자료구조 및 실습
보고서

[제10주] DS_10_201502273_김현종

2018.05.27.

201502273/김현종

1.내용

```java
public interface PriorityQueue {
	boolean isEmpty();
	boolean isFull();

	void add(int element);
	int max();
	int removeMax();
	int size();
}
```

PriorityQueue.java

```java
public interface Heap extends PriorityQueue {
	void heapify(int index);
	void buildHeap();
}
```

Heap.java

```java
public class MaxHeap implements Heap {
		private int[] heap; //
		private int size;
		// Don't add field!!!!!!!
		// Don't add field!!!!!!!
		// Don't add field!!!!!!!
		// Don't add field!!!!!!!

		/*
		 * MaxHeap의 Default Constructor
		 * heap에 해당하는 array를 크기 10짜리 array로 초기화
		 * size = 0
		 */
		public MaxHeap() {
			this.heap = new int[10];
			this.size = 0;
		}

		/*
		 * HeapSort에서 사용되는 private constructor
		 * heap에 해당할 array를 입력받으며 해당 array의 size도 입력받는다.
		 */
		private MaxHeap(int[] array, int size) {
			int[] temp = new int[size + 1];		//입력된 array보다 하나 더 큰 array를 만든다.
			System.arraycopy(array, 0, temp, 1, size);	//새로운 어레이의 0번자리는 비우고 1번부터 입력된 어레이를 복사해넣는다.
			this.heap = temp;
			this.size = size;
			this.buildHeap();  //heap 구성
		}

		/*
		 * MaxHeap이 비었는지를 확인하는 메소드
		 */
		@Override
		public boolean isEmpty() {
			return size == 0;
		}

		/*
		 * MaxHeap이 꽉 찼는지 확인하는 메소드
		 * heap array의 길이보다 하나 작은 값이 size와 같으면
		 * 꽉 찬것이다. (0번자리가 비어있기때문)
		 */
		@Override
		public boolean isFull() {
			return size == heap.length - 1;
		}

		/*
```

```
	 * 새로운 값을 Heap에 넣는 메소드
	 * Heap이 꽉 찼으면 resize를 해주고 다시 add를 호출한다.
	 * size를 하나 올리고 heap의 맨 뒤에 새로운 값을 넣어준 뒤
	 * buildHeap을 호출한다.
	 */
	@Override
	public void add(int element) {
		// TODO : Fill it !!
		if (this.isFull()) {
			resize();
			add(element);
		} else {
			this.size++;
			this.heap[this.size] = element;
			this.buildHeap();
		}
	}

	/*
	 * 현재 Heap에서 가장 큰 값을 반환하는 메소드
	 * 비었으면 201502273을 반환하고
	 * 값이 있으면 heap array의 1번 자리에 있는 값을 반환한다.
	 */
	@Override
	public int max() {
		// TODO : Fill it !!
		if (this.isEmpty()) {
			return 201502273;
		} else {
			return this.heap[1];
		}
	}

	/*
	 * 가장 큰 값을 제거하는 메소드
	 * Heap이 비었으면 201502273을 반환하며
	 * 1번위치의 값을 따로 저장한 뒤
	 * 값이 있으면 가장 뒤에 있는 값과 1번자리의 값을 바꾼 후 size를 1 줄이고
	 * buildHeap을 호출한다.
	 * 그리고 저장해뒀던 1번자리 값을 반환한다.
	 */
	@Override
	public int removeMax() {
		// TODO : Fill it !!
		if (this.isEmpty()) {
			return 201502273;
		} else {
			int temp = this.heap[1];
			MaxHeap.swap(this.heap, 1, this.size);
			// this.heap[this.size] = Integer.MIN_VALUE;
			this.size--;
			this.buildHeap();
			return temp;
		}

	}

	/*
	 * size의 getter
	 */
	@Override
	public int size() {
		return size;
	}

	/*
	 * 해당 노드와 그 하위 노드를 MaxHeap으로 만들기 위한 메소드
	 * 해당 index의 노드와 그 노드의 child를 비교하여 가장 큰 값을 root로 설정하고
	 * 바뀐 child를 root로 하여 재귀적으로 실행한다.
	 */
	public void heapify(int index) {
		// TODO : Fill it !!
```

```
                if (!this.isEmpty()) {
                        int left = Integer.MIN_VALUE, right = Integer.MIN_VALUE;
                        if (2 * index <= this.size) {
                                left = this.heap[2 * index];
