

알고리즘-5주차-Sort

이름 : 유형곤

학번 : 201902722

사용 언어 : Java

<점수표>

RANK	TEAM	SCORE	ANAGRAMS  [3 POINTS]	BUBBLE  [2 POINTS]	MERGE  [2 POINTS]	QUICK  [2 POINTS]	SORT  [1 POINT]
1	201902722	10 701	148 8 tries	22 2 tries	76 1 try	182 5 tries	13 2 tries

문제 1 : ANAGRAMS

문제 : 서로 anagram 관계에 속하는 문자열들끼리 그룹을 나누고 정렬하는 문제

해결 방법 :

처음 문제를 봤을 때 어려웠던 점은, 저번에 푼 ANAGRAM 문제 같은 경우에는 두 단어 간의 관계를 파악하는 것이었다면 이번에는 **다수의 단어가 같은 anagram인지를 파악하는 것이었습니다**. 그래서 생각하다가 같은 anagram이라면 문자열을 정렬했을 때 같은 값이 나오므로 이 것을 key로 하는 Map을 만들어야겠다고 생각했습니다. **key가 정렬된 문자열이고, value가 리스트인 map**을 만들고, 그 map에 문자열을 분류했습니다. 따라서 서로 anagram인 문자열들은 같은 그룹에 들어갑니다.

하지만 아직 해결하지 못한 게 있습니다. 바로 정렬입니다. 같은 그룹에 있는 문자열을 정렬하는 것은 쉽습니다. String에 대한 Comparator가 이미 정의되어있으므로, Collections.sort() 함수를 호출하면 됩니다. 그런데 각 그룹을 출력하는 순서를 정하는 게 어려웠습니다. 왜냐하면 **Map에 문자열을 사용할 때 사용한 key는 정렬된 문자열인데, 각 그룹을 출력하는 순서는 각 그룹의 첫 번째 문자열을 기준으로 정하기 때문**입니다. 그래서 두 가지 풀이를 생각했습니다.

1. Map의 원래 key와 해당 그룹의 첫 번째 문자열을 연결시켜주는 Map을 하나 더 만든다.
2. 현재 Map의 Key를 각 그룹의 첫 번째 문자열로 업데이트한다.

저는 2번째 방법으로 풀이를 했습니다. 그런데 map.keySet()을 순회하면서 map을 수정하면 ConcurrentModification (동시성) 문제가 발생하므로, keySet을 새로운 List에 넣어서 iterator로 돌리고, map을 수정하여 key를 업데이트했습니다.

시간복잡도 :

1. anagram의 그룹을 저장하는 Map 생성

$O(N * M \log M)$ N : 전체 문자열의 개수, M : 문자열의 길이

2. anagram 그룹을 정렬하고 출력

$O(K * L \log L) = O(N \log L)$, K : 그룹의 수, L : 그룹별 문자열의 수

3. 정렬된 anagram 출력

$O(K * L) = O(N)$, K : 그룹의 수, L : 그룹 내 문자열의 수, N : 전체 문자열의 개수

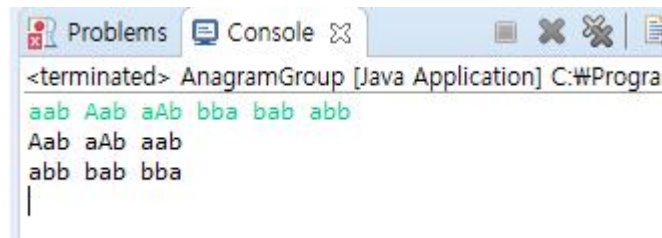
따라서, 이 풀이의 시간복잡도는 $O(N * M \log M + N * L \log L)$ 입니다.

