INDIAN INSTITUTE OF TECHNOLOGY (INDIAN SCHOOL OF MINES), DHANBAD

DETECTION OF ROTATION OF CONVEX OBJECT USING 8 DIRECTIONAL CHAIN CODE

Under Mentorship of **Dr. Sushanta Mukhopadhyay**

Team Members:

1) Utkarsh Agarwal
2) Madhur Chauhan
3) Abhishek Tyagi

Introduction

IMAGE PROCESSING

Image Processing is a technique to enhance raw images received from cameras/sensors placed on satellites, space probes and aircrafts or pictures taken in normal day-today life for various applications.

Various techniques have been developed in Image Processing during the last four to five decades. Most of the techniques are developed for enhancing images obtained from unmanned space-crafts, space probes and military reconnaissance flights. Image Processing systems are becoming popular due to easy availability of powerful personnel computers, large size memory devices, graphics softwares etc.

IMAGE COMPRESSION

Image compression is the process of encoding or converting an image file in such a way that it consumes less space than the original file. The best image quality at a given bit-rate (or compression rate) is the main goal of image compression, however, there are other important properties of image compression schemes:

Scalability generally refers to a quality reduction achieved by manipulation of the bitstream or file (without decompression and re-compression). Other names for scalability are *progressive coding* or *embedded bitstreams*. Despite its contrary nature, scalability also may be found in lossless codecs, usually in form of coarse-to-fine pixel scans. Scalability is especially useful for previewing images while downloading them (e.g., in a web browser) or for providing variable quality access to e.g., databases. There are several types of scalability:

- **Quality progressive or layer progressive**: The bitstream successively refines the reconstructed image.
- **Resolution progressive**: First encode a lower image resolution then encode the difference to higher resolutions.
- **Component progressive**: First encode grey then color.

Region of interest coding- Certain parts of the image are encoded with higher quality than others. This may be combined with scalability (encode these parts first, others later).

Meta information- Compressed data may contain information about the image which may be used to categorize, search, or browse images. Such information may include color and texture statistics, small preview images, and author or copyright information.

Processing power- Compression algorithms require different amounts of processing power to encode and decode. Some high compression algorithms require high processing power.

The quality of a compression method often is measured by the Peak signal-to-noise ratio. It measures the amount of noise introduced through a lossy compression of the image, however, the subjective judgment of the viewer also is regarded as an important measure, perhaps, being the most important measure.

