

ASCII Art Generator Using Brute Force Algorithm

Yusuf Rahmatullah / 13512040
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13512040@std.stei.itb.ac.id

Abstract—ASCII Art is the new kind of art that create image from ASCII character. The ASCII character generated randomly, but the color of character is set by pixel of the image. In order to show the color of character, ASCII Art Generated to .html file format. So the image can be show in an internet browser application.

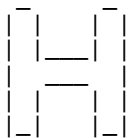
Index Terms—Brute Force, ASCII Art, Image, Picture

I. INTRODUCTION

ASCII is abbreviation for American Standard Code for Information Interchange. ASCII is a character encoding scheme originally based on the English alphabet that encodes 128 specified character, the numbers 0-9, the small letters a-z, the big letters A-Z, some basic punctuation symbols, some control codes that originated with Teletype machines, and a blank space, into 7-bit binary integers.[2]

ASCII art is a graphic design technique that uses computers for presentation and consists of pictures pieced together from printable ASCII character. ASCII art has two form, first is the shape-oriented form and the second is the color-oriented form.

Shape-oriented form of ASCII art created by look the shape of the image. Example the shape of letter “H” in the shape-oriented form ASCII art is like this :



In this era of high quality technology, the image that will be generated is from picture that take by camera. So the shape of the image depends on density of every pixel in the picture. the explanation can be shown with this image :

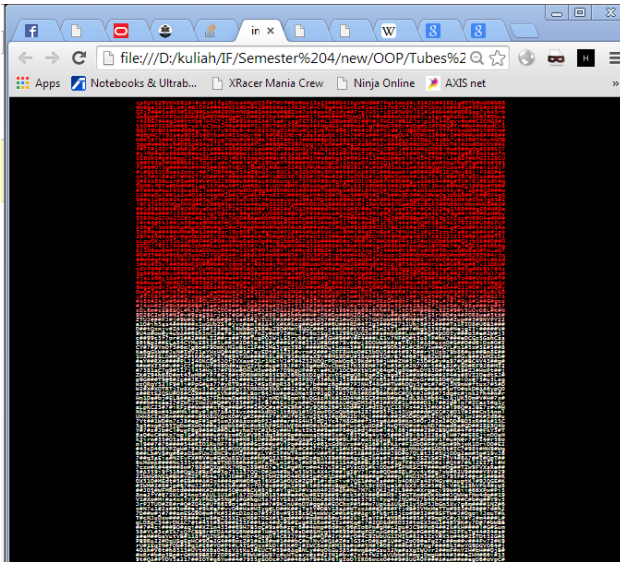


Source : collerctorsquest.com

Pixel that has tiny density can be drawn by tiny density character such as apostrophe (‘), caret (^), asterisk (*), dot (.), comma (,), and quotation mark (“); pixel tha has medium density can be drawn by medium density character such as O, D, exclamation mark (!), question mark (?), open bracket (‘(‘ and close bracket (‘). And pixel that has large density can be drawn by large density character such as M, N, W, at sign (@), number sign (#), and ampersand (&).

Due to the the number of the possibility which character draw the pixel, second form of ASCII art has created. Color-oriented form ASCII art will change every pixel by any character but the color of the character set by color of the pixel. Color-oriented form ASCII art doesn’t determine which character that draw every pixel of an image but determine the color of the text.

Color-oriented form of ASCII art needs a font-face that height and width has the same size. This form need colorable text so every pixel of the picture can be changed by any character. The example of color oriented form ASCII art can be shown by this :



II. FUNDAMENTAL THEORY

A. Brute Force Algorithm

Brute force algorithm is kind of algorithm that has straightforward approach to solving a problem. The brute force algorithm is based on problem statement and entangled concept. The brute force algorithm solve the problem with simple algorithm, directly, and obvious way.[1]

Generally, the brute force algorithm is “not smart” algorithm and not effective algorithm because the brute force algorithm need a lot of computation time and need much time to solve the problem. The “force” from brute force indicate “power” then “brain”. Sometimes the brute force algorithm called naive algorithm.

Brute force algorithm is fit for small problem. The brute force algorithm is used by programmer to compare his algorithm, to check the effectiveness of his algorithm.

Brute force algorithm isn't effective algorithm, but this algorithm can solve any problem. it's too hard to show a problem that can't be solve by brute force algorithm. In fact, there is a problem that can be solved only by brute force algorithm.

Example use of brute force algorithm is finding the biggest element (or the smallest element) in an array. Algorithm to solve for this problem is like the following pseudo-code :

```

Procedure FindBiggestElement(input A : array of
integer, output max : integer)

var i : integer

max := A[1]
for i := 1 to n do
  if A[k] > max then
    max := A[k]
  endif
endfor

```

B. Image File Formats

Image file formats are standardized means of organizing and storing digital images. Image file are composed of digital data in one of these formats that can be rasterized for use on a computer display or printer. An image file format may store data in uncompressed, compressed, or vector formats. Once rasterized, an image becomes a grid of pixels, each of which has a number of bits to designate its color equal to the color depth of the device displaying it.