<ANAGRAMS 주요 소스코드>

```
//Map을 이용하여 anagram 그룹을 만들어서 단어를 분류함
//O(N * M log M) N : 전체 문자열의 개수, M : 문자열의 길이
while(st.hasMoreTokens()) {
    String token = st.nextToken();
    String anagram = sort(token.toLowerCase()); //anagram key
    if(!map.containsKey(anagram)) {
        LinkedList<String> list = new LinkedList<String>();
        list.add(token);
        map.put(anagram, list);
    } else {
        map.get(anagram).add(token);
    }
}

//Group 안의 문자열을 정렬 & Map의 key를 각 그룹의 첫 번째 요소로 변경
//Map을 순회하면서 Map을 수정하면 ConcurrentModificationException이 발생하므로 새로운 리스트를 생성
//O(K * L log L) = O(N log L), K : 그룹의 수, L : 그룹별 문자열의 수
Iterator<String> iter = new LinkedList<String>(map.keySet()).iterator();
while(iter.hasNext()) {
    String originalKey = iter.next();
    LinkedList<String> list = map.get(originalKey);
    Collections.sort(list);
    String key = list.get(0);
    if(!map.containsKey(key)) {
        map.put(key, list);
        map.remove(originalKey);
    }
}

//정렬된 anagram 그룹 (Map) 출력
//O(K * L) = O(N) K : 그룹의 수, L : 그룹 내 문자열의 수, N : 전체 문자열의 개수
LinkedList<String> keyList = new LinkedList<String>(map.keySet());
Collections.sort(keyList); //O(K log K)
iter = keyList.iterator();
while(iter.hasNext()) {
    Iterator<String> iterList = map.get(iter.next()).iterator();
    while(iterList.hasNext()) {
        bw.write(iterList.next());
        if(iterList.hasNext()) {
            bw.write(" ");
        }
    }
    bw.write("\n");
}
```



<ANAGRAMS 실행결과>

문제 2 : BUBBLE

문제 : 버블정렬을 구현하는 문제

해결 방법 :

일반적인 버블정렬 알고리즘을 사용하여 문제를 해결했는데, 버블정렬 알고리즘을 간단하게 설명해보면 (오름차순 정렬 기준) 한 번 순회할 때마다, 원소를 두 개씩 비교해서, 왼쪽 원소가 오른쪽 원소보다 작으면 교체하는 방식으로, 정렬되지 않은 원소 중, 가장 큰 원소가 오른쪽으로 이동한다는 특징이 있습니다.

예를들어, [4, 2, 1, 3]이라는 리스트가 있다고 할 때, 한 번 순회하면

4가 2보다 크므로 교체하고, [2, 4, 1, 3] 4가 1보다 크므로 교체하고,

[2, 1, 4, 3], 4가 3보다 크므로 교체합니다. [2, 1, 3, 4]

이렇게 한 번 순회할 때마다 정렬되지 않은 원소 중 가장 큰 것이 오른쪽으로 이동합니다.

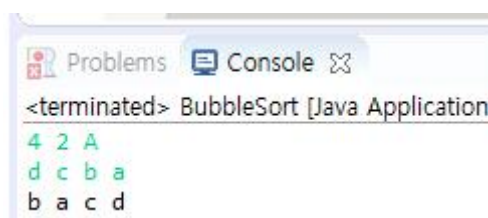
따라서 순회를 N번하면 확실하게 모든 원소를 정렬할 수 있습니다.

문제를 풀때 순회하는 횟수가 정해져있으므로 순회한 횟수에 따라 정렬을 중지하였습니다.

시간복잡도 : $O(N^2)$ (N번 순회하고, 한 번 순회할 때 N번 비교 * 교체함)

```
public static void bubbleSort(String arr[], int m, int isAscending) {  
    int cnt = 0;  
    for(int i = arr.length-1; i >= 0; i--) {  
        if(cnt == m) {  
            break;  
        }  
        cnt++;  
        for(int j = 0; j < i; j++) {  
            //isAscending : 오름차순이면 1, 내림차순이면 -1로, compareTo가 들어간 식의 부등호를 역전시키는 역할  
            if(isAscending * arr[j].compareTo(arr[j+1]) > 0) {  
                String tmp = arr[j];  
                arr[j] = arr[j+1];  
                arr[j+1] = tmp;  
            }  
        }  
    }  
}
```

<BUBBLE 소스코드>



<BUBBLE 수행결과>

문제 3 : MERGE

문제 : 병합정렬을 구현하는 문제

해결 방법 : 병합정렬도 일반적인 병합정렬 알고리즘으로 구현했는데, 병합정렬 같은 경우에는 크게 두 가지로 나눌 수 있습니다. 1. 정렬할 리스트를 반으로 나눈다. 2. 리스트를 합치되, 합칠 때는 원소의 크기 순서대로 합쳐서 정렬이 되도록 한다.

