HW01

수리과학부 2019-16022 박채연

1. Measure the actual time usage of the two algorithms on inputs of various sizes and compute the ratio of the constants hidden in the asymtotic time complexities of the two algorithms.

randomized selection 알고리즘과 deterministic selection 알고리즘의 asymtotic time complexities 에서 숨겨진 상수비를 구하기 위해서는 두 selection 알고리즘의 실제 수행시간을 계산해야 한다. 이 때, N의 최댓값이 5,000,000으로 주어졌으므로 500,000부터 5,000,000까지 500,000 크기 단위로 총 10개의 input 파일을 생성하고 각 알고리즘별 수행시간을 계산하였다.

input 파일을 생성하는 소스 코드를 'generator.c'에 작성하고 'gcc -std=gnu99 -O2 generator.c -o generator' 명령어를 통해 컴파일하였다. 그 결과 generator라는 실행파일이 생성되는데, 이는 N과 M, input 파일 경로를 순서대로 인자로 입력받는다. 예를 들어 './generator 100 10 1.in' 명령어는 '1.in'이라는 input 파일에 N 값인 100과 M 값인 10을 첫번째 줄에 차례로 출력하고 다음 줄에 랜덤으로 생성된 100(=N)개의 정수를 출력하라는 의미이다. 해당 코드에 대한 구체적인 설명은 3번 문항에 제시하였다. 아래의 명령어를 통해 총 10개의 input 파일을 생성하였다.

```
# ./generator 500000 250000 1.in
# ./generator 1000000 500000 2.in
# ./generator 1500000 750000 3.in
# ./generator 2000000 1000000 4.in
# ./generator 2500000 1250000 5.in
# ./generator 3000000 1500000 6.in
# ./generator 3500000 1750000 7.in
# ./generator 4000000 2000000 8.in
# ./generator 4500000 2250000 9.in
# ./generator 5000000 2500000 10.in
```

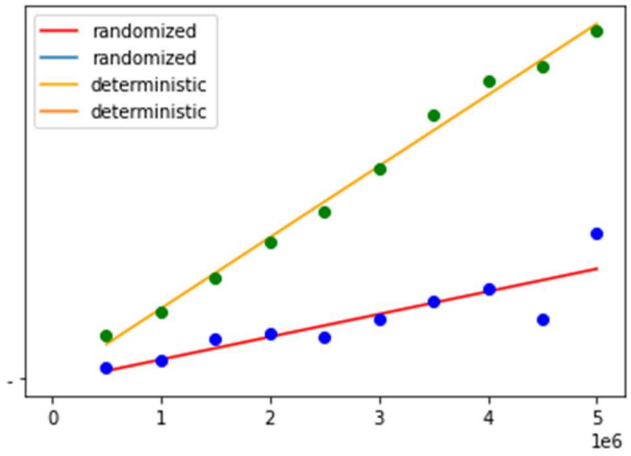
한편, randomized selection 알고리즘과 deterministic selection 알고리즘은 main.c 파일에 구현되어 있다. 'gcc -std=gnu99 -O2 main.c -o main' 명령어를 통해 컴파일하면 main 실행파일이 생성되는데 이는 실행모드, input 파일 경로, output 파일 경로를 순서대로 인자로 입력받는다. 이때, 실행모드가 1이면 randomized-select가, 2이면 deterministic-select가 수행된다.

앞서 생성한 10개의 input 파일에 대해 randomized-select와 deterministic-select를 수행하고 수행시간을 출력하면 다음과 같다.

```
# ./main 1 1.in 1.out
Total running time : 0.003437s
# ./main 1 2.in 2.out
Total running time : 0.006321s
# ./main 1 3.in 3.out
Total running time : 0.013969s
# ./main 1 4.in 4.out
Total running time : 0.015384s
# ./main 1 5.in 5.out
Total running time : 0.014315s
# ./main 1 6.in 6.out
Total running time : 0.020719s
# ./main 1 7.in 7.out
Total running time : 0.026691s
# ./main 1 8.in 8.out
Total running time : 0.031193s
# ./main 1 9.in 9.out
Total running time : 0.020757s
# ./main 1 10.in 10.out
Total running time : 0.051179s
```

```
# ./main 2 1.in 1.out
Total running time : 0.014897s
# ./main 2 2.in 2.out
Total running time : 0.023453s
# ./main 2 3.in 3.out
Total running time : 0.035070s
# ./main 2 4.in 4.out
Total running time : 0.047828s
# ./main 2 5.in 5.out
Total running time : 0.058179s
# ./main 2 6.in 6.out
Total running time : 0.073377s
# ./main 2 7.in 7.out
Total running time : 0.092407s
# ./main 2 8.in 8.out
Total running time : 0.104330s
# ./main 2 9.in 9.out
Total running time : 0.109302s
# ./main 2 10.in 10.out
Total running time : 0.122282s
```

