# Algorithm HW2

## 1. Problem Set

1)

2) 연산의 다양성

3) 회문

4) 놀이판

5) 탑 쌓기

각각의 폴더에는 문제설명, 뼈대코드, 샘플입력, 샘플정답이 들어있습니다.

**뼈대코드의 이름과 클래스명을 바꾸지 않고 그대로 그 위에 구현하도록 합니다.**

작성한 알고리즘의 **시간 복잡도에 대한 분석**을 주석으로 뼈대코드마다 서술합니다.

## 2. 유의사항

시작 전에 반드시 최신 버전의 Java를 설치하고 **사용하는 편집기에서 설치한 최신 Java 빌드를 사용하도록 세팅해야 합니다. 소스코드의 인코딩은 utf8**으로 설정하도록 합니다.
(Eclipse의 경우 지원이 안 될 수 있어서 한글 깨짐 발생)

멀티 쓰레딩 및 기타 외부 라이브러리 사용을 금지합니다.

## 3. 제출 양식 및 기한

각각 문제에 해당하는 java 파일명(ex. Solution1.java) 그대로 제출합니다. 파일명이 일치하지 않아 채점 코드가 작동하지 않을 시 감점 5%가 있습니다. etl.snu.ac.kr 과목페이지에 문제번호 별로 각각 제출하도록 합니다. 이는 압축했을 시 다양한 파일들(특히나 이미 컴파일 된 파일)이 같이 첨부되는 것을 방지하기 위함입니다. 딜레이 패널티는 1일마다 20% 입니다.

- 제출 기한 : 2020-05-19 화 11:59 pm
- 문제 2), 3), 4), 5) 4개 모두 제출

## 2. 연산의 다양성

{a, b, c} 세 원소로 이루어지는 연산 테이블이 아래 그림과 같이 주어진다. 예를 들어, ab=b, ba=c, cc=c 와 같이 계산된다. a(b((cb)a)) = a 이다. 괄호를 치는 방법에 따라 연산의 순서가 달라지고 결과도 달라진다. {a, b, c}로 이루어진 임의의 문자열 $x_1x_2\cdots x_n$ ($x_i \in \{a, b, c\}$)에 대하여 결과가 a, b, c가 되는 괄호치기 방법의 총 수는 각각 몇 가지인지 계산하는 알고리즘을 작성하라. 예를 들어, 문자열 abaa의 경우를 보자. 결과가 a가 되는 괄호치기 방법은 ((ab)a)a 한 가지이다. b가 되는 방법은 ((ab)(aa)), (a(ba))a, a((ba)a), a(b(aa)) 4가지이다. c가 되는 방법은 없다.

a: ac, bc, ca
b: aa, ab, bb
c: ba, cb, cc

| | a | b | c |
|---|---|---|---|
| a | b | b | a |
| b | c | b | a |
| c | a | c | c |

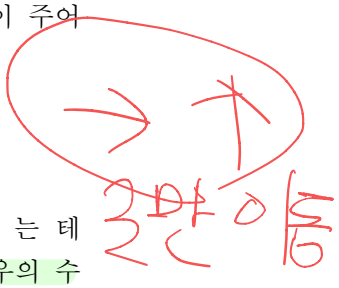
알고리즘은 최대한 효율적으로 작성하라. 10개의 케이스 모두를 수행한 알고리즘의 총 수행시간이 1초를 넘지 않아야 한다. 채점을 위한 컴파일 시에 최적화 옵션은 쓰지 않는다.

**[입력]**

입력 파일에는 10 개의 테스트 케이스가 주어진다. 각 케이스는 2 줄로 이루어진다. 첫 줄에는 N 이 주어지고, 다음 줄에는 a, b, c로 이루어진 문자열(길이는 1부터 30사이)이 주어진다. 입력파일의 이름은 “input2.txt”이다.

