

알고리즘 과제(01)

201404377 진승언

1. 과제 목표 및 해결 방법

➔ 이번 과제는 삽입, 버블, 선택 정렬을 직접 구현하는데 배열이 아닌 이중연결리스트를 직접 구현해서 정렬을 구현하는 것이 목표였다. 정렬을 하기 전에 이중연결리스트에 데이터를 저장해 놔야했으므로 이중연결리스트부터 구현을 하였다. 이중연결리스트 안에 실제로 이어질 노드클래스를 내부클래스로 정의하였다. 그리고 이중연결리스트의 메소드로 삽입 삭제 출력의 기능을 구현하였다. 그 다음 정렬을 구현하였다. (ppt에 나와있는 sudo코드를 참고하였다.)

2. 주요 부분 코드 설명(알고리즘 부분 코드 캡처)

```
public class DoubleLinked {
    private Node head;
    private Node tail;
    private int size;

    public DoubleLinked() {
        size = 0;
    }

    public class Node{
        public Node next;
        public Node prev;
        public int data;
        Node(int data){
            this.data = data;
            this.next = null;
            this.prev = null;
        }
    }
}
```

이중연결리스트를 구현한 코드중 일부분이다. Head는 가장 앞의 노드를 의미하고 tail은 가장 뒤에 있는 노드를 의미한다. Size는 말 그대로 노드의 개수이다. 그리고 내부클래스로 Node를 만들어서 이 Node로 연결을 시킨다.

```

public void addFirst(int line) {
    Node node = new Node(line);

    if(head != null) {
        node.next = head;
        head.prev = node;
    }

    head = node;
    if(head.next == null) {
        tail = head;
    }
    size++;
}

```

맨 앞에 데이터를 추가 시키는 이중연결리스트에 구현한 메소드이다. 해당값을 가진 Node를 생성한 후 head가 비어있지 않으면 삽입 할려는 노드의 다음연결을 head에 연결하고 그리고 현재 head의 prev를 현재 삽입한 노드로 연결해준다. 마지막으로 가장 첫 노드를 현재삽입한 노드로 변경해준다.(head = node) 그 밑 코드는 지금 삽입할려는 노드가 첫 노드일 경우이다.

```

public Object removeFirst() {
    if(size == 0) {
        return null;
    }
    Node removeNode = head;
    head = null;
    head = removeNode.next;
    head.prev = null;
    size--;
    return removeNode.data;
}

```

다음은 노드 삭제 함수인데 연결리스트에 노드가 존재하는 경우 해당 삭제할 노드를 저장 후, head를 현재 삭제할려는 다음에 있는 노드를 가리키게하고 head의 이전연결은 null로 해줌으로서 연결을 끊어준다. 마지막으로 삭제한 노드를 반환한다.

```

public String printAll() {
    StringBuffer result = new StringBuffer();
    if(head == null) {
        System.out.println("Data is not exist");
    }
    if(head != null) {
        /* System.out.println(head.data);
        while(head.next != null) {
            System.out.println(head.next.data);
            head= head.next;
        }
        */
        result.append(head.data);
        result.append(",\n");
        while(head.next != null) {
            result.append(head.next.data);
            result.append(",\n");
            head= head.next;
        }
    }
    return result.toString();
}

```

다음은 이중연결리스트의 출력 메소드이다.(한번에 모든 데이터를 출력하게 해주었다.) 다음 노드가 null일 때 까지 다음 연결된 노드로 이동하면서 StringBuffer에 데이터 값을 차례대로 저장해서 String으로 한번에 출력하게 해주었다.

```

2 public class InsertionSort {
3     void sort(DoubleLinked list) {
4         System.out.println("Start InsertionSort");
5         int i, j, key;
6         for(i =1; i<list.size(); i++) {
7             key = (list.getNode(i)).data;
8             for(j= i-1; j>=0 && (list.getNode(j)).data > key; j--) {
9                 (list.getNode(j+1)).setData((list.getNode(j)).data);
10            }
11            (list.getNode(j+1)).setData(key);
12        }
13    }
14 }
15 }

```

삽입정렬 알고리즘이다. 이중연결리스트를 매개변수로 받아서 두번째 노드부터 시작해서 size만큼 반복하고 해당 순번 노드보다 전에있는 노드들과 비교해서 알맞은 자리로 이동시킨다. 즉 i=1 에서 size만큼 반복하고 i번째 값을 key로 저장 후 i-1번째 밑에 있는 값들과 비교해서 자신이 있어야할 자리로 스왑해주면 된다.

