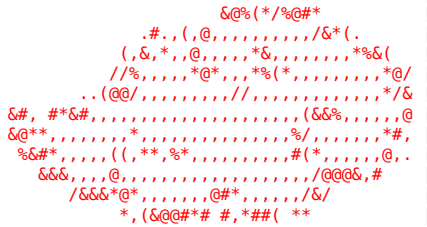
COMPRESSION AND INFORMATION



The original Akatsuki cloud represented justice and fairness. Later however it began to represent the rain of blood that fell in Amegakure during its wars.

This game is a game of information.

How can we represent information as compact as possible.

What is the smallest amount of symbols we need in order to still think of Akatsiku when we see the image?

GAME RULES

This game is more of a puzzle, you can play it alone, or with friends, but try to start with the easy and obvious cards.

Remember that it is actually very difficult to read the encoded images, so do not get discouraged, and just try it, one card at a time.

> 1. WATCH NARUTO IF YOU HAVENT

> 2. PICK AN IMAGE CARD

> 3. FIND THE ENCODING CARD
> MATCHING THE OUTPUT IMAGE

> 4. IF YOU ARE HAVING FUN GOTO 2

> 5. WATCH:

- Hunter x Hunter
- One Piece
- Bleach
- One Punch Man
- Dragonball Z
- Hajime no Ippo
- My Hero Academia
- Sword Art Online

WHAT IS AN IMAGE

Images are just an array of pixels, a pixel is just a dot of color. Our eyes can see any color as a combination of red, green and blue.
Each pixel has 3 one byte values:

- red: from 0 to 255
- green: from 0 to 255
- blue: from 0 to 255

When we blend those primitive color values we can create 16777216 colors, (0,0,0) is black, (255,255,255) is white
So the image is list of pixels, and each pixel has a value for each of the 3 colors. For example, we could have one red and one magenta pixel next to each other:

```
[...,(255,0,0),(255,0,255),...]
```

In this game we will use text symbols to represent pixels, but the idea is the same. A pixel is just a piece of display information. Each card has 40 columns and 31 rows, so it has 1240 pixels.
This tree has 42 pixels:

```
 .......
 ...*...   6 rows (height)
 ..***..   7 columns (width)
 .*****.   6*7 = 42 pixels
 ...|...   your monitor has at least
 .......   2 million pixels (1080*1920)
```


ENCODE AND DECODE

First we will convert our text image into something easier to use, like a list of numbers. We will convert each symbol to a number by creating a table of symbols and we can also use it to convert back. For example, having this 6x7 image:

```
 .......
 ...*...  "encoding" is the process
 ..***..  of converting information
 .*****.  from one form to another.
 ...|...  "decoding" is converting
 .......  it back.
```

The first unique symbol we see is `.`, we will give it the number 0, next is `*`, so it will be number 1, and then the last symbol we see `|`, which will be number 2.

Our symbol table will look like this:

```
{'.': 0, '*': 1, '|': 2}
```

Our encoding process works like this:

```
 ....... every '.' becomes 0 ->  0000000
 ...*... every '*' becomes 1 ->  0001000
 ..***..                     ->  0011100
 .*****.                     ->  0111110
 ...|... every '|' becomes 2 ->  0002000
 .......                     ->  0000000
```


ENCODE AND DECODE

```
def encode(x, sym={}):
  r = []

  for v in x:
    if v not in sym:
      # first time we see a symbol
      # we put it in the dictionary
      sym[v] = len(sym)

    # append the number of the symbol
    r.append(sym[v])

  # return both the list and the table
  return [r, sym]

def decode(x, sym):
  # invert keys and values from
  #   { ".": 1, "@": 2}
  # to
  #   {1: ".", 2: "@"}
  rs = {sym[k] : k for k in sym}
  r = ''

  for v in x:
    # lookup symbol from number
    r += rs[v]

  return r
```


RUNLENGTH ENCODING

If we know the height and width of an image, we can just flatten it into a giant list of numbers:

```
 0000000
 0001000 Later when we decode the list
 0011100 to draw it on screen, we can
 0111110 draw new row every 'width'
 0002000 pixels.
 0000000
```

becomes:

```
[
  0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,1,1,1,
  0,0,0,1,1,1,1,1,0,0,0,0,2,0,0,0,0,0,0,
  0,0,0,0
]
```

There is a lot of repetition in this list, so we can just write how many times each number appears in the sequence:

```
[
  10,0,1,1,5,0,3,1,3,0,5,1,4,0,1,2,10,0
]
```

meaning, 10 zeroes, 1 time one, and again 5 zeroes, 3 ones etc..

This is called RunLength Encoding. To decode simply do the reverse operation.

