# COMPRESSION AND INFORMATION \$\&\(\alpha\)\(\delta\)\(\d

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?

ENCODE AND DECODE

# 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 do what you are about to do, so do not get discouraged.

- > 1. WATCH NARUTO IF YOU HAVENT
- > 2. PTCK AN TMAGE 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 Haikyuu Hajime no Ippo My Hero Academia Sword Art Online

# .....

Images are just an array of pixels, a pixel is just a dot of color, your monitor probably has more than 2,073,600 pixels.

WHAT IS AN IMAGE

Each pixel has 3 one byte values:

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

So the image is list of pixels, and each pixel has the 3 colors. For example, we could have one red and one magenta pixel next to each other:

```
[\ldots, (255,0,0), (255,0,255), \ldots]
```

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

...\*.. ..\*\*\*. .\*\*\*\*.

## ENCODE AND DECODE

First we will transform our text image into something we can use, like a list of numbers. We will encode each symbol from the image with a number, and we will create a symbol table so we can use it to decode the image later. For example, having this 6x7 image:

```
...*
..***..
.****
```

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

. . . . . . .

Our symbol table will look like this:

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

and our encoded image would look like: 0000000 0001000 0011100 0111110 0002000

0000000

```
def encode(x):
 sym = \{\}
 r = []
 for v in x:
   if v not in svm:
     # first time we see a symbol
     # we put it in the dictionary
     svm[v] = len(svm)
   # 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. "a": 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
Lets flatten this image to how it
actually looks as a list of numbers:
               0000000
               0001000
               0011100
              0111110
              0002000
               0000000
becomes:
 0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,1,1,1,
 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 = 11
  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])
```

```
DELTA ENCODING
Lets look at a list of numbers:
[1, 3, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
If we runlength encode this list it will
blow up, most elements appears only once
 1,1,3,2,4,1,5,1,6,1,7,1,8,1,9,1,
  10.1.11.1.12.1
However we can encode the difference
between each element, so
 1, # first element is 1
  2, # 3 - 1
 0, #3 - 3
 1, #4 - 3
This is called delta encoding, we encode
the the difference (the delta).
So the list becomes very repetitive:
[1, 2, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1]
Now we can runlength encode it:
[1, 1, 1, 2, 1, 0, 9, 1]
To go back to the original list first we
need to run length decode and then to
delta decode.
```

```
DELTA ENCODING
# delta encode a list of numbers
# from:
# [1,2,3,4,5]
# to:
# [1.1.1.1.1]
def delta(x):
 r = [x[0]]
  for i in range(1, len(x)):
    current = x[i]
    prev = x[i-1]
    d = current - prev
    r.append(d)
  return r
# delta decode a list of numbers
# from:
# [1.1.1.1.1]
# to:
# [1.2.3.4.5]
def undelta(x):
  r = [x[0]]
  for i in range(1, len(x)):
    r.append(x[i] + r[-1])
  return r
```

```
REDUCE INFORMATION
Going back to our tree:
                ...*...
                ***
                ****
                . . . | . . .
                . . . . . . .
How would it look if we try to squeeze
it in fewer pixels? For example take
every 2 pixels and average them together
and then explode it back:
original
            squeeze
                          exploded
0000000
          0 0 0 0
                      0000000
0001000
          0 0 0 0
                      0000000
0011100 -> 0 1 0 0 -> 0011000
0111110
          0 1 1 0
                      0011110
0002000
                      0011000
          0 1 0 0
0000000
           0 0 0 0
                      0000000
Then if we draw it again:
                . . . . . . .
                **
                ****
                ..**...
                . . . . . . .
Its ugly, but it .. kind of looks like
```

```
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):
   avq = 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 = 11
  for v in x:
    for i in range(n):
      r.append(v)
  return r
```

```
FTI TERS
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 nixel:
   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.
```

```
FILTERS
def blur(x):
 # fake blur, averaging every 3 element
  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
```

