Pseudocódigo e explicação

Inicialmente, tive um pouco de dificuldade para encontrar uma solução para o problema. Experimentei algumas alternativas que tentavam rastrear diretamente as expressões que atingiam a profundidade desejada, mas nessa abordagem não consegui fazer funcionar.

Uma solução elegante que facilitou bastante o cálculo foi a ideia de calcular o total de expressões com profundidade menor ou igual a `depth` e subtrair o total de expressões com profundidade menor ou igual a `depth - 1`. Essa abordagem simplificou a implementação e aproveitou a memoização para evitar cálculos redundantes.

Essa ideia foi baseada no código encontrado no seguinte repositório: https://github.com/yubinbai/pcuva-problems/blob/master/UVa%2010157%20-%20Expressions/expression.py

Adaptei para não ler de um arquivo mas sim receber como input.

Como executar

Rode o script solution.py com:

`python3 solution.py`

Informe os valores de n e d, por exemplo 6 e 2 cujo o resultado é 3.

Para sair basta dar um ctrl + d

Pseudocódigo

```
Função depthLowerSum(pares, profundidade, memo):
    Se memo[pares][profundidade] já foi calculado:
        Retornar memo[pares][profundidade]

    Se pares == 0 OU profundidade == 1:
```

```
        memo[pares][profundidade] = 1
        Retornar 1

    soma = 0
    Para i de 0 até pares - 1:
        soma += depthLowerSum(i, profundidade - 1, memo) *
                depthLowerSum(pares - 1 - i, profundidade, memo)

    memo[pares][profundidade] = soma
    Retornar soma


Função principal:
    Inicializar memo como uma tabela 2D preenchida com -1
    Enquanto houver entradas do usuário:
        Ler length e depth
        Calcular pares = length // 2
        resultado = depthLowerSum(pares, depth, memo) -
                    depthLowerSum(pares, depth - 1, memo)
        Exibir resultado
```