수는 19,17,15,15,14,14,13,10,10,7,6,5,4,4,1

프로그램의 비교

출력문의 비교

다음은 각각의 언어에서 사용한 출력문에 대한 캡쳐사진이다.
Go의 경우 fmt에서 제공하는 표준 출력 함수를 사용하였으며, 위에서 사용한 Println과 Printf등의 출력 메소드는 아래와 같다. C의 printf와 scanf와 유사한 함수를 제공한다. 더 자세한 명세는 [Go document]: https://golang.org/pkg/fmt/ 에서 확인할 수 있다.

- func Print(a …interface{}) (n int, err error): 값을 그 자리에 출력(새 줄로 넘어가지 않음)
- func Println(a …interface{}) (n int, err error): 값을 출력한 뒤 새 줄로 넘어감(개행)
- func Printf(format string, a …interface{}) (n int, err error): 형식을 지정하여 값을 출력

Go 출력문 특징은 자동으로 toString형태로 변환되지 않다는 것이다. 그래서 String과 int를 동시에 출력할 때는 위에서 사용한 것처럼 strconv.Itoa(n Int)를 이용하여 Int를 String으로 변환해 주어야 한다.

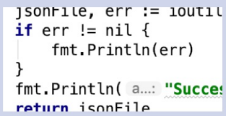
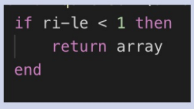
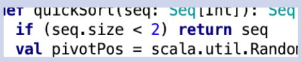
Lua는 프로그램에서 print함수와 write함수를 사용하였다. print는 개행이 있으며, 개행하고 싶지 않을 경우 io.write()함수를 이용하였다. print문에서 .. 으로 string을 연결할 수 있다.

Scala역시 유사한 형태의 출력 함수들을 내장하고 있으며, 한 가지 특징은 변수를 출력함에 있어 ${}를 이용하여 바로 변수를 출력할 수 있다. 스칼라 도큐먼트에 따르면 스칼라에서 제공하는 출력 관련 함수는 다음과 같다.

- def flush() : Unit
  - flushed the output stream
- def print(obj: Any):Unit
  - Prints an object to out using its toString method
- def printf(text:String, args:Any*):Unit
  - Prints its arguments as a formatted string to the default output, based on a string pattern
- def println(x:Any):Unit
  - Prints out an object to the default output, followed by a newline character

조건문의 비교

조건문은 세 언어에서 크게 다른 점이 없다. 다음은 프로그램에서 각각 if문을 사용했을 때를 캡쳐한 것이다.

Go	Lua	Scala
		

Go에서는 if에서 조건에 괄호가 없다. 하지만 {}는 필요하다. Go에서는 삼항연산자가 존재하지 않기 때문에, 간단한 연산자라도 전체 if문을 사용하여야 한다. 조건문으로 Switch를 제공한다. if문에 관한 문서와 switch에 관한 문서는
[Go if document]: https://golang.org/doc/effective_go.html#if 와
[Go if document]: https://golang.org/doc/effective_go.html#switch 이다.

Lua의 경우 역시 크게 다르지 않다. lua의 특징은 {}대신 then 과 end를 사용한다는 것이다. 기본 틀은 다음과 같다.

```
if op == "+" then
  r = a + b
elseif op == "-" then
  r = a - b
elseif op == "*" then
  r = a * b
elseif op == "/" then
  r = a/b
```

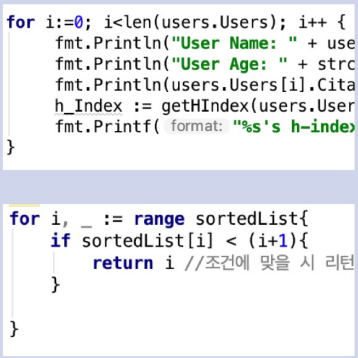
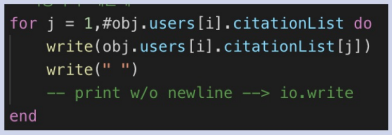
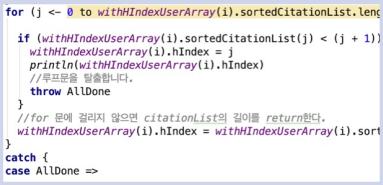
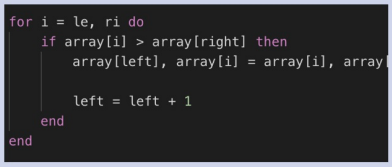
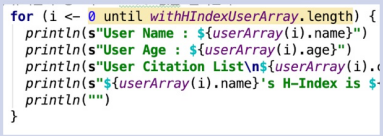
```
    else
      error("invalid operation")
    end
```

Lua의 if문에 관한 문서는
[Lua if document]:https://www.lua.org/pil/4.3.1.html 에서 확인할 수 있다.