**[출력]**

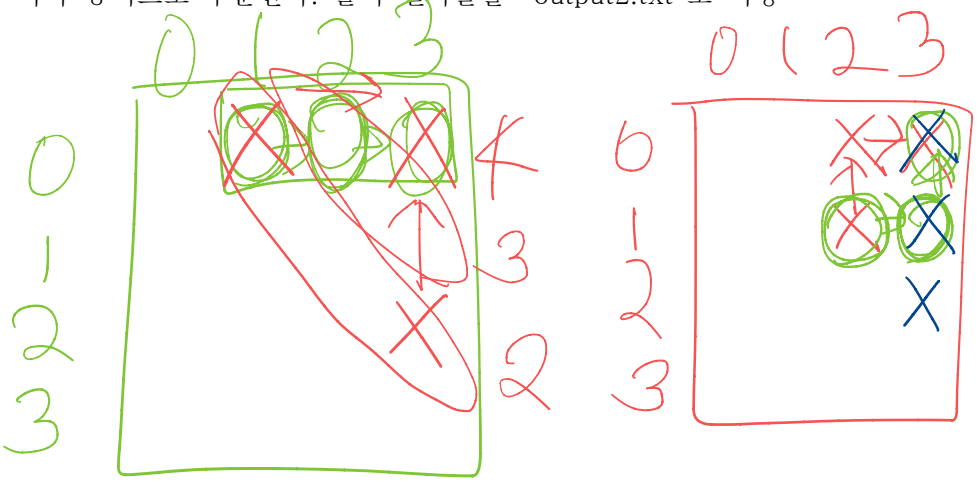
각 테스트 케이스에 대해서, 케이스의 번호를 "#x" 의 형식으로 출력한 후(여기서 x 는 테스트 케이스 번호), 공백을 하나 둔 다음 주어진 케이스에서 각각 a, b, c가 되는 경우의 수들을 나열한다. 각 수 사이는 역시 공백으로 구분한다. 출력 결과물을 “output2.txt”로 저장한다.



**[예제]**

**입력 (input2.txt)**

→ ↑ 2만 이동
시작 지점에서
연속한 것



```
2                                                    ← 1번 케이스
ac
3                                                    ← 2번 케이스
bbc
...
```

출력 (output2.txt)

```
#1 1 0 0
#2 1 0 1
...
```

이미 계산된 결과

↓ ↓

더큰발호 ← 이전괄호 + 이전 발호

일단 재귀라도 생각

merge 방법 ① (aa)(aa)

② ((aa)a)

③ (a(aa))

S를 배열에 넣고

근데 괄치는 부분이 있는데... ✱을 어떻게 저장?

(a (✱)) ((✱)a)

## 3. 회문

읽는 순서를 뒤에서 시작하여도 앞에서부터 읽는 것과 똑같은 문장, 혹은 문자열을 회문이라고 한다. 문자의 집합 {A, B, C, D}를 알파벳으로 삼는 언어가 있다고 가정해보자. ABBCDAA는 이 언어에 속하는 문자열의 한 예이다. 이 때 문자열의 부분순서라 함은 A, B, ABC, ABBA, ACDAA, ACA, BD, BDA 등이 될 수 있다. 이 예시 중에서 앞에서부터 읽거나 뒤에서부터 읽어도 같은 부분순서는 A, B, ABBA와 ACA, 이 넷이다.

이와 같이 {A, B, C, D}로 이루어진 문자열에는 부분순서가 있을 것이다. 이 부분순서들 중에서 회문의 조건을 만족함과 동시에 가장 긴 부분순서의 길이를 구하는 프로그램을 작성하라. 알고리즘은 최대한 효율적으로 작성하라. 10개의 케이스 모두를 수행한 알고리즘의 총 수행 시간이 10초를 넘지 않아야 한다. 채점을 위한 컴파일 시에 최적화 옵션은 쓰지 않는다.

**[입력]**

입력 파일에는 10 개의 테스트 케이스가 주어진다. 각 케이스는 2 줄로 이루어진다. 첫 줄에는 문자열의 길이가 주어지고, 다음 줄에는 A, B, C, D로 이루어진 문자열(길이는 10 이상 5000 이하)이 주어진다. 입력파일의 이름은 “input3.txt”이다.