[요약]삽입 정렬은 두 번째 인덱스부터 시작한다. 현재 인덱스는 별도의 변수에 저장해두고, 비교 인덱스를 현재 인덱스 -1로 잡는다. 별도로 저장해 둔 삽입을 위한 변수와, 비교

배열 값을 비교한다. 삽입 변수의 값이 더 작으면 현재 인덱스로 비교 인덱스의 값을 저장해주고, 비교 인덱스를 -1하여 비교를 반복한다. 만약 삽입 변수가 더 크면, 비교 인덱스 + 1에 삽입 변수를 저장한다.

```

2 public class BubbleSort {
3     void sort(DoubleLinked list) {
4         System.out.println("Start BubbleSort");
5         int temp = 0;
6         for (int i = 0; i < list.size(); i++) {
7             for (int j = 1; j < list.size() - i; j++) {
8                 if ((list.getNode(j)).data < (list.getNode(j-1)).data) {
9                     temp = (list.getNode(j-1)).getData();
10                    (list.getNode(j-1)).setData((list.getNode(j)).data);
11                    (list.getNode(j)).setData(temp);
12                }
13            }
14            //System.out.println("bubble"+i);
15        }
16    }
17 }
18

```

버블정렬 알고리즘이다. size개수만큼 반복하며 처음 노드부터 시작해서 size-반복횟수 까지(끝에 정렬이 된 노드부분 전 까지) 차례대로 비교 및 swaps를 해준다.

[요약] 두 번째 인덱스부터 시작하며 현재 인덱스 값과 , 바로 이전의 인덱스 값을 비교한다. 만약 이전 인덱스가 더 크면, 현재 인덱스와 바꿔준다. 현재 인덱스가 더크면, 교환하지 않고 다음 두 연속된 배열값을 비교한다. 이를 반복한다.

```

2 public class SelectionSort {
3     void sort(DoubleLinked list) {
4         System.out.println("Start selectionsort");
5         int i, j , min, temp;
6         for(i = 0; i < list.size()-1; i++ ) {
7             min = i;
8             for(j = i+1; j<list.size(); j++) {
9                 if((list.getNode(j)).data < (list.getNode(min)).data)
10                    min = j;
11            }
12            //swap
13            temp = (list.getNode(i)).data;
14            (list.getNode(i)).setData((list.getNode(min)).data);
15            (list.getNode(min)).setData(temp);
16        }
17    }
18 }
19

```

선택정렬 알고리즘이다. i=0 번부터 size-1번 만큼 반복하며 i+1 에서 size만큼 비교 및 최소값을 저장해 준다. 그리고 최종최소값을 정렬할 부분으로 swap 해준다.

[요약] 정렬 되지 않은 인덱스의 맨 앞에서부터, 이를 포함한 그 이후의 배열값 중 가장 작은 값을 찾아간다. 가장 작은 값을 찾으면, 그 값을 현재 인덱스의 값과 바꿔준다. 다음 인덱스에서 위 과정을 반복해준다.