위에서 제시한 randomized_selection 알고리즘과 deterministic_selection 알고리즘의 수행시간을 파이썬의 matplotlib를 통해 산점도로 나타내고 추세선을 그리면 아래와 같다.



randomized selection 알고리즘 수행시간의 추세선은 y=7.96575758e-09*x+-1.50933333e-03이고, deterministic selection 알고리즘 수행시간의 추세선은 y=2.50017333e-08*x-6.42266667e-04이다. 따라서 randomized selection 알고리즘과 deterministic selection 알고리즘의 asymtotic time complexities에서 숨겨진 상수비는 2.50017333e-08/7.96575758e-09 로, 약 3.14이다.

한편, 10개 input 파일에 대해 randomized-select와 deterministic-select을 수행하고 아래와 같이 정답이면 1을, 오답이면 0을 출력하는 checker program을 실행한 결과 아래와 같이 두 경우 모두 정답을 출력하는 모습을 확인할 수 있었다.

```
# ./checker 1.in 1.out
1
# ./checker 2.in 2.out
1
# ./checker 3.in 3.out
1
# ./checker 4.in 4.out
1
# ./checker 5.in 5.out
1
# ./checker 6.in 6.out
1
# ./checker 7.in 7.out
1
# ./checker 8.in 8.out
1
# ./checker 9.in 9.out
1
# ./checker 10.in 10.out
1
```

```
# ./checker 1.in 1.out
1
# ./checker 2.in 2.out
1
# ./checker 3.in 3.out
1
# ./checker 4.in 4.out
1
# ./checker 5.in 5.out
1
# ./checker 6.in 6.out
1
# ./checker 7.in 7.out
1
# ./checker 8.in 8.out
1
# ./checker 9.in 9.out
1
# ./checker 10.in 10.out
1
```

2. Explain how your checker program works.

'checker.c'파일에 Checker 프로그램을 구현하고 'gcc -std=gnu99 -O2 checker.c -o checker' 명령어를 통해 컴파일하였다. 그 결과 checker 실행파일이 생성되는데 이는 input 파일 경로와 output 파일 경로를 순서대로 인자로 입력받는다. 즉, input 파일과 output 파일 경로가 각각 './1.in', './1.out'일 때, './checker ./1.in ./1.out' 명령어를 통해 실행할 수 있다.

```
28      //We should check whether the integer in the output file is the correct answer for the given input file.
29      c = fscanf(fpout, "%d", &target);
30      c = fscanf(fpin, "%d", &N);
31      c = fscanf(fpin, "%d", &M);
32
33      //If each integer in the given input file is smaller than 'target', then increment the 'smallcnt' by 1.
34      //If each integer in the given input file is equal to 'target', then increment the 'samecnt' by 1.
35      for (int i=0;i<N;i++){
36          c = fscanf(fpin, "%d", &num);
37          if (num<target)
38              smallcnt++;
39          else if (num==target)
40              samecnt++;
41      }
```

checker 프로그램 내부에서는 우선 input 파일과 output 파일을 open하고 만약 에러가 발생하면 프로그램을 종료한다. 다음으로 위의 스크린샷 이미지와 같이 fscanf()를 이용하여 output 파일로부터 integer를 읽어 변수 target에 저장하고 input 파일로부터 첫 번째 줄에 있는 두 개의 integer를 읽어 순서대로 변수 N과 M에 저장하였다. 한편, 'fscanf(fpout, "%d", &target);'과 같이 프로그램을 작성하고 앞서 소개한 컴파일 명령어로 컴파일을 시도하면 경고 메시지가 뜨므로 'c=fscanf(fpout, "%d", &target);'과 같이 작성하였다.