**[출력]**

각 테스트 케이스에 대해서, 케이스의 번호를 "#x" 의 형식으로 출력한 후(여기서 x는 테스트 케이스 번호), 공백을 하나 둔 다음 주어진 케이스에서 입력된 문자열에 대해 회문인 최장 부분순서의 길이를 출력한다.

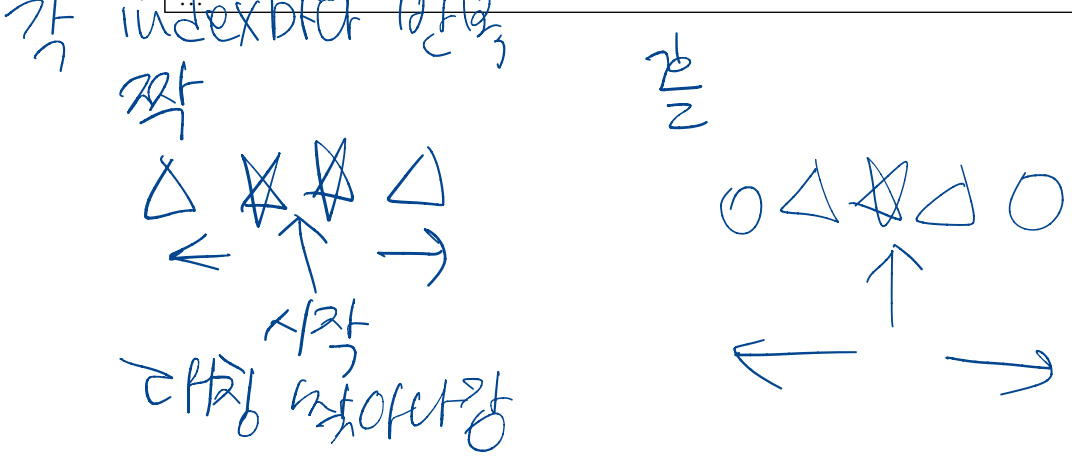
**[예제]**

**입력 (input3.txt)**

```
ABBCDAAAAA                ← 1번 케이스
ABCDDDDCCDDAABB           ← 2번 케이스
…
```

**출력 (output3.txt)**

```
#1 6
#2 10
…
```



# 4. 놀이판

길이가 N 이고 3 줄짜리 놀이판이 주어지고 놀이판의 각 칸에는 1 부터 1000 양의 정수가 쓰여 있다. 이 놀이판의 모든 칸에 ○, △, ×를 써서 표시하는데, 세로 열마다 ○, △, ×를 각각 한 번씩 모두 사용해야 하고, **가로로 이웃하는 두 칸에는 같은 표시가 나타나서는 안된다**. 이렇게 하면 ○, △, ×가 각각 n 번씩 나타나게 된다.

모든 칸에 ○, △, ×를 표시하고 난 다음 이에 대해 점수를 매기는데, ○로 표기한 칸의 수는 더해주고, ×로 표기한 칸의 수는 빼주고, △로 표시한 칸의 수는 무시한다(0 으로 처리). 아래 그림은 놀이판을 규칙대로 ○, △, × 표시한 예시이다..

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 8 | 11 | 15 | 9 | 28 | 19 | 16 | 41 | 34 | 28 | 9 |
| 10 | 13 | 9 | 16 | 20 | 18 | 32 | 26 | 15 | 37 | 24 | 3 |
| 5 | 6 | 7 | 16 | 25 | 31 | 21 | 29 | 39 | 29 | 19 | 10 |

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| ○ | △ | ○ | × | ○ | × | △ | ○ | △ | ○ | × | ○ |
| × | ○ | × | △ | × | △ | ○ | × | ○ | × | △ | × |
| △ | × | △ | ○ | △ | ○ | × | △ | × | △ | ○ | △ |

↓

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| +2 | | +11 | −15 | +9 | −28 | | +16 | | +34 | −28 | +9 |
| −10 | +13 | −9 | | −20 | | +32 | −26 | +15 | −37 | | −3 |
| | −6 | | +16 | | +31 | −21 | | −39 | | +19 | |

첫번째 표는 주어진 놀이판, 두번째 표는 ○, △, ×를 채운 방법, 세번째 표는 계산규칙에 따라 더하는 수, 빼는 수, 무시하는 수를 나타내고 있다. 위의 방식으로 채울 때 점수는 2 − 10 + 13 − 6 + … + 9 − 3 = −35 가 된다.