                        }
                        if (2 * index + 1 <= this.size) {
                                right = this.heap[2 * index + 1];
                        }
                        if (left < right) {
                                if (right > this.heap[index]) {
                                        MaxHeap.swap(this.heap, index, 2 * index +
1);
                                        this.heapify(2 * index + 1);
                                }
                        } else {
                                if (left > this.heap[index]) {
                                        MaxHeap.swap(this.heap, index, 2 * index);
                                        this.heapify(2 * index);
                                }
                        }
                }
        }

        /*
         * heapify를 child가 있는 모든 노드에서 실행시켜 가장 큰 값을 root로 보내는
함수이다.
         */
        public void buildHeap() {
                // TODO : Fill it !!
                for (int i = this.size / 2; i > 0; i--) {
                        this.heapify(i);
                }
        }

        /*
         * target과 같은 값을 갖고있는 노드의 값을 num으로 바꾸고 다시 MaxHeap을
만족하도록 해준다.
         */
        public boolean increaseKey(int target, int num) {
                // TODO : Fill it !!
                for (int i = 1; i <= this.size; i++) {
                        if (target == this.heap[i]) {
                                this.heap[i] = num;
                                this.buildHeap();
                                return true;
                        }
                }
                return false;

        }

        /*
         * MaxHeap을 이용하여 sorting을 해주는 static method
         * sorting을 원하는 array를 입력해주고 해당 array로 MaxHeap을 만들어
         * Max값을 뽑아내며 sorting을 진행한다.
         */
        public static int[] heapSort(int[] array, int size) {
                // TODO : Fill it !!
                MaxHeap sorting = new MaxHeap(array, size);
                int[] sorted = new int[size];
                int i = 0;
                while (!sorting.isEmpty()) {
                        sorted[i++] = sorting.removeMax();
                }

                return sorted;
        }

        /*
         * print했을때의 모습을 지정해주는 함수
         * 현재 heap array에 들어있는 순서대로 출력해준다.
         */
```

```
	@Override
	public String toString() {
		StringBuilder stringBuilder = new StringBuilder();
		stringBuilder.append("{ ");
		for (int i = 1; i <= size; ++i) {
			stringBuilder.append(heap[i]).append(" ");
		}
		stringBuilder.append("}");

		return stringBuilder.toString();
	}

	/*
	 * array의 size를 키워주는 함수
	 * 기존의 2배크기의 array를 만들고 기존 array를 복사해준다.
	 */
	private void resize() {
		int[] newHeap = new int[this.heap.length * 2];
		System.arraycopy(this.heap, 0, newHeap, 0, this.heap.length);
		this.heap = newHeap;
	}

	/*
	 * 두 노드를 바꾸어주는 메소드
	 * 입력된 array에서 i와 j에 있는 값을 서로 바꾸어준다.
	 */
	private static void swap(int[] array, int i, int j) {
		int temp = array[i];
		array[i] = array[j];
		array[j] = temp;
	}
}
```