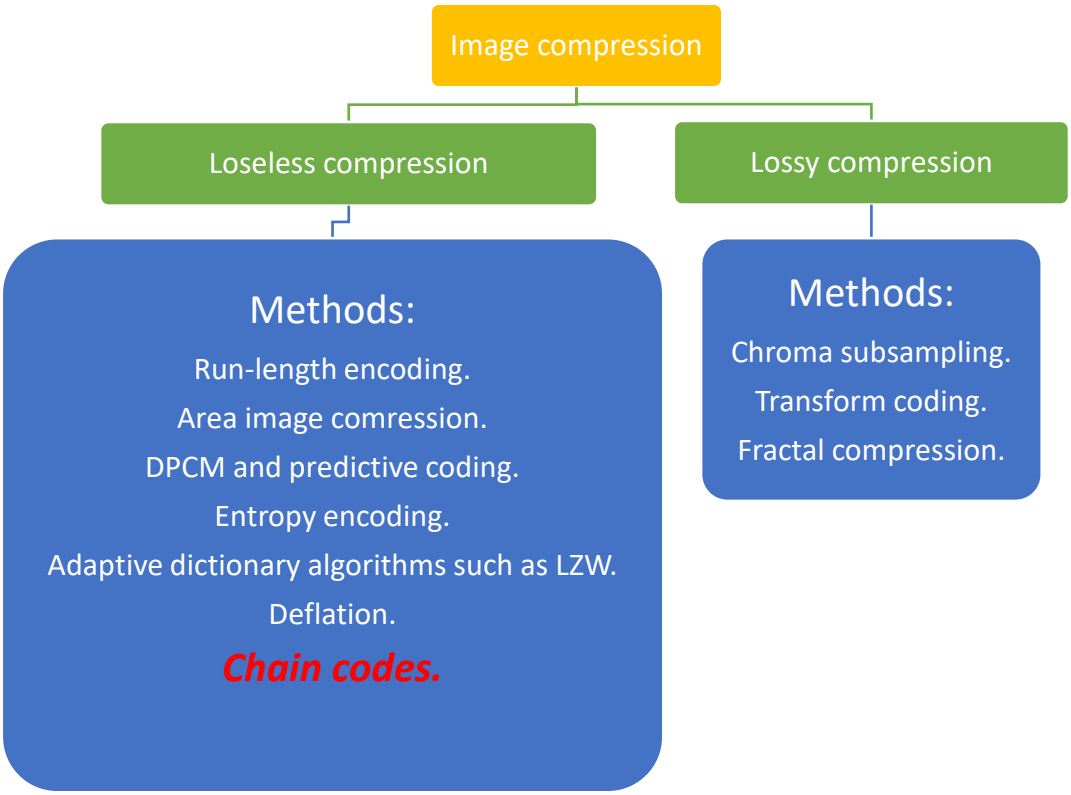


Fig.1. Classification of compression methods

Lossless compression:

Lossless compression is a class of data compression algorithms that allows the original data to be perfectly reconstructed from the compressed data.

Lossless compression is preferred for archival purposes and often for medical imaging, technical drawings, clip art, or comics.

Lossless compression is used in cases where it is important that the original and the decompressed data be identical, or where deviations from the original data would be unfavourable. Typical examples are executable programs, text documents, and source code. Some image file formats, like PNG or GIF, use only lossless compression.

Lossy compression:

In information technology, **lossy compression** or **irreversible compression** is the class of data encoding methods that uses inexact approximations and partial data discarding to represent the content. These techniques are used to reduce data size for storage, handling, and transmitting content. The amount of data reduction possible using lossy compression is much higher than through lossless techniques.

Lossy compression is most commonly used to compress multimedia data (audio, video, and images), especially in applications such as streaming media and internet telephony.

Lossy compression methods, especially when used at low bit rates, introduce compression artifacts. Lossy methods are especially suitable for natural images such as photographs in applications where minor (sometimes imperceptible) loss of fidelity is acceptable to achieve a substantial reduction in bit rate.

Chain code:

- A **chain code** is a lossless compression algorithm for monochrome images. The basic principle of chain codes is to separately encode each connected component, or "blob", in the image.
- For each such region, a point on the boundary is selected and its coordinates are transmitted. The encoder then moves along the boundary of the region and, at each step, transmits a symbol representing the direction of this movement.
- This continues until the encoder returns to the starting position, at which point the blob has been completely described, and encoding continues with the next blob in the image.
- This encoding method is particularly effective for images consisting of a reasonably small number of large connected components.

Variations

Some popular chain codes include-

1) The Freeman Chain Code of Eight Directions (FCCE).
2) Vertex Chain Code (VCC).
3) Three Orthogonal symbol chain code (3OT).
4) Directional Freeman Chain Code of Eight Directions (DFCCE).
5) Unsigned Manhattan Chain Code (UMCC).

In particular, FCCE, VCC, 3OT and DFCCE can be transformed from one to another.

Applications:

1) Digit Recognition.
2) Contour Detection.
3) Face Recognition.
4) Shape Recognition & Matching.

Chain Code of Eight Directions:

There are of two types, of this, one is ***Absolute chain code*** and the other is ***Relative chain code***.

Absolute Chain code:

Absolute chain codes do not need to be fixed to anything as large as the whole earth, however. We can also fix the direction points to something smaller (like a piece of paper for example, or your computer monitor). If we do that, it is natural to make the top of the paper (or monitor) "N", the bottom "S", and so on. Once we've designated the official "North" end of the paper, then it doesn't matter which direction we are facing as we look at the paper. "N" will always be in a fixed (absolute) direction in relation to the dimensions of the paper. **We can also use numbers in chain code as in fig. 3.**

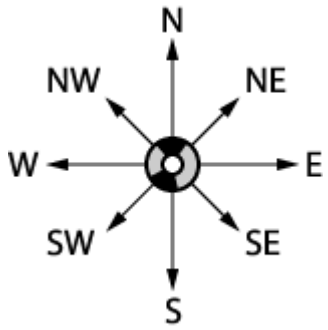


Fig. 2.

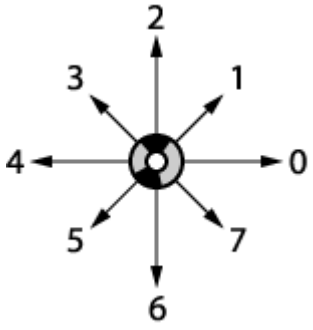


Fig. 3.

