사칙연산

문제 설명

사칙연산에서 더하기(+)는 결합법칙이 성립하지만, 빼기(-)는 결합법칙이 성립하지 않습니다.
예를 들어 식 1 - 5 - 3은 연산 순서에 따라 다음과 같이 다른 결과를 가집니다.

- ((1 - 5) - 3) = -7
- (1 - (5 - 3)) = -1

위 예시와 같이 뺄셈은 연산 순서에 따라 그 결과가 바뀔 수 있습니다.
또 다른 예로 식 1 - 3 + 5 - 8은 연산 순서에 따라 다음과 같이 5가지 결과가 나옵니다.

- (((1 - 3) + 5) - 8) = -5
- ((1 - (3 + 5)) - 8) = -15
- (1 - ((3 + 5) - 8)) = 1
- (1 - (3 + (5 - 8))) = 1
- ((1 - 3) + (5 - 8)) = -5

위와 같이 서로 다른 연산 순서의 계산 결과는 `[-15, -5, -5, 1, 1]` 이 되며, 이중 최댓값은 1입니다.
문자열 형태의 숫자와, 더하기 기호("+"), 뺄셈 기호("-")가 들어있는 배열 arr가 매개변수로 주어질 때, 서로 다른 연산순서의 계산 결과 중 최댓값을 return 하도록 solution 함수를 완성해 주세요.

제한 사항

- arr는 두 연산자 "+", "-" 와 숫자가 들어있는 배열이며, 길이는 3 이상 201 이하 입니다.
  - arr의 길이는 항상 홀수입니다.
  - arr에 들어있는 숫자의 개수는 2개 이상 101개 이하이며, 연산자의 개수는 (숫자의 개수) -1 입니다.
  - 숫자는 1 이상 1,000 이하의 자연수가 문자열 형태로 들어있습니다.. (ex : "456")
- 배열의 첫 번째 원소와 마지막 원소는 반드시 숫자이며, 숫자와 연산자가 항상 번갈아가며 들어있습니다.

입출력 예

arr	result
["1", "-", "3", "+", "5", "-", "8"]	1
["5", "-", "3", "+", "1", "+", "2", "-", "4"]	3

입출력 예시

입출력 예 #1
위의 예시와 같이 (1-(3+(5-8))) = 1 입니다.

입출력 예 #2
(5-(3+((1+2)-4))) = 3 입니다.

괄호 사칙연산 ? stack을 통해서 우선 순위 ?

작은 범위의 최대값 or 최솟값을 구해서 큰 범위의 최소 최대 값을 순차적으로 구한다.

dp[i][j] 는 i ~ j 까지의 합의 최대를 나타냄

1. 수식이 최대가 되고싶은데, 내 뒷 수식이 (-) 인 경우

   => 내 뒷 구간의 최소를 뺀다.

2. 수식이 최대가 되고싶은데, 내 뒷 수식이 (+) 인 경우

   => 내 뒷 구간의 최대를 더한다.

3. 수식이 최소가 되고 싶은데 , 내 뒷 수식이 (-) 인 경우

   => 내 뒷 구간의 최대를 뺀다.

4. 수식이 최소가 되고 싶은데, 내 뒷 수식이 (+) 인 경우

   => 내 뒷 구간의 최소를 더한다.

```
          0 1 2 3 4 5 6
String =  1 - 3 + 5 - 8
```

(2, 4)

(0, 4)

(0, 0), (2,4)

(2,2) 4,4)

```
     최대                        최대         최소
dp ( 0 ~ 6 ) 를 구하는 방법  ( 0, 0 ) - ( 2, 6 )    1  -  0 = 1

                               최대         최대
                            ( 0, 2 ) + ( 4, 6 )    -2 + -3 = -5

                               최대        최소
                            ( 0 ,4 ) - ( 6, 6 )    3  -  8 = -5

dp ( 2 ~ 6 ) 를 구하는 방법  ( 2, 2) + ( 4, 6 )   3   +   -3 = 0
                            ( 2, 4) - ( 6, 6 )   8   -  8  = 0
```