Scala의 if문 역시 크게 다르지 않다. 한 줄의 실행문은 {}를 생략할 수 있고, 여러 줄은 {}가 필요하다. 한 가지 특징은 if문도 expression이기 때문에 if문으로 삼항연산자를 대신할 수 있다는 점이다.

반복문의 비교

반복문은 언어마다의 특징이 존재한다. 다음은 각각의 언어에서 반복문을 사용한 것을 캡쳐한 사진이다.

Go	Lua	Scala
		
		

Go에서 반복문은 for루프를 이용한다. Go는 for이 유일한 반복문이다. 다른 언어와 유사하게 for 초기값;조건식;증감{...}의 형식을 따른다. 또한 상황에 따라 각각을 생략할 수 있다. 이것을 둘러싸는 괄호를 생략하는데, ()를 사용할 시 에러가 난다. break를 이용하여 탈출할 수 있다.

```
//Like a C for
for init; condition; post {
}
//Like a C while
for condition {
}
```

초기값과 증감식을 생략하고 조건식만 for문에 사용하게 되면 다른 언어의 while루프와 같게 동작할 수 있다. 모두를 생략하게 되면 무한루프를 만들 수 있다.

```
for key, value := range oldMap {
    newMap[key] = value
```

```
}
for key := range m {
    if key.expired() {
        delete(m, key)
    }
}
sum := 0
for _, value := range array {
    sum += value
}
```

for range문이 존재하는데, key와 value값을 이용하여 loop문에 사용할 수 있다. key값만 필요하다면 value를 생략할 수 있으며, value값만 필요하다면 `_,value` 와 같은 식으로 사용할 수 있다.

관련된 문서는 [Go for document]:https://golang.org/doc/effective_go.html#for 에서 확인할 수 있다.

Lua의 for statement는 두 가지가 있다. : numeric for과 generic for

numeric for의 syntax는 다음과 같다. 위 프로그램에서 사용한 for문은 numeric for이다. Go 의 for 과 유사하다.

```
for var = exp1, exp2, exp3 do
  something
end
--example
for i=10, 1, -1 do print(i) end
```

Generic for의 syntax는 다음과 같다.

```
-- print all values of array 'a'
for i,v in ipairs(a) do print(v) end
```

Lua는 for 이외에도 while 문과 repeat문을 제공한다. while문은 일반적으로, while condition을 확인하여 condition이 false이면 loop가 종료되고, 그렇지 않으면 body를 실행하고 이 과정을 반복한다. repeat-until state는 condition이 true일 때까지 body를 반복한다. while과 다른 점은 body가 한번은 실행된다는 것이다. 다른 언어에서의 do while문과 비슷하다.

관련된 문서는 [Lua numeric for document]:https://www.lua.org/pil/4.3.4.html

에서 확인할 수 있다.

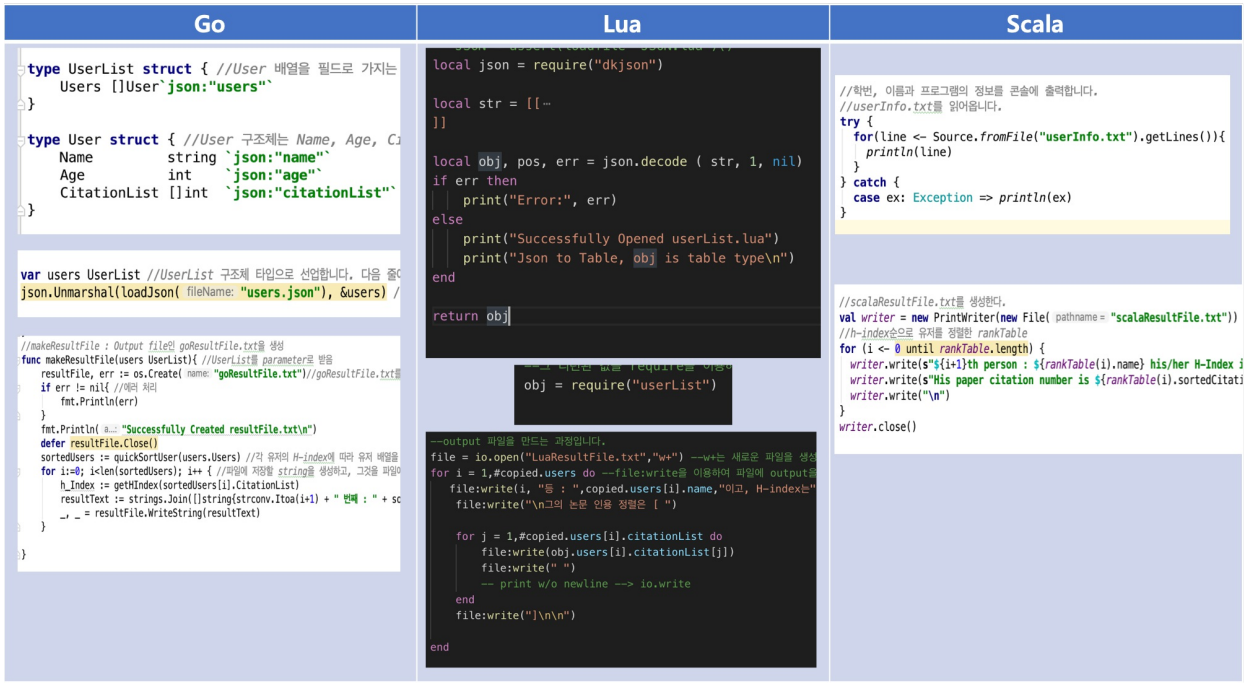
Scala는 세 언어중 가장 다양한 형태의 for문의 조건식을 작성할 수 있다.

```scala
// <-
val languages = List("Java", "Scala", "#F")
for (language <- languages)
  println(language)
// to, until
for (i <- 0 to 5)
  print(i) //0,1,2,3,4,5
for (i <=0 until 5)
  print(i) //0,1,2,3,4
//range
for (i<- Range(5,8))
  print(i) //5,6,7
for (i<- Range(0,5,2)) //step of 2
  print(i) //0,2,4
```

