Algorithms

HW#1 Report

2013-11706
기계항공공학부 강인재

Introduction to Algorithms 교재에 소개된 선형 시간 선택 알고리즘과 최악의 경우에도 선형 시간을 보장하는 선택 알고리즘을 C언어로 구현하였다. 선형 시간 선택 알고리즘의 경우, 교재에는 입력 배열의 가장 오른쪽 원소를 분할 과정에서 기준 원소로 정하였으나, 본 알고리즘에서는 (srand()로 랜덤화된) rand() 함수를 이용하여 임의로 정할 수 있도록 하였다.

알고리즘을 구현한 뒤, 다양한 입력 크기에 대해 걸리는 시간을 계산하였다. 그 결과는 다음과 같았다. (테스트는 윈도우 환경에서 진행되었다. 리눅스 환경에서도 계산해 본 결과, 리눅스에서 더 작은 값이 나왔다.)

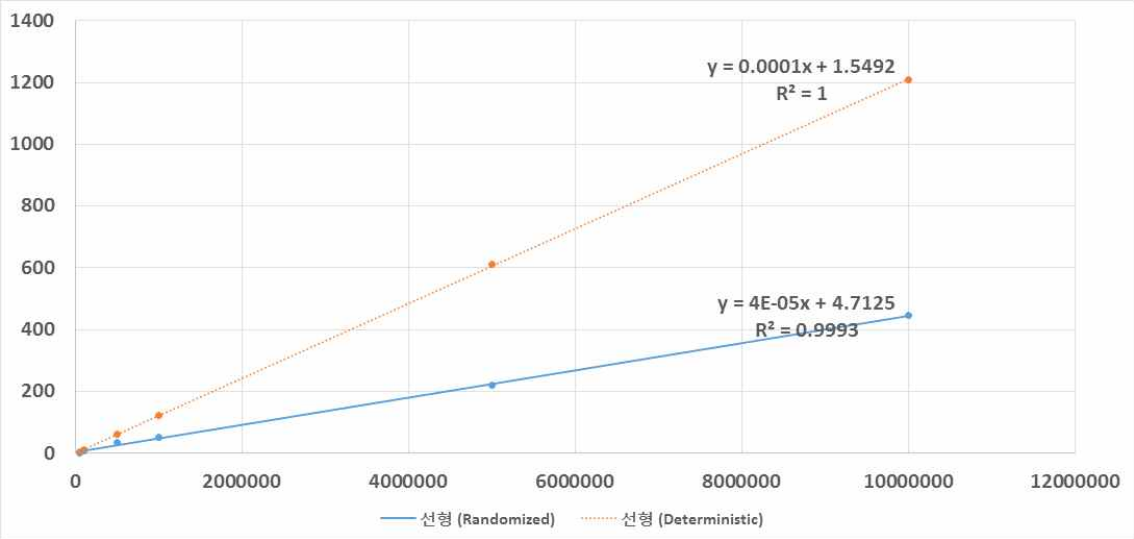
<Randomized-Select>

단위 : ms	5만	10만	50만	100만	500만	1000만
test 1	2	8	35	57	159	485
test 2	2	5	24	52	235	396
test 3	2	7	44	45	269	461

<Deterministic-Select>

단위 : ms	5만	10만	50만	100만	500만	1000만
test 1	6	12	59	123	610	1206
test 2	6	12	62	122	615	1212
test 3	6	12	62	128	608	1206

세 번 시행된 테스트 결과의 평균을 취한 뒤, 이를 그래프로 나타내어 입력의 크기와 수행 시간 사이에 어떤 관계식이 있는지 도출해 보았다. 그 결과는 다음과 같았다.



가로축(x)은 입력의 크기를, 세로축(y)은 수행 시간(단위 : ms)을 의미한다. 수업 시간에 살펴본 바와 같이, 추세선의 R^2 값이 모두 1에 아주 근접한 것으로 보아 두 알고리즘은 모두 $\Theta(n)$의 시간 복잡도를 가지고 있다고 볼 수 있다. 평균 선형 시간 알고리즘은 최악의 경우 성능이 더 나빠지지만, 이 실험은 임의의 입력값과 임의의 분할 기준 원소를 사용했기 때문에 평균의 경우를 나타내었다고 생각할 수 있다.

결과로부터 얻은 추세선의 관계식은 Randomized-Select의 경우 $y = 0.0001x + 1.5492$, Deterministic-Select의 경우 $y = 4{\cdot}10^{-5}x + 4.7125$였다. 일차항의 계수의 비를 계산하면 약 2.5로, 충분히 큰 입력의 경우 Deterministic의 시간이 Randomized의 2.5배 정도 걸리는 것을 알 수 있다. 이는 Deterministic-Select 알고리즘에서 '5개씩의 원소를 가진 서브그룹의 중앙값'들의 중앙값을 찾기 위한 오버헤드로부터 기인한다. 만일 이 오버헤드가 일반적인 선형 선택 알고리즘의 빈도를 고려한 최악의 경우로부터 오는 손해보다 작다면, Deterministic 알고리즘을 사용하는 편이 나을 것이다.