Relative Chain code:

There will be times when absolute chain codes won't do the job. Either, because no absolute point of reference is available or because absolute chain codes are not capable of doing the job that needs to be done. In such cases, using a relative perspective might be preferable.

It is possible to base an eight-directional coding system on relative directions. Each of the eight directions will be defined **in relation to** a moving perspective. Think of it this way. Imagine that you are driving a car around the perimeter of the object, marking a line behind you as you go. From any given point you have eight options. You can continue to go forward ("F") in the same direction that you have been going, you can go backward ("B") in the opposite direction, you can turn to the left ("L") or to the right ("R"). These are the four main points of the compass. The other four directions are derived from the four basic ones, as can be seen in the compass to the right. **We can also use numbers in chain code as in fig. 5.**

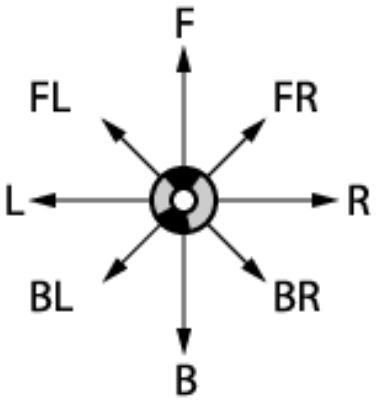


Fig. 4.

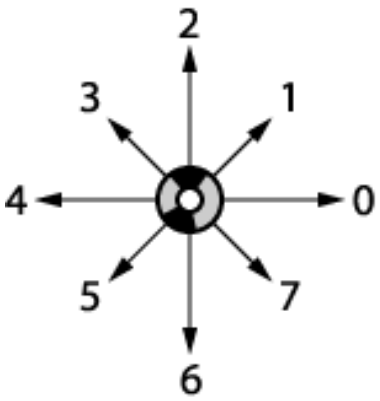


Fig. 5.

Comparison

Absolute Chain code & Relative Chain Code:

One of the strengths of the *absolute chain code* is that it carries information not only about the size and shape of the object but also about its rotation. Suppose we want to encode the information about the size and shape of the triangle, but also about the position of the figure on the page. An absolute chain code can distinguish between the triangle on the left (below) and the one on the right, that has been rotated 180 degrees.

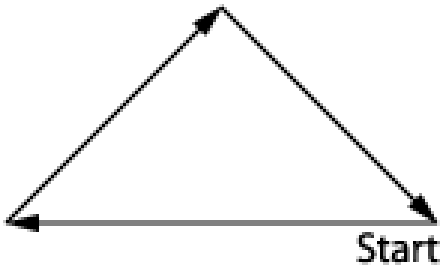


Fig. 6.

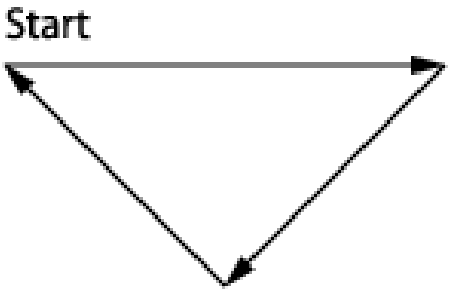


Fig. 7.

The absolute chain code for the triangle in fig. 6. is: 4,1,7 and for the one in fig. 7, it is: 0, 5, 3. They don't share even one number in common. This is because the rotation of the triangle as well as the size and the shape are being represented in the code. So, if we want to build a machine that can preserve the orientation of an object (its rotation) as well as its shape and size, then the machine would need to represent the object with an absolute chain code.

Relative chain codes do not behave in the same way. The relative code for the first triangle is 7, 7, 0. The code for the triangle on the right will also be 7, 7, 0 if the starting point remains the same (as shown by the words "start" above). Things only change slightly if we **move the starting point to the right-angled corner** of the triangle. The code then becomes**: 0, 7, 7** regardless of whether the base of the triangle is pointed up or down.

The similarity in these chain codes has a beneficial feature, one that our machine can take advantage of. Since the numbers that are used are the same, a computer can be programmed to compare the chain codes of two different objects and determine if they are the same size and shape.

If we wanted to give a machine the ability to compare chain codes, a fairly simple program can be written that can determine whether or not two chain codes are exactly the same. This would give the machine the ability to judge that two shapes were the same, even if they were rotated quite differently.