예를 들어 리스트 [4, 1, 2, 3]이 있다고 할 때, 우선 리스트의 길이가 1이 될 때까지 리스트를 절반으로 나눕니다.

[4, 1, 2, 3]

[4, 1], [2, 3]

[4], [1], [2], [3]

이제 길이가 1이 될 때까지 반으로 나누었으므로, 다시 합쳐야합니다. 단, 무작위로 합치는 게

아니라 정렬이 되도록 합칩니다.

[4], [1], [2], [3]

[1, 4], [2, 3]

[1, 2, 3, 4]

문제를 풀 때는 merge의 횟수가 정해져있으므로, 위와 같은 알고리즘을 사용하되 merge를 한 횟수에 따라서 함수를 종료시켰습니다.

시간복잡도 :

한 번 병합(merge)할 때마다 N번의 연산이 필요하고, 병합의 횟수는 $\log N$ 입니다. (반씩 쪼개서 길이가 1일 될 때까지 쪼개므로, 예를 들어 8을 반씩 쪼개서 1로 만들려면 $\log 8 = 3$ 번 쪼개야한다.)

```

//리스트를 절반으로 나눔
public static List<String> mergeSort(List<String> arr, int isAscending) {
    if(arr.size() <= 1) {
        return arr;
    }

    List<String> left = new LinkedList<String>();
    List<String> right = new LinkedList<String>();

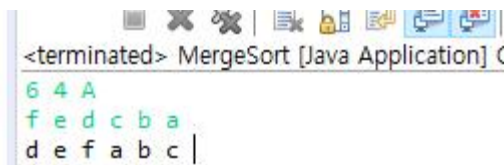
    int mid = arr.size()/2;
    for(int i = 0; i < mid; i++) {
        left.add(arr.get(i));
    }
    for(int i = mid; i < arr.size(); i++) {
        right.add(arr.get(i));
    }

    left = mergeSort(left, isAscending);
    right = mergeSort(right, isAscending);
    return merge(left, right, isAscending);
}

//리스트를 정렬하면서 합침
public static List<String> merge(List<String> left, List<String> right, int isAscending)
{
    m--;
    List<String> result = new LinkedList<String>();
    if(m >= 0) {
        while((left.size() != 0) && (right.size() != 0)) {
            if(isAscending * left.get(0).compareTo(right.get(0)) <= 0) {
                result.add(left.remove(0));
            } else {
                result.add(right.remove(0));
            }
        }
        while(left.size() > 0) {
            result.add(left.remove(0));
        }
        while(right.size() > 0) {
            result.add(right.remove(0));
        }
    } else {
        while(left.size() > 0) {
            result.add(left.remove(0));
        }
        while(right.size() > 0) {
            result.add(right.remove(0));
        }
    }
    return result;
}
}

```

<MERGE 소스코드>



```

<terminated> MergeSort [Java Application] (
6 4 A
f e d c b a
d e f a b c |

```

<MERGE 실행결과>

문제 4 : QUICK

문제 : 퀵 정렬을 구현하는 문제

해결 방법 : 퀵 정렬도 일반적인 퀵 정렬 알고리즘으로 구현했는데, 퀵 정렬 같은 경우도 두 가지 연산으로 나눌 수 있습니다.

1. Pivot을 기준으로 작은 원소를 왼쪽으로, 큰 원소를 오른쪽으로 분할하는 파티션 작업
 2. 분할된 Pivot의 왼쪽 부분과 오른쪽 부분에 대한 파티션 작업을 진행
- 위의 1, 2 작업을 정렬이 모두 될 때까지 작업하여 정렬을 수행합니다.

QuickSort같은 경우, Pivot을 선택하는 방법에 따라서 다르지만, 여기서는 `arr[high]`를 Pivot으로 정하므로, 이를 기준으로 예시를 들어보겠습니다.

[5, 1, 4, 2, 3]이 있다고 하면

pivot은 3이 되고, `low~high-1` (여기선 [5, 1, 4, 2])를 순회합니다.

`i = low = 0` // `i`는 pivot보다 작거나 같은 숫자들이 들어갈 인덱스

[5, 1, 4, 2] -> `j = 0`, `3 < 5`이므로 건너뜁니다.

