# 

# 알고리즘 재귀(2)

## 📌 백준

### 1. 2054 괄호의 값

#### 문제

4개의 기호 '`(`', '`)`', '`[`', '`]`'를 이용해서 만들어지는 괄호열 중에서 올바른 괄호열이란 다음과 같이 정의된다.

1. 한 쌍의 괄호로만 이루어진 '`()`'와 '`[]`'는 올바른 괄호열이다.
2. 만일 `X`가 올바른 괄호열이면 '`(X)`'이나 '`[X]`'도 모두 올바른 괄호열이 된다.
3. `X`와 `Y` 모두 올바른 괄호열이라면 이들을 결합한 `XY`도 올바른 괄호열이 된다.

예를 들어 '`(()[[]])`'나 '`(())[][]`' 는 올바른 괄호열이지만 '`([)]`' 나 '`(()()[]`' 은 모두 올바른 괄호열이 아니다. 우리는 어떤 올바른 괄호열 `X`에 대하여 그 괄호열의 값(괄호값)을 아래와 같이 정의하고 값(`X`)로 표시한다.

1. '`()`' 인 괄호열의 값은 2이다.
2. '`[]`' 인 괄호열의 값은 3이다.
3. '`(X)`' 의 괄호값은 2×값(`X`) 으로 계산된다.
4. '`[X]`' 의 괄호값은 3×값(`X`) 으로 계산된다.
5. 올바른 괄호열 `X`와 `Y`가 결합된 `XY`의 괄호값은 값(`XY`)= 값(`X`)+값(`Y`) 로 계산된다.

예를 들어 '`(()[[]])([])`' 의 괄호값을 구해보자. '`()[[]]`' 의 괄호값이 2 + 3×3=11 이므로 '`(()[[]])`'의 괄호값은 2×11=22 이다. 그리고 '`([])`'의 값은 2×3=6 이므로 전체 괄호열의 값은 22 + 6 = 28 이다.

여러분이 풀어야 할 문제는 주어진 괄호열을 읽고 그 괄호값을 앞에서 정의한대로 계산하여 출력하는 것이다.

#### 입력

첫째 줄에 괄호열을 나타내는 문자열(스트링)이 주어진다. 단 그 길이는 1 이상, 30 이하이다.

## 출력

첫째 줄에 그 괄호열의 값을 나타내는 정수를 출력한다. 만일 입력이 올바르지 못한 괄호열이면 반드시 0을 출력해야 한다.

## 예시 및 출력

| 예제 입력 1 복사 | 예제 출력 1 복사 |
| --- | --- |
| (()[[]])([]) | 28 |
| 예제 입력 2 복사 | 예제 출력 2 복사 |
| [][](()) | 0 |

## 구현 아이디어

1. 완전한 괄호인지 확인한다.
2. stack에 괄호가 닫힐 때까지 넣는다.
3. 괄호가 닫히면 숫자를 스택에 넣는다.
4. 괄호가 닫히지 않고 숫자가 나오면 total이라는 변수에 더하고 변수를 빼낸다.
5. 괄호가 닫히면 숫자를 스택에 넣는다.

## 코드

```
s = input()

def is_right(x):
  length = len(x)
  count = 0
  while True:
    x = x.replace('[]','')
    x = x.replace('()','')
    count+=1
    if x=='':
      return True
    if count>=length//2+1:
      return False

stack = []
```

```
if is_right(s):
  for i in range(len(s)):
    if s[i] =='(':
      stack.append('(')
    elif s[i] == '[':
      stack.append('[')

    elif s[i] == ')':
      if stack[-1]== '(':
        stack.pop()
        stack.append(2)
      else:
        total = 0
        for j in range(len(stack)-1,-1,-1):
          if stack[j] == '(':
            total *=2
            stack.pop()
            stack.append(total)
            break
          else:
            total += stack[j]
            stack.pop()

    else :
      if stack[-1]== '[':
        stack[-1] = 3
      else:
        total = 0
        for j in range(len(stack)-1,-1,-1):
          if stack[j] == '[':
            total *=3
            stack.pop()
            stack.append(total)
            break
          else:
            total += stack[j]
            stack.pop()
  print(sum(stack))
else:
  print(0)
```

## 2.별찍기-10

### 문제

재귀적인 패턴으로 별을 찍어 보자. N이 3의 거듭제곱(3, 9, 27, ...)이라고 할 때, 크기 N의 패턴은 N×N 정사각형 모양이다.

크기 3의 패턴은 가운데에 공백이 있고, 가운데를 제외한 모든 칸에 별이 하나씩 있는 패턴이다.

```
***
* *
***
```

N이 3보다 클 경우, 크기 N의 패턴은 공백으로 채워진 가운데의 (N/3)×(N/3) 정사각형을 크기 N/3의 패턴으로 둘러싼 형태이다. 예를 들어 크기 27의 패턴은 예제 출력 1과 같다.

## 입력

첫째 줄에 N이 주어진다. N은 3의 거듭제곱이다. 즉 어떤 정수 k에 대해 N=3k이며, 이때 1 ≤ k < 8이다.

## 출력

첫째 줄부터 N번째 줄까지 별을 출력한다.