놀이판이 주어졌을 때 ○, △, ×를 표시하여 얻을 수 있는 최대 점수를 구하는 알고리즘을 작성하라. N 은 1,000,000 을 넘지 않는다. 여러 가지 크기의 문제로 테스트하므로 효율이 좋은 알고리즘일수록 점수를 많이 받을 것이며 원시적인 알고리즘으로는 큰 점수를 기대할 수 없다. 컴파일 시 옵티마이제이션 옵션은 사용하지 않는다. 10 개의 케이스를 모두 수행한 총 시간의 합이 1 초를 넘지

않아야 한다.

**[입력]**

입력 파일에는 10 개의 테스트 케이스가 주어진다. 각 케이스는 네 줄로 이루어진다. 첫 줄에는 N 이 주어지고, 이후의 세 줄에는 차례대로 1,2,3 행의 숫자들이 주어진다. 입력 파일의 이름은 “input4.txt”이다.

**[출력]**

각 테스트 케이스에 대해서, 케이스의 번호를 "#x" 의 형식으로 출력한 후(여기서 x 는 테스트 케이스 번호), 공백을 하나 둔 다음 주어진 케이스에서 받을 수 있는 최대 점수를 기록한다. 출력 결과물을 “output4.txt”로 저장한다.

**[예제]**

**입력 (input4.txt)**

```
10                                                                  ← 1번 케이스
1 17 17 19 2 17 14 6 19 11
1 19 16 6 13 15 16 13 7 2
13 11 1 9 10 12 16 20 14 12
30                                                                  ← 2번 케이스
35 45 22 15 26 15 19 38 2 42 15 34 20 4 40 28 23 16 26 15 10 36 35 30 34 6 49 22 24 11
6 17 29 40 19 7 29 16 18 29 21 47 30 29 25 9 47 23 23 25 30 32 9 5 6 43 39 1 11 46
35 12 46 26 32 24 37 44 15 26 47 2 27 12 40 50 45 11 16 6 45 43 45 14 8 12 49 21 3 44
…
```

**출력 (output4.txt)**

```
#1 75
#2 521
…
```

## 5. 탑 쌓기

윗면에 구멍이 파인 블록 $n$개가 주어진다. 블록 $i$의 높이는 $h_i$, 구멍의 깊이는 $d_i(< h_i, h_{i+1})$이다. ($h_i \geq 1$, $d_i \geq 0$, $h_i$, $d_i$는 정수) 블록 $i+1$의 아랫면은 정확히 블록 $i$의 윗면의 구멍과 맞물리지만, 그 이외 블록의 아랫면은 블록 $i$의 구멍에 조금이라도 들어갈 수 없다. 따라서 블록 $i$ 위에 다른 블록을 올려서 이층 탑을 만들 때, 블록 $i+1$을 올리면 높이는 $h_i - d_i + h_{i+1}$이 되고 블록 $j(\neq i, i+1)$를 올리면 높이는 $h_i + h_j$가 된다. 그림 1은 연속한 세 블록의 모양의 예시를 단면도로 보인다.

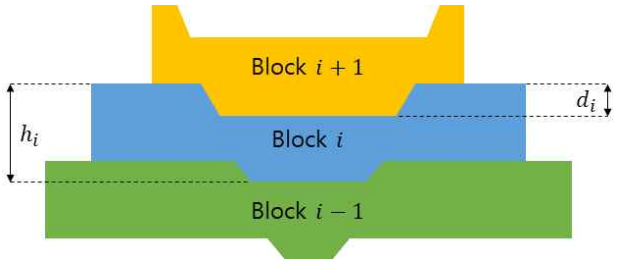



그림 1

이제 다음 조건들을 만족시키면서 블록들 중 일부를 선택하여 탑을 쌓으려 한다.