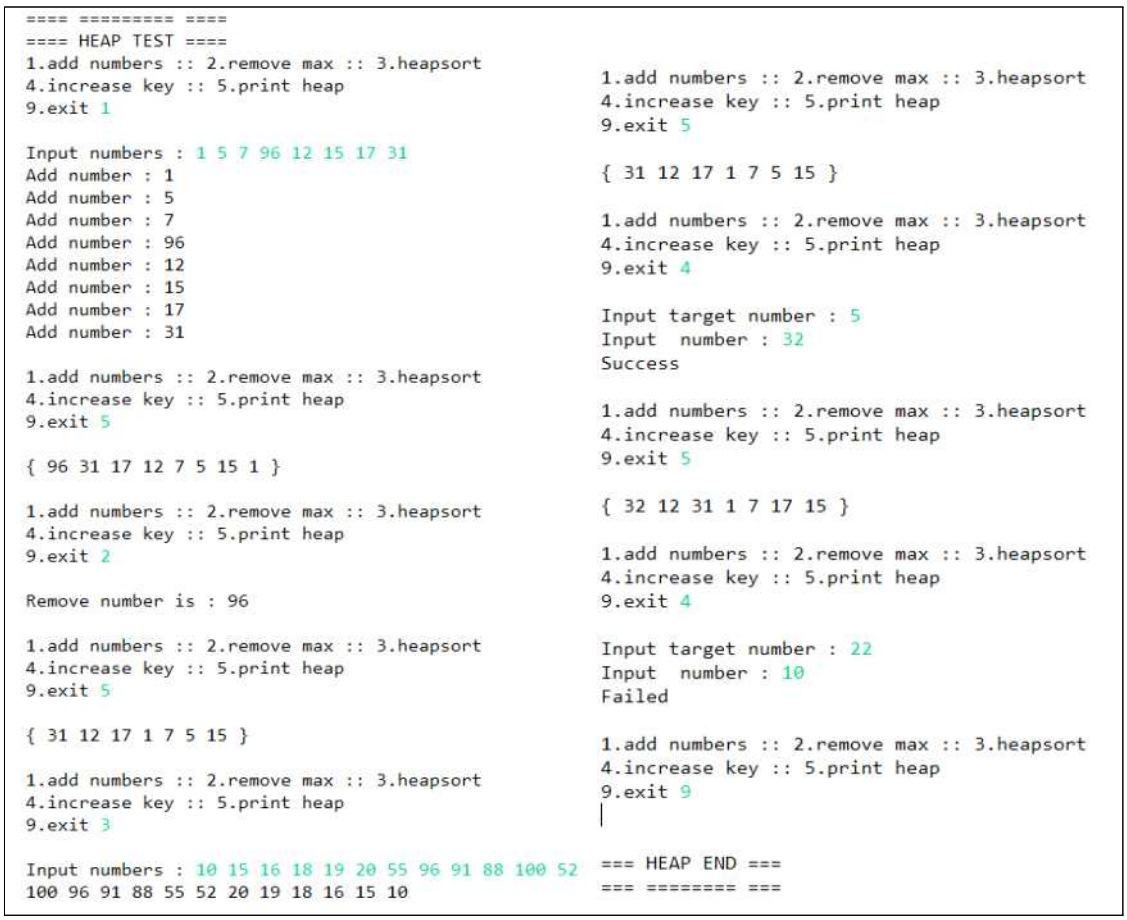
MaxHeap.java

```
import java.util.Scanner;

public class MainClass_10_201502273 {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("==== ========= ====");
        System.out.println("==== HEAP TEST ====");

        int num = 0;

        MaxHeap maxHeap = new MaxHeap();
        while (num != 9) {
            System.out.println("1.add numbers :: 2.remove max :: 3.heapsort");
            System.out.println("4.increase key :: 5.print heap");
            System.out.print("9.exit ");
            num = scanner.nextInt();
            System.out.println();
            switch (num) {
                case 1: {
                    scanner.nextLine();
                    System.out.print("Input numbers : ");
                    String numbers = scanner.nextLine();
                    String[] nums = numbers.split(" ");
                    int[] ns = new int[nums.length];
                    for (int i = 0; i < nums.length; ++i) {
                        ns[i] = Integer.parseInt(nums[i]);
                    }
                    for (int n : ns) {
                        maxHeap.add(n);
                        System.out.println("Add number : " + n);
                    }
                    break;
                }
                case 2: {
                    System.out.print("Remove number is : ");
                    int max = maxHeap.removeMax();
                    System.out.println(max);
```

```
                    break;
                }
                case 3: {
                    scanner.nextLine();
                    System.out.print("Input numbers : ");
                    String numbers = scanner.nextLine();
                    String[] nums = numbers.split(" ");
                    int[] ns = new int[nums.length];
                    for (int i = 0; i < nums.length; ++i) {
                        ns[i] = Integer.parseInt(nums[i]);
                    }
                    int[] maxs = MaxHeap.heapSort(ns, ns.length);
                    for (int i : maxs) {
                        System.out.print(i + " ");
                    }
                    System.out.println();
                    break;
                }
                case 4: {
                    System.out.print("Input target number : ");
                    int target = scanner.nextInt();
                    System.out.print("Input  number : ");
                    int next = scanner.nextInt();
                    if (maxHeap.increaseKey(target, next))
                        System.out.println("Success");
                    else
                        System.out.println("Failed");
                    break;
                }
                case 5: {
                    System.out.println(maxHeap);
                    break;
                }
                default: {
                    break;
                }
            }
            System.out.println();
        }

        System.out.println("=== HEAP END ===");
        System.out.println("=== ======== ===");
    }
}
```

