10-31-1992

Design and implementation of two text recognition algorithms

Madhumathi Yendamuri
New Jersey Institute of Technology

Copyright Warning & Restrictions

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be "used for any purpose other than private study, scholarship, or research." If a, user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of "fair use" that user may be liable for copyright infringement,

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

Please Note: The author retains the copyright while the New Jersey Institute of Technology reserves the right to distribute this thesis or dissertation

Printing note: If you do not wish to print this page, then select "Pages from: first page # to: last page #" on the print dialog screen

ABSTRACT

Design and Implementation of Two Text Recognition Algorithms

by

Madhumathi Yendamuri

This report presents two algorithms for text recognition. One is a neural-based orthogonal vector with pseudo-inverse approach for pattern recognition. A method to generate N orthogonal vectors for an N-neuron network is also presented. This approach converges the input to the corresponding orthogonal vector representing the prototype vector. This approach can restore an image to the original image and thus has error recovery capacility. Also, the concept of sub-networking is applied to this approach to enhance the memory capacity of the neural network. This concept drastically increases the memory capacity of the network and also causes a reduction of the convergence time to stable states. Another approach is to use the Levenshtein algorithm for string matching following the application of rules to recognise a given character. Both these methods are discussed and the results are presented.

DESIGN AND IMPLEMENTATION OF TWO TEXT RECOGNITION ALGORITHMS

by
Madhumathi Yendamuri

A Thesis
Submitted to the Faculty of New Jersey
Institute of Technology in Partial Fulfillment of the Requirements
for the Degree of Master of Science
Department of Computer and Information Sciences
October 1992

APPROVAL PAGE

Design and Implementation of Two Text Recognition Algorithms

by

Madhumathi Yendamuri

Dr. David T. Wang, Thesis Advisor
Assistant Professor of Computer and Information Sciences,
New Jersey Institute of Technology

ACKNOWLEDGEMENT

The author wishes to express her sincere gratitude to Dr. David T. Wang, thesis advisor for his guidance and support throughout this research.

The author wishes to take this opportunity to thank Dr. Daniel Y. Chao, for his guidance and helpful suggestions in this research.

Special thanks to Tzung-i Li, Kwon Churhee and Kamal Dheri for their help as members of the research team.

This thesis is dedicated to Daddy
for his love and encouragement

BIOGRAPHICAL SKETCH

Author: Madhumathi Yendamuri

Degree: Master of Science in Computer and Information Sciences

Date: October, 1992

Date of Birth:

Place of Birth:

Undergraduate and Graduate Education:

- Master of Science in Computer and Information Sciences, New Jersey Institute of Technology, Newark, NJ, 1992
- Master of Science (Technology) in Computer Science, Birla Institute of Technology, Pilani, India, 1990

Major: Computer and Information Sciences

TABLE OF CONTENTS

CHAPTER 1
INTRODUCTION

There are two methods for pattern recognition --- syntactic and statistical. One of the syntactic approaches is string matching. This report discusses the use of Levenshtein distance algorithm for string matching; and also a neural-network based algorithm for the same.

1.1 Levenshtein Algorithm

Structural representation used in pattern recognition most commonly are the string, trees, graphs and arrays. With respect of computational complexity, strings are very efficient since similarity measures between strings can be computed very fast. However, string are limited in their representational power. At the other extreme, graphs are most powerful approach to structural pattern recognition. But, graph matching is conceptually rather complicated and expensive with respect to computational cost.

Using the Levenshtein algorithm for string matching with a combination of recognition rules is found to yield 100% recognition rate. The experiment was conducted on 2600 real samples of lower and upper case characters.

1.2 Neural-based approach

Using the neural-based approach has yielded 86% recognition rate in the first implementation. Further study can be made on improving this recognition rate using a rule generator for character recognition. The neural-based algorithm used higher-order multi-valued subnetworking of Hopfield network, which is one of the

most successful conventional neural networks. It allows the retrieval of auto-associative patterns, given an input pattern.

Subsequent chapters describe both the algorithm in more detail. The experimental results of each method are included in chapters 7 and 9. The program listings are included in Appendix A and B.