RUNLENGTH ENCODING

```
# run length encode a list of numbers
# from:
#   [1,1,1,1,1,1,1,2]
# to:
#   [7,1,1,2]
def rle(x):
  r = []
  for v in x:
    if len(r) == 0 or r[-1] != v:
      r.append(0)
      r.append(v)
    if v == r[-1]:
      r[-2] += 1
  return r

# run length decode
# from:
#   [7,1,1,2]
# to:
#   [1,1,1,1,1,1,1,2]
def rld(x):
  r = []

  for i in range(0, len(x), 2):
    for k in range(x[i]):
      r.append(x[i+1])

  return r
```


REDUCE INFORMATION

Going back to our tree:

```
 .......
 ...*...    we will inspect
 ..***.. <- this row to observe
 .*****.    how it changes with
 ...|...    compression
 .......
```

How would it look if we try to squeeze it in fewer pixels? For example take every 2 pixels and average them together and round down and then explode it back:

```
original   squeeze    exploded
0000000    0 0 0 0    0000000 (0+0)/2=0
0001000    0 0 0 0    0000000 (1+1)/2=1
0011100 -> 0 1 0 0 -> 0011000 (1+0)/2=0
0111110    0 1 1 0    0011110
0002000    0 1 0 0    0011000 (0+2)/2=1
0000000    0 0 0 0    0000000
```

Then if we draw it again:

```
 .......
 ....... its ugly, but
 ..**... it kind of looks
 ..****. like a tree...
 ..**... kind of..
 .......
```

This approach of grouping a block of pixels into some compressed value, is a common way to compress with losing data and it is called 'lossy compression'.

-> 09 <

REDUCE INFORMATION

```
# average every n elements
# from:
#  [1,2,4,4,9,5] with n=2
# to:
#  [1,4,7]
def squeeze(x, n):
  r = []

  for i in range(0, len(x), n):
    avg = sum(x[i:i+n])/n
    r.append(int(avg))

  return r


# explode the elements
# from:
#   [1,4,7] with n=2
# to:
#   [1,1,4,4,7,7]
def unsqueeze(x, n):
  r = []

  for v in x:
    for i in range(n):
      r.append(v)

  return r
```

-> 10 <

FILTERS

You can do all kinds of manpulations of the image data.

An example is Black And White filter:

```
  for every pixel
     if the pixel is not zero
       set the pixel to WHITE
     else
       set the pixel to BLACK
```

Simple Blur filter:

```
  for every block of 8x8 pixels
    replace them with their
    average
```

Invert filter:

```
  for each pixel:
    set it to the opposite
    e.g. ORANGE <-> BLUE
         RED <-> GREEN
         WHITE <-> BLACK
```

In our example we use simplified versions of those filters, but the fundamental idea is the same.

Take the pixels and manipulate them.

-> 11 <

FILTERS

```
def blur(x):
  # fake blur, averaging every 3 values
  s = squeeze(x, 3)
  return unsqueeze(s,3)

def invert(x):
  # invert the values
  # [1,1,3,0,0] -> [2,2,0,3,3]
  r = []

  m = max(x)
  for v in x:
    r.append(m - v)

  return r

def bw(x):
  # make everything "black and white"
  # [2,7,0,0] -> [1,1,0,0]
  r = []

  for v in x:
    if v == 0:
      r.append(0)
    else:
      r.append(1)

  return r
```

-> 12 <

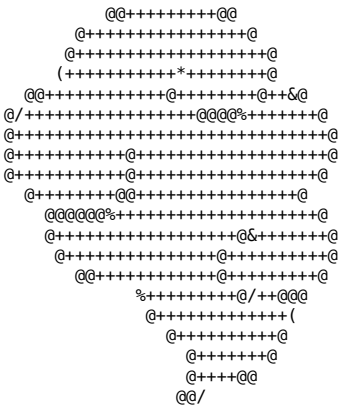
SYMBOL TABLE

```
{
    " ": 0,
    "@": 1,
    "+": 2,
    "(": 3,
    "*": 4,
    "&": 5,
    "/": 6,
    "%": 7
}
```

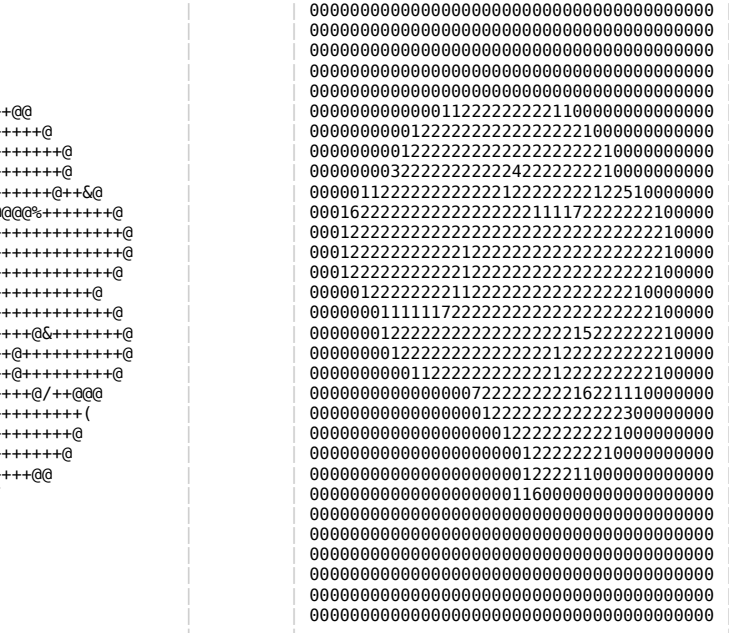
A symbol table is a table used to encode and decode from one symbol to another. In our case it is from a character to a number.