until과 to 의 차이점은 마지막 값의 포함 유무이다. until은 마지막 값이 포함되지 않고, to는 마지막 값이 포함된다.이 외에도 while 문과 do while이 존재하는데, C 언어를 포함한 다른 언어와 유사하다.

파일 입출력의 비교

다음은 각각의 언어에서 file을 읽고, 값을 file에 쓰는 과정을 캡쳐한 것이다. Scala의 경우 Json파일을 파싱하는 데 어려움이 있어 input data는 object로 바로 생성하고, 학생의 정보가 적힌 txt 파일을 읽어왔고, Output File은 동일하게 진행하였다.



파일 입력

Go는 파일을 읽고 쓰기 위해 표준 패키지인 os 패키지를 제공한다. `os.Open()` 함수는 기존 파일을 열 때 사

용하고, `os.Create()` 는 새 파일을 생성할 때 사용한다. Open, Create함수는 File타입을 리턴하는데, 이 file타입의 메서드인 Read(), Write()을 사용하여 파일을 읽고 쓸 수 있다.

```
import(
  "io"
  "os"
  )
//파일 열기//
fi,err := os.Open("1.txt")
fi.Close()
//파일 만들기
fo,err := os.Create("2.txt")
fo.Close()
//파일 읽기
buff := make([]byte, 1024)
cnt, err := fi.Read(buff)
//파일에 쓰기
_, err = fo.Write(buff[:cnt])
```

이상으로 Golang에서 기본적인 파일 입력이었다. 추가로 JSON파일을 읽어오기 위해 다음 정보들을 이용하였다. Go는 언어에서 JSON의 encoding/decoding을 지원한다. 프로그램에서는 오직 decoding만을 사용하였다. Marshal function을 이용하여 decoding을 진행하였다.

```
//to Encode
//func Marshal(v interface{}) ([]byte, error)
type Message struct {
    Name string
    Body string
    Time int64
}
m := Message{"Alice", "Hello", 1294706395881547000}
b, err := json.Marshal(m)
b == []byte(`{"Name":"Alice","Body":"Hello","Time":1294706395881547000}`)
//true

//to Decode
//func Unmarshal(data []byte, v interface{}) error
var m Message
err := json.Unmarshal(b, &m)
m = Message{
    Name: "Alice",
    Body: "Hello",
    Time: 1294706395881547000,
}
```

관련된 문서는 [Go blog - json]:https://blog.golang.org/json-and-go 에서 확인할 수 있다.

```
file = io.open("sample.txt", "r")
if file then
  local data = file:read("*all")
  file:close()
  print(data)
end
```

Lua에서 file을 여는 방법은 다음과 같다. `io.open` 함수는 file name과 option을 받는다. 본 프로그램에서는 "w+"를 이용하였는데, write전용으로 열고 +는 항상 새로운 파일을 만드는 것이다. 옵션에 관한 추가적인 정보는 [Lua io.open document]:https://www.lua.org/manual/5.3/manual.html#pdf-io.open 에서 확인할 수 있다.

