알고리즘 과제(03)

201404377 진승언

1. 과제 목표 및 해결 방법

이번 과제는 최대/최소 힙정렬과 계수정렬을 구현하는게 목표였다. 힙정렬이란 이진트리로서 맨 아래 층을 제외하고는 완전히 채워져 있고 맨 아래층은 왼쪽부터 꽉 채워져있다. 그리고 최소힙은 각 노드의 값은 자신의 자식의 값보다 작다는 성질을 갖고있고 최대힙은 각 노드의 값은 자신의 자식의 값보다 크다는 성질을 갖고있다.

이번과제에서는 힙정렬을 크게 Max-Heap-Sort(A), Build-Max-Heap(A), Max-Heapify(A, i) 3가지로 메소드를 분리해서 구현하였다. 먼저 주 메소드인 Max-Heap-Sort(A) 에서 Build-Max-Heap(A)를 호출해준다. Build-Max-Heap(A)는 자식이 존재하는 가장 마지막 노드부터 시작해서 처음 노드까지 기준노드를(부모노드 or i값) 정해서 Max-Heapify(A, i)을 호출하는 역할을 한다. 여기서 Max-Heapify(A, i)는 해당 i인덱스에 있는 값을 기준으로 자식노드들과 비교해서 만약 자식노드가 더 큰 값을 가지고 있다면 i인덱스와 swap을 해주고 swap을 했다면 i를 largest으로(최대값) 하여 이것을 그 밑의 자손노드들(서브트리)까지 Max-Heapify(A, i) 재귀를 돌려준다. 이렇게 해주면 큰값이 이진트리의 상위 높이에 존재하게되고 작은값은 하위 높이에 존재하게 된다. 다시 말하면, 이 과정을 Build-Max-Heap(A)에서 Max-Heapify(A, i)를 자식이 있는 노드의 가장아래부터 가장 최상위에 있는 노드까지 반복문을 돌려 주어 자식이 있는 노드들은 모두 Max-Heapify(A, i) 시켜줌으로서 모든 부모노드가 자식노드보다는 큰 값을 가지게 해주는 것이다.

최소 힙정렬은 heapify 부분을 최대 힙정렬에서 자식노드가 작은값일 때 swap 해주는 걸로만 바꿔주면 된다.

계수정렬은 각 데이터들의 개수를 계산하여 배열[데이터값]에(count 배열이라 하겠다) 그 데이터의 개수를 넣는다. 그 다음 이 배열의 누적합을 구해서 배열에 다시 담아준다. 누적합은 그냥 count[1]은 count[0] + count[1]해주고 count[2]는 앞서 누적합이 넣어진 count[1] + count[2] 이런식으로 누적되는합을 말한다. 이 누적합 배열을 이용해서 원래 처음 데이터배열의 크기만큼 임시 배열을 만들어 거기에 차례대로 넣어주면 된다. 그리고 임시배열을 원래배열에 다시 복사해주면된다. 그럼 예를들어 누적합이 담긴 배열이 count[4]={1,4,6,10}이렇게 됐다면 인덱스 0인 부분이 1이니깐 데이터 0은 0번째에서 1번째 사이에 위치한다는 의미이고, 즉 임시배열의 인덱스 0에 데이터 값 0이 넣어진다. 인

덱스 1인 부분은 4니까 1번째에서 4번째 사이에 위치한다는 의미로서 1,2,3인덱스에 데이터 값 1이 들어가면 된다. 마지막으로 인덱스 2인 부분은 4번째에서 6번째 사이니깐 데이터 값 2가 4,5번째에 들어가면 된다.

2. 주요 부분 코드 설명(알고리즘 부분 코드 캡쳐)

알고리즘의 저유 부분코드 설명은 코드의 주석에 자세히 쓰는게것이 더 보기도 좋아서 주석으로 작성하였다. 또한 이 코드에 대한 설명은 앞서 1번에서 자세히 작성하였다.

추가적으로, 힙정렬에서 맨 상위노드 인덱스를 1로하는 경우가 많은데 나 같은 경우 0 인덱스로 했다.