Use this card to decode the encoded cards.

-> 13 <



-> 14 <



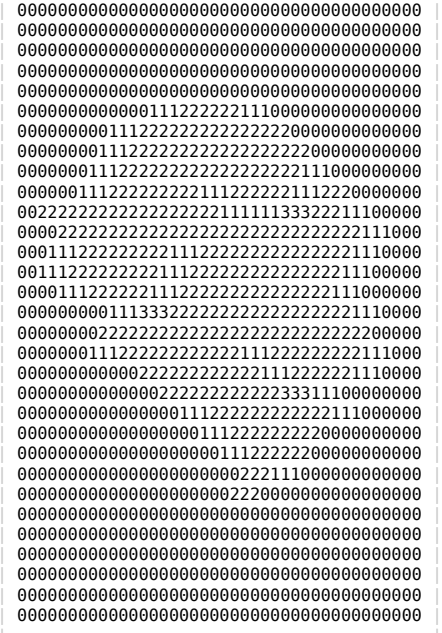
-> 15 <

rle(encoded)

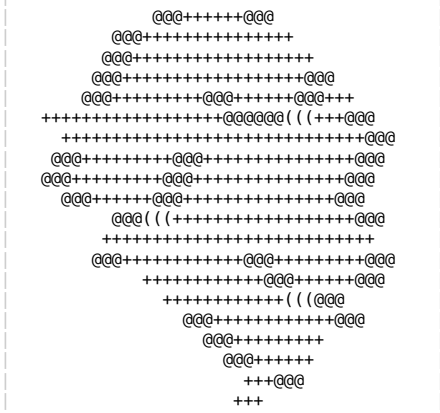
size: 216

213	0	2	1	9	2	2	1	24	0
1	1	16	2	1	1	21	0	1	1
19	2	1	1	18	0	1	3	11	2
1	4	8	2	1	1	15	0	2	1
12	2	1	1	8	2	1	1	2	2
1	5	1	1	10	0	1	1	1	6
17	2	4	1	1	7	7	2	1	1
8	0	1	1	31	2	1	1	7	0
1	1	11	2	1	1	19	2	1	1
7	0	1	1	11	2	1	1	18	2
1	1	10	0	1	1	8	2	2	1
16	2	1	1	14	0	6	1	1	7
20	2	1	1	12	0	1	1	18	2
1	1	1	5	7	2	1	1	12	0
1	1	15	2	1	1	10	2	1	1
14	0	2	1	12	2	1	1	9	2
1	1	21	0	1	7	9	2	1	1
1	6	2	2	3	1	24	0	1	1
13	2	1	3	27	0	1	1	10	2
1	1	30	0	1	1	7	2	1	1
31	0	1	1	4	2	2	1	32	0
2	1	1	6	257	0				

-> 16 <



-> 17 <



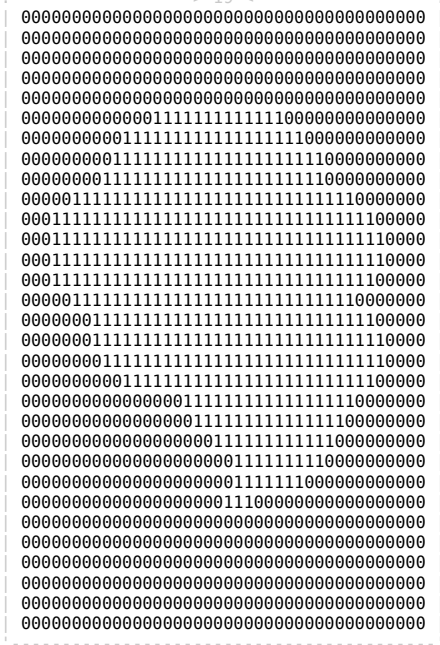
-> 18 <

rle(blur(encoded))

