A Review SEARCH BITMAP IMAGE FOR SUB IMAGE AND THE PADDING PROBLEM

Omeed Kamal Khorsheed
School of Engineering, Dept. of S\w Engineering
Koya University, Iraq - Erbil

ABSTRACT
In this paper we will discuss the algorithm of search Bitmap image for sub image or duplicate images ,It is difficult for people to find or recognize if sub images are Found in large Image .This algorithm is a part of Image Processing Technique to help people solve Search problem .This algorithm has possibility to Convert the image to Binary Matrix and to individualize The Reading Buffer Size according to the padding edge. This algorithm is the first step to create Image Search Engine which can find matching image without looking to image name text or alternative text tag.

KEYWORDS: *Sub Image, Image Search Engine, Image Processing, Bitmap, Padding, algorithm, Duplicate, Matrix, Bitmap file header, Java, Looping, Coordinate.*

I. INTRODUCTION

After The Huge a evolution in multimedia world ,Images search and images manipulation process must grow up to be powerful and effective especially with Image sharing web site , Millions of images are published on the web every day , How we can make search for an image or a part of it easy like we search text in Google?
Images are not usually named with conveniently text, and the alternative Tag all the time is empty. So the images need to be searched by their pixel.
Now it is important to compare images with images or sub image and to locate images that contain the pattern which we are looking for it. The pattern can be a picture of anything Number, Character, person, building or just duplicate to the same image, and the search operation is to retrieve matching images. This retrieval is called the sub image searching.
In this paper, we propose an algorithm for searching sub image into large image and to locate the indexes to all matching Pattern. This algorithm can solve any Image search for 24-bit Bitmap image format , And we will discuss The padding problem in bitmap file .The search process need big loops in four levels deep and the Inner condition that find any different in pixel must rest the looping and exit out the Inner loop to the main loop which start with New Line[7] .

II. SURVEY OF BITMAP IMAGE

The bitmap is very strictly bound to hardware platform architecture, Bitmap was created by Microsoft and IBM, Bitmap file format compatible with Intel File format which called the little endian format [1, 2, 3, 5, and 8] because of the byte order that an Intel processor uses internally to store values.
Bitmap file has four types of headers; those types are differentiated by the size member, which is the first dword in each of the structures.

1.	BITMAP CORE HEADER
2.	BITMAP INFO HEADER
3.	BITMAP V4 HEADER
4.	BITMAP V5 HEADER

In "BITMAP INFO HEADER" there is a color table describes how pixel values correspond to RGB color values. RGB is a model for describing colors that are produced by emitting light [4].Bitmap pixels stored in order binary integers format this order can by (red, blue, green) but in 24-bit there is reverse ordering (blue, green, and red) [5] so each image format has deferent bits order which known as color Mask . Bitmap pixels stored in upside way (down to up) and that mean The first line in the file is the bottom line of the image and the last line in the file is the first line of the image.[6]

2.2. Bitmap Image Most popular Format:

- 8-bit image is 8 bits file and it use 1 bytes for color information.
- 16-bit image is 16 bits file and it use 2 bytes for color information.
- 24-bit image is 24 bits file and it use 3 bytes for color information.
- 32-bit image is 32 bits file and it use 4 bytes for color information.

2.3. 24-bit Bitmap Image Padding Problem:

Bitmap file BITMAPINFOHEADER Header has two parts (14-byte file header and a 40-byte information header) , These two parts Contain information about Data stored in the file and the order of that data .IF we look at The 24-bit Bitmap Header We will find Describe For Image Size, Row Length (width) , Column Length (height) and the RGB (Red/Green/Blue) color information values (pixel) which are used in the file to present the image Details , RGB need 3 bytes in the 24-bit Bitmap image. [1, 2, 3, 5, 6, 8]

Each pixel is represented by Triple of RGB. Depending on the width of the image, pad maybe used at the end of each Row. If width is a multiple of 4, no pad is required, otherwise at the end of each line we need to add a number which will make the width divisible by 4 this number is Gap with Zero Value.

When OS Load an Image to the memory there is a scan Operation, each scan line is padded out to an even 4-byte boundary (even 4-byte has odd 3 bytes of RGB). That mean every line in the 24-bit image file has point to determine the end of line this point called Pad and it must be even number of 4-bytes Triple, in the same time and at the same scan line it must has RGB of 3-bytes number, and we can calculate the scan line length by counting the RGB 3- bytes according to this equation:

Scan Row length=Scan Row width * 3

Dependence on the above equation the result could be odd number, but the boundary must be even number which means maybe the Row has gap between Row length and Row width, this gap known as Padding Problem, we can get the Image size, Width and Height from the Bitmap Header.