Lua는 언어에서 JSON형식을 지원하지 않기 때문에 외부 라이브러리를 import하는 방식을 사용하였다. dkjson이라는 라이브러리를 이용하여 json 형태를 table로 파싱할 수 있다. userList.lua에서 str이 json형식으로 되어 있고, obj가 파싱한 결과이다. userList.lua가 obj를 return 하였고, main.lua에서 `obj = require("userList")` 를 이용하여 obj에 파싱된 table을 담았다.

Scala의 파일 읽기는 다음과 같다. Java의 File class와 비슷한 Source class가 존재한다. 값을 읽고 쓰는 방식이 Java의 Files와 유사하다.

```
/* file reading example */
import scala.io.Source
object fileReadObj {
    def main(args: Array[String]){
        try {
            for(line <- Source.fromFile("/home/path/path.txt").getLines()){
                println(line)
            }
        } catch {
            case ex: Exception => println(ex)
        }
    }
}
```

파일 출력

Go의 파일 출력 역시 입력 크게 다르지 않다. 다음은 파일을 읽는 대표적인 방법이다.

```
import(
  "io"
  "os"
  )
//파일 만들기
fo,err := os.Create("2.txt")
fo.Close()
```

```
//파일에 쓰기
_, err = fo.Write(buff[:cnt])
```

Lua의 파일 출력은 io class와 그 메소드를 이용하여 이루어진다. 다음은 파일을 열어 값을 출력하는 예제이다.

```
local write = io.write
file = io.open("sampleFile.txt", "w+")
-- +가 없을 경우 기존 파일에 추가로 이어서 출력
file:write("hello Lua")
```

Scala의 Source Object는 간단한 문자열 파일을 읽어오는데는 좋지만 Java의 File만큼 다양한 기능을 제공하지 않기 때문에 자바의 File클래스를 사용하기도 한다. Source Object는 파일 쓰기를 지원하지 않기 때문에 Java의 클래스를 사용하여 파일 출력을 하여야 한다.

```
import java.io._
val writer = new PrintWriter(new File("writeFile.txt"))
writer.write("\n")
writer.close()
```

그 외 비교

Go 의 goroutine을 Quick Sort에 적용하기

Go언어의 특징 중 하나인 goroutine을 다루어 보고자 한다. Go루틴(goroutine)은 Go Runtime이 관리하는 논리적 쓰레드이다. 함수를 동시에 실행시키는 기능이라고 할 수 있다. Go에서 `go` 키워드를 사용하여 함수를 호출하면, 런타임시 새로운 goroutine을 실행한다. 고루틴은 다른 언어의 스레드 생성 방법보다 문법이 간단하고, 운영체제의 쓰레드보다 훨씬 가볍게 비동기 Cocurrent처리를 하기 위해 만들어졌다. 다음은 go를 사용한 간단한 예제이다.

```
package main

import (
    "fmt"
    "sync"
)

func main() {
    // WaitGroup 생성. 2개의 Go루틴을 기다림.
    var wait sync.WaitGroup
    //여러 Go루틴이 끝날 때까지 기다리는 역할
```

```
    wait.Add(2)
    //두 개의 Go루틴을 기다릴 것임

    // 익명함수를 사용한 goroutine
    go func() {
        defer wait.Done() //끝나면 .Done() 호출
        fmt.Println("Hello")
    }()

    // 익명함수에 파라미터 전달
    go func(msg string) {
        defer wait.Done() //끝나면 .Done() 호출
        fmt.Println(msg)
    }("Hi")

    wait.Wait() //Go루틴 모두 끝날 때까지 대기
}
```

본 프로그램에서, 퀵 소트에 goroutine을 적용해 보았다. 프로그램에는 각 언어들과의 알고리즘 유사성을 위해 적용시키지 않았지만, 코드를 다음에 명시하고 설명하고자 한다. 간단한 in-place 퀵 소트를 구현하는 방법이다. 본 프로그램은 한 함수안에서 partition을 진행하였지만, partition을 다른 함수로 분리시킨 퀵 소트를 구현한다. partition함수는 랜덤 피봇을 고르고 배열을 재배열하는 역할을 한다. quick sort의 주된 수행은 Partition함수에서 이루어진다. 다음은 goroutine이 적용되지 않고, partition함수를 사용하는 퀵 소트이다.

```
func quickSort(s []int) {
    if len(s) <= 1 {
        return
    }
    pivot := partition(s)
    quickSort(s[:pivot+1])
    quickSort(s[pivot+1:])
    return
}
```