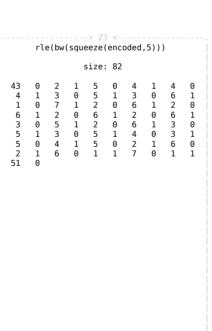
```
.....
....0/+++++++++++++++0000%+++++++0
....@+++++++++0++++++++++++++++++++++
.....%++++++++0/++000......
...........
```

```
00000000000011222222221100000000000000
00000000012222222222222100000000000
00000000122222222222222222210000000000
000000032222222224222222210000000000
000001122222222222222222222222510000000
00016222222222222222111172222222100000
00012222222222222222222222222222220000
00012222222221222222222222222222210000
0001222222222122222222222222222100000
000001222222211222222222222222210000000
0000000111111722222222222222222222100000
0000000122222222222222221522222210000
000000001222222222222222222222222220000
0000000000112222222222222222222222200000\\
00000000000000072222222216221110000000
000000000000000012222222222230000000
00000000000000000012222222221000000000
```

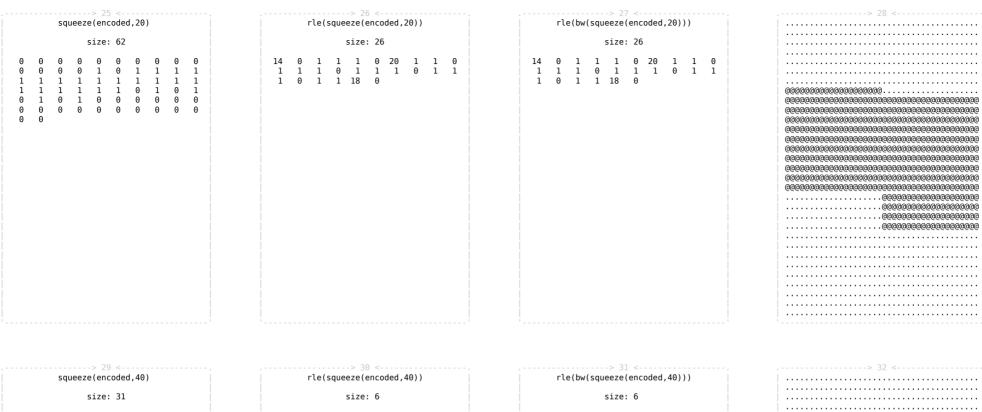
### rle(encoded) size: 216 213 0 2 1 9 2 2 1 24 0 1 16 2 1 1 21 0 1 1 19 2 1 1 18 0 1 3 11 2 8 2 1 1 15 0 2 12 2 1 1 2 2 1 1 2 1 1 10 0 1 1 1 6 4 1 1 7 7 2 1 1 1 1 31 2 1 1 7 0 11 2 1 1 19 1 18 1 1 11 2 1 1 10 0 1 1 8 1 14 1 12 0 1 1 18 2 2 1 7 2 1 1 12 0 1 5 1 10 1 15 1 2 1 12 2 1 1 21 Θ 1 7 9 2 3 1 24 0 1 1 2 1 3 27 0 1 1 10 2 1 30 0 1 1 7 2 1 1 31 0 1 1 4 2 2 1 32 0 2 1 1 6 257

> 19 <	
++9000+++++9000+++++9000	
+++++++++++++++++++++++++++++++++	
+++++++++++++++++++++++++++++++++++	
000++++++++000++++++++++++++000	
++++++++++++++++++++++++++++++++	
+++++++++++(((@@	
+++@@@	
+++	

		s	quee	ze(ei	ncode	ed,5	)		
			S	ize:	248				
0 0 0 0 1 2 0 1 2 1 0 0 1 0 0 1 0 0 0 0	0 0 0 0 0 2 1 0 2 2 2 0 0 0 0 0 0 0 0 0	0 0 0 0 2 0 0 2 2 2 2 0 0 0 2 2 2 0	0 0 0 0 1 2 2 0 0 1 2 2 0 0 0 0 0 0 0 0	0 0 0 0 1 0 0 2 1 2 0 0 0 2 0 0 0 0 0 0	0 0 0 0 0 1 1 2 0 0 0 2 2 1 2 0 0 0 0 0	0 0 0 0 0 0 2 2 1 0 1 2 1 0 0 0 0 0 0 0	0 0 0 0 0 0 2 1 0 2 2 1 2 0 0 0 0 0 0 0	0 0 0 0 0 2 2 1 0 2 2 1 0 0 0 0 0 0 0 0	0 0 0 0 0 0 2 1 1 2 2 0 0 0 0 0 0 0 0 0