3. 결과(시간 복잡도 포함)

콘솔 출력결과 화면

```
workspace - [AI]01_201404377_진승언/src/Main.java - Eclipse IDE
Edit Source Refactor Navigate Search Project Run Wir
Markers Properties Servers Data Source Explorer
<terminated> main [Java Application] C:\Java\jdk1.8.0_181\bin#
start 100 sort
Start InsertionSort
Start BubbleSort
Start selectionsort
start 1000 sort
Start InsertionSort
Start BubbleSort
Start selectionsort
start 10000 sort
Start InsertionSort
Start BubbleSort
Start selectionsort
```

100개 결과화면

```
Main.java SelectionSo... result10000.txt result1000.txt result100.txt
1 InsertionSort result-----
2 253,
3 4564,
4 5340,
5 7441,
6 8543,
7 8796,
8 10909,
9 10923,
10 12440,
11 15034,
12 15314,
13 15747,
14 15754,
15 16862,
16 18615,
17 18644,
18 19063,
19 20316,
20 20410,
21 20972,
22 21267,
23 21345,
24 21404,
25 21428,
26 21483,

Main.java SelectionSo... result10000.txt result1000.txt result100.txt
103 BubbleSort result-----
104 253,
105 4564,
106 5340,
107 7441,
108 8543,
109 8796,
110 10909,
111 10923,
112 12440,
113 15034,
114 15314,
115 15747,
116 15754,
117 16862,
118 18615,
119 18644,
120 19063,
121 20316,
122 20410,
123 20972,
124 21267,
125 21345,
126 21404,
127 21428,
128 21483,
```

```
205 SelectionSort result-----
206 253,
207 4564,
208 5340,
209 7441,
210 8543,
211 8796,
212 10909,
213 10923,
214 12440,
215 15034,
216 15314,
217 15747,
218 15754,
219 16862,
220 18615,
221 18644,
222 19063,
223 20316,
224 20410,
225 20972,
226 21267,
227 21345,
228 21404,
229 21428,
230 21483,
```

1000개 결과화면

```
Main.java InsertionSo... SelectionSo... result1000.txt
1 InsertionSort result-----
2 286,
3 326,
4 384,
5 399,
6 670,
7 736,
8 813,
9 814,
10 836,
11 846,
12 914,
13 1033,
14 1090,
15 1111,
16 1229,
17 1248,
18 1457,
19 1689,
20 1723,
21 1790,
22 1817,
23 1935,
24 2025,
25 2087,
26 2607,
```

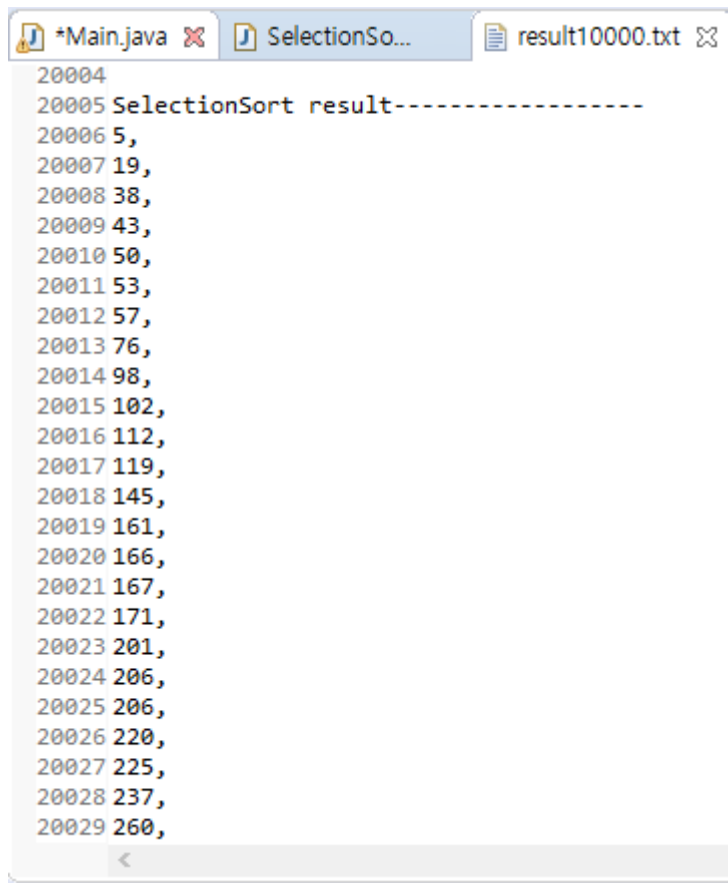
```
Main.java InsertionSo... SelectionSo... result1000.txt result100.txt
1003 BubbleSort result-----
1004 286,
1005 326,
1006 384,
1007 399,
1008 670,
1009 736,
1010 813,
1011 814,
1012 836,
1013 846,
1014 914,
1015 1033,
1016 1090,
1017 1111,
1018 1229,
1019 1248,
1020 1457,
1021 1689,
1022 1723,
1023 1790,
1024 1817,
1025 1935,
1026 2025,
1027 2087,
```

```
Main.java InsertionSo... SelectionSo... result1000.txt
2003 99999,
2004
2005 SelectionSort result-----
2006 286,
2007 326,
2008 384,
2009 399,
2010 670,
2011 736,
2012 813,
2013 814,
2014 836,
2015 846,
2016 914,
2017 1033,
2018 1090,
2019 1111,
2020 1229,
2021 1248,
2022 1457,
2023 1689,
2024 1723,
2025 1790,
2026 1817,
2027 1935,
2028 2025,
```

