**Problem Statement**

This problem was asked by Quantcast.

You are presented with an array representing a Boolean expression. The elements are of two kinds:

* T and F, representing the values True and False.
* &, |, and ^, representing the bitwise operators for AND, OR, and XOR.

Determine the number of ways to group the array elements using parentheses so that the entire expression evaluates to True.

**Scenario**

Imagine you are working on a system that processes Boolean expressions and evaluates their possible outcomes. You need to determine how many different ways you can parenthesize an expression such that it results in True. This is particularly useful in optimization problems where different arrangements of operations can lead to the desired outcome.

**Example**

Suppose the input is ['F', '|', 'T', '&', 'T']. In this case, there are two acceptable groupings that result in True:

1. (F | T) & T → True
2. F | (T & T) → True

Thus, the function should return 2.

**Input Format**

1. The first line contains an integer n, representing the number of elements in the Boolean expression array.
2. The second line contains the Boolean expression array with n space-separated elements, which include T, F, &, |, and ^.

**Output Format**

Return an integer representing the number of ways to parenthesize the expression such that it evaluates to True.

**Example**

**Input:**

5

F | T & T

**Output:**

2

**Explanation:**

The two valid groupings that result in True are:

1. (F | T) & T
2. F | (T & T)

**Constraints**

* The expression will always have an odd number of elements.
* The expression is guaranteed to be valid.
* 1 ≤ n ≤ 19.

**Solution Approach**

To solve this problem, you can use dynamic programming to count the number of ways to parenthesize the expression such that it evaluates to True.

1. **Base Case:** When there's only one element, it's either True or False.
2. **Recursive Case:** Break the expression at every operator and calculate the number of ways the left and right sub-expressions can evaluate to True or False.

**Implementation**

python

def count\_ways\_to\_evaluate(expr, result, memo):

if len(expr) == 1:

return 1 if (expr[0] == 'T' and result == True) or (expr[0] == 'F' and result == False) else 0

if (tuple(expr), result) in memo:

return memo[(tuple(expr), result)]

ways = 0

for i in range(1, len(expr), 2):

left\_expr = expr[:i]

right\_expr = expr[i + 1:]

operator = expr[i]

for left\_result in [True, False]:

for right\_result in [True, False]:

total\_left = count\_ways\_to\_evaluate(left\_expr, left\_result, memo)

total\_right = count\_ways\_to\_evaluate(right\_expr, right\_result, memo)

if operator == '&':

if left\_result and right\_result == result:

ways += total\_left \* total\_right

elif operator == '|':

if left\_result or right\_result == result:

ways += total\_left \* total\_right

elif operator == '^':

if (left\_result != right\_result) == result:

ways += total\_left \* total\_right

memo[(tuple(expr), result)] = ways

return ways

def count\_true\_ways(expr):

memo = {}

return count\_ways\_to\_evaluate(expr, True, memo)

# Input

n = int(input())

expr = input().split()

# Output

print(count\_true\_ways(expr))

**Example Usage**

**Input:**

5

F | T & T

**Output:**

2

**Explanation:**

There are two ways to parenthesize the expression so that it evaluates to True:

1. (F | T) & T → True
2. F | (T & T) → True

**Additional Test Cases**

**Test Case 1:**

Input:

5

T ^ F | T

Output:

2

Explanation:

The valid groupings are (T ^ (F | T)) and ((T ^ F) | T).

**Test Case 2:**

Input:

7

T & F ^ T | F

Output:

3

Explanation:

The valid groupings are ((T & F) ^ (T | F)), ((T & (F ^ T)) | F), and (T & ((F ^ T) | F)).

This approach will efficiently compute the number of ways to evaluate the Boolean expression to True while handling different scenarios and test cases.