1)MAX HEAP SORT

```
//MAX HEAP SORT(인덱스 0부터 시작)
public void maxHeapSort(int[] arr) {
    int tmp;
    buildMaxHeap(arr);  //max heap 구조로 만듬
    for(int i =arr.length-1; i>=1; i--) { //최대 힙으로 되있는 것을 마저 완벽하게 정렬해줌, 노드가 n개라면 n-1번 반보해야함
        //swap
        tmp = arr[0];  //힙에서 최댓값을 가장 마지막값과 바꿈 (최댓값 최솟값 스왑)
        arr[0] = arr[i];
        arr[i] = tmp;

        heap_size -= 1;  //하나씩 정렬하는 것이므로 하나의 노드가 정렬이 완료되면 힙사이즈를 1줄여준다.(즉 스왑 후 마지막노드값은 고정이됨)(마지막값이 가장 큰 노드값으로 고정)
        maxHeapify(arr, 0);  //첫노드엔 가장 큰 값이 마지막노드엔 가장 작은값이 오도록 heapify재귀를 돌려줌
    }

}

public void buildMaxHeap(int[] arr) {
    heap_size = arr.length-1; //힙사이즈 최대크기 설정
    for(int i=arr.length/2-1; i>=0; i--) {  //자식노드들가 있는노드에서부터 상위노드까지 차례대로 힙구조로 만듬
        maxHeapify(arr, i);
    }

}
```

```java
public void maxHeapify(int[] arr, int i) { //부모노드와 부모노드의 서브트리를 비교(재귀), 스왑하면서 maxheap구조로 만들어줌
    int largest;   //최대값 갖고있는 인덱스
    int left = 2*i+1;   //왼쪽 자식노드 인덱스
    int right = 2*i+2; //오른쪽 자식노드 인덱스


    if(left <= heap_size && arr[left]>arr[i]) { //왼쪽 자식노드 힙사이즈보다 작거나같고 부모노드(기준) 값보다 크면 largest에 왼쪽자식노드 인덱스 저장
        largest = left;
    }
    else { //아니면 부모노드 인덱스를 largest에 저장
        largest = i;
    }

    //largest에 부모인덱스 or 왼쪽자식인덱스 저장되있는 상태
    if(right <= heap_size && arr[right] > arr[largest]) { //오른쪽자식 인덱스가 힙사이즈보다 작거나같고 largest인덱스 값보다 크면 largest에 오른쪽자식노드 인덱스저장
        largest =right;
    }
    if(largest != i){ //가장 큰 값의 인덱스가 부모노드가 아니라면 자식노드와 부모노드를 swap 해줌
        swap(arr, i, largest);
        maxHeapify(arr, largest); //가장 컸던 자식노드를(largest) 기준으로 재귀로 heapify반복
    }

}
```

2) MIN HEAP SORT

최대 힙정렬과 heapify부분만 자식노드가 부모노드값 보다 작은 값일 때 swap되게 바꿔주면 나머진 동일하다.

```java
//MIN HEAP SORT (max heap과 반대로 heapify메소드에서 자식노드가 더 작은값이면 부모노드와 스왑해주면됨
 public void minHeapify(int[] arr, int i) {
     int smaller;
     int left = 2*i+1;
     int right = 2*i+2;
     if(right<= heap_size) {
         if(arr[left] < arr[right]) {
             smaller = left;
         }
         else {
             smaller = right;
         }
     }
     else if(left <= heap_size) {
         smaller = left;
     }
     else {
         return;
     }
     if(arr[smaller] < arr[i]) {
         swap(arr, i, smaller);
         minHeapify(arr, smaller);
     }
 }
```

3) COUNTING SORT

```java
public void countingSort(int[] arr) {
    int i, j, max=0;
    int count[];
    int tmp[] = new int[arr.length];
```

