알고리즘

- HW05 : Greedy Algorithm-

제 출 일	2017년 10월 26일
분 반	02반
담당교수	공은배
학 과	컴퓨터공학과
학 번	201302423
이 름	신종욱

ㄱ.Lateness

1. 해결 방법

Dj기준으로 정렬된순서로 입력이 되기 때문에 구현이 아주 편하였다.
앞의 일의양은 workT에 저장하고 끝나는 시간은 DeadT배열에 저장했다
처음에는 총갯수가 들어오기 때문에 0인덱스에는 workT에 개수가 저장되고 DeadT는 초기값인 0이 저장된다.
그후 time과 lateness를 0으로 초기화후 1번 인덱스부터 읽으면서 time을 더하고 Dj가 넘었는지 확인하면서 구현하였다. 넘었을시에는 lateness보다 넘었을 경우 변경했다.

```
int n= workT[0];//0번째 배열엔 읽은 일의 갯수가 저장되어있다.
int time = 0;//시작시간은 0으로 초기화
int lateness = 0;//lateness는 최소 0이기때문에 0으로 초기화했다.
System.out.println("Input Data : \n" + n);
for(int i=1;i<n+1;i++) {//0에는 일의 갯수라서 1부터 읽으면서 for문을 진행한다
    System.out.println(workT[i]+ "   " +deadT[i]);
time+=workT[i];//일단 일한 시간을 현재 time에 더한다
if(deadT[i]<time) {//그시간이 종료시간을 넘었으면 Lateness 발생
    if(lateness<time-deadT[i])//현재 저장된 lateness보다 크다면
        lateness=time-deadT[i];//lateness 값을 바꿔준다.
}
}
System.out.println("Output Data : "+lateness);
```

2. 실행 결과

```
<terminated> Lateness [Java Application] C:₩Program Files₩Java₩jre1.8.0_144₩bin₩javaw.exe (2017. 10. 25. 오후 11:14:19)
Input Data :
6
3  6
2  8
1  9
4  9
3  14
2  15
Output Data : 1
```

ㄴ.Huffman

1. 해결 방법

일단 MinHeap은 저번에 사용한 MaxHeap에서 부등호만 바꿔서 구현하였다.

```
                for (int i = 0; i < word.length(); i++) {
                    char temp = word.charAt(i);
                    if (huf.containsKey(temp))
                        huf.put(temp, huf.get(temp) + 1);
                    else
                        huf.put(temp, 1);// 해쉬맵을 이용해서 문자를 읽으면서 가중치를 더한다 최초이면
                                         // 1로지정 이미 있는 거라면 +1을 해준다.
                }}
            line = in.readLine();
        }
        in.close();
    } catch (IOException e) {}

    for (char Cha : huf.keySet()) {
        // 현재 해쉬맵에 있는 문자를 한개씩 읽으면서 인쇄할방법을 찾다가 for문기능중 ':'가 있다는걸 알았다 Cha에
        // huf.keyset값을 한개씩 넣으면서 진행하는 for문이다
        Huffman temp = new Huffman(Cha, huf.get(Cha));
        tree.add(temp);// 그와 동시에 그값들을 우선순위 큐에 추가한다.
    }
```

문자열을 읽으면서 같은 문자가있으면 빈도수를 증가시켜야하는데 그런구조에 적합한 자료구조는 HashMap이다 자바내에 내장된 HashMap을 사용하여서 문자열을 저장시켜줬다.
그리고 그값들을 허프만트리(Minheap)에 추가하여서 초기상태를 만들었다.

그런다음 이제 최소 빈도수 두 개를 뽑고 다시 힙에 넣어주는 과정을 반복해야한다.

```
public Huffman maketree(HeapPriorityQueue A, int n) {// 우선순위 큐인 tree를 이용하여서
                                                     // 최종적으로 사용할 허프만 트리를
                                                     // 구현하는 과정이다
    for (int i = 1; i < n; i++) {
        Huffman temp = new Huffman(' ', 0);
        temp.lchild = (Huffman) A.remove();
        temp.rchild = (Huffman) A.remove();// string값이 공백이고 왼쪽오른쪽 자식이 현재
                                           // 우선순위큐에있는 최소가중치 두개를 뽑아서 만든다.

        temp.freq = temp.rchild.freq + temp.lchild.freq;// 해당 가중치는 자식두개를
                                                        // 더한값으로 한다
        A.add(temp);// 그리고 그 트리도 다시 들어가서 허프만코드를 구성한다
    } // 총 갯수가 n개라면 n-1번하면 모두 완성된다.
    return (Huffman) A.remove();// 그럼 최종적으로있는 루트에있는 트리가 허프만트리가 구현되었으니 리턴한다

}
```

일단 Root가 공백인 트리를 만들고 우선순위큐에서 최소값 2개를 가져와서 left,rightchild에 추가시켜준다.
그런다음 만든 트리를 다시 우선순위큐에 넣는다
이과정을 n-1번 반복하면 허프만 트리가 완성된다.

이제 트리를 이용하여서 0과 1로 나타내면된다.

```
public static void makecode(Huffman A, String S, HashMap<Character, String> B) {
    if (A == null)
        return;
    makecode(A.rchild, S + "1", B);// 오른쪽 자식으로 갈경우 '1'스트링을 더하고 계속 실행
    makecode(A.lchild, S + "0", B);// 왼쪽 자식으로 갈경우 '0'스트링을 더하고 계속 실행
    if (A.alphabet != ' ') {// 만약 허프만코드의 문자가 공백이아니라면 해당문자열에 도착하였음으로 그값을 해쉬맵인
                            // B에 저장한다
        B.put(A.alphabet, S);// 순회하는 방식으로 코드를 구현하였다.
    }
}
```

재귀하는 방식으로 트리를 순회하면서 문자열을 만날때까지 0과 1을 추가한다 만나면 hasp맵에 해당문자열과 코드를 저장한다.
그후 만들어진 해쉬맵을 출력하면된다.

2. 실행 결과

```
<terminated> HuffmanCode [Java Application] C:₩Program Files₩Java₩jre1.8.0_144₩bin₩javaw.exe (2017. 10. 25. 오후 11:21:57)
Input data :
aabbccadefaabcdffg

OutPutData :
a : 10
b : 110
c : 111
d : 010
e : 0110
f : 00
g : 0111
```

3. 느낀 점

Lateness는 Dj가 정렬된 채로 입력하여서 쉽게구현하였는데 만약 정렬이 안되어있었다면 comparable을 이용하여서 정렬후 for문을 돌리면 될것같다.
huffman코드는 해본적이 있었지만 다시 새롭게 할려니 조금 걸렸다 그래도 해쉬맵을 사용해야한다는걸 알고있어서 저번에 한것보단 쉽게구현한편이였다.