2.4. Example for padding:

If we have a Bitmap image with depth 24-bit and the area (X * Y) of this image is 15 * 15.Now we can calculate the color pixel information like that (15 * 3=45 bytes) in each row.

45 bytes is the length of every row so now we have to compute if row has pad or not, since 24-bit image is an odd numbers 3 bytes color depth we have to find the modulo of row by 4 like this (45 MOD 4=1), by looking to the modulo result if it's Zero that meaning image format depends on Padding and no gap found.

In our example the modulo give us the result 1, now we have to find out how many bytes of padding in this image.

Image_row_length=image_width * 3;
Image_row_length=15 * 3;
Padding= 4 –(image_row_length %4);
Padding=4 –(1);
Padding=(3);
Image_real _Row_Length= Image_row_length +Padding
Image_real _ Row _Length= 45+3;

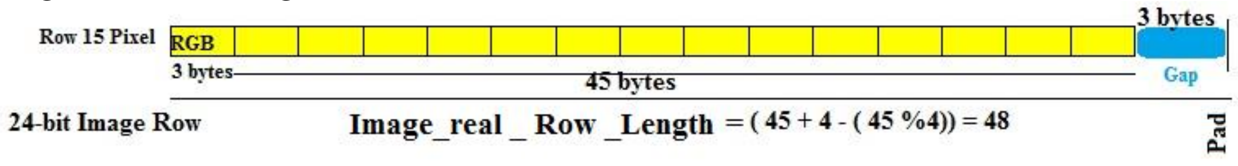


Figure 1:24-bit Image Row With Padding Gap

Padding result tell us we need to use 3 bytes to fill the gap between image row color info and the end of the each row, that gap has the value of zero[9].
So we can find The Image Length By this equation:
Image_Length=((Image_row_length + Padding) * Image_ column _length *3)

2.5. Another Example:

In this Example we will see an 24-bit bitmap image with area 12 * 12.
When we calculate the color pixel info it will be (12 * 3=36 bytes) in each row 300 bytes is the length of each row so now we have to compute if row has padding or not like this (36 MOD 4=0), the modulo result is Zero that meaning Padding will be Zero and No gap between image row color information and the end of row [9].
Image_real _Row_Length= Image_row_length +0.
Image_real _Row_Length= 36 +0 =36.

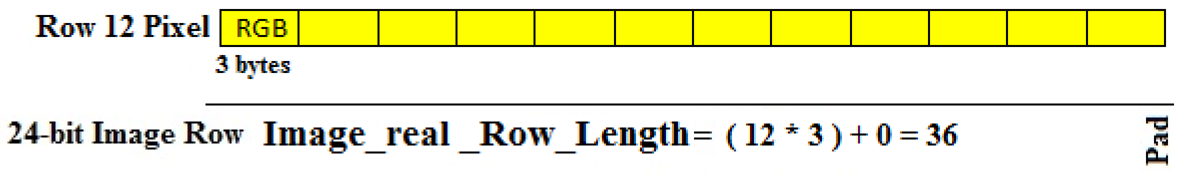


Figure 2:24-bit Image Row with No Padding Gap

The Image Length:
Image_Length=((Image_row_length) * Image_ column _length *3)

2.6. Algorithm to solve 24-bit Padding Problem:

```
//Find 24-bit Image Real Length Padding
 Image_row_length=image_width * 3;
if (image_row_length % 4 !=0){
Padding= 4 –( image_row_length %4);
}
else{
Padding=0;
}
Image_real _Length=(( Image_row_length +Padding) *Image_ column _length *3)
//End
```

After we have explained the problem of Padding in 24-bit image we can suggest a way to calculate the padding for many deferent format of Bitmap Image like 8-bit ,16-bit,24-bit and 32-bit depends on Bit Count :

2.7. Algorithm to solve 8-bit, 16-bit, 24-bit and 32-bit Padding:

```
Image_row_length=image_width *( image_Bit_Count / 8);
if (image_row_length % 4 !=0){
Padding= 4 –( image_row_length % 4);
}
else{
Padding=0;
}
Image_real _Length=(( Image_row_length +Padding) *Image_ column _length *3);
```

III. IMPLEMENTATION

In this section we will describe the implementation details of our algorithm . This algorithm has two class .Sub Image Search class and Convert Image To Matrix class, Sub Image Search class will implement the search operation to find the matching indexes, Convert Image To Matrix class has the responsibility to Read The Bitmap image file header and individualize the padding and File Buffer Size , After that it will convert the image RGB color data to a matrix of integer .