There are two kind of image, bitmap image and vector image. Bitmap image is kind of image that data of every pixels in the image is stored in the file. Vector image is kind of image that data of every pixel is generated by formulas. These formulas are stored in the file.

Bitmap Image file size is correlated to the number of pixels in a image and the color depth. Every pixel in a image is stored into an matrix by its color value. The color value is depended on image's color depth.

Color depth is max of bit value that representate three fundamental color, red, green, and blue (RGB). Image that has 24-bit color depth has 8-bits for each in RGB. 8-bits representate integer from 0—255. So 24-bit color depth has $256 * 256 * 256 = 16,777,216$ possibility color.

Now, there is image that has 32-bit color depth. It's contain an alpha number for every pixel. Alpha is transparent of pixel. 255 if pixel is shown and 0 if pixel is not shown. This color depth also called alpha, red, green, blue (aRGB) format.

Including proprietary types, there are hundreds of image file types. The major of bitmap image file formats is PNG, JPEG, and GIF formats. The major of vector image file formats is CDR (CorelDRAW), AI(Adobe Illustrator), and SVG (Scalable Vector Graphics).

There are compound image file format that contain both pixel and vector data. The major of this types is PDF (Portable Document Format) and SWF (Shockwave Flash).

III. BRUTE FORCE ALGORITHM IMPLEMENTATION IN ASCII ART GENERATOR

ASCII art generator that discussed in this paper is second form ASCII art generator. The generator read the image then convert it to simple webpage. The generator is draw every pixel in the image by any character include pixel's color to coloring every character.

The generator is made by me in JAVA programming language with command line interface (CLI). The generator read an bitmap image file format because this type of image file format stores it's data in pixel format. So the generator can convert every pixels to any character and can coloring the character depend on it's color in pixel data.

The generator only use one bitmap image file format and that is JPEG file format. The generator read one image file that has name img.jpg then convert it to webpage that has name img.html.

Pixel data in img.jpg read by generator and stored in array of integer that has size as image size. If the image has h pixel in height and w pixel in width so the array of integer has size h*w. Each integer (4-bytes / 32-bits) contains 32-bit color depth format.

Generator put the pixel data to the memory use method ImageIO.read(File) from package javax.imageio.ImageIO in JAVA programming language. Then the stored data are put to the array of integer.

After the generator read the pixel data, the generator generate the ASCII character, coloring each character, and write it into img.html using method write(String) from package java.io.Writer in JAVA Programming Language.

The conversion of each pixel data use brute force algorithm. The brute force algorithm is used to converting each pixel data to colored ASCII character because the brute force algorithm iterate the array of integer from the start to the end of array.

The brute force algorithm for draw each pixel can be shown by the following pseudo code :

```

var i ,j : integer
var w : integer
var h : integer
var A : array of integer
var image : PixelData
function generateRandomChar() : char

w := image.width
h := image.height
for i := 0 to h-1 do
  for j := 1 to w do
    write(A[i*w+j] + generateRandomChar())
  endfor
  write(newline)
endfor

```

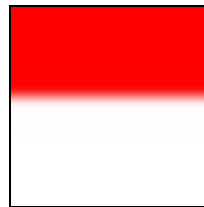
That algorithm has algorithm complexity $T(m,n) = mn$ where m is the height of the image and n is the width of the image. So the algorithm has time complexity $T(m,n) = O(mn)$.

Because of the the generator using HTML webpage as the output of ASCII art, the generator must write each character using HTML format. For each character, it's written `(_char_)` with (_color_) is color format in hex-string and (_char_) is randomly generated ASCII character.

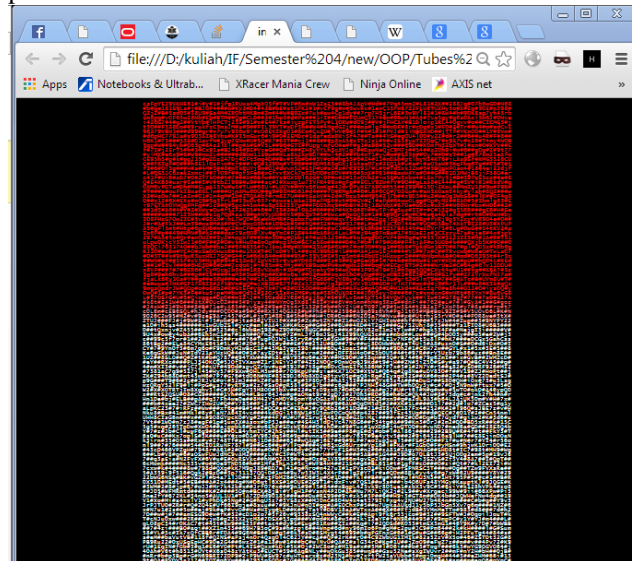
The generator must draw each character like each pixel, so the size of one character width is must the same as the other width. I use the HTML's font tag to change the font-face of all character. I use Lucida Console font because this font has same width for all character.