값을 담아놀 count와 tmp배열이 필요하다.

```java
//arr배열 최댓값 max에 저장
for(i=0; i<arr.length; i++ ) {
    if(arr[i] > max) {
        max = arr[i];
    }
}

//cout배열 생성
count = new int[max+1];

//카운팅할 배열 초기화
for(i=0; i<=max; i++) {
    count[i] = 0;
}

//카운팅 저장(ex)arr배열에 1이 3개면 count[1]에 3저장
for(i=0; i<arr.length; i++) {
    count[arr[i]]++;
}

//count에 누적된 합저장
//count[j]에 j보다 작거나 같은 원소의 총 개수 저장 (ex) count[1]은 count[0]과 count[1]의 합, count[2]는 누적합이저장된 count[1]과 count[2]의 합
for(j=1 ; j < count.length; j++) {
    count[j] = count[j] + count[j-1];
}


//위에서한 누적된 합을(count) 이용해 정렬
for(j=arr.length-1; j >= 0; j--) {
    tmp[count[arr[j]]-1] = arr[j]; // arr값이 들어있는 count 누적된값에서 -1한 인덱스 위치를 사용하여 tmp에 해당 arr의 값을 저장
    count[arr[j]]--; //count에 누적된합에서 1감소시켜줌
}

//tmp배열을 arr배열에 복사
System.arraycopy(tmp, 0, arr, 0, tmp.length);
```

주석에 자세히 설명해놓았다.

3. 결과(시간 복잡도 포함)

1) MAX HEAP SORT

```
Below is MAX Heap_Sort 100 result
0
0
7
9
10
12
13
14
18
20
27
28
33
34
36
50
52
53
54
57
57
58
60
61
61
62
64
64
65
70
72
72
72
75
76
76
78
```

```
Below is MAX Sort 1000 result
1
2
3
4
4
6
7
8
9
13
15
15
16
16
17
19
20
20
22
24
24
25
27
31
33
34
36
37
37
38
38
42
43
49
51
58
65
```

2) MIN HEAP SORT

```
199
--------------------------------------------------
Below is MIN Heap_Sort 100 result
199
195
192
186
185
184
183
181
180
178
176
175
175
173
172
172
171
171
171
168
167
166
166
166
166
164
157
156
154
154
153
153
151
147
145
```

```
1998
--------------------------------------------------
Below is MIN Heap_Sort 1000 result
1998
1997
1996
1994
1994
1994
1994
1993
1985
1983
1983
1982
1980
1980
1978
1976
1975
1968
1967
1966
1966
1966
1964
1962
1960
1959
1959
1958
1955
1955
1952
1951
1949
1948
1943
```

3) COUNTING SORT

```
0
--------------------------------------------------
Below is Counting_Sort 100 result
0
0
7
9
10
12
13
14
18
20
27
28
33
34
36
50
52
53
54
57
57
58
60
61
61
62
64
64
65
70
72
72
72
75
76
```

```
1
-----------------------------------------
Below is Counting_Sort 1000 result
1
2
3
4
4
6
7
8
9
13
15
15
16
16
17
19
20
20
22
24
24
25
27
31
33
34
36
37
37
38
38
42
43
49
```

먼저 시간복잡도는 최대힙정렬이나 최소힙정렬 똑같은데 buildMaxHeap에서 n/2번 반복하면서 maxheapify부르는데 트리깊이만큼 비교하고 스왑하는 것이므로(스왑은 그냥 두개 바꿔주는것이므로 시간복잡도 1) logn이다. 그래서 여기서도 O(nlogn) 시간복잡도가 걸리고 그 후 maxHeapSort에서 이어서 n-1번 반복하면서 maxheapfiy하는데 logn이 걸린다. 그러므로 총 시간복잡도는 O(nlogn)이다. 또한 좋을 때나 나쁠 때나 항상 평균 O(logn)이다.

계수정렬은 반복문 4개다 대게 n-1번만큼의 이동연산밖에 일어나지 않으므로 시간복잡도가 O(n)이라고 볼 수 있다. 그러나 계수정렬은 n개 크기의 임시배열도 2개가 필요하고 그 배열의 빈곳도 0으로 채워넣어야하므로 메모리 낭비가 크다는 단점이 있다.