output 파일에서 읽어들인 target 값이 정답과 일치하는지 확인하기 위해 for 반복문 내에서 fscanf()를 이용하여 input 파일의 두 번째 줄에 있는 정수를 하나씩 읽어들이고 그 값이 target 값보다 작다면 smallcnt를 1씩 증가시켰다. 만약 input 파일에서 읽어들인 값이 target 값과 일치할 경우 samecnt 값을 1씩 증가시켰다. 즉, for 반복문이 종료되면 smallcnt는 input array에서 target보다 작은 정수의 개수를 나타내고, samecnt는 target과 같은 값의 정수의 개수를 나타낸다.

```
43      //If smallcnt is 7 and samecnt is 3, it means that there are 7 integers smaller than 'target' and 3 integers equal to 'target'.
44      //Therefore, 'target' is the 8(=7+1)th smallest, the 9(=7+2)th smallest, and the 10(=7+3)th smallest value.
45      //Hence, we should check whether M is greater than M and less than or equal to (smallcnt + samecnt).
46      //If it is, we should print 1 to stdout. Otherwise, we should print 0 to stdout.
47      if (M>smallcnt&&M<=smallcnt+samecnt)
48          printf("1\n");
49      else
50          printf("0\n");
```

만약 for문이 종료되었을 때 smallcnt 값이 7이고 samecnt 값이 3이라면 input array에서 target보다 작은 정수는 7개, target과 같은 값의 정수는 3개라는 것을 의미한다. 따라서 target은 8(=7+1)번째로 작은 수이자 9(=7+2)번째로 작은 수이자 10(=7+3)번째로 작은 수이다. 즉, 만약 M값이 7(=smallcnt)보다 크고 10(=smallcnt+samecnt)보다 작거나 같다면 정답이고, 그렇지 않다면 오답이라는 것을 의미한다. 따라서 위의 스크린샷과 같이 M이 smallcnt보다 크고 smallcnt+samecnt보다 작거나 같다면 1을 출력, 아니면 0을 출력하였다.

마지막으로 fclose()를 이용하여 input과 output file을 모두 닫고 프로그램을 종료하였다.

3. Explain how your program works.

(1) main.c

main.c 파일 내에 randomized selection을 수행하는 함수와 deterministic selection을 수행하는 함수를 모두 정의하였다. 'gcc -std=gnu99 -O2 main.c -o main' 명령어를 통해 컴파일하면 main 실행파일이 생성되는데, 이는 실행 모드, input 과 output 파일 경로를 순서대로 인자로 입력받는다. input 파일의 두번째 줄에 있는 정수값들을 arr 배열에 모두 저장하고 실행모드가 1이면 randomized select를 수행하고 2이면 deterministic select를 수행한다. 이때, selection 알고리즘을 수행할 때의 오버헤드를 줄이기 위해 arr 배열을 전역으로 선언하였다.

i) randomized_selection 알고리즘

Lecture note에서 제시된 pseudo code와 거의 동일하다. 다만, arr 배열이 전역이기 때문에 함수 호출 시 argument로 arr을 전달할 필요가 없다. 즉, Lecture note에서 제시된 randomized selection을 수행하는 함수는 인자로 배열을 전달받지만 main.c에서 정의된 함수는 아래와 같이 더 이상 배열을 인자로 받지 않는다.

```
//randomized_selection algorithm
int randomized_select(int p, int r, int i){
    int q, k;

    if (p==r)
        return arr[p];
    q = partition(p, r);
    k = q-p+1;
    if (i==k)
        return arr[q];
    else if (i<k)
        return randomized_select(p,q-1,i);
    else
        return randomized_select(q+1,r,i-k);
}
```

ii) deterministic_selection 알고리즘

Deterministic selection을 수행하는 함수 역시 Lecture note와 거의 동일하지만 Randomized selection을 수행하는 함수와 마찬가지로 호출 시 배열을 인자로 받지 않는다. 아래는 Lecture note에서 제시된 deterministic selection 함수의 pseudo code를 캡쳐한 이미지이다.