Image type used:

This algorithm is used to detect a convex object in an image and compare two images whether they contain same object, the images used are binary images. Binary images by definition consist of only black and white pixels. Black is taken as '0' and white as '1'.

Implementation:

In C language using ANSI standard of C99, implemented using gcc compiler version 6.3.3 provided by GNU.

Algorithm:

Input:

The algorithm takes input of two .pgm files with binary entries and equal boundary length to be processed by algorithm.

Output:

There will be total of four output to files – two files containing the absolute chain code of both images and two files containing the relative chain code. Output to console shows whether two images are equivalent (in terms of rotation).

Assumptions:

Assume black pixels are the zeros and white pixels are ones and that the image(binary) input is a rectangular array or matrix containing N by M elements with either zero or one value as input.

The algorithm works in four parts for all zero (black) pixels it encounters.

P3	P2	P1
P4	**P**	P0
P5	P6	P7

Steps:

1. Finding boundary in image in absolute terms and outputting the result in a file.

   A. Any pixel will have 8 neighbours discounting the edges and corners of image, for this exception a border of 1 pixel length containing white(0) is taken.
   B. Now the first encountered black pixel is marked and named top-left. Similarly, for top-right, bottom-left and bottom-right, taking all cases in consideration as we move downwards.
   C. Traversing and outputting the value of directions (0 to 7) from top-left to bottom-left.
   D. Traversing and outputting the value of directions (0 to 7) from bottom-left to bottom-right.
   E. Traversing and outputting the value of directions (0 to 7) from bottom-right to top-right.
   F. Traversing and outputting the value of directions (0 to 7) from top-right to top-left.

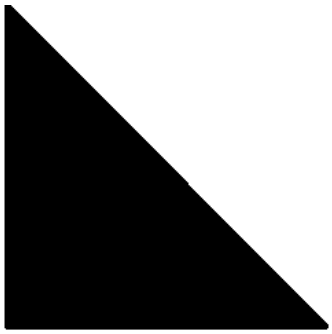
2. Converting the absolute edge traversal directions into relative directions.

   A. Relative value of edges depends on the current entry and the previous entry in the file, taking first direction as reference value.
   B. All the different cases are compared and relative direction values are input into another file.

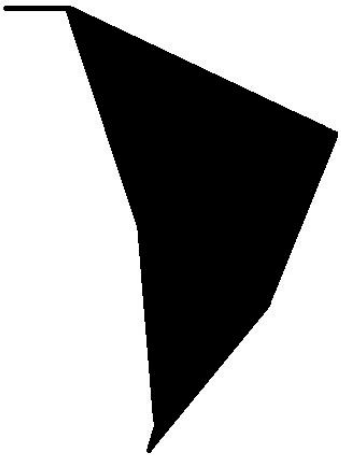
3. For given two image files after the above two steps are completed, a cyclic comparison between the two relative arrays of directions is done to compare whether or not the images contain the same object.

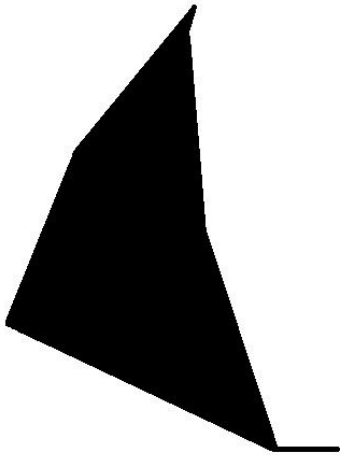
Sample Output:

1. Successfully detected the rotations of the triangle below.

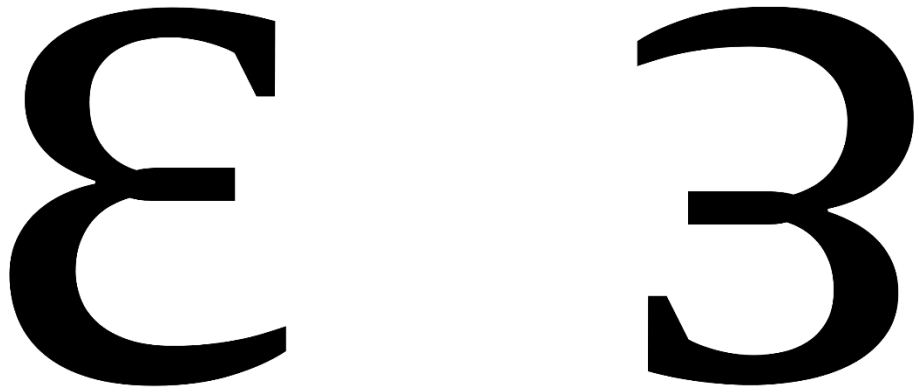


2. Successfully detected the following rotation of Capricorn constellation.





3. Successfully detected rotation of ‘epsilon’.



Advantages:

1) The algorithm is very fast in execution.
2) The algorithm is simple and thus easy to implement.
3) Can be expanded and afterwards used for various other applications.