To testing the generator, I use three images file that has image file format JPEG. For each image file, it's name is renamed to img.jpg in order that generator can read each file.

First image that I use is square image that size is 100 x 100 pixel. The original image is like the following picture :



And the result of generated page is like the following picture :



The size of image is 1.84 KB, but the size of output file, img.html, is 293 KB. It's 159 times large from the original file. it can be done because for each character is written with HTML's font tag (``)

The image has homogenous color. If character that has some color and wrote in order is written by one HTML's font tag, the size of output file can be decreased. But with this concept, the worst case is when the each color of the character (each color of pixel too) is different. But the best case is when all color of the character is same.

The implementation of this concept needs change recent algorithm. The algorithm is added with previous color checking algorithm. If the current color is the same as the previous color, so the generator just write the generated character, but if the current color is different from the previous color, the generator write the character within HTML's font tag like the recent algorithm.

The new algorithm of this concept can be shown by the following pseudocode :

```

var i ,j : integer
var w : integer
var h : integer
var A : array of integer
var image : PixelData
function generateRandomChar() : char

w := image.width
h := image.height
for i := 0 to h-1 do
  for j := 1 to w do
    if(A[i*w+j] = A[i*w+j-1]) then
      write(generateRandomChar())
    else
      write(A[i*w+j] + generateRandomChar())
    endif
  endfor
endfor

```



```
write(newline)
endfor
```

This algorithm has algorithm complexity $T_1(m,n) = mn$ for conversion algorithm and $T_2(m,n) = mn$ for previous color checking algorithm. Each algorithm complexity has same time complexity and that is $T(m,n) = O(mn)$. So the total of time complexity is $O(mn) + O(mn) = O(mn+mn) = O(2mn)$.

But this algorithm will error when $i = 0$ and $j = 1$, because the check algorithm will access $A[0*w+1-1] = A[0]$ that element doesn't exist. So in the implementation to the JAVA programming language, I use *try-catch* scope then while the program throws *ArrayIndexOutOfBoundsException* the program can catch the *Exception* then write the character as the previous algorithm.

The size of new output file from first image (square red-white colored image) become 32.4 KB. It just 17 times large then original file, and the new size is 11% from the previous size.

The first image is take 150 ms to execute when use the previous algorithm (algorithm without previous color checking algorithm) and take 298 ms to execute when use the current algorithm (algorithm within previous color checking algorithm).

The second image that I use is ganesha logo of Bandung Institute of Technology. This image has 400 pixel as it's width and 537 pixel as it's height. the image has 79.8KB as it's size. The original second image is shown by following image :



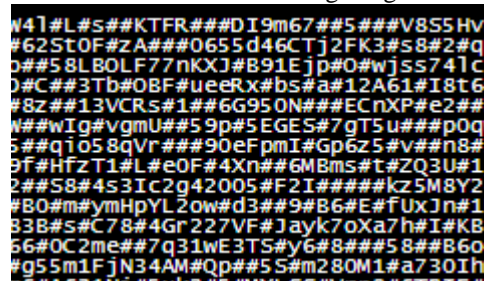
The result is generated with Lucida Console font and 2pt size. And the result of the second image is like the following image :



The size of the output file using the previous algorithm is 6.02 MB. It takes $6.02MB/79.8KB = 77$ times large than the original file size. The generator with the previous algorithm takes 488 ms to execute.

When the generator uses the current algorithm, the size of output file become 1.86 MB. It just takes $1.86MB/79.8KB = 24$ times large than the original file size. the generator with this algorithm takes 961 ms to execute.

This result is like has some noise (little black dot that has a size as a pixel). While I take a zoom into the generated image, the result doesn't has a noise. The zoomed-result is like following image :



I think that the noise is come from the character that has small or tiny density. The noise is come from the So, I modify the function generateRandomChar to return only large density character and I use the at sign (@). The new result is like following image :



The noise has cleared.