퀵 소트의 분할 및 정복은 동일한 메모리에 접근하지 않으므로 병렬 처리가 가능함을 의미한다. MAXGOROUTINE을 설정해 주지 않으면 과도하게 많은 고루틴이 생성되어 속도가 저하되므로, 이를 설정한 코드는 다음과 같다.

```
func coQuick(s []int) {
    if len(s) <= 1 {
        return
    }
    workers := make(chan int, MAXGOROUTINES-1)
    for i := 0; i < (MAXGOROUTINES - 1); i++ {
        workers <- 1
    }
    coQuick(s, nil, workers)
```

```
}

func cqsort(s []int, done chan int, workers chan int) {
    // report to caller that we're finished
    if done != nil {
        defer func() { done <- 1 }()
    }
    if len(s) <= 1 {
        return
    }
    doneChannel := make(chan int, 1)
    pivotIdx := partition(s)
    select {
    case <-workers:
        go cqsort(s[:pivotIdx+1], doneChannel, workers)
    default:
        cqsort(s[:pivotIdx+1], nil, workers)
        doneChannel <- 1
    }
    cqsort(s[pivotIdx+1:], nil, workers)
    <-doneChannel
    return
}
```

Go의 arrayToString function

Go는 write함수를 사용할 때 toString을 지원하지 않는다. 파일에 output을 출력하는 과정에서 Array를 String으로 변환하기 위한 함수를 선언하여 사용하였다. 다음과 같다.

```
func arrayToString(a []int, delim string) string {
    return strings.Trim(strings.Replace(fmt.Sprint(a), " ", delim, -1), "[]")
```

Lua의 slice function

Lua의 Table은 파이썬의 slice와 같은 syntax를 지원하지 않기 때문에, 프로그램 내에서 slice를 구현하였다. 아래는 그 코드이다. 퀵 소트에서 테이블을 분할함에 있어 아래 함수를 이용하였다.

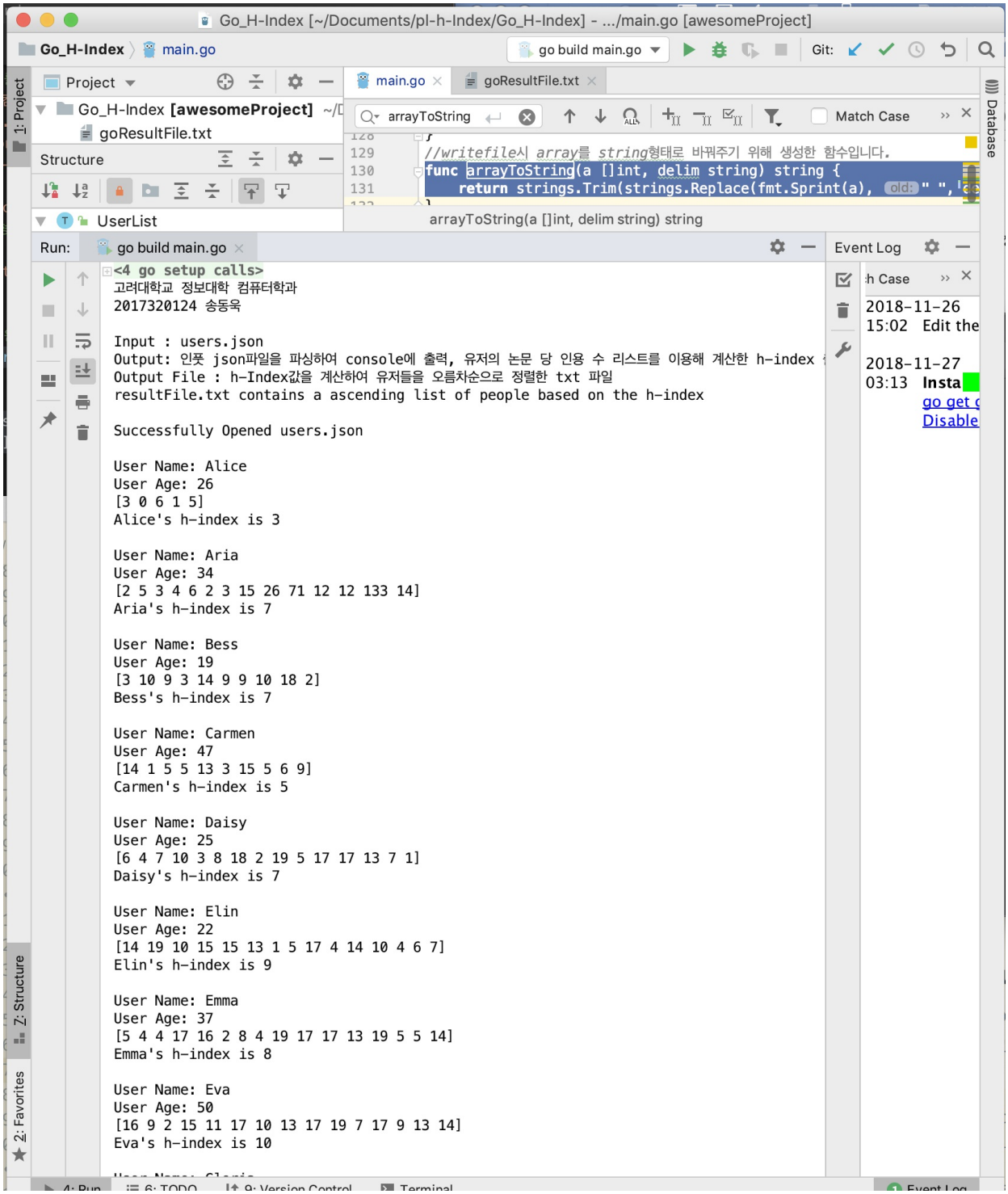
```
function slice(tbl, first, last)
    local sliced = {}

    for i = first , last   do
      sliced[#sliced+1] = tbl[i]
    end

    return sliced
end
```