Disadvantages & Scope for improvement:

1) The algorithm is to be improved for concave and other random shapes.
2) For a discontinuous shape the algorithm has to be improved upon. Afterwards, will find application in fingerprint recognition and other such procedures.

Implemented Code:

```
#include<stdio.h>
#include<stdlib.h>
#include<stdbool.h>
#define black false
#define white true

int size(FILE *a){
  int temp,ct=0;
  while(fscanf(a," %d",&temp)!=EOF)
    ct++;
  return ct;
}
void absolute(FILE * , FILE *);
bool check(FILE *, FILE *,int,int);
void torelative(FILE *f, FILE *fw,int sz);

int main()
{
  FILE *f, *f1;
  int size1,size2;
  //FIRST FILE
      f=fopen("1.pgm", "r");
  f1 = fopen("abs1.txt", "w");
  absolute(f,f1);
  fclose(f);
  fclose(f1);
  f=fopen("abs1.txt", "r");
  size1=size(f);
  f=fopen("abs1.txt", "r");
  f1=fopen("rel1.txt","w");
  torelative(f,f1,size1);
  fclose(f);
  fclose(f1);
  //SECOND FILE
  f=fopen("2.pgm", "r");
  f1 = fopen("abs2.txt", "w");
  absolute(f,f1);
  fclose(f);
  fclose(f1);
  f=fopen("abs2.txt", "r");
  size2=size(f);
  f=fopen("abs2.txt", "r");
  f1=fopen("rel2.txt","w");
  torelative(f,f1,size2);
  fclose(f);
  fclose(f1);
  //CHECK
  f=fopen("rel1.txt","r");
```

```
  f1=fopen("rel2.txt","r");
  bool ans=false;
  if(size1==size2)
    ans = check(f,f1,size1,size2);
  if(ans)
  {
    printf("The objects are same\n");
  }
  else
  {
    printf("The objects are different\n");
  }
  return 0;
}

void absolute(FILE *f, FILE *f1)
{
  char format[3];
  fscanf(f, "%s",format);
  int r, c, b, ulx, uly, urx, ury, llx, lly, lrx, lry, count1 =0,
count2=0;
  fscanf(f, "%d %d %d", &r, &c, &b);
  bool mat[r+2][c+2];
  for(int i=0; i<=r+1; i++)
  {
          for(int j=0; j<=c+1; j++)
    {
      mat[i][j]=true;
    }
  }
     for(int i=1; i<=r; i++)
  {
          for(int j=1; j<=c; j++)
    {
                int temp;
     fscanf(f,"%d", &temp);
     mat[i][j] = (temp==255?true:false);
          }
     }
     for(int i=1; i<=r && count1!=1; i++)
  {
          for(int j=1; j<=c; j++)
    {
                if(count1==0&&mat[i][j]==0)
     {
        uly = i;
       ulx = j;
       count1 =1;
       break;
                }
          }
```

```
          for(int j=c; j>0; j--)
  {
                if(count2==0&&mat[i][j]==0)
    {
     ury = i;
      urx = j;
      count2 =1;
      break;
                }
          }
    }
    count1=0;
count2=0;
    for(int i=r; i>0 && count1!=1; i--)
{
          for(int j=1; j<=c; j++)
  {
                if(count1==0 && mat[i][j]==0)
    {
       lly = i;
      llx = j;
      count1 =1;
      break;
                }
          }
          for(int j=c; j>0; j--)
  {
                if(count2==0 && mat[i][j]==0)
    {
     lry = i;
      lrx = j;
      count2 =1;
      break;
                }
          }
    }
    //UPPER TO LOWER
    int px=ulx,py=uly;
    for(int i=uly+1;i<=lly;++i)
{
          int tx;
  for(int j=0;;++j)
  {
    if(mat[i][j]==black)
    {
      tx=j;
      break;
    }
  }
          if(abs(tx-px)<=1)
  {
```

```
                if(tx<px)
      fprintf(f1, "5\n");
                else if(tx==px)
      fprintf(f1, "6\n");
                else
      fprintf(f1, "7\n");
          }
          else
   {
                if(mat[px][i+1]==black)
     {
                      fprintf(f1, "6\n");
                      if(tx>px)
       {
         for(int tt=0;tt<tx-px;++tt)
          fprintf(f1, "0\n");
        }
                    else
       {
         for(int tt=0;tt<px-tx;++tt)