### 예시 및 출력

예제 입력 1 복사

```
27
```

예제 출력 1 복사

```
***************************
* ** ** ** ** ** ** ** ** *
***************************
***   ******   ******   ***
* *   * ** *   * ** *   * *
***   ******   ******   ***
***************************
* ** ** ** ** ** ** ** ** *
***************************
*********         *********
* ** ** *         * ** ** *
*********         *********
***   ***         ***   ***
* *   * *         * *   * *
***   ***         ***   ***
*********         *********
* ** ** *         * ** ** *
*********         *********
***************************
* ** ** ** ** ** ** ** ** *
***************************
***   ******   ******   ***
* *   * ** *   * ** *   * *
***   ******   ******   ***
***************************
* ** ** ** ** ** ** ** ** *
***************************
```

### 구현 아이디어

1. 처음에 모든 칸이 가득 찬 별 사각형을 만든다.
2. 가장 큰 사각형에서 시작한다.
3. 가운데 1/3 만큼을 비운다.
4. 2와 3을 9등분하여 재귀함수로 반복한다.

### 코드

```
n = int(input())

matrix =[['*']*n for _ in range(n)]

def show_matrix(matrix):
  for i in matrix:
    print(''.join(i))

def hole(n, x, y):
  if n == 1:
    return
  else:
    hole(n//3, x, y)
    hole(n//3, x+n//3, y)
    hole(n//3, x+2*n//3, y)
    hole(n//3, x, y+n//3)
    for i in range(n//3):
      for j in range(n//3):
        matrix[x+n//3+i][y+n//3+j]= ' '
    hole(n//3, x+2*n//3, y+n//3)
    hole(n//3, x, y+2*n//3)
    hole(n//3, x+n//3, y+2*n//3)
    hole(n//3, x+2*n//3, y+2*n//3)

hole(n,0,0)
show_matrix(matrix)
```

## 3.종이의 개수

### 문제

N×N크기의 행렬로 표현되는 종이가 있다. 종이의 각 칸에는 -1, 0, 1 중 하나가 저장되어 있다. 우리는 이 행렬을 다음과 같은 규칙에 따라 적절한 크기로 자르려고 한다.

1. 만약 종이가 모두 같은 수로 되어 있다면 이 종이를 그대로 사용한다.
2. (1)이 아닌 경우에는 종이를 같은 크기의 종이 9개로 자르고, 각각의 잘린 종이에 대해서 (1)의 과정을 반복한다.

이와 같이 종이를 잘랐을 때, -1로만 채워진 종이의 개수, 0으로만 채워진 종이의 개수, 1로만 채워진 종이의 개수를 구해내는 프로그램을 작성하시오.

## 입력

첫째 줄에 N(1 ≤ N ≤ 37, N은 3k 꼴)이 주어진다. 다음 N개의 줄에는 N개의 정수로 행렬이 주어진다.

## 출력

첫째 줄에 -1로만 채워진 종이의 개수를, 둘째 줄에 0으로만 채워진 종이의 개수를, 셋째 줄에 1로만 채워진 종이의 개수를 출력한다.

예제 입력 1 복사

```
9
0 0 0 1 1 1 -1 -1 -1
0 0 0 1 1 1 -1 -1 -1
0 0 0 1 1 1 -1 -1 -1
1 1 1 0 0 0 0 0 0
1 1 1 0 0 0 0 0 0
1 1 1 0 0 0 0 0 0
0 1 -1 0 1 -1 0 1 -1
0 -1 1 0 1 -1 0 1 -1
0 1 -1 1 0 -1 0 1 -1
```

예제 출력 1 복사

```
10
12
11
```

## 구현 아이디어

1. 가장 큰 사각형에서 시작한다.
2. 부분행렬 내의 모든원소가 같은지를 확인하는 함수를 만든다.
3. 같다면 하나의 원소를 집어 그원소로 이루어진 사각형의 개수를 하나 더한다.(Dict)
4. 2와 3을 9등분하여 재귀함수로 반복한다.

## 코드

```
from collections import defaultdict
n = int(input())
matrix = [list(map(int,input().split())) for _ in range(n)]

Dict = defaultdict(int)


def show_matrix(mat):
  for i in matrix:
    for j in i :
      print(j, end=" ")
    print()
```

```
def is_same(mat):
  prev = mat[0][0]
  for i in range(len(mat)):
    for j in range(len(mat)):
      if prev != mat[i][j]:
        return False
      else:
        prev = mat[i][j]
  return True

def recursive(mat):
  size = len(mat)
  if is_same(mat):
    Dict[mat[0][0]]+=1
    return

  recursive([row[0:size//3] for row in mat[0:size//3]])
  recursive([row[size//3:(size//3)*2] for row in mat[0:size//3]])
  recursive([row[(size//3)*2:(size//3)*3] for row in mat[0:size//3]])

  recursive([row[0:size//3] for row in mat[(size//3):(size//3)*2]])
  recursive([row[(size//3):(size//3)*2] for row in mat[(size//3):(size//3)*2]])
  recursive([row[(size//3)*2:(size//3)*3] for row in mat[(size//3):(size//3)*2]])

  recursive([row[0:size//3] for row in mat[(size//3)*2:(size//3)*3]])
  recursive([row[(size//3):(size//3)*2] for row in mat[(size//3)*2:(size//3)*3]])
  recursive([row[(size//3)*2:(size//3)*3] for row in mat[(size//3)*2:(size//3)*3]])

recursive(matrix)
print(Dict[-1])
print(Dict[0])
print(Dict[1])
```