3.1.Test Media:

We have an image for Numbers from 1 to 20 ,this image is 24-bit bitmap image

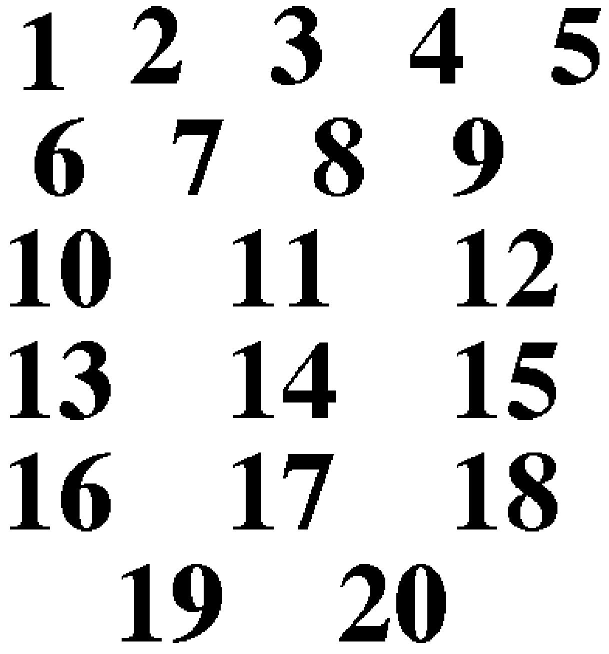


Figure 3: LargeImageNumbers.bmp

And the sub image could by any Number image, Let’s take number five as example

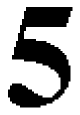


Figure 4 :SubImageN5.bmp

3.2 Java Class:

3.2.1 ConvertImageToMatrix Class:

Our first step is converting the images to matrix of integers , In order to create the matrix we must read the image header to get the file size ,Image width , Image height and the Bit count , After that we have to Check if it’s 24-bit Image or not [9].

There is a java code designed to read image header and get the Bitmap details [9], You can find this code on:

http://www.javaworld.com/javaworld/javatips/jw-javatip43.html

3.2.1.2 Read Bitmap Header:

In order to get the bitmap details we have to read the two parts of bitmap header BITMAPFILEHEADER(14-byte) and BITMAPINFOHEADER(40-byte), All the date in header stored in binary and that mean we have to use the binary operators {|,&,<} [6,7,9] .

- Get File Size:

```
  try {
FileInputStream fs = new FileInputStream(imageFile);
        int bflen = 14;           // 14 byte BITMAPFILEHEADER
             byte bf[] = new byte[bflen];
             fs.read(bf,0, bflen);      // Read the file header
        int nsize = (((int)bf[5] & 0xff) << 24)
             | (((int)bf[4] & 0xff) << 16)
             | (((int)bf[3] & 0xff) << 8)
             | (int)bf[2] & 0xff;
}
catch (Exception e) {
```

```
                    System.out.println(e.getMessage());
}
```

- Get image width , Height ,Bit count:

```
        try {
FileInputStream fs = new FileInputStream(imageFile);
            int bilen=40;               // 40-byte BITMAPINFOHEADER
                    byte bi[] = new byte[bilen];
                    fs.read(bi, 0, bilen);      // Read the bitmap header
            int nbisize = (((int)bi[3] & 0xff) << 24)
                    | (((int)bi[2] & 0xff) << 16)
                    | (((int)bi[1] & 0xff) << 8)
                    | (int)bi[0] & 0xff;
int nwidth = (((int)bi[7] & 0xff) << 24)
                    | (((int)bi[6] & 0xff) << 16)
                    | (((int)bi[5] & 0xff) << 8)
                    | (int)bi[4] & 0xff;
int nheight = (((int)bi[11] & 0xff) << 24)
                    | (((int)bi[10] & 0xff) << 16)
                    | (((int)bi[9] & 0xff) << 8)
                    | (int)bi[8] & 0xff;
int nbitcount = (((int)bi[15] & 0xff) << 8) | (int)bi[14] & 0xff;
 int nsizeimage = (((int)bi[23] & 0xff) << 24)
                    | (((int)bi[22] & 0xff) << 16)
                    | (((int)bi[21] & 0xff) << 8)
                    | (int)bi[20] & 0xff;
if (nbitcount != 24){
                      System.err.println("Use only 24bit color .bmp files");
                      System.exit(1);
                    }
}
catch (Exception e) {
                    System.out.println(e.getMessage());
}
```

In this step we checked if the image is 24-bit , If not the code will print error Message and exit.