dp (4 ~ 6) 를 구하는 방법 (4, 4) - (6, 6) = -3

dp (2 ~ 4) 를 구하는 방법 (2, 2) + (4, 4) = 8

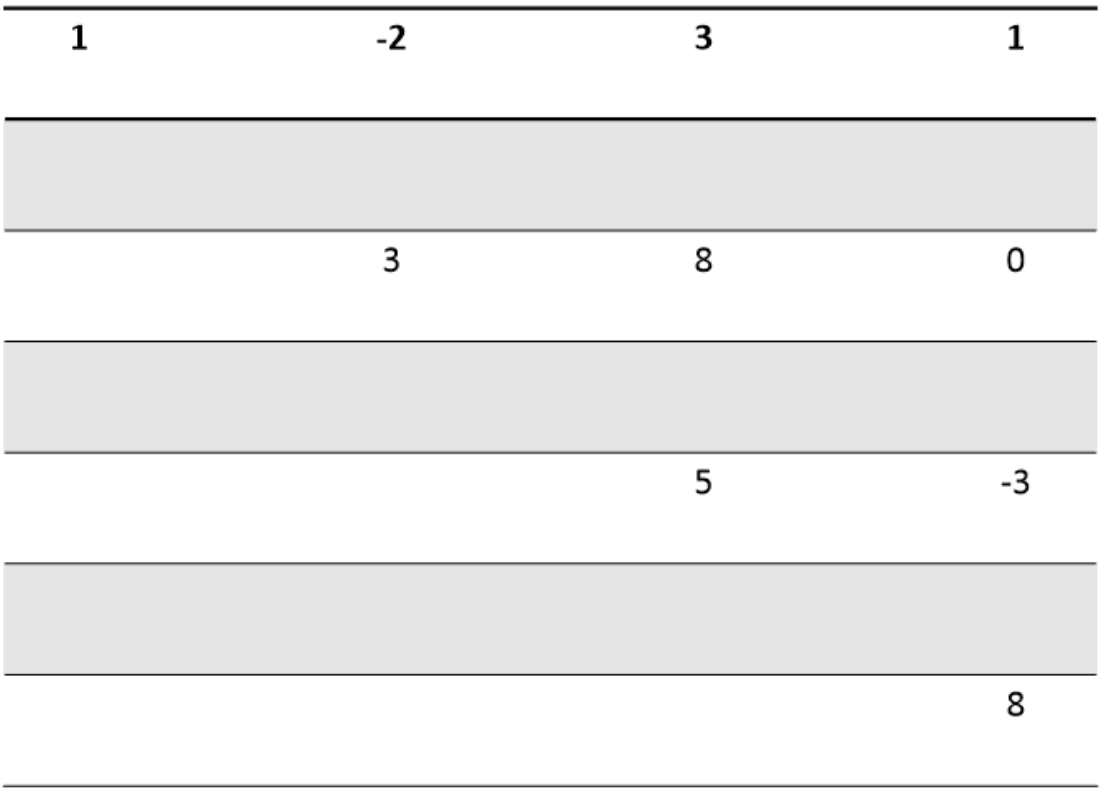
dp (0 ~ 2) 를 구하는 방법 (0, 0) - (2 , 2) = -2

dp (0 ~ 4) 를 구하는 방법 (0, 0) - (2, 4) 1 - 8 = -7

```
                        ( 0, 2 ) + ( 4, 4 ) -2  + 5  = 3
```

최대값은 dp[0][6] 을 만들 수 있는 최대값인 1

dp 최대값 최소값 0, 6



코드

```
package Algo_Study_Programmers;


public class Solution_사칙연산 {
  static int[][] dp;

  private static int calc(String[] arr, int i, int j) {


          0 1 2 3 4 5 6
String =  1 - 3 + 5 - 8
```

```
    if (dp[i][j] != Integer.MIN_VALUE) {
      return dp[i][j];
    }

    // 내 앞 기호가 (-) 이면,
    // 내 뒤 연산에서 구할 수 있는 최소값을 찾아야한다.
    if (i - 1 >= 1 && arr[i - 1].equals("-")) {
      int min = Integer.MAX_VALUE;

      for (int k = i; k < j; k += 2) {
        if (arr[k + 1].equals("-")) {
          min = Math.min(min, calc(arr, i, k) - calc(arr, k + 2, j));
        } else {
          min = Math.min(min, calc(arr, i, k) + calc(arr, k + 2, j));
        }
      }
      dp[i][j] = min;
    }
    else {
      int max = Integer.MIN_VALUE;
       -0, 0  2,6

      for(int k = i; k < j; k += 2) {
        if(arr[k + 1].equals("-")) {
          max = Math.max(max, calc(arr, i, k) - calc(arr, k + 2, j));
        }
        else {
          max = Math.max(max, calc(arr, i, k) + calc(arr, k + 2, j));
        }
      }
      dp[i][j] = max;
    }
    return dp[i][j];

  }

  public static int solution(String arr[]) {
    int N = arr.length;
    dp = new int[N][N];

    for (int i = 0; i < N; i++) {
      for (int j = 0; j < N; j++) {
        dp[i][j] = Integer.MIN_VALUE;
      }
      if (i % 2 == 0) {
        dp[i][i] = Integer.parseInt(arr[i]);
      }
    }

    return calc(arr, 0, N - 1);
  }

  public static void main(String[] args) {

    String[] arr = { "1", "-", "3", "+", "5", "-", "8" };
    solution(arr);
  }
}
```