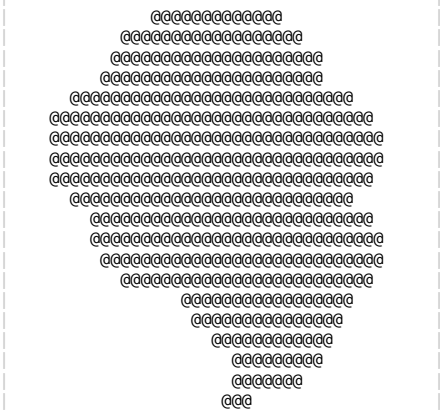
size: 172

213	0	3	1	6	2	3	1	24	0
3	1	15	2	21	0	3	1	18	2
18	0	3	1	18	2	3	1	15	0
3	1	9	2	3	1	6	2	3	1
3	2	9	0	18	2	6	1	3	3
3	2	3	1	9	0	30	2	3	1
6	0	3	1	9	2	3	1	15	2
3	1	6	0	3	1	9	2	3	1
15	2	3	1	9	0	3	1	6	2
3	1	15	2	3	1	15	0	3	1
3	3	18	2	3	1	12	0	27	2
12	0	3	1	12	2	3	1	9	2
3	1	15	0	12	2	3	1	6	2
3	1	18	0	12	2	3	3	3	1
24	0	3	1	12	2	3	1	24	0
3	1	9	2	30	0	3	1	6	2
33	0	3	2	3	1	33	0	3	2
258	0								

-> 19 <



-> 20 <



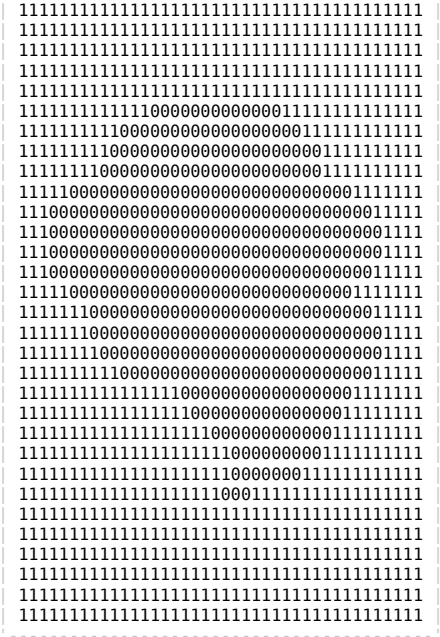
-> 21 <

rle(bw(encoded))

size: 82

213	0	13	1	24	0	18	1	21	0
21	1	18	0	22	1	15	0	28	1
10	0	32	1	8	0	33	1	7	0
33	1	7	0	32	1	10	0	28	1
14	0	28	1	12	0	29	1	12	0
28	1	14	0	25	1	21	0	17	1
24	0	15	1	27	0	12	1	30	0
9	1	31	0	7	1	32	0	3	1
257	0								

-> 22 <



-> 23 <

rle(invert(bw(encoded)))

size: 82

213	1	13	0	24	1	18	0	21	1
21	0	18	1	22	0	15	1	28	0
10	1	32	0	8	1	33	0	7	1
33	0	7	1	32	0	10	1	28	0
14	1	28	0	12	1	29	0	12	1
28	0	14	1	25	0	21	1	17	0
24	1	15	0	27	1	12	0	30	1
9	0	31	1	7	0	32	1	3	0
257	1								

-> 24 <

squeeze(encoded,10)

size: 124

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	1	1	0	0	1	1	0	0	2
1	0	0	2	2	0	0	1	1	0
1	2	2	0	1	2	2	1	1	1
2	1	1	1	2	0	0	1	2	0
0	2	2	0	0	2	2	1	0	2
1	1	0	1	1	0	0	1	2	0
0	0	2	0	0	0	2	0	0	0
1	0	0	0	1	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0						

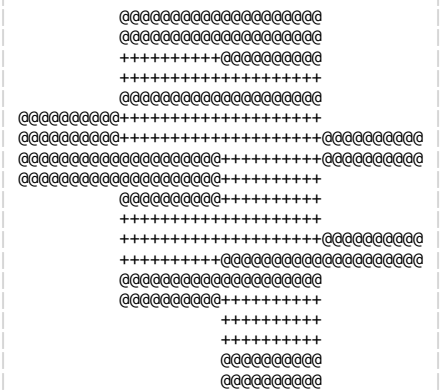
--> 25 <--

rle(squeeze(encoded,10))

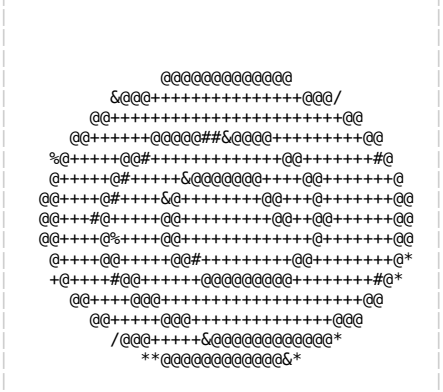
size: 92

```
 21   0   2   1   2   0   2   1   2   0
  1   2   1   1   2   0   2   2   2   0
  2   1   1   0   1   1   2   2   1   0
  1   1   2   2   3   1   1   2   3   1
  1   2   2   0   1   1   1   2   2   0
  2   2   2   0   2   2   1   1   1   0
  1   2   2   1   1   0   2   1   2   0
  1   1   1   2   3   0   1   2   3   0
  1   2   3   0   1   1   3   0   1   1
 29   0
```