MainClass_10_201502273.java

2.결과



```
==== ========= ====
==== HEAP TEST ====
1.add numbers :: 2.remove max :: 3.heapsort
4.increase key :: 5.print heap
9.exit 1

Input numbers : 1 5 7 96 12 15 17 31
Add number : 1
Add number : 5
Add number : 7
Add number : 96
Add number : 12
Add number : 15
Add number : 17
Add number : 31

1.add numbers :: 2.remove max :: 3.heapsort
4.increase key :: 5.print heap
9.exit 5

{ 96 31 17 12 7 5 15 1 }

1.add numbers :: 2.remove max :: 3.heapsort
4.increase key :: 5.print heap
9.exit 2

Remove number is : 96

1.add numbers :: 2.remove max :: 3.heapsort
4.increase key :: 5.print heap
9.exit 5

{ 31 12 17 1 7 5 15 }

1.add numbers :: 2.remove max :: 3.heapsort
4.increase key :: 5.print heap
9.exit 3

Input numbers : 10 15 16 18 19 20 55 96 91 88 100 52
100 96 91 88 55 52 20 19 18 16 15 10

1.add numbers :: 2.remove max :: 3.heapsort
4.increase key :: 5.print heap
9.exit 5

{ 31 12 17 1 7 5 15 }

1.add numbers :: 2.remove max :: 3.heapsort
4.increase key :: 5.print heap
9.exit 4

Input target number : 5
Input  number : 32
Success

1.add numbers :: 2.remove max :: 3.heapsort
4.increase key :: 5.print heap
9.exit 5

{ 32 12 31 1 7 17 15 }

1.add numbers :: 2.remove max :: 3.heapsort
4.increase key :: 5.print heap
9.exit 4

Input target number : 22
Input  number : 10
Failed

1.add numbers :: 2.remove max :: 3.heapsort
4.increase key :: 5.print heap
9.exit 9

=== HEAP END ===
=== ======== ===
```

주어진 출력 예시와 똑같이 잘 작동하는 것을 볼 수 있다. 정렬되지 않은 숫자 배열을 입력해도 해당 배열 중 가장 큰 값은 가장 왼쪽에, 즉, Root에 저장되는 것을 볼 수 있고 static method인 heapsort를 사용해 sorting을 한 뒤 print heap을 해보면 원래 있던 heap이 그대로 있는 것도 확인할 수 있었다. 또한 increaseKey를 한 경우 값이 heap에 존재하면 잘 찾아서 바꾸고 heap에 값이 없으면 failed라고 뜨는 것을 확인했다.

깨달은 점 및 결론

array의 0번째 자리를 비워두고 구현을 하였는데 기존에 짜여져 있던 코드는 0번을 비우지 않고 구현했던 것이라 좀 맞지 않는 부분이 있었다. toString에서 0번부터 size-1까지 순회하는 것을 1번부터 size까지 도는 것으로 수정하고 private Constructor의 경우 받은 array를 한 칸 Right shift하여 0번 자리를 비우는 전처리를 하는 등 자잘한 부분들을 손봐야 했다. 매 연산을 할 때마다 index에 + 1 연산을 해주는 것보다 0번 자리에는 heap의 size를 저장하고 1번부터 heap을 만들면 RAM도 낭비되지 않고 연산량도 줄어들 것으로 생각된다.