IDE(or editor), 개발 버전

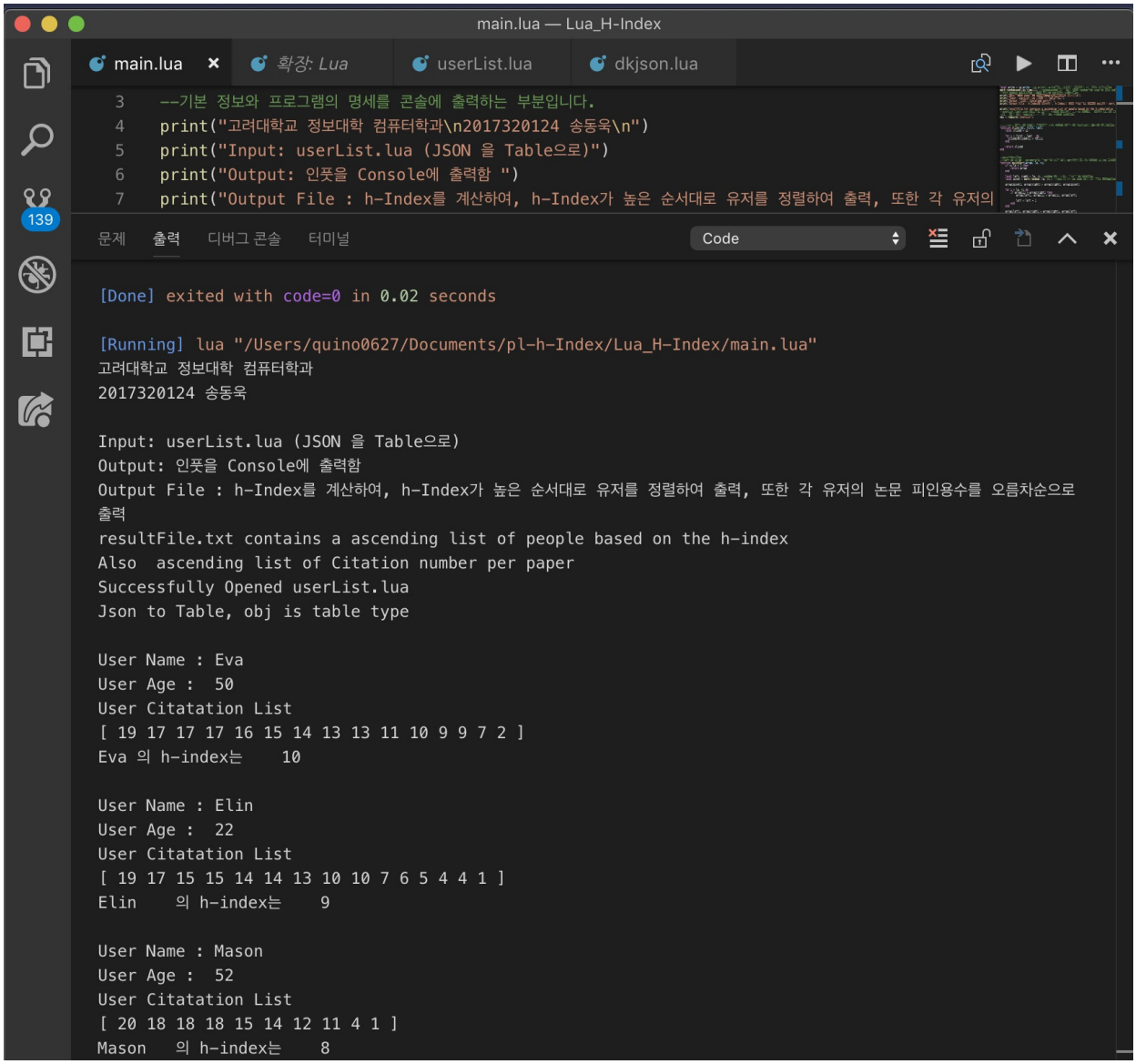
Go : go version1.11.2, GoLand (on Mac)
Lua : Lua 5.3.5, VSCode (on Mac)
Scala : Scala 2.12 IntelliJ IDEA CE (on Mac and Windows10)