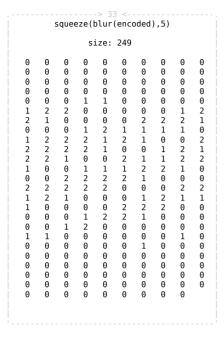


..... ..... ..... .........++++++++++++++000000...... ..... ..... @@@@@@@@@@@@@+++++++++++++ .....++++++++++++++++++000000..... .....+++++++++0@0000+++++++++++ ..... .....++++++++ ..... 



squeeze(encoded,40)	rle(squeeze(encoded,40))	rle(bw(squeeze(encoded,40)))
size: 31	size: 6	size: 6
0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1	7 0 12 1 12 0	7 0 12 1 12 0

> 32 <
000000000000000000000000000000000000000
999999999999999999999999999999999999999
09
999999999999999999999999999999999999999
000000000000000000000000000000000000000
000000000000000000000000000000000000000
000000000000000000000000000000000000000
000000000000000000000000000000000000000
000000000000000000000000000000000000000
000000000000000000000000000000000000000
000000000000000000000000000000000000000
000000000000000000000000000000000000000



rle(squeeze(blur(encoded),5)) size: 140											
43 5 3 4 1 2 1 3 3 3 5 6	0 0 2 1 2 0 1 1 0 0 0 2 0 0	2 1 1 1 1 2 2 4 2 1 5 1 1	1 1 1 0 1 1 0 0 2 2 1 0 1 1 1	5 2 3 1 2 1 1 3 1 1 1 1 1 7	0 2 0 1 0 2 2 1 1 1 2 1 2 0	1 1 1 3 5 1 2 2 3 1 3 2 6 1	1 1 1 2 2 1 1 2 0 2 1 2 0 1	2 4 1 1 2 2 1 5 1 4 1 2 552	2 0 2 1 1 2 2 1 0 1 1 0		

rle	(del	ta(		> 3 ze(	blur(	enc	oded	)),5	))
			si	ze:	180				
1 2 1 2 1 2 1 4 1 1 2 2 1	0 1 0 -1 -1 0 -1 1 0 -1 -1 0 -1 -1 0 -1 -1 -1	1 1 2 2 2 1 1 1 1 1 2 1 4 1 6	1 0 -1 0 1 0 0 1 0 -2 1 0 -2 0 0 0 0 0 0 0 0 0 0 0 0 0	1 1 3 2 2 1 1 1 2 2 2 2 1 4 2 1 1	0 -2 0 1 0 2 -1 2 -1 0 -1 -1 0 1 -1 1	1 4 1 1 1 1 2 2 1 2 3 2 1 5 1	-1 0 2 -1 -1 0 -1 0 -1 0 2 0 0 1 -2 0 -2	4 2 3 1 2 1 1 1 2 1 1 1 5 1 51	0 1 0 0 1 1 -1 0 0 1 2 0 0 1 2 0 0 1 0 0 0 0 0 0 0 0

> 36 <	
ļ	
+++++++++++++0@@@@	
0000000000000000000000000000000000	
@@@@@++++++++++++++++++++++++++++++++++	
+++++++++++++++++++++++++++++++++++	
000000+++++000000+++++++++000000	
+++++000000000000000000000000000000	
++++++++++++++++++++++++++++++++++	
++++++++++++++++++++++++++++++++++	
+++++++++++++++++++++++++++++++	
0000000000000000000000000000000000	
++++++++++++++	
	• • •
	٠٠٠ إ
	• • •
	٠٠٠ إ
	• • •
ļ	• • •
	• • •
ļ	• • •
	• • •