CHAPTER 2
GENERATION OF N ORTHOGONAL STATE VECTORS FOR N-NEURON NETWORKS

2.1 Introduction

Consider a network of N neurons, the state vector of the system is **u**, which is a tuple of binary states of each neuron.

In the Hopfield-Little model[9] of associative memory, the neurons have binary states with threshold value assumed to be zero. The network is fully connected and symmetric, on which Hebb's rule applies. If the input vector is partially corrupted to some extent, the network can still converge to the correct prototype. Therefore, this kind of network can do robust pattern recognition. In addition, it is crucial to determine the prototypes to be stored in the network.

In order to achieve good error recover, it is desirable that the orthogonal prototype vectors are equally spaced apart in terms of the *Hamming Distance*. Further, this Hamming Distance should be as large as possible. For a N-neuron network, the largest Hamming Distance between any two orthogonal vectors is N/2. A neural network constructed from such a set of prototype vectors demonstrates a better retrieval capability, and this, in turn, also results in higher memory capacity.

2.2 Direct Method for N Orthogonal Vectors of N elements

There are various approaches to find orthogonal vectors. One is to usesinusoidal functions and another non-sinusoidal series. Of the latter, there are some well-known funcitons: Walsh function, Harr function and Rememacher function. Both Walsh- and Harr-function can form a complete orthogonal set while Rememacher

function provides another set of two-level orthogonal functions which are incomplete but true subset of the Walsh function. In this section, a direct approach is discussed.

In the recursive approach, 2N orthogonal vectors for a 2N-neuron network can be constructed from N orthogonal vectors of a N-neuron network and these N orthogonal vectors, in turn, can be constructed in a similar manner from the N/2 orthogonal vectors of a neural network with N/2 neuron, etc. The disadvantage of this approach is the necessity of constructing orthogonal vectors using the recursive successive-doubling approach. The proposed direct approached, to be studied in this section, can circumvent this drawback by calculating the N elements of each of the N orthogonal vectors from a formula. The resulting orthogonal vectors are identical to those constructed using the recursive successive-doubling approach.

The direct approach is to construct N orthogonal vectors of a N-neuron network based on N/2 orthogonal vectors for a N/2-neuron network. The first n/2 orthogonal vectors of a N-neuron network are constructed in the same manner as that for a n/2 neuron network except that the N elements of each of the former can be decomposed into N/2 blocks of

$$\begin{bmatrix} 1 \\ 1 \end{bmatrix} \quad \text{or} \quad \begin{bmatrix} -1 \\ -1 \end{bmatrix}$$

With the substitutions 1 for

$$\begin{bmatrix} 1 \\ 1 \end{bmatrix} \quad \text{and } -1 \text{ for} \quad \begin{bmatrix} -1 \\ -1 \end{bmatrix}$$

the first N/2 orthogonal vectors of N elements become N/2 orthogonal vectors of N/2 elements. Applying the division procedure further, the next N/2 orthogonal

vectors of N elements can be constructed in exactly the same manner as that of N/2 elements and it is obvious that the resulting vectors are orthogonal to each other.

CHAPTER 3
COMBINED ORTHOGONAL VECTOR AND PSEUDO-INVERSE APPROACH FOR CHARACTER RECOGNITION

3.1 Introduction

The neural network of the associative memory type is extensively applied in the field of classification and pattern recognition. There are two kinds of associative memory[4]; autoassociative and heteroassociative. Here, the former is discussed.

In the autoassociative memory neural network system, certain number of binary prototype vectors are integrated into the structure of the system. Then a vector is input to the system for recognition. Starting from this vector, the network system can reach either a cycle or a fixed point.

This chapter presents a combined orthogonal vector and pseudo-inverse approach to achieve robust pattern recognition. The upper and lower case letters of the English alphabet are used as an example to implement the proposed approach. The theory of the combined orthogonal vector and pseudo-inverse method is studied in Section 2.

3.2 Theory of the Recognition Algorithm

In this section, the proposed algorithm is presented. The Hamming distance between two binary vectors is defined.