10000개 결과화면

```
*Main.java SelectionSo... result10000.txt
1 InsertionSort result-----
2 5,
3 19,
4 38,
5 43,
6 50,
7 53,
8 57,
9 76,
10 98,
11 102,
12 112,
13 119,
14 145,
15 161,
16 166,
17 167,
18 171,
19 201,
20 206,
21 206,
22 220,
23 225,
24 237,
25 260,
26 262,
```

```
*Main.java SelectionSo... result10000.txt
10001 99993,
10002
10003 BubbleSort result-----
10004 5,
10005 19,
10006 38,
10007 43,
10008 50,
10009 53,
10010 57,
10011 76,
10012 98,
10013 102,
10014 112,
10015 119,
10016 145,
10017 161,
10018 166,
10019 167,
10020 171,
10021 201,
10022 206,
10023 206,
10024 220,
10025 225,
10026 237.
```


A screenshot of a Java IDE window. The title bar shows three tabs: '*Main.java', 'SelectionSo...', and 'result10000.txt'. The 'result10000.txt' tab is active, displaying a list of numbers from 5 to 260, each preceded by a line number from 20004 to 20029. The numbers are: 5, 19, 38, 43, 50, 53, 57, 76, 98, 102, 112, 119, 145, 161, 166, 167, 171, 201, 206, 206, 220, 225, 237, 260. The text is formatted as 'SelectionSort result-----' followed by the list of numbers.

```
20004
20005 SelectionSort result-----
20006 5,
20007 19,
20008 38,
20009 43,
20010 50,
20011 53,
20012 57,
20013 76,
20014 98,
20015 102,
20016 112,
20017 119,
20018 145,
20019 161,
20020 166,
20021 167,
20022 171,
20023 201,
20024 206,
20025 206,
20026 220,
20027 225,
20028 237,
20029 260,
```

삽입정렬은 최악의 경우엔 $n-1$ 개, $n-2$ 개, ..., 1 개 씩 비교를 반복하여 시간복잡도는 (n^2) 이지만, 이미 정렬되어 있는 경우에는 한번씩 밖에 비교를 하지 않아 시간복잡도는 $O(n)$ 이다. 하지만 상한을 기준으로하는 빅오표기법은 따라서 시간복잡도는 $O(n^2)$ 이다.

버블정렬은 1부터 비교를 시작하여 $n-1$ 개, $n-2$ 개, ..., 1 개씩 비교를 반복하여, 선택 정렬과 같이 배열이 어떻게 되어있던지간에 전체 비교를 진행하므로 시간복잡도는 $O(n^2)$ 이다.

선택정렬은 $n-1$, $n-2$, ..., 1 개씩 비교를 반복한다. 그리고 배열이 어떻게 되어있던지간에 전체 비교를 진행하므로 시간복잡도는 $O(n^2)$ 이다.