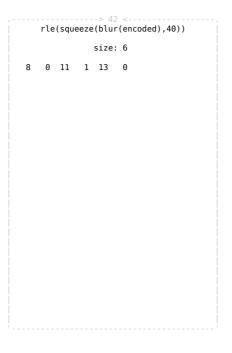
;		s	queez	ze(b	-> 37 lur(e	<	ed),	20)		
				:	size:	63				
	0 0 1 1 0 0 0	0 0 1 1 1 0 0	0 0 1 1 0 0 0	0 0 1 1 0 0	0 1 1 1 0 0	0 0 1 1 0 0	0 1 1 0 0 0	0 0 1 1 0 0	0 1 1 0 0 0	0 1 1 1 0 0

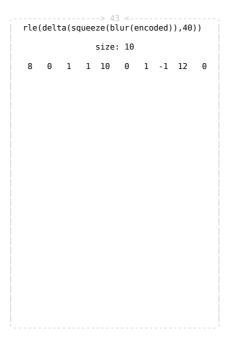
 	rle(squeeze(blur(encoded),20))												
					size:	26							
	14 18 1	0 1 0	1	1 0 1	1 1 21	1	1	1 0		0			

rle	rle(delta(squeeze(blur(encoded)),20))											
			9	size	: 30							
	1	17	0	1	-1 -1 1	1	1	1	-1			

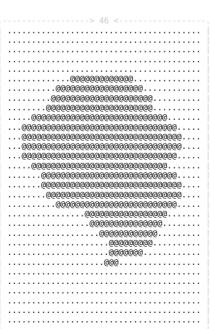
> 40 <	
40 <	i
	i
(මය) මය වැනි වැනි වැනි වෙන	i
@@@@@@@@@@@@@@@@@	
<u> </u>	İ
<u> </u>	ĺ
09	İ
<u> </u>	
000000000000000000000000000000000000	
000000000000000000000000000000000000	
000000000000000000000000000000000000000	ļ
000000000000000000000000000000000000000	ļ
000000000000000000000000000000000000000	
000000000000000000000000000000000000000	
000000000000000000000000000000000000000	
000000000000000000000000000000000000000	ŀ
	ŀ
	i

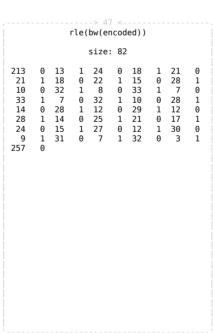
į	squeeze(blur(encoded),40)											
					size:	32						
	0 1 0 0	0 1 0 0	0 1 0	0 1 0	0 1 0	0 1 0	0 1 0	0 1 0	1 1 0	1 0 0		





> 44 <	
000000000000000000000000000000000000000	
000000000000000000000000000000000000000	
000000000000000000000000000000000000000	
000000000000000000000000000000000000000	
000000000000000000000000000000000000000	
000000000000000000000000000000000000000	
000000000000000000000000000000000000000	
000000000000000000000000000000000000000	
000000000000000000000000000000000000000	
0990999999999999999999999999999999999	
099099999999999999999999999999999999999	99999





<u>හර්තර්තර්තර්තර්තර්තර්තර්තර්තර්තර්තර්තර්තර</u>
(වල් වෙර විට විට විට විට විට විට විට විට විට විට
තුව හිට වෙන වෙන වෙන වෙන වෙන වෙන වෙන වෙන වෙන වෙන
බෙනවානවානවානවානවානවානවානවානවානවානවානවානවාන
000000000000000000000000000000000000000
000000000000000000000000000000000000000
000000000000000000000000000000000000000
000000000000000000000000000000000000000
000000000000000000000000000000000000000
000000000000000000000000000000000000000
0000000000
00000000
00000000
000000000
00000000000000
00000000000000
00000000000000
000000000000000000000000000000000000000
000000000000000000000000000000000000000
000000000000000000000000000000000000000
000000000000000000000000000000000000000
000000000000000000000000000000000000000
000000000000000000000000000000000000000
000000000000000000000000000000000000000
000000000000000000000000000000000000000
000000000000000000000000000000000000000
000000000000000000000000000000000000000
000000000000000000000000000000000000000
000000000000000000000000000000000000000
000000000000000000000000000000000000000
000000000000000000000000000000000000000