linearSelect (A, p, r, i)
▷ find i-th smallest element in A[p ... r]
① if $n \leq 5$, find answer by insertion sort and return
② divide n elements into $\lceil n/5 \rceil$ groups of 5 elements
(if n is not a multiple of 5, size of one group is < 5)
③ find median in each group by insertion sort
let $m_1, m_2, \ldots, m_{\lceil n/5 \rceil}$ be these medians.
④ find median M of $m_1, m_2, \ldots, m_{\lceil n/5 \rceil}$ recursively
(if number of elements is even, find lower median) ▷ call linearSelect( )
⑤ partition n elements using M as pivot
⑥ same as select() ▷ call linearSelect( )

2단계에서 배열을 5개의 원소로 구성된 여러 개의 소그룹을 나누고 3단계에서 각 소그룹별 중앙값을 구하며 4단계에서 재귀호출을 이용하여 최종적인 중앙값 M을 구하는 것이 본 알고리즘의 핵심이다. 2단계에서 각 소그룹이 5개의 원소를 가지도록 배열을 나눌 때, 각 소그룹의 인덱스를 0, 1, 2, 3, ..., gcnt-1이라 하자. (단, gcnt는 소그룹의 개수) index가 0인 소그룹에서 insertion sort를 통해 중앙값을 구하고 이를 배열의 맨 앞(arr[p]) 원소와 swap한다. 다음으로 index가 1인 소그룹에서 insertion sort를 통해 중앙값을 구하고 이를 arr[p+1] 원소와 swap한다. 계속해서 반복하여 마지막으로 index가 gcnt-1인 소그룹에서 insertion sort를 통해 구한 중앙값을 arr[p+gcnt-1] 원소와 swap한다. 그러면 arr의 index p부터 index p+gcnt-1까지 소그룹들의 중앙값들이 위치하게 된다. 따라서 deterministic_select(p, p+gcnt-1, (gcnt+1)/2)를 호출하여 재귀적으로 최종적인 중앙값 M을 구하고, 이를 pivot으로 하여 selection을 수행하였다.

(4) generator.c

제출 목록에 포함되지 않아 해당 파일을 제출하진 않았지만 본 소스 코드는 input 파일을 생성하기 위해 작성하였다. 'gcc -std=gnu99 generator.c -o generator'명령어로 컴파일하면 'generator'이라는 이름의 실행파일이 생성되는데 이는 N, M, input 파일의 경로를 순서대로 인자로 입력받는다. 예를 들어 './generator 100 40 ./1.in'이라는 명령어는 1.in이라는 파일을 쓰기 모드로 open하는데 첫 번째 줄에 N과 M을 출력하고 두 번째 줄에 N개의 임의의 정수를 출력한다.

```
for (int i=0;i<N;i++){
    if (rand()%2)
        fprintf(fp, "%d ", rand()%2147483647);
    else
        fprintf(fp, "%d ", (-1)*(rand()%2147483647));
}
```

위와 같이 rand()를 사용하여 임의의 정수를 출력하도록 하였다.

4. Write down the environment you run your program and how to run it.

Window에서 VS Code를 이용해 프로그램을 작성하였고, Docker Desktop을 다운받아 도커 환경 내에서 asymtotic time complexities에서 숨겨진 상수비를 구하기 위해 테스트를 수행하였다. Docker Desktop에서 'docker pull yehyun/cta:assignment' 명령어를 통해 도커 이미지를 불러오고 'docker run -it yehyun/cta:assignment bash' 명령어를 통해 실행하였다.

'gcc -std=gnu99 -O2 main.c -o main' 명령어로 main.c 파일을 컴파일하고, './main x ./1.in ./1.out' 명령어로 프로그램을 실행하였다. 이때 'x' 자리에는 1과 2가 올 수 있는데, 1은 randomized select를 수행하고 2는 deterministic select를 수행한다.

'gcc -std=gnu99 -O2 checker.c -o checker' 명령어로 checker.c 파일을 컴파일 하고, './checker 1.in 1.out'명령어로 프로그램을 실행하였다.