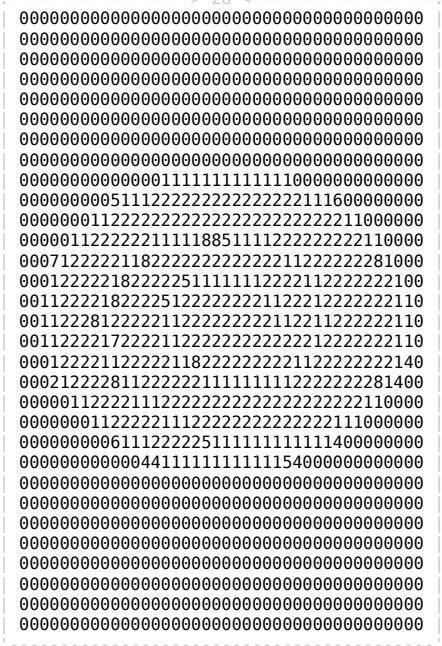
--> 26 <--



--> 27 <--



--> 28 <--



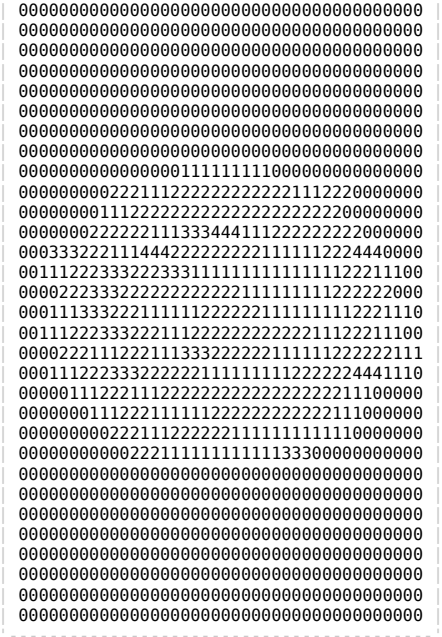
--> 29 <--

rle(encoded)

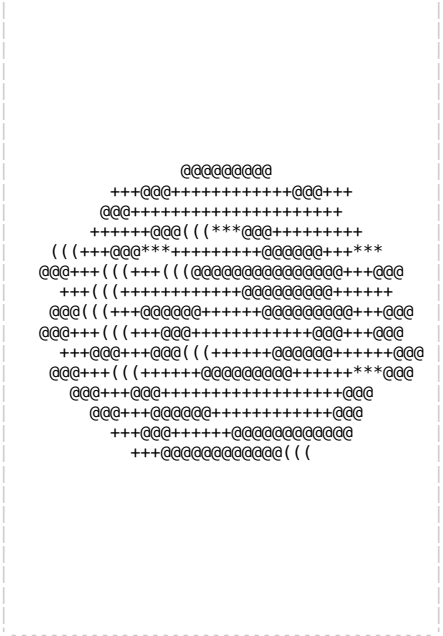
size: 262

```
334   0  13   1  22   0   1   5   3   1
 15   2   3   1   1   6  15   0   2   1
 23   2   2   1  11   0   2   1   6   2
  5   1   2   8   1   5   4   1   9   2
  2   1   7   0   1   7   1   1   5   2
  2   1   1   8  13   2   2   1   7   2
  1   8   1   1   6   0   1   1   5   2
  1   1   1   8   5   2   1   5   7   1
  4   2   2   1   7   2   1   1   4   0
  2   1   4   2   1   1   1   8   4   2
  1   5   1   1   8   2   2   1   3   2
  1   1   7   2   2   1   3   0   2   1
  3   2   1   8   1   1   5   2   2   1
  9   2   2   1   2   2   2   1   6   2
  2   1   3   0   2   1   4   2   1   1
  1   7   4   2   2   1  13   2   1   1
  7   2   2   1   4   0   1   1   4   2
  2   1   5   2   2   1   1   8   9   2
  2   1   8   2   1   1   1   4   4   0
  1   2   1   1   4   2   1   8   2   1
  6   2   9   1   8   2   1   8   1   1
  1   4   7   0   2   1   4   2   3   1
 20   2   2   1  11   0   2   1   5   2
  3   1  14   2   3   1  15   0   1   6
  3   1   5   2   1   5  12   1   1   4
 20   0   2   4  12   1   1   5   1   4
332   0
```