<pre>rle(invert(bw(encoded)))</pre>									
				size:	82				
213 21 10 33 14 28 24 9 257	1 0 1 0 1 0 1	13 18 32 7 28 14 15 31	0 1 0 1 0 1 0	24 22 8 32 12 25 27 7	1 0 1 0 1 0 1	18 15 33 10 29 21 12 32	0 1 0 1 0 1 0 1	21 28 7 28 12 17 30 3	1 0 1 0 1 0 1

51 <
000000000000000000000000000000000000000
000000000000000000000000000000000000000
000000000000000000000000000000000000000
000000000000000000000000000000000000000
000000000000000000000000000000000000000
000000000000111111111111100000000000000
000000000111111111111111111110000000000
000000001111111111111111111111110000000
0000000111111111111111111111111111000000
00000011111111111111111111111111111110000
001111111111111111111111111111111111110000
000011111111111111111111111111111111111
000111111111111111111111111111111111111
001111111111111111111111111111111111110000
00001111111111111111111111111111111110000
00000000111111111111111111111111111110000
00000000111111111111111111111111111110000
000000011111111111111111111111111111111
0000000000011111111111111111111111110000
000000000000011111111111111111110000000
0000000000000000111111111111111111110000
000000000000000000111111111111111000000
000000000000000000001111111111000000000
000000000000000000000011111100000000000
000000000000000000000111000000000000000
000000000000000000000000000000000000000
000000000000000000000000000000000000000
000000000000000000000000000000000000000
000000000000000000000000000000000000000
000000000000000000000000000000000000000
000000000000000000000000000000000000000

> 52 <
1 1
6000000000000000000000000000000000
0909090909090909090909090909090909
000000000000000000000000000000000
İ j

<pre>rle(bw(blur(encoded)))</pre>									
				size	: 82				
213	0	12	1	24	0	18	1	21	0
21	1	18	0	24	1	15	0	27	1
9	0	33	1	9	0	33	1	6	0
33	1	6	0	33	1	9	0	30	1
15	0	27	1	12	0	27	1	12	0
30	1	15	0	24	1	18	0	18	1
24	0	18	1	24	0	12	1	30	0
9	1	33	0	6	1	33	0	3	1
258	0								

777777777777777777777777777777777777777
777777777777777777777777777777777777777
777777777777777777777777777777777777777
777777777777777777777777777777777777777
777777777777777777777777777777777777777
777777777776655555555566777777777777
7777777776555555555555555567777777777
777777776555555555555555555556777777777
777777745555555555555555555556777777777
77777665555555555555555555555555555
7776155555555555555555666605555555677777
7776555555555555555555555555555555
777655555555555555555555555555555
777655555555555555555555555555555
7777765555555556655555555555555556777777
77777766666605555555555555555555677777
7777776555555555555555555555555555
777777765555555555555555555555555
7777777766555555555555555555555677777
7777777777777770555555556155666777777
777777777777777776555555555555547777777 7777777777
777777777777777777777777777777777777777
777777777777777777777776555566777777777
777777777777777777777661777777777777777
777777777777777777777777777777777777777
777777777777777777777777777777777777777
777777777777777777777777777777777777777
777777777777777777777777777777777777777
777777777777777777777777777777777777777
777777777777777777777777777777777777777

%%%%%%%%%/&&&&&&&&&&&&&&&&&&&& %%%%%%%%\*\*&&&&&&&&&&&&&&&&&& %%%/&&&&&&&&&&&&&&&&&&&&&&&&&&& %%%%/&&&&&&\/&&&&&&&&&&&&&&& %%%%%%//////. &&&&&&&&&&&&&&&&&& %%%%%%/&&&&&&&&&&&&&&&/+&&&&&&/%%% %%%%%%%%%//&&&&&&&&&\\d&&&&&\\d&&&&&\