(B) Extend to n = 4 gray values

```
import argparse
import numpy as np
import cv2

# parse the command line arguments
parser = argparse.ArgumentParser()
parser.add_argument('-i', '--image', type=str,
                    default='img/default.jpg', help='path to the image')
parser.add_argument('-o', '--output', type=str,
                    default='', help='name of the output image')
parser.add_argument('-t', '--type', type=str, default='',
                    help='type of the output image')
args = parser.parse_args()

if(args.output == ''): args.output = args.image.split('/')[-1].split('.')[0]
if(args.type == ''): args.type = args.image.split('.')[-1]

# Load grayscale image and save original
img = cv2.imread( args.image, cv2.IMREAD_GRAYSCALE)
cv2.imwrite(args.output + '_original.' + args.type, img)

# Compute quantized image
Q = np.floor_divide(img, 85)

# Define dithering matrix
D1 = np.array([[0, 56], [84, 28]])

# Repeat dithering matrix to match image size
D = np.tile(D1, (img.shape[0]//2+1, img.shape[1]//2+1))[:img.shape[0],
:img.shape[1]]

# Threshold image
I_thresholded = Q.copy().astype('uint8')
I_thresholded += (img - Q*85 > D).astype('uint8')
I_thresholded = (I_thresholded * 85).astype('uint8')

# Scale values to [0, 255]
I_scaled = (I_thresholded ).astype('uint8')

# Save extended image
cv2.imwrite(args.output + '_extended.' + args.type, I_scaled)
```