--> 30 <--



--> 31 <--



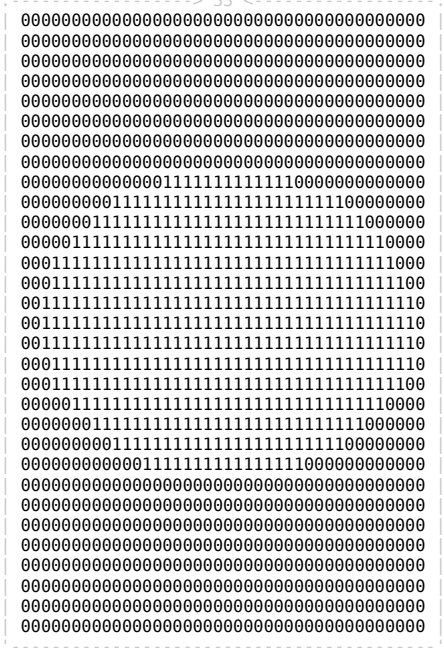
--> 32 <--

rle(blur(encoded))

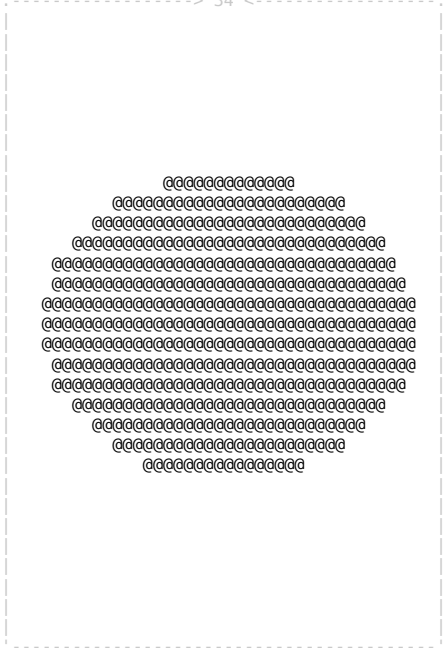
size: 204

```
336   0   9   1  24   0   3   2   3   1
 12   2   3   1   3   2  15   0   3   1
 21   2  15   0   6   2   3   1   3   3
  3   4   3   1   9   2   9   0   3   3
  3   2   3   1   3   4   9   2   6   1
  3   2   3   4   6   0   3   1   3   2
  3   3   3   2   3   3  15   1   3   2
  3   1   6   0   3   2   3   3  12   2
  9   1   6   2   6   0   3   1   3   3
  3   2   6   1   6   2   9   1   3   2
  3   1   3   0   3   1   3   2   3   3
  3   2   3   1  12   2   3   1   3   2
  3   1   6   0   3   2   3   1   3   2
  3   1   3   3   6   2   6   1   6   2
  3   1   3   0   3   1   3   2   3   3
  6   2   9   1   6   2   3   4   3   1
  6   0   3   1   3   2   3   1  18   2
  3   1  12   0   3   1   3   2   6   1
 12   2   3   1  15   0   3   2   3   1
  6   2  12   1  18   0   3   2  12   1
  3   3 333   0
```

-> 33 <



-> 34 <



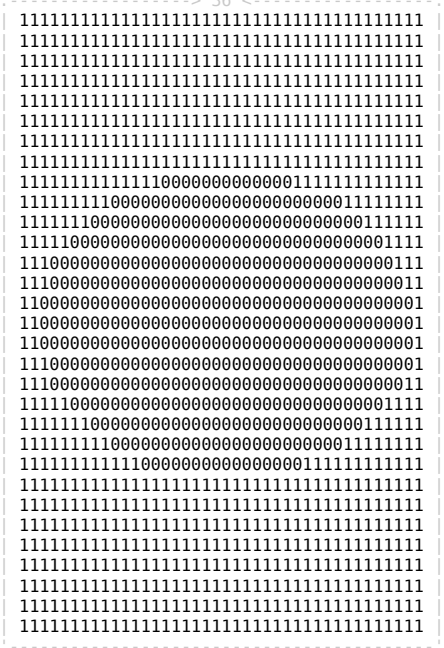
-> 35 <

rle(bw(encoded))

size: 62

334	0	13	1	22	0	23	1	15	0
27	1	11	0	31	1	7	0	34	1
6	0	35	1	4	0	37	1	3	0
37	1	3	0	37	1	4	0	36	1
4	0	35	1	7	0	31	1	11	0
27	1	15	0	23	1	20	0	16	1
332	0								

-> 36 <



-> 37 <

rle(invert(bw(encoded)))

size: 62

334	1	13	0	22	1	23	0	15	1
27	0	11	1	31	0	7	1	34	0
6	1	35	0	4	1	37	0	3	1
37	0	3	1	37	0	4	1	36	0
4	1	35	0	7	1	31	0	11	1
27	0	15	1	23	0	20	1	16	0
332	1								

-> 38 <

squeeze(encoded,10)

size: 124

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	1	0
0	2	2	0	0	2	1	1	1	2
1	1	1	2	1	1	1	2	1	1
1	1	1	1	1	1	1	1	1	2
1	1	1	1	1	2	0	1	2	1
0	1	2	0	0	1	1	0	0	1
1	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0						