3.2.1.3 Handling Padding problem and Buffer Size:

After we get the image details, We must handling Padding problem in order to set the Buffer Reader Size:

```
int padding =(nwidth * 3)%4;
                   int npad=0;
                   if(padding!=0){
                   npad = 4-(  (nwidth * 3)%4); // the Gap Size
                   }
                   else{
                       npad=0;// No Gap
                   }
                   //-----Set Buffer Size
                  int imageBufferSize=( nwidth + npad) * 3 * nheight;
```

3.2.1.3 Reading Image Data And Store it in matrix :

Now we can store image data into imageFileRGBData matrix depend on the buffer reader size imageBuffersize.

```
  int[ ][ ] imageFileRGBData = null; is the Matrix to representing the image pixels
 try {
   int ndata[] = new int [nheight * nwidth];
this.imageFileRGBData = new int[nheight][nwidth]; // [line][column]
  byte brgb[] = new byte [imageBufferSize];
  fs.read (brgb, 0, imageBufferSize); // Read the bitmap
int nindex = 0; // current postion in brgb
for (int j = 0; j < nheight; j++){ // by lines, from bottom to top
for (int i = 0; i < nwidth; i++){ // by columns
int rgbValue = (255 & 0xff) << 24
                                | (((int)brgb[nindex + 2] & 0xff) << 16)
                                | (((int)brgb[nindex + 1] & 0xff) << 8)
                                | (int)brgb[nindex] & 0xff;
                                ndata[nwidth * (nheight - j - 1) + i] = rgbValue;
                                this.imageFileRGBData[nheight - j - 1][i] = rgbValue;

                                nindex += 3;
                              }
                              nindex += npad;
                          }
}
catch (Exception ee) {
                          System.out.println(ee.getMessage());
}
```

Conversion step to both Image (the large image and sub image) will done as object of ConvertImageToMatrix class.

3.2.2 SubImageSearch Class:

In this class the first step is to create the two object of image matrix like this:

```
ConvertImageToMatrix  largeImage = new ConvertImageToMatrix("LargeImageNumbers.bmp");
ConvertImageToMatrix subImage = new ConvertImageToMatrix("SubImageN5.bmp");
int[][] large=largeImage.imageFileRGBData;//The Matrix of Large Image
int[][] sub=subImage.imageFileRGBData;//The Matrix of Sub Image
```

Now we have two matrix of integers, large matrix and sub matrix so we can now search for sub matrix in large matrix.To start search for matching we must get the first pixel value from the sub matrix as start point for search.

```
int pixelIndex = sub[0][0];//Search Start Point  is the First Pixel(x,y) From Sub Image
```

3.2.2.1 SubImageSearch Four Level Looping:

We have to start our search loop by row lines scan, In each row we will scan all column with nested loop to find matching point (x,y), And from the matching point we will search for any different between sub and large depend on sub image size ,that mean we need new two nested loop looking for different .

If we found the first different we have to leave the row looping index and start over with next index. In this case we must to use Branching Statement (Continue Label) [7] between row loop and column so we can exit the internal fourth loop to the main loop.

If different not found we will print out the point of matching (x,y) and continue searching for another matching point in case the sub image exists more than one time.

We used foundCount as matching Counter. And rowIndex to save matching Row index, and columnIndex to save matching Column index , Both have the initial value -1 to be sure index is out of large image.

```
public static int foundCount = 0;//counter for Matching if found
  public static int rowIndex = -1;//-1 is out of image
  public static int columnIndex = -1;//-1 is out of image
//-----------------Row lines  scan
 for ( int row = 0 ; row <= large.length - sub.length  ; row++ ){
  // Column  Scan
  continuescan://Column Scan Label
                for ( int column = 0 ; column < large[0].length - sub[0].length; column++ ){
                if (large[row][column] != pixelIndex) continue; // No Match With Start Point Go to Next
row
```