[5, 1, 4, 2] -> `j = 1`, `1 <= 3`이므로 `arr[i]`, `arr[j]`를 교체합니다, `i += 1` -> `i = 1`

[1, 5, 4, 2] -> `j = 2`, `3 < 4`이므로 건너뜁니다. // `i = 1`

[1, 5, 4, 2] -> `j = 3`, `2 < 3`이므로 `arr[i]`와 `arr[j]`를 교체합니다.

[1, 2, 4, 5] -> pivot보다 작거나 같은 원소들과, pivot보다 큰 원소가 나뉩니다.

[1, 2, 3, 5, 4] -> pivot의 왼쪽은 pivot보다 작거나 같게, pivot의 오른쪽은 pivot보다 크도록 pivot을 원래 자리로 위치시킵니다. 원래 자리에 있던 (인덱스 `i`에 있던) 원소는 pivot과 교체됩니다.

이러한 파티션 작업을 다시 pivot의 왼쪽 [1, 2], pivot의 오른쪽 [5, 4]에 대해서 진행하면 QuickSort가 끝납니다. 단, 이 문제에선 파티션 작업의 횟수에 따라 퀵 소트를 중지시키므로 파티션 함수의 호출 횟수를 카운트 하는 부분을 추가하였습니다.

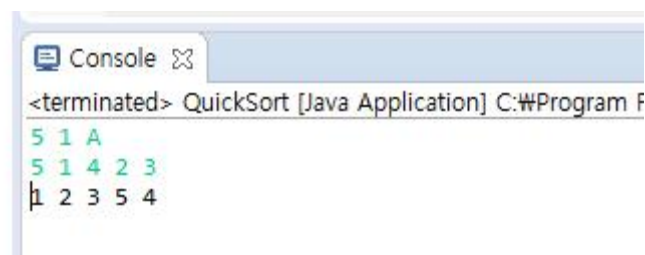
```

public static void quickSort(String[] arr, int low, int high) {
    if(cnt == m) {
        return;
    }
    if(low < high) {
        int p = partition(arr, low, high);
        quickSort(arr, low, p-1);
        quickSort(arr, p+1, high);
    }
}

public static int partition(String[] arr, int low, int high) {
    cnt++;
    String pivot = arr[high];
    int i = low;
    for(int j = low; j <= high - 1; j++) {
        if(isAscending * arr[j].compareTo(pivot) <= 0) {
            String tmp = arr[i];
            arr[i] = arr[j];
            arr[j] = tmp;
            i++; //이 부분 참고
        }
    }
    String tmp = arr[i];
    arr[i] = arr[high];
    arr[high] = tmp;
    return i;
}

```

<QUICK 소스코드>



```

Console
<terminated> QuickSort [Java Application] C:\Program F
5 1 A
5 1 4 2 3
1 2 3 5 4

```

<QUICK 실행결과>

문제 5 : SORT

문제 : 정렬을 활용하는 문제

해결 방법 : 글 안에 들어있는 모든 단어를 리스트에 넣어두고, 정렬한 후 출력했습니다.

시간복잡도 :

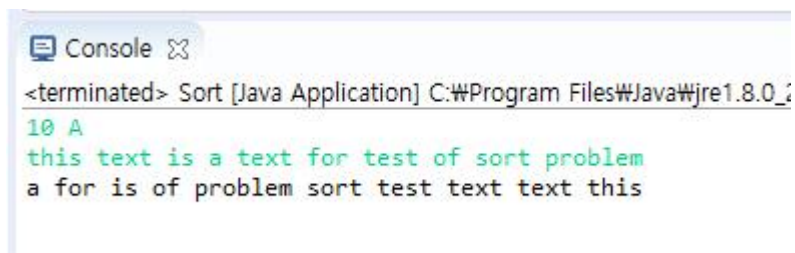
Java에서는 참조형 변수에 대하여 timsort를 진행하므로 시간복잡도는 $O(N \log N)$ 입니다.

```
while(st.hasMoreTokens()) {
    list.add(st.nextToken());
}

if(ch.equals("A")) {
    Collections.sort(list);
} else {
    Collections.sort(list, Collections.reverseOrder());
}

Iterator<String> iter = list.iterator();
while(iter.hasNext()) {
    bw.write(iter.next());
    bw.write(" ");
}
```

<SORT 소스코드>



```
Console
<terminated> Sort [Java Application] C:\Program Files\Java\jre1.8.0_2
10 A
this text is a text for test of sort problem
a for is of problem sort test text text this
```

<SORT 실행결과>