           fprintf(f1, "4\n");
       }
                }
                else
     {
                      if(tx>px)
       {
          fprintf(f1, "7\n");
          for(int tt=1;tt<tx-px;++tt)
           fprintf(f1, "0\n");
       }
                    else
       {
         fprintf(f1, "5\n");
         for(int tt=1;tt<px-tx;++tt)
         fprintf(f1, "4\n");
       }
            }
     }
 px=tx;
}
    //FROm lower left to lower right
    for(int i= llx;i<lrx;i++ )
  fprintf(f1, "0\n");
    //LOWER TO UPPER
    px=lrx;
py=lry;
    for(int i=lry-1;i>=ury;--i)
{
          int tx;
  for(int j=c;;--j)
```

```
 if(mat[i][j]==black)
 {
   tx=j;
   break;
 }
       if(abs(tx-px)<=1)
{
            if(tx>px)
  fprintf(f1, "1\n");
            else if(tx==px)
  fprintf(f1, "2\n");
            else
  fprintf(f1, "3\n");
       }
       else
{
            if(mat[px][i-1]==black)
  {
                 fprintf(f1, "2\n");
                 if(tx<px)
    {
      for(int tt=0;tt<px-tx;++tt)
        fprintf(f1, "4\n");
    }
                 else
    {
      for(int tt=0;tt<tx-px;++tt)
        fprintf(f1, "0\n");
    }
            }
            else
  {
                 if(px>tx)
    {
      fprintf(f1, "3\n");
      for(int tt=1;tt<tx-px;++tt)
        fprintf(f1, "4\n");
    }
                 else
    {
      fprintf(f1, "1\n");
      for(int tt=1;tt<tx-px;++tt)
        fprintf(f1, "0\n");
    }
            }
       }
       px=tx;
  }
  //FROM upper right to upper left
  for(int j=urx; j>ulx; j--)
fprintf(f1, "4\n");
```

```
}
bool check(FILE *a, FILE *b,int size1,int size2){   //ASSUMES FRESH
FILES AND SIZES
  int arr1[size1], arr2[size2];
  for(int i=0;i<size1;++i)
    fscanf(a," %d",&arr1[i]);
  for(int i=0;i<size2;++i)
    fscanf(b," %d",&arr2[i]);
  for(int i=0;i<size1;++i)
  {
    if(arr1[i]==arr2[0])
    {
      int j=0;
      bool stop=false;
      for(;j<size2;++j)
      {
        if(arr2[j]!=arr1[(j+i)%size1])
        {
          stop=true;
          break;
        }
      }
      if(!stop) return true;
    }
  }
  return false;
}
void torelative(FILE *f,FILE *f1,int size)
{
     int v[size], temp=0;
  while(fscanf(f," %d\n",&v[temp])!=EOF)
    temp++;
  int p=v[temp-1],c=v[0];
  for(int i=0;i<size; ++i)
  {
    int c=v[i];
    switch(c)
    {
      case 0:
      {
          switch(p)
          {
            case 0: {
              fprintf(f1,"2 \n");
              break;
            }
            case 1: {
              fprintf(f1,"3 \n");
              break;
            }
            case 2: {
```

```
          fprintf(f1,"4 \n");
          break;
        }
        case 3: {
          fprintf(f1,"5 \n");
          break;
        }
        case 4: {
          fprintf(f1,"6 \n");
          break;
        }
        case 5: {
          fprintf(f1,"7 \n");
          break;
        }
        case 6: {
          fprintf(f1,"0 \n");
          break;
        }
        case 7: {
          fprintf(f1,"1 \n");
          break;
        }
      }

  } break;
case 1 :{
    switch(p){
      case 0: fprintf(f1,"1 \n");
        break;
      case 1: fprintf(f1,"2 \n");
        break;
      case 2: fprintf(f1,"3 \n");
        break;
      case 3: fprintf(f1,"4 \n");
        break;
      case 4: fprintf(f1,"5 \n");
        break;
      case 5: fprintf(f1,"6 \n");
        break;
      case 6: fprintf(f1,"7 \n");
        break;
      case 7: fprintf(f1,"0 \n");
        break;
    }
  } break;
case 2 :
{
    switch(p)
    {
      case 0: fprintf(f1,"0 \n");
```

```
        break;