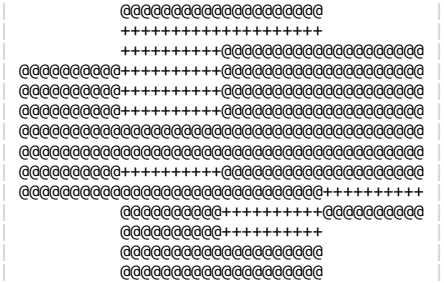
-> 39 <

rle(squeeze(encoded,10))

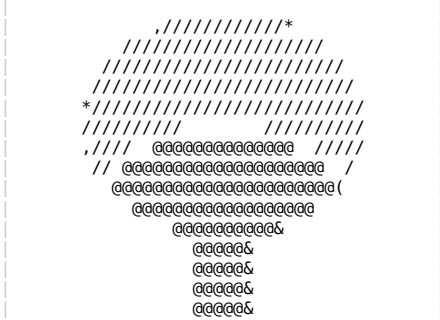
size: 56

37	0	2	1	2	0	2	2	2	0
1	2	3	1	1	2	3	1	1	2
3	1	1	2	11	1	1	2	5	1
1	2	1	0	1	1	1	2	1	1
1	0	1	1	1	2	2	0	2	1
2	0	2	1	33	0				

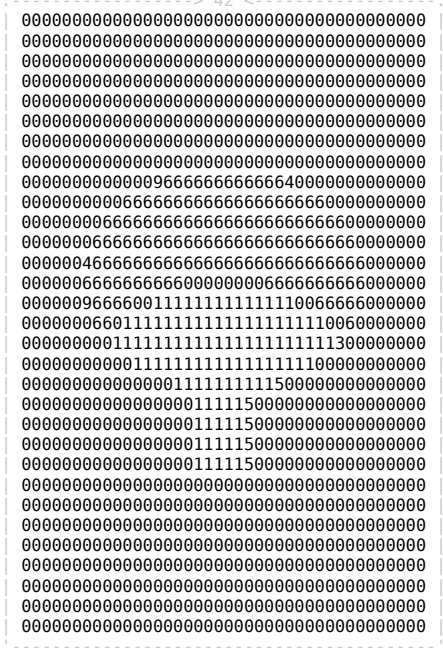
-> 40 <



> 41 <



> 42 <



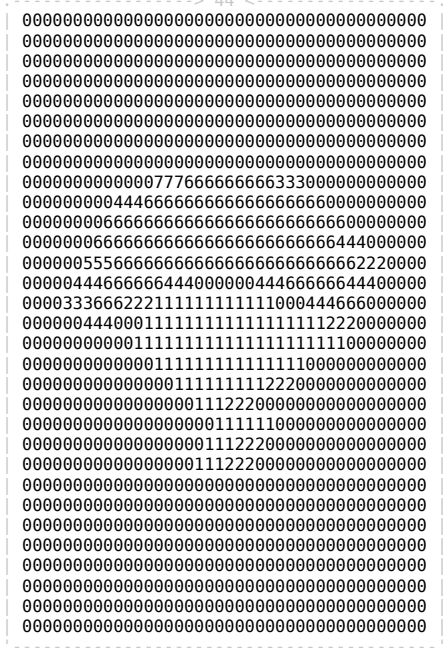
> 43 <

rle(encoded)

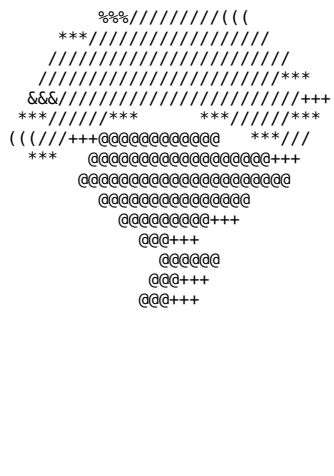
size: 102

333	0	1	9	12	6	1	4	23	0
20	6	18	0	24	6	15	0	26	6
13	0	1	4	27	6	12	0	10	6
8	0	10	6	12	0	1	9	4	6
2	0	14	1	2	0	5	6	13	0
2	6	1	0	20	1	2	0	1	6
16	0	22	1	1	3	19	0	18	1
26	0	10	1	1	5	31	0	5	1
1	5	34	0	5	1	1	5	34	0
5	1	1	5	34	0	5	1	1	5
337	0								