결과 스크린샷(Console)

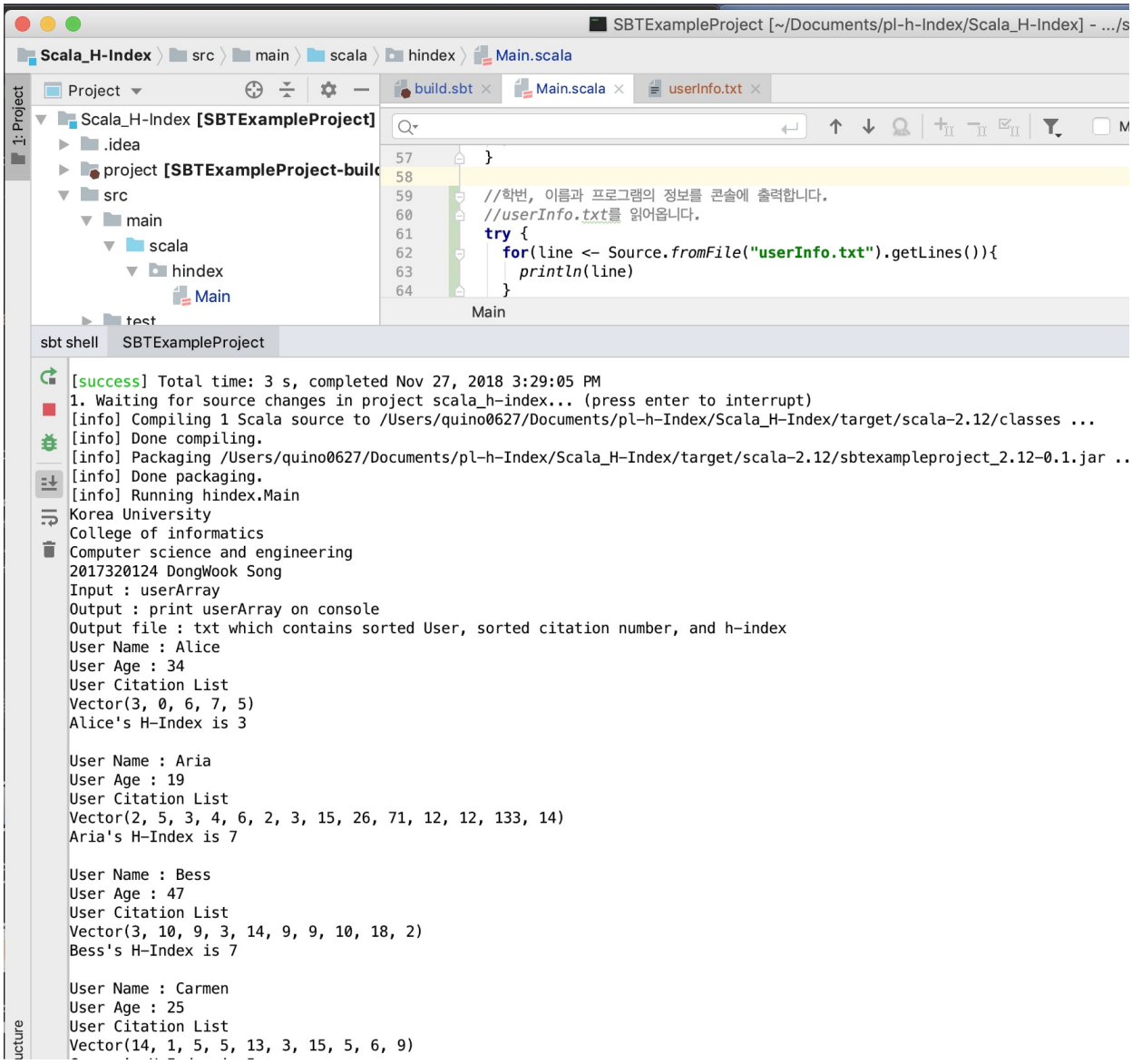
Go


Go_H-Index [~/Documents/pl-h-Index/Go_H-Index] - .../main.go [awesomeProject]
Go_H-Index
main.go
go build main.go
Git:
Project
Go_H-Index [awesomeProject]
goResultFile.txt
Structure
UserList
main.go
goResultFile.txt
arrayToString
Match Case
//writefile시 array를 string형태로 바꿔주기 위해 생성한 함수입니다.
func arrayToString(a []int, delim string) string {
return strings.Trim(strings.Replace(fmt.Sprint(a),
arrayToString(a []int, delim string) string
Run:
go build main.go
Event Log
<4 go setup calls>
고려대학교 정보대학 컴퓨터학과
2017320124 송동욱
Input : users.json
Output: 인풋 json파일을 파싱하여 console에 출력, 유저의 논문 당 인용 수 리스트를 이용해 계산한 h-index
Output File : h-Index값을 계산하여 유저들을 오름차순으로 정렬한 txt 파일
resultFile.txt contains a ascending list of people based on the h-index
Successfully Opened users.json
User Name: Alice
User Age: 26
[3 0 6 1 5]
Alice's h-index is 3
User Name: Aria
User Age: 34
[2 5 3 4 6 2 3 15 26 71 12 12 133 14]
Aria's h-index is 7
User Name: Bess
User Age: 19
[3 10 9 3 14 9 9 10 18 2]
Bess's h-index is 7
User Name: Carmen
User Age: 47
[14 1 5 5 13 3 15 5 6 9]
Carmen's h-index is 5
User Name: Daisy
User Age: 25
[6 4 7 10 3 8 18 2 19 5 17 17 13 7 1]
Daisy's h-index is 7
User Name: Elin
User Age: 22
[14 19 10 15 15 13 1 5 17 4 14 10 4 6 7]
Elin's h-index is 9
User Name: Emma
User Age: 37
[5 4 4 17 16 2 8 4 19 17 17 13 19 5 5 14]
Emma's h-index is 8
User Name: Eva
User Age: 50
[16 9 2 15 11 17 10 13 17 19 7 17 9 13 14]
Eva's h-index is 10
2018-11-26
15:02 Edit the
2018-11-27
03:13 Insta
go get
Disable
1: Project
2: Favorites
Z: Structure
Database

Lua