3.2.1 Algorithm Statement

The proposed algorithm is stated as follows:

Phase I. Training Stage

1. Generate p orthogonal vectors $r^{(i)}$, i = 1,..., p and assign one such vector to each of the p prototype vectors $t^{(i)}$, i = 1, ..., p.

2. Construct matrix T whose columns are the protoype vectors.

3. Generate the pseudo-inverse T^+, of matrix T.

4. Construct matrix R with the orthogonal vectors generated in step 1 as its columns.

5. Construct matrix $P = RT^+$.

Phase II. Recognition Stage

1. Use matrix P as the operator to perform $a_1 = Pa_0$ is an input pattern vector to be recognized.

2. Computer $a_2 (= Pa_1)$, ... $a_X = Pa_{X-1}$.

3. Find out an orthogonal vector r_j, $1 <= i <= p$, whose

Hamming distance with vector a_X is the minimum.

4. If this Hamming distance is within the tolerance, the

corresponding prototype is the answer to the recognition, otherwise, the input pattern vector is not recognized.

3.2.2 Hamming distance

The Hamming distance of two binary-valued vectors with elements in {1,-1} is defined as follows. We first EXCLUSIVE OR the two vectors, then count the number of 1-bits in the result. This number of 1-bits is then divided by 2 to obtain the Hamming distance.

3.2.3 The Use of Pseudo-Inverse Matrix

Given a prototype matrix T defined as follows:

$$T = [t^{(1)}, t^{(2)}, ..., t^{(p)}], \quad p < N$$

where $t^{(i)}$,(i = 1,...,p) are the N x 1 prototype vectors,

the pseudo-inverse of matrix A is given by:

$$T^{+} = (T^{T}T)^{-1}T^{T}$$

where superscript T means matrix transpose.

3.3 Conclusions

A neural-based orthogonal vector with pseudo-inverse approach for pattern recognition has been presented. A method to generate N orthogonal vectors for a N-neuron network is also proposed. Using approach, patterns with 20% error can be recognized, using English capital and lower case letters for experiment. The robustness of this approach is obvious.

CHAPTER 4
ENHANCEMENT OF MEMORY CAPACITY OF NETWORKS USING SUB-NETWORKING

4.1 Concept of Subnetworking

Subnetworking is to break a large neural network into a number of independent subnetworks provided that the memory pattern can be partitioned into a number of independent subpatterns which is often the case. There is an output neuron O_k connected to each neuron i of a subnetwork with synaptic weight a_{Ki}. These weights are such that different stable states of the subnetwork assume different output values. The following example illustrates how this increases the memory capacity. Consider a 75 neuron network. Suppose each subfeature requires 25 neurons and there are three sub-features corresponding to three subneural networks. Each subnetwork has 25 neurons and an output neuron. The state of the network can be represented as a vector (O_1,O_2,O_3). For instance, (1,3,5) indicates a pattern consisting of the first subpattern of subnetwork 1, the third subpattern of subnetwork 2 and the fifth subpattern of subnetwork 3.

If each subnetwork can store 10 patterns, then the total number of patterns that can be stored is 10_3. On the other hand, assuming the number of stored patterns grows linearly with the number of neurons, the 75-neuron network can store only 30 patterns --- much less than 1000. This idea is used in this text recognition process.

4.2 Application of Subnetworking to Text Recognition

The above concept is used to recognise a given input text character by comparing it with different base character sets. A subnetwork is used to compare the input

character with each base character set. All the results of comparision are then compared at a higher level and the character is recognised.

In the implementation of the algorithm, we have used 10 different sub-networks, five for lower case and five for the upper case. So, for each input character, we get 1 result from each of the sub-networks. These results are further compared to yield the right answer.

4.3 Advantages of Sub-networking

Advantages of subnetworking include :

* Faster convergence time to reach a fixed point

* Faster learning rate

* Amenable to VLSI implementations (because each subnetwork can be fabricated in a single VLSI chip.)

There are three ways to demonstrate that the convergence time is a quadratic function of the number of neurons. Because each subnetwork has smaller number of neurons, it can therefore converge to stable states faster than the whole neural network. Although we discuss emmory capacity and convergence time using Hopfield network, the concept also applies to other types of neural networks.

