2nd Assignment

-Divide&Conquer: 선택 문제-

2014147550 컴퓨터과학과
강효림

1. 개요

정렬되지 않은 n개의 원소가 있는 배열에서 k번째 작은 원소를 뽑는 소위 컴퓨터과학의 "선택 문제"에 대한 Lower bound가 O(n)인 알고리즘 및 Randomized 알고리즘(최악의 경우 $O(n^2)$)을 구현하고 그 성능을 비교한다.

2. 핵심 알고리즘

가. babyArr와 재귀적인 방법으로 median 도출

```
int miniIdx = 0;
int babyIdx = 0;
for(int i : arr) {
    miniArr[miniIdx] = i;
    miniIdx++;
    if (miniIdx == a) {
        babyArr[babyIdx] = miniMedian(miniArr);
        babyIdx++;
        miniIdx = 0;
    }
}
if (arr.length % a != 0) {
    miniIdx = 0;
    miniArr = new int[arr.length % a];
    for (int i = arr.length-1;miniIdx < arr.length % a;i--) {
        miniArr[miniIdx] = arr[i];
        miniIdx++;
    }
    babyArr[babyIdx] = miniMedian(miniArr);
}
int median;
if (babyArr.length % 2 == 0) {
    median = Det_Select(babyArr, babyArr.length/2, a, 0);
} else {
    median = Det_Select(babyArr, babyArr.length/2 + 1, a, 0);
}
if (flag == 1) {
    babyMedian = median;
}
```

babyArr은 길이 a의 작은 소배열(miniArr)에서 산출한 중간값들을 저장하는 babyMedian들의 배열이다. 이 때 이 배열을 인풋으로 하여 Det_Select를 호출하여 재귀를 구성한다. 결과적으로 median에 babyArr의 median이 저장된다.

나. Det_Select의 return 부분

```
if (k == rank) {
    return median;
} else if (k < rank) {
    int[] tmpArr = new int[smallSize];
    System.arraycopy(arr, 0, tmpArr, 0, smallSize);
    return Det_Select(tmpArr, k, a, 0);
} else if (k > rank && k <= e_idx+1) {
    return median;
} else {
    int[] tmpArr = new int[bigSize];
    System.arraycopy(arr, e_idx+1, tmpArr, 0, bigSize);
    return Det_Select(tmpArr, k - (e_idx + 1), a, 0);
}
```

Det_Select의 return 부분으로써, 기준이 된 median의 rank가 함수의 인자로 주어진 k와 같다면 그대로 median을 return하고, 아니라면 median을 중심으로 partition된 upper part나 lower part에 대하여 재귀적으로 동일한 알고리즘을 수행한다.

다. partition

```
public static int[] partition(int[] arr, int pivot) {
    ArrayList<Integer> small = new ArrayList<>(500);
    ArrayList<Integer> big = new ArrayList<>(500);
    ArrayList<Integer> same = new ArrayList<>();
    int idx = 0;
    int s_idx = -1;
    int e_idx = -1;
    int smallSize = 0, bigSize = 0;
    int[] retArr = new int[4];
    for (int i : arr) {
        if (i == pivot) {
            same.add(i);
        } else if (i < pivot) {
            small.add(i);
        } else {
            big.add(i);
        }
    }
```

arr를 pivot을 중심으로 partition 연산을 실행하는 코드로써(arr자체가 바뀐다) lower part의 크기, upper part의 크기, pivot의 rank등 정보를 포함하는 size 4인 int 배열을 반환한다. 위는 코드의 일부이다.

라. *Randomized Select에서의 pivot 선택

```
int pivot = randomSelect(arr, r);
int[] partArr = partition(arr, pivot);
```

제출되는 소스코드에서는 포함되지 않은 함수로써, 성능 비교를 위해 정의한 함수 Ran_Select가 있다. 그 함수 내에서 partition을 행할 pivot을 선택하는 부분으로써, 여기서 r은 Java의 Random 객체이다.

3. 성능 비교

가. input의 크기 및 측정 단위

워낙 빠른 시간에 시행되는 알고리즘이므로, 유의미한 running time을 얻기 위해 10,000,000개의 정수를 대상으로 하였다. 시간 측정은 milisecond로 하였으며 (System.currentTimeMillis), 파일을 읽어들이는 시간은 제외된 값이므로 순수 알고리즘 수행시간이다.

나. Ran_Select와 Det_Select간 성능 비교

Ran_Select(a=5)	Det_Select(a=5)
1907ms	2763ms

다. a 값에 따른 Det_Select의 성능 비교

a value	time(ms)
3	4406
5	2763
7	2664
9	2641
11	2835
13	2722
15	2681
117	2564
1117	2899
11117	3388

라. 결과분석

1) Ran_Select/Det_Select 간 비교

몇 번의 실험을 해 보아도 언제나 Ran_Select가 더 빠른 성능을 보였다. 비록 Det_Select 알고리즘이 lower bound가 선형시간으로 보장이 되나, i) 매번 babyMedian의 재귀적 호출과 관련해서 동적으로 새로운 배열의 할당이 이루어지는 등 공정한 pivot 상정의 오버헤드가 걸린다는 점, ii) Ran_Select가 bad pivot selection에 따른 bad partitioning을 연속해서 할 가능성은 인풋이 커짐에 따라 그 가능성이 극히 적어진다는 점 등이 그 이유이다.

2) Det_Select에서 a 값의 선정

a=3인 경우를 제외하고 거의 비슷한 성능을 보이다가, a 값이 극히 커진다면 (a=11117) 나쁜 성능을 보이는 것이 확인되었다. a=3인 경우는 a가 5 이상인 경우와 달리 recursive한 divide&conquer에서 subproblem의 크기가 줄어들지 않는 현상 때문이며, a가 지나치게 커지면 답안과 직접적으로 관계없음에도 a 크기 배열을 정렬하는 연산의 오버헤드가 커지기 때문이다.

-끝-