




알고리즘 -4주차

201902694 – 박재우

RANK	TEAM	SCORE	1-ANAGRAM  [1 POINT]	2-LPS  [1 POINT]	3-KAKAODART  [3 POINTS]	4-CHANGE  [5 POINTS]
9	201902694	10 244	11 1 try	15 1 try	46 1 try	172 1 try

Submissions			
time	problem	lang	result
09/28/20-17:02	4-CHANGE	JAVA	CORRECT
09/28/20-14:57	3-KAKAODART	JAVA	CORRECT
09/28/20-14:26	2-LPS	JAVA	CORRECT
09/28/20-14:21	1-ANAGRAM	JAVA	CORRECT

문제 1

a. 문제/목표

아나그램 판별하기

b. 해결방법(주요 소스코드 첨부. 스크린샷 or 코드 cv)

```
package algorithm.week5;

import java.util.Arrays;

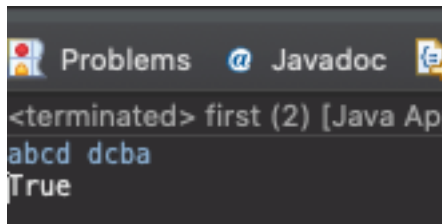
public class first {

    public static boolean Anagram(String str1, String str2) {
        str1 = str1.toLowerCase();
        str2 = str2.toLowerCase();
        char[] a = str1.toCharArray();
        char[] b = str2.toCharArray();
        Arrays.sort(a);
        Arrays.sort(b);
        if (Arrays.equals(a, b))
            return true;
        return false;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String s1 = sc.next();
        String s2 = sc.next();
        if (Anagram(s1, s2))
            System.out.print("True");
        else
            System.out.print("False");
    }
}
```

두개의 string을 입력받아 전부 알파벳 순서대로 정렬한뒤에 배열의 시작부터 끝까지 비교하여 끝까지 같다면 true를, 다르다면 false를 출력한다.

c. 결과



시간복잡도 : $O(n)$ -> string의 크기만큼 비교

자신의 생각 및 느낀점 : 이론시간에 다른 내용이라 딱히 어려움은 없었다.

질문 : 이론시간에 다른 문제이므로 딱히 질문사항은 없다.

문제 2

a. 문제/목표

입력된 문자열의 palindrome 확인

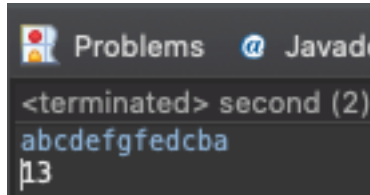
b. 해결방법(주요 소스코드 첨부. 스크린샷 or 코드 cv)

```
package algorithm.week5;
import java.util.Scanner;
public class second {
    public static int is_palindrome(String s) {
        boolean[][] table = new boolean[s.length()][s.length()];
        int longest = 0;
        for (int i=0; i<s.length(); i++) {
            for (int j = 0; j<s.length()-i; j++) {
                if(i<2) {
                    if (s.charAt(j) == s.charAt(i+j)) {
                        table[j][i+j] = true;
                        longest = i + 1;
                    }
                    else
                        table[j][j+i] = false;
                }
                else {
                    if (s.charAt(j) == s.charAt(i+j) && table[j+1][i+j-1]) {
                        table[j][i+j] = true;
                        longest = i + 1;
                    }
                    else
                        table[j][i+j] = false;
                }
            }
        }
        return longest;
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String str = sc.nextLine();
        str = str.replace(" ", "");
        str.toLowerCase();
        System.out.print(is_palindrome(str));
    }
}
```

왼쪽 끝과 오른쪽 끝이 같다면, 그 사이의 숫자가 팰린드롬이라면, 이 숫자는 팰린드롬이 된다. 위와 같이 길이순으로 루프를 돌면서, 계산된 값을 이용하게 되는 메모라이즈 방식을 사용한다. 1 일때는 무조건 팰린드롬이고, 길이가 2 일 때는, 두 수가 같으면 팰린드롬이다.

For 문을 돌면서 중간에 palindrome 인 문자열의 길이를 기록하여 최장인값을 리턴한다.

c. 결과



시간복잡도 : $O(n^2)$ -> 이중 for문

자신의 생각 및 느낀점 : 이론시간에 다른 내용이라 딱히 어려움은 없었다.

질문 : 이론시간에 다른 문제이므로 딱히 질문사항은 없다.