1) $i < j$면 블록 $i$는 무조건 블록 $j$보다 탑의 아래층에 있어야 한다.
2) 탑의 높이는 주어진 자연수 $H$ 이하여야 한다. ($H \geq 1$)

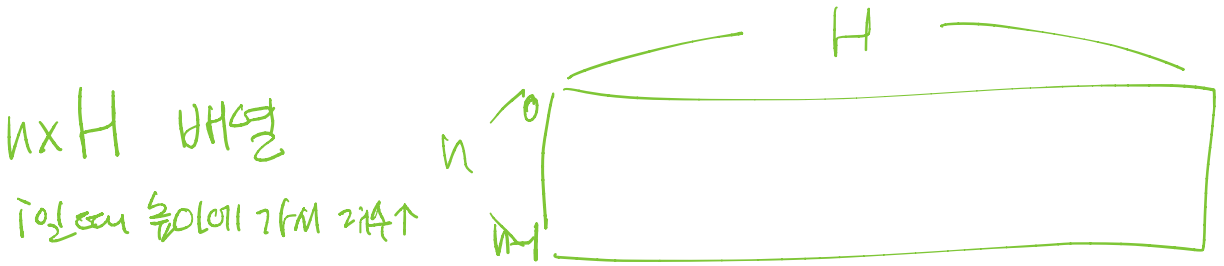
물론 탑의 각 층에는 한 개의 블록만 있고, 탑을 쌓을 때 블록을 뒤집거나 비스듬히 세울 수 없다. 만약 블록 1, 블록 2, 블록 5, 블록 6, 블록 7, 블록 9를 선택하여 탑을 만들었다면, 그 높이는 $h_1 - d_1 + h_2 + h_5 - d_5 + h_6 - d_6 + h_7 + h_9$가 된다.

주어진 $h_i$, $d_i$, $H$에 대해, 위 조건대로 탑을 쌓을 수 있는 모든 경우의 수(블록 조합의 수)를 구하는 알고리즘을 작성하라. 단, 경우의 수가 int 최대값 $2^{31}-1$를 초과할 가능성이 있으므로, 대신에 해당 값을 1,000,000으로 나눈 나머지를 구하는 것으로 한다.

**[제약사항]**

시간 제한: 10개의 입력 전부를 해결하는 시간 총 1초
메모리 제한: 128MB
블록의 개수 $n$은 [10, 1000] 범위의 정수, 높이 제한 $H$는 [100, 10000] 범위의 정수
각 블록의 높이 $h_i$는 [1, 100] 범위의 정수 ($0 \leq i \leq n-1$)
각 블록의 구멍의 깊이 $d_i$는 $[0, \min\{h_i, h_{i+1}\} - 1]$ 범위의 정수 ($0 \leq i \leq n-2$)

$$\sum_{i=1}^{n-1} i \times H = H \frac{n^2}{2} \qquad \frac{(n-1)n}{2}$$




**[입력]**

입력 파일에는 10개의 테스트 케이스가 주어진다. 각 케이스는 세 줄로 이루어진다. 첫 줄에는 블록의 개수 n과 높이 H가 주어지고, 둘째 줄에는 $h_i$ 값들이 주어진다. 셋째 줄에는 $d_i$ 값들이 주어진다. 각각의 값은 공백으로 구분된다. 입력 파일의 이름은 “input5.txt”이다.

**[출력]**

각 테스트 케이스에 대해서, 케이스의 번호를 "#x" 의 형식으로 출력한 후(여기서 x는 테스트 케이스 번호), 공백을 하나 둔 다음 주어진 케이스에서 탑을 쌓을 수 있는 모든 경우의 수를 1,000,000으로 나눈 나머지를 출력한다. 출력 결과물을 “output5.txt”로 저장한다.

**[예제]**

**입력 (input5.txt)**

```
10 100                                                    ← 1번 케이스
49 64 69 76 45 40 50 36 57 87
18 13 58 30 30 31 26 32 50
20 200                                                    ← 2번 케이스
75 87 34 5 24 16 7 91 50 13 15 66 94 93 37 63 19 80 12 62
65 17 4 3 12 1 0 26 3 0 4 4 69 3 3 1 11 9 1
…
```

**출력 (output5.txt)**

```
#1 34
#2 20825
…
```