Algorithm 1: Parenthesis

```
Data: arr[], n
Result: boolMemoi[][]
Let memoi[n][n], boolMemoi[n][n]
for i = 0 to n do
    memoi[i][i] = arr[i];
    if memoi[i][i] == a then
        boolMemoi[i][i] = true;
    else
        boolMemoi[i][i] = false;
    end
end
for l = 1 to n do
    for i = 0 to n − 1 do
        j = i + l for k = i to j do
            q = multipleTable(memoi[i][k], memoi[k + 1][j]);
            if q == a then
                memoi[i][j] = q;
                boolMemoi[i][j] = true;
                break;
            else
                memoi[i][j] = q;
                boolMemoi[i][j] = false;
            end
        end
    end
end
return boolMemoi[n − 1][n − 1]
```

그림 1

Algorithm 2: multipleTable

```
Data: leftData, rightData
if (leftData == b) AND (rightData == a) then
    return a;
else
    return b;
end
```

그림 2

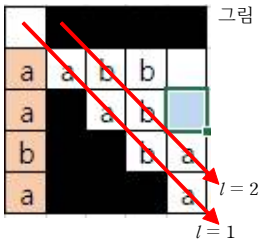

그림 3

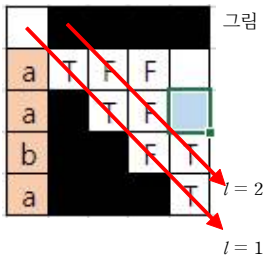

그림 4

주어진 문자열 X의 가장 앞부분부터 뒷부분까지 모든 구간을 조사한다. 연산결과를 저장하는 배열을 memoi, 연산결과에 따라 a이면 true, b이면 false를 저장하는 boolmemoi를 생성한다. 처음 memoi 배열의 초기화는 배열의 index를 기준으로 row, column의 index가 일치할 경우 본래 자리의 값을 저장하고 boolmemoi의 값은 memoi 배열 값 기준으로 a이면 true, b이면 false를 저장하여 초기화한다. 이제 [그림 3]처럼 화살표 방향으로 계산을 하여 값을 구하여 저장한다. 이때 첫 번째 화살표가 있는 곳을 $l = 1$로 시작하여 채워나간다.

memoi[i][j] = multipleTable(memoi[i][k], memoi[k+1][j])

boolmemoi[i][j] = true (if memoi[i][j] == a)
false (if memoi[i][j] == b)

예를 들어 [그림 3]에서 memoi[1][3]의 값을 구하기 위해서는 (memoi[1][1]*memoi[2][3])과 (memoi[1][2]*memoi[3][1])의 값을 구해서 값을 배열에 저장하게 된다. 이때 a라면 boolmemoi[1][3]에도 true를 저장, b라면 boolmemoi[1][3]에 false를 저장한다.

시간복잡도를 구해보면 [그림 1]에서 전체 반복문이 3번 중첩되고 가장 오래걸리는 경우를 생각해보면 n만큼 3번 반복하게 된다. 따라서 시간복잡도는 $T(n) = O(n^3)$이다. 공간복잡도의 경우 $\Theta(n^2)$이다.

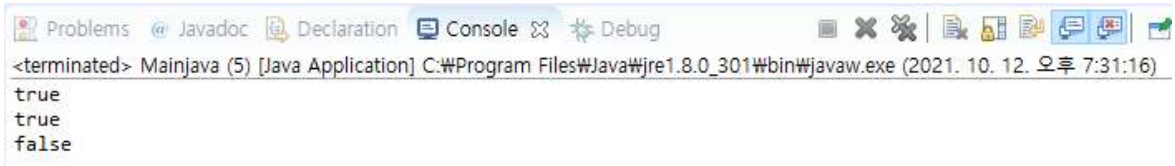

그림 5

<결과>