      case 1: fprintf(f1,"1 \n");
        break;
      case 2: fprintf(f1,"2 \n");
        break;
      case 3: fprintf(f1,"3 \n");
        break;
      case 4: fprintf(f1,"4 \n");
        break;
      case 5: fprintf(f1,"5 \n");
        break;
      case 6: fprintf(f1,"6 \n");
        break;
      case 7: fprintf(f1,"7 \n");
        break;
    }

  } break;
case 3 :
{
  switch(p)
    {
      case 0: fprintf(f1,"7 \n");
        break;
      case 1: fprintf(f1,"0 \n");
        break;
      case 2: fprintf(f1,"1 \n");
        break;
      case 3: fprintf(f1,"2 \n");
        break;
      case 4: fprintf(f1,"3 \n");
        break;
      case 5: fprintf(f1,"4 \n");
        break;
      case 6: fprintf(f1,"5 \n");
        break;
      case 7: fprintf(f1,"6 \n");
        break;
    }

  } break;
case 4 :{
  switch(p)
    {
      case 0: fprintf(f1,"6 \n");
        break;
      case 1: fprintf(f1,"7 \n");
        break;
      case 2: fprintf(f1,"0 \n");
        break;
      case 3: fprintf(f1,"1 \n");
```

```
          break;
        case 4: fprintf(f1,"2 \n");
          break;
        case 5: fprintf(f1,"3 \n");
          break;
        case 6: fprintf(f1,"4 \n");
          break;
        case 7: fprintf(f1,"5 \n");
          break;
      }

    } break;
  case 5 :{
    switch(p)
      {
        case 0: fprintf(f1,"5 \n");
          break;
        case 1: fprintf(f1,"6 \n");
          break;
        case 2: fprintf(f1,"7 \n");
          break;
        case 3: fprintf(f1,"0 \n");
          break;
        case 4: fprintf(f1,"1 \n");
          break;
        case 5: fprintf(f1,"2 \n");
          break;
        case 6: fprintf(f1,"3 \n");
          break;
        case 7: fprintf(f1,"4 \n");
          break;
      }

    } break;
  case 6 :{
    switch(p)
      {
        case 0: fprintf(f1,"4 \n");
          break;
        case 1: fprintf(f1,"5 \n");
          break;
        case 2: fprintf(f1,"6 \n");
          break;
        case 3: fprintf(f1,"7 \n");
          break;
        case 4: fprintf(f1,"0 \n");
          break;
        case 5: fprintf(f1,"1 \n");
          break;
        case 6: fprintf(f1,"2 \n");
          break;
```

```
            case 7: fprintf(f1,"3 \n");
              break;
          }

        } break;
      case 7 :{
        switch(p)
          {
            case 0: fprintf(f1,"3 \n");
              break;
            case 1: fprintf(f1,"4 \n");
              break;
            case 2: fprintf(f1,"5 \n");
              break;
            case 3: fprintf(f1,"6 \n");
              break;
            case 4: fprintf(f1,"7 \n");
              break;
            case 5: fprintf(f1,"0 \n");
              break;
            case 6: fprintf(f1,"1 \n");
              break;
            case 7: fprintf(f1,"2 \n");
              break;
          }

        } break;
    }
    p=c;
  }
}
```

Reference:

1) ***Lecture on chain codes by- Fredrik Georgsson, Umeå University.***
https://www8.cs.umu.se/kurser/TDBC30/VT04/material/lect13_kap_11.pdf

2) ***Encoding of arbitrary geometric configuration by- H. Freeman.***
http://electronicimaging.spiedigitallibrary.org/article.aspx?articleid=1838782

3) ***Digital Picture processing by- A. Rosenfeld, A. C. Kak.***
http://dl.acm.org/citation.cfm?id=578095

4) ***A new chain code and pattern recognition by- E. Briebesca.***
http://www.sciencedirect.com/science/article/pii/S0031320398001320

5) ***Introduction to chain codes, by- Andrew L. Anderson.***
http://www.mind.ilstu.edu/curriculum/chain_codes_intro/chain_codes_intro.php