4.4 Theory

Consider a neural network W consisting of m subnetworks $S_1,...,S_m$; each containing N/m neurons, where N is the total number of neurons in W. The subnetworks are independent of each other in the sense that there are no synaptic connections between any two subnetworks, i.e.,

$W_{\{i,j\}} = 0$, i belongs to S_h, j belongs to S_k,

$h <> k, h,k = 1,...,m.$

Each subnetwork has its own stable states or fixed points. Let F^i $(i=1,...,m)$ be a stable state in S_i, then the vector $F^1F^2 ... F^m$ is a fixed point of W.

A network consisting of a number of independent networks has higher memory capacity than a fully connected network with the same number of neurons. Also, the convergence time of a subnetwork to a stable state from a nearby initial state is smaller than that of the fully connected networks.

CHAPTER 5
PROPERTIES OF BINARY ORDER MULTI-VALUED HOPFIELD NEURAL NETWORKS

The binary order Hopfield neural network of multi-valued neurons is proposed. There are at least four advantages of this kind of neural network: higher memory capacity, direct storage of multi-valued patterns, allowance of patterns with larger Hamming distance, and more robust pattern recognition. Consequently, it is important to study the properties of such a multi-valued Hopfield network. An energy function is proposed, using a potential function, for synchronous and asynchronous operation, respectively. The convergence of this type of network, for both synchronous and asynchronous operation, is verified an d the upper bounds of convergence time are studied. The number of threshold funcitons for multi-valued neurons is investigated. The enhancement of memory capacity is also discussed.

The advantages of multi-valued Hopfield networks over the conventional binary order networks are as follows:

* higher memory capacity can be achieved
* multi-valued patterns can be stored
* patterns with larger Hamming distance can be used
* more robust pattern recognition can be achieved

5.1 Discussion

It is therefore important to study the properties of such a promising neural network. However, the research work in this field is rare, if not seen, to the authors' knowledge. The convergence time and memory capacity of such networks have been investigated.

Neurons of conventional neural networks have two values (k=2), namely 1 and -1 (or 0). Information is stored as stable states. Consider a neural network with N neurons and each of which can store and retrieve either 0 or 1. The vector of neuron values $U=(U_1,U_2,...,U_N)$ represents the state of the system. Each pair of neurons i and j are interconnected through a synaptic weight W_{ij}. A threshold t_i is associated with each neuron i. It changes its state according to the value of its *potential function* P_i, i = 1,..., N which is defined as

$$P_i = \sum_{j=1}^{N} W_{ij}U_j - t_j.$$

In the remainder of this chapter, $t_i=0$, i=1, ... , N. The memory capacity of such a network is bounded by N. The increase of this capacity can be achieved either by enlarging the size of network or by increasing the number of states of each neuron. This study is focussed on the latter. Let each neuron have k+1 ($=2^a$, a=1,2, ...) stable

states; the value of the potential function of a neuron determines its state according to Eqn's (1) and (2), with Eqn. (2) as follows:

$U_i = 0$ if $P_i < k$,

$= P_i = v$ if $v <= P_i < v + 1$

$= k-1$ if $P_i >= k-1$

The information capacity of such a network of size N is $N \log_2 (k+1)$.

5.2 Conclusion

We propose a binary order multi-valued Hopfield neural network. Using a potential function, we first define an energy function for synchronous and asynchronous operation, respectively. The convergence of this kind of network is

then proved, followed by discussion of the upper bound of convergence time. The number of threshold functions for N (k+1)-valued neurons has been investigated, using the concept of hyperplanes. The result of this investigation is used to study the memory capacity of the proposed neural network. It is also found that the memory capacity of (k+1)-valued prototypes in a (k+1)-valued binary order Hopfield network is independent of the value of neurons.

Compared with conventional binary-valued Hopfield neural network, the proposed (k+1)-valued network (k > = 2) has the following features:

* longer convergence time due to the longer radii of attraction
* greater memory capacity with growth rate $\log_2 (k+1)