실험 결과로부터 한 가지 더 알 수 있는 사실은, Deterministic-Select에 비해 Randomized-Select의 결과값의 편차가 크다는 것이었다. 이는 Randomized-Select 알고리즘의 경우 최선과 최악의 경우 간의 시간차가 Deterministic-Select 알고리즘에 비해 크다는 것을 의미한다.

알고리즘의 무결성을 검증하기 위한 체크 프로그램을 구현하였다. 체크 프로그램은 입력 리스트 외에도 두 개의 값을 더 입력받는데, 하나는 몇 번째로 작은 수인지를 나타내고(이를 i라 하자) 다른 하나는 예상되는 결과값이다(이를 res라 하자). 체크 프로그램은 다음과 같이 동작한다. 먼저 아래와 같이 4개의 인수를 입력받는다.

```
./rdcheck i input res
```

3번째 인수인 input을 읽기 모드로 열어 저장되어 있는 정수를 차례대로 읽는다. 이 때,

count와 dup 변수를 이용하였다. count는 1로, dup는 0으로 초기화하였다. count는 읽어들인 값이 res보다 작을 경우 1씩 증가하고, dup는 읽어들인 값이 res와 같을 경우 1씩 증가한다. 즉, 파일의 끝까지 읽어들인 후 count는 res가 몇 번째로 작은 값인지를 나타내고, dup는 res와 같은 수가 몇 번 등장했는지를 나타낸다. 이후 dup가 0일 경우 res와 같은 수가 존재하지 않으므로 WRONG을 표시한다. dup가 1 이상이라면 res가 존재하는 범위를 계산하여 결과를 판단한다.

이 프로그램은 처음에 Visual Studio로 작성되어 윈도우의 cmd 창에서 검증 과정 및 시간 측정을 거쳤다. 이후 Linux 환경에서 소스 코드가 약간 수정되었다.

구현된 알고리즘은 책에서 구현된 것과 전반적으로 크게 다르지 않다. 다른 점은 Deterministic-Select 알고리즘에서 '중간값들의 중간값'을 구하는 과정을 재귀적으로 구현해야 하므로, 각각의 중간값들을 계산한 즉시 배열의 앞쪽부터 차례로 swap하여 연속되도록 하였다. 또한 중간값들의 중간값을 구하고 나서 분할하기 위해서는 그 값의 index를 알아야 하므로 배열의 앞쪽부터 차례대로 검색하여 index를 구하였다.(물론 다른 방법이 있을 수도 있겠지만, 분할의 기준 원소를 맨 오른쪽 원소와 교환하는 방법을 사용하기 위함이었다.)

[리눅스 환경에서 컴파일 방법]

rdselect(선택 알고리즘의 실행 파일)

```
gcc -o rdselect dselect.c rselect.c swap.c main.c
```

rdcheck(테스트 프로그램의 실행 파일)

```
gcc -o rdcheck rdcheck.c
```

[리눅스 환경에서 실행 방법]

rdselect 실행

```
./rdselect i input
```

(i는 몇 번째로 작은 수인지를 나타냄)

rdcheck 실행

```
./rdcheck i input res
```

(res는 결과를 알고 싶은 수를 나타냄)

Example Running

(test1_100K, test2_500K, test3_1000K, test4_5000K 이렇게 테스트 input 4개가 함께 들어 있다. Example Running은 test1_100K과 test4_5000K에 대해 검증한 결과이다.)

<test1_100K>

```
abcinje@martini:~$ ./rdselect 12345 test1_100K
*** Randomized select result ***
[12345]th smallest element : 1235466
Program running time : 1ms

*** Deterministic select result ***
[12345]th smallest element : 1235466
Program running time : 9ms
```

```
abcinje@martini:~$ ./rdcheck 12345 test1_100K 1235466
*** Check correctness of rdselect ***
1235466 is the [12345]th smallest element -> CORRECT
```

<test4_5000K>

```
abcinje@martini:~$ ./rdselect 2564854 test4_5000K
*** Randomized select result ***
[2564854]th smallest element : 51275220
Program running time : 132ms

*** Deterministic select result ***
[2564854]th smallest element : 51275220
Program running time : 410ms
```

```
abcinje@martini:~$ ./rdcheck 2564854 test4_5000K 51275220
*** Check correctness of rdselect ***
51275220 is the [2564854]th smallest element -> CORRECT
```