문제 3

a. 문제/목표

입력받은 다트점수 계산

b. 해결방법(주요 소스코드 첨부. 스크린샷 or 코드 cv)

```
package algorithm.week5;
import java.util.Scanner;
public class third {
    public static int getScore(String point) {
        int[] ar = new int[3];
        int curIdx = 0;
        int stars = 0;
        int binx = 0;
        String tempNum = "";
        for (char c : point.toCharArray()) {
            if (Character.isDigit(c)) {
                tempNum = tempNum + c;
            }
            else {
                if (!tempNum.equals("")) {
                    ar[curIdx++] = Integer.parseInt(tempNum);
                    tempNum = "";
                }
                switch (c) {
                    case 'S':
                        ar[curIdx-1] = (int) Math.pow(ar[curIdx-1], 1);
                        break;
                    case 'D':
                        ar[curIdx-1] = (int) Math.pow(ar[curIdx-1], 2);
                        break;
                    case 'T':
                        ar[curIdx-1] = (int) Math.pow(ar[curIdx-1], 3);
                        break;
                    case '*':
                        stars++;
                        ar[curIdx-1] *= 2;
                        binx = curIdx-1;
                        break;
                    case '#':
                        ar[curIdx-1] = ar[curIdx-1] * -1;
                        break;
                }
                if (stars != 0 && curIdx-1 != binx) {
                    ar[curIdx-1] *= 2;
                    stars = 0;
                }
            }
        }
        return ar[0] + ar[1] + ar[2];
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String str = sc.next();
        System.out.print(getScore(str));
    }
}
```

입력받은 점수 string 을 char 단위씩 하나하나 인덱스로 탐색한다.

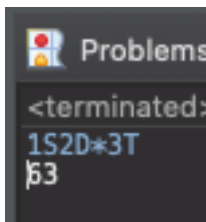
S,D,T,*,#일 경우 직전 인덱스가 점수임으로 각각의 문자에 맞게 점수 배수해서 더한다.

*인경우 현재 점수와 다음 점수가 2 배가 되므로 현재점수를 2 배를 시키고 *이었던 것을 star 카운터를 올려 저장한다. 동시에 *이었던 index 번호를 저장한다.

다음점수를 계산할때 star 카운터가 올라가 있고 앞서 저장한 index 번호와 다르다면 직전에 *이 있었다는 것 이므로 현재 점수를 2 배를 한다.

Star 카운터는 0 으로 초기화 해준다.

c. 결과



시간복잡도 : $O(n)$ ->string 길이만큼 작동

자신의 생각 및 느낀점 :

이론시간에 다루었었지만 조건이 달라지면서 어떻게 처리해야 할지 꽤 오래 고민했던것 같다.

질문 : 우선 이론시간에 배운대로 사용하였지만 내가 사용한 방법이외에 무엇이 있는지 궁금하다

문제 4

a. 문제/목표

b. 해결방법(주요 소스코드 첨부. 스크린샷 or 코드 cv)

```
package algorithm.week5;

import java.util.*;

public class fourth {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        int n = sc.nextInt();
        int[] a1 = new int[n];
        int[] a2 = new int[n];
        for(int i = 0; i < n; i++) {
            a1[i] = sc.nextInt();
        }
        int num = sc.nextInt();
```

```

Arrays.sort(a1);
for(int i = n-1; i >= 0; i--) {
    if(num >= a1[i]) {
        a2[i] = num/a1[i];
        num %= a1[i];
    }
}
for(int i = 0; i < n; i++) {
    if(a2[i] != 0) {
        System.out.println(a1[i] + " " + a2[i]);
    }
}
}

```

입력받은 액권수만큼 배열을 2개 생성한다

하나를 2번째로 입력된 금액들의 수를 배열에 넣는다

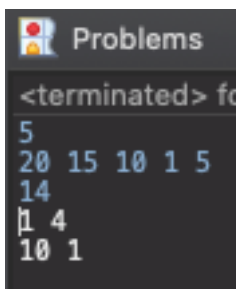
이를 순서대로 정렬한뒤에 for문을 통해 큰수부터 3번째로 입력받은 수를

나누고 몫을 2번째로 생성한 배열에 for문의 i에 맞게 넣는다.

Num은 나눈 나머지로 재저장해서 num이 나눌 수보다 작을 때까지 반복한다.

그후 결과를 output에 맞게 출력한다.

c. 결과



```

Problems
<terminated> fo
5
20 15 10 1 5
14
4
10 1

```

시간복잡도 : $O(n)$ -> 액권수 만큼 반복

자신의 생각 및 느낀점 :

처음에는 지난주에 했던 가위바위보 최단거리를 통해

최소 개수를 구한뒤에 더해진 숫자의 개수를 통해

각각 출력하려 했지만 재귀문에서 그 개수별로 출력하는

것이 매우 까다로웠다. 그래서 상당히 시간을 잡아먹었으며

이방법을 사용하면 시간복잡도가 상당히 커져 포기했는데

단순하게 나누고 계산하는 방법으로 쉽게 생각하니 금방 풀렸다.

너무 어렵게 생각하지 않고 단순하게 생각해봐야한다는 것을 느꼈다.