main.lua — Lua_H-Index
main.lua
확장: Lua
userList.lua
dkjson.lua
3 --기본 정보와 프로그램의 명세를 콘솔에 출력하는 부분입니다.
4 print("고려대학교 정보대학 컴퓨터학과\n2017320124 송동욱\n")
5 print("Input: userList.lua (JSON 을 Table으로)")
6 print("Output: 인풋을 Console에 출력함 ")
7 print("Output File : h-Index를 계산하여, h-Index가 높은 순서대로 유저를 정렬하여 출력, 또한 각 유저의
문제
출력
디버그 콘솔
터미널
Code
[Done] exited with code=0 in 0.02 seconds
[Running] lua "/Users/quino0627/Documents/pl-h-Index/Lua_H-Index/main.lua"
고려대학교 정보대학 컴퓨터학과
2017320124 송동욱
Input: userList.lua (JSON 을 Table으로)
Output: 인풋을 Console에 출력함
Output File : h-Index를 계산하여, h-Index가 높은 순서대로 유저를 정렬하여 출력, 또한 각 유저의 논문 피인용수를 오름차순으로
출력
resultFile.txt contains a ascending list of people based on the h-index
Also ascending list of Citation number per paper
Successfully Opened userList.lua
Json to Table, obj is table type
User Name : Eva
User Age : 50
User Citatation List
[19 17 17 17 16 15 14 13 13 11 10 9 9 7 2]
Eva 의 h-index는 10
User Name : Elin
User Age : 22
User Citatation List
[19 17 15 15 14 14 13 10 10 7 6 5 4 4 1]
Elin 의 h-index는 9
User Name : Mason
User Age : 52
User Citatation List
[20 18 18 18 15 14 12 11 4 1]
Mason 의 h-index는 8

Scala



Github Repository

본 프로그램들에 대한 github repository는 다음과 같다. 각각의 폴더 안에 input file과 output file이 포함되어 있고, code역시 존재한다. 2017320124_송동욱.md파일은 보고서이다.

https://github.com/quino0627/pl-h-Index

참고 문헌

https://ko.wikipedia.org/wiki/JSON
https://golang.org/doc/
https://blog.golang.org

https://www.lua.org/docs.html
https://docs.scala-lang.org
https://ko.wikipedia.org/wiki/Go_(프로그래밍_언어)
https://ko.wikipedia.org/wiki/루아_(프로그래밍_언어)
https://ko.wikipedia.org/wiki/스칼라_(프로그래밍_언어)