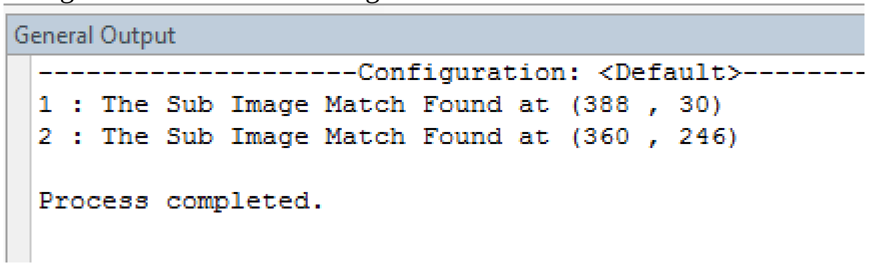
```
    // If there is matching With Start Point we have to Check All Pixel in sub with Pixel in large From
this Point
                for ( int r2 = 0 ; r2 < sub.length ; r2++ )
                for ( int c2 = 0 ; c2 < sub[0].length ; c2++ )
                // Now we are looking for Any different Pixel
           if ( large[row + r2][column + c2] != sub[r2][c2]){
                 continue continuescan;//there is no match Go to New Line (Next Y)
           }//end if Any different
            //-------Save The Matching Point Index
               rowIndex = column;
               columnIndex = row;
   //--------Check if Index is positive and the Point is In Image
   if ( rowIndex != -1 && columnIndex !=-1){

        System.out.println(++foundCount+" : The Sub Image Match Found at (" + rowIndex + " , " +
columnIndex+")");
                  }//end if positive
                 }//end for column
                }//end for row
         //--------Check if Index is negative and the Point is Out Image
           if( rowIndex == -1 || columnIndex ==-1){
            System.out.println("No Match Found");

              }//end if negative
```

3.2.2.2 The Test Result:

When we run the algorithm with the tow image the result will be :



General Output

```
--------------------Configuration: <Default>--------
1 : The Sub Image Match Found at (388 , 30)
2 : The Sub Image Match Found at (360 , 246)

Process completed.
```

Figure 5 :result of searching to the sub image

Now we can see the coordinate of matching point like this:

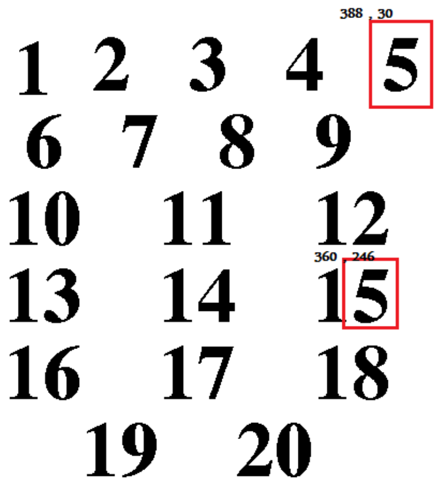


Figure 6 :Matching found in two Places

IV. LIMITATIONS

Our algorithm can work only with 24-bit bitmap image format. But as we know there are many ways to convert none 24-bit images to this format.

V. CONCLUSION

In this algorithm, we handled the problem of Searching for Sub Image and how to find out the Padding and real Buffer size when we are reading bitmap image .
Our proposal algorithm Can detect the matching Indexes in the large image for sub image however it occurs in the target large image .For the implementation of this algorithm we used the java language.

VI. Future Work

Our future work will be to create and development an image search engine which can search in image contains instead of Text or HTML meta data, This search engine can be helpful to find the target image from internet or any image storage, after that we will try to handle video frame search to lookup for still image. The next future research will be about Low-Pass and High-Pass Image filters.

REFERENCES

[1] Dr. William T. Verts, "An Essay on Endian Order" , April 19, 1996
[2] Dr. David Kirkby (G8WRB) , atlc "bmp format", http://atlc.sourceforge.net/bmp.html
[3] Microsoft MSDN , "Bitmap Storage”,
[4] Microsoft MSDN ,"Bitmap Header Types" ,
[5] John Miano, "Compressed Image File Formats: JPEG, PNG, GIF, XBM, BMP"
[6] (DUE 9816443) and the Hypermedia and Visualization Laboratory, "24-bit BMP FILES", Georgia State University, G. Scott Owen
[7] The Java Tutorials, ”Branching Statements",
[8] Charlap, David, "The BMP File Format: Part I," Dr. Dobb's Journal, March 01, 1995
[9] John D. Mitchell , "How to read 8- and 24-bit Microsoft Windows bitmaps in Java ", JavaWorld ,Dec 1, 1997

AUTHORS BIOGRAPHY

O. K. Khorsheed born in Baghdad in 1974. BSC computer science from al-mustansiriya University. Higher diploma degree from Informatics Institute of Higher Studies. Master degree from Jordan (Arab academic for banking and financial science) school of (IT) computer information system. Lecturer in Koya University since 2005 in school of engineering S\W engineering department.