A brute force strategy is sufficient here. Let n be the length of the binary string s.

Suppose the value of c has been selected; then, we do the following.

- Do the "text wrapping" to convert the binary string into a grid. This takes $\mathcal{O}(n)$ time, since the characters in the grid are just the same n characters from s, just rearranged.
- Check each of the 16×16 subgrids and count how many are Manlilinlang patterns. This also only takes $\mathcal{O}(n)$ time because there are $\leq n$ subgrids to check (consider all possible top-left corners), and checking if a subgrid matches a Manlilinlang pattern only takes a fixed $\mathcal{O}(16) = \mathcal{O}(1)$ amount of time.

If c is fixed, then the answer can be computed in $\mathcal{O}(n)$. If we try this for all values of c from 1 to n and just keep track of the best result, then the total running time is $\mathcal{O}(n^2)$. But $n \leq 750$, so this very comfortably passes.

Note that we can save time by only trying c from 4 to $\approx n/4$ (because otherwise there wouldn't be enough columns or rows), but this does not change the asymptotic running time of $\mathcal{O}(n^2)$, and is not necessary in order to pass within the time limit.

```python
n = int(input())
s = input()

amogus1 = [
    '1000',
    '0011',
    '0000',
    '1010'
]
amogus2 = [
    '0111',
    '1100',
    '1111',
    '0101'
]

best = 0
for c in range(1, n+1):
    rows = [s[i:i+c] for i in range(0, n, c)]
    rows[-1] = rows[-1].ljust(c, ' ')  # pad the last row with empty space
    r = len(rows)

    total = 0
    for i in range(r - 3):
        for j in range(c - 3):
            if any(
                all(rows[i+k][j:j+4] == amogus[k] for k in range(4))
                for amogus in [amogus1, amogus2]
            ):
                total += 1
    best = max(best, total)

print(best)

```