The third image that used to test the generator is my

picture. this image has 244 pixel as it's width and has 282 pixel as it's height. The image has 64.5 KB as it's size. the original image of the picture is like following image:



The result of the original picture is like the following image :



This result is generated with previous setting of function generatRandomChar() (use A-Z, a-z, 0-9, and # character). But to clear the noise I try to only use at sign (@) character. Then the result is :

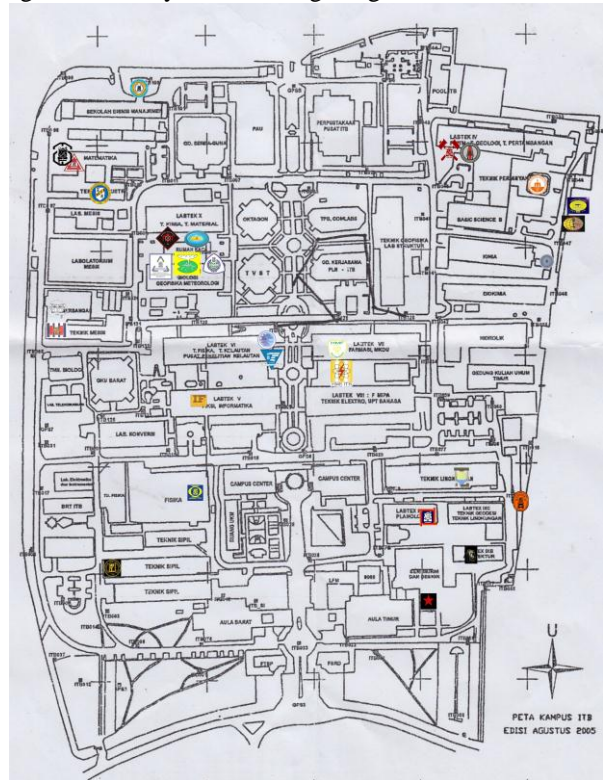


The size of output file using first algorithm (algorithm without previous color checking algorithm) is 1.96MB. It takes $1.96\text{MB}/64.5\text{KB} = 31$ times large than the original image. The generator take 382 ms to execute with this algorithm.

When the generator use the second algorithm (algorithm within previous color checking algorithm), size of the output file become 1.94MB. it takes $1.94\text{MB}/64.5\text{KB} = 30$ times large than the original image. The generator take 679 ms to execute with this algorithm

The first and the second algorithm has the similar output size. it's because the color of each pixel is different with the previous pixel's color.

For each picture that takes to be testing image for the generator, the generator take under 1 second to execute them. Because size of the original image is relatively small. So I test the ASCII Art Generator using the big size image file. This image has 4937 pixel as it's width and 6469 pixel as it's height. This image has 3.74MB as it's size. The image is shown by the following image :



The image is Bandung Institute of Technology's Map. This image will generate by the ASCII Art generator to an HTML webpage format file with Lucida Console font and 1pt as it's size. This font size is needed because of the image's pixel size. And the function generateRandomChar() is modify to return only at sign (@) character.

The result of this image isn't displayed because the size is too large. With the first algorithm, the size of the output file is 913MB. it takes $913\text{MB}/3.74\text{MB} = 244$ times large than the original image. The generator takes 16177 ms to execute this algorithm.

With the second algorithm, the size of output file is 511MB. it takes $511\text{MB}/3.74\text{MB} = 136$ times large than the original image. The generator takes 12714 ms to execute this algorithm.

IV. CONCLUSION

Brute force algorithm is fit to use by ASCII art generator because the ASCII art generator stores the pixel data in memory and brute force algorithm iterates this from the start of array of integer to the end of array of integer. The ASCII art generator makes smaller size of the output file when the algorithm is included with the previous color checking algorithm. The output file will have a small size when the color of each pixel in the image is homogenous. When the picture has heterogeneous color, the size of the output file with first algorithm is similar with the size of output file with second algorithm.

VII. ACKNOWLEDGMENT

Thanks to our lord Allah SWT who gives me power and spirit to finish this paper that has title *ASCII Art Generator Using Brute Force Algorithm*.

Thanks to my teacher, Mr. Rinaldi and Mrs. Masayu for their guidance in Algorithm Strategy course in Bandung Institute of Technology. Without their teaching and their encouragement, this paper wouldn't be completed.

Thanks to my parent for their support and my friends who had helped me in the completion of this paper.

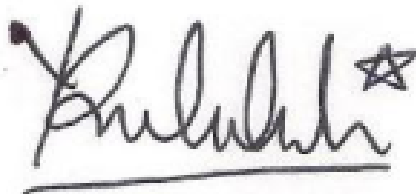
REFERENCES

- [1] Munir, Rinaldi. *Algoritma Brute Force*. [http://informatika.stei.itb.ac.id/~rinaldi.munir/Smik/2013-2014-genap/Algoritma%20Brute%20Force%20\(2014\).ppt](http://informatika.stei.itb.ac.id/~rinaldi.munir/Smik/2013-2014-genap/Algoritma%20Brute%20Force%20(2014).ppt) (retrieved 18 Mei 2014)
- [2] R., Shirley. *RFC 4949*. August 2007

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 18 Mei 2014



Yusuf Rahmatullah
13512040