Let's define a correct string of parentheses (round brackets):

- () is a correct string of parentheses;
- If A is a correct string of parentheses, then (A) also is a correct string of parentheses;
- If both, A and B, are correct parentheses strings, then the concatenation AB is also a correct string of brackets.

In other words, a correct parentheses string is a correct string of brackets, but containing no square brackets.

Let us take an arbitrary correct string of parentheses and replace some (perhaps all, but at least one) of the closing (right) parentheses by the opening (left) parentheses. We shall name the resulting string a broken string of parentheses.

It is easy to see that every broken string of brackets is also a broken string of parentheses, and vice versa. In fact, both of them fulfil the following two conditions:

- the number of opening brackets is greater than the number of closing brackets;
- the number of opening brackets is not less than the number of closing brackets for every prefix of the string.

Accordingly, we will name both, the broken string of brackets and the broken strings of parentheses, broken strings.

For further explanation, let s be a broken string. Let B(s) denote the set of correct strings of brackets that match the broken string of brackets s. Let P(s) denotes the set of correct strings of parentheses that match the broken string of brackets s.

Lemma

For any broken string s the sets B(s) and P(s) contain the same number of elements.

Proof

Let s be a broken string. We set a one-to-one correspondence between the elements of sets B(s) and P(s).

1. Let b be an arbitrary correct string of brackets from B(s). Substitute its every square bracket by a corresponding round bracket (opening with opening, closing with closing). As a result we obtain some correct string of parentheses that matches s.
2. Let p be an arbitrary correct string of parentheses from P(s). We mark all closing round brackets of p that correspond to closing round brackets in s (that appear at the same position in p and s). Then we mark all opening round brackets in p that correspond to the already marked closing round brackets. Finally, we substitute all unmarked round brackets in p by square brackets (opening with opening, closing with closing). As a result we obtain some correct string of brackets that matches s.

It follows from (1) and (2) that B(s) and P(s) contain the same number of elements. QED.

Thus, we have reduced the task to the following: calculate the number of distinct correct strings of parentheses that correspond to the given broken string. This task is solved using standard dynamic programming.

Let r(L,i) denote the number of distinct prefixes with length L of correct strings of brackets that match the given broken string s such that the difference between opening and closing brackets is i.

- r(0,0)=1, r(0,i)=0 for all i>0
- If s[L]=')' then r(L,i) = r(L-1,i+1) for all i=0, L>=0.
- If s[L]='(' then r(L,0) = r(L-1,1), r(L,i) = r(L-1,i-1) + r(L-1,i+1) for all i>0, L>=0

(s[L] denotes the L-th symbol of the given broken string s).

r(N,0) contains the answer, where N is the length of the given broken string s.

This directly leads to a solution with O(N^2) time and O(N^2) memory complexity. However, this is insufficient to score maximum points.

Firstly, we have to avoid using a two-dimensional array; we have to somehow obtain the same result by a manipulation of one-dimensional arrays. Note that to calculate r(L,i) for all i, we need to remember only r(L-1,j) for all j (we don't need any r(L',j) where L'<L-1). Thus, it is enough to maintain only a one-dimensional array throughout the calculations.

Secondly, straight forward implementation of the algorithm is a bit too slow. The correct solution is still O(N^2), but we should speed it up by some constant factor.

Note that if L and i have different parity then r(L,i)=0.

Note also that we are not interested in the values of r(L,i) when i>L or i>N-L. The first case does not matter because it essentially means that there have been more opening brackets than brackets in total in the prefix, which is impossible. The second case does not matter because it essentially means that there are currently more unclosed opening brackets in the prefix than the number of brackets left until reaching the length N; this is also an impossible case.

The overall time complexity is O(N^2) and memory complexity O(N). Note that the observations mentioned in the previous two paragraphs greatly decrease the number interesting states and speeds up the execution approximately 8 times.

Side note: This task was proposed for IOI 2011.