> 44 <



> 45 <



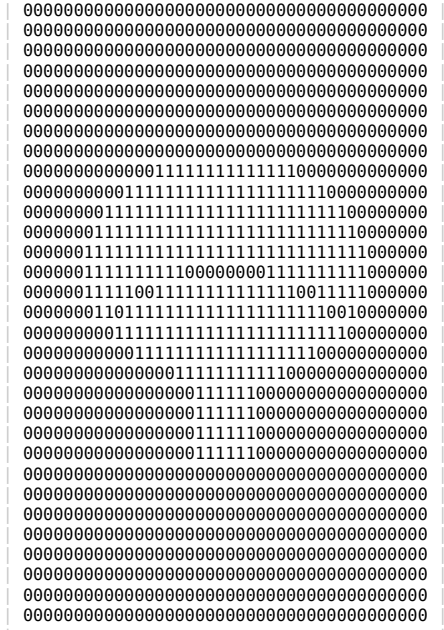
> 46 <

rle(blur(encoded))

size: 112

333	0	3	7	9	6	3	3	21	0
3	4	18	6	18	0	24	6	15	0
24	6	3	4	12	0	3	5	24	6
3	2	9	0	3	4	6	6	3	4
6	0	3	4	6	6	3	4	9	0
3	3	3	6	3	2	12	1	3	0
3	4	3	6	12	0	3	4	3	0
18	1	3	2	18	0	21	1	21	0
15	1	27	0	9	1	3	2	30	0
3	1	3	2	36	0	6	1	33	0
3	1	3	2	33	0	3	1	3	2
339	0								

> 47 <



> 48 <

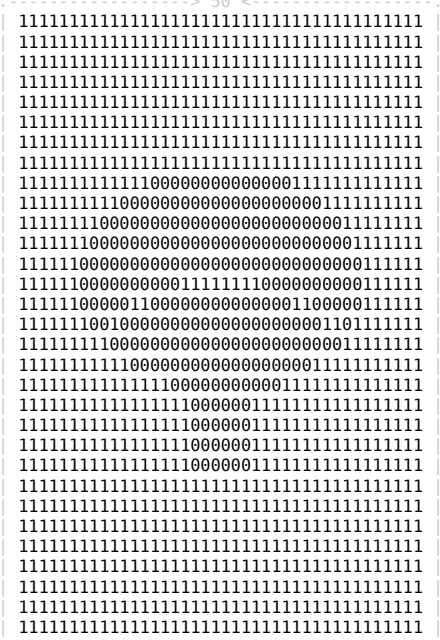
-> 49 <

rle(bw(encoded))

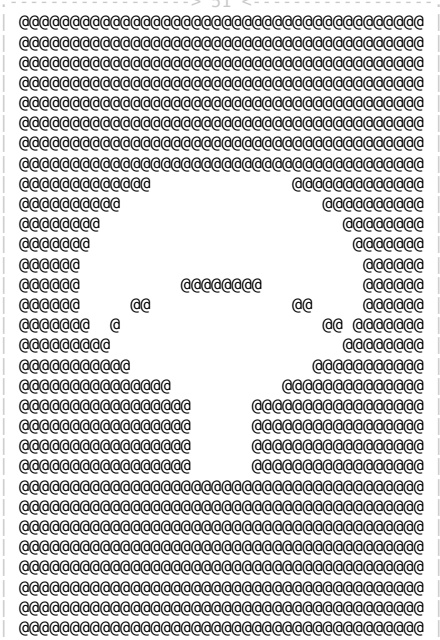
size: 82

```
333   0  14   1  23   0  20   1  18   0
 24   1  15   0  26   1  13   0  28   1
 12   0  10   1   8   0  10   1  12   0
  5   1   2   0  14   1   2   0   5   1
 13   0   2   1   1   0  20   1   2   0
  1   1  16   0  23   1  19   0  18   1
 26   0  11   1  31   0   6   1  34   0
  6   1  34   0   6   1  34   0   6   1
337   0
```

-> 50 <



-> 51 <



-> 52 <

rle(invert(bw(encoded)))

size: 82

```
333   1  14   0  23   1  20   0  18   1
 24   0  15   1  26   0  13   1  28   0
 12   1  10   0   8   1  10   0  12   1
  5   0   2   1  14   0   2   1   5   0
 13   1   2   0   1   1  20   0   2   1
  1   0  16   1  23   0  19   1  18   0
 26   1  11   0  31   1   6   0  34   1
  6   0  34   1   6   0  34   1   6   0
337   1
```

-> 53 <

squeeze(encoded,10)

size: 124

```
  0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0
  0   0   0   4   4   0   0   6   6   0
  1   6   6   1   1   6   6   1   2   6
  6   2   2   3   3   2   2   1   1   2
  1   1   1   0   0   1   1   0   0   0
  0   0   0   0   1   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0
  0   0   0   0
```

-> 54 <

rle(squeeze(encoded,10))

size: 46

```
 33   0   2   4   2   0   2   6   1   0
  1   1   2   6   2   1   2   6   1   1
  1   2   2   6   2   2   2   3   2   2
  2   1   1   2   3   1   2   0   2   1
  7   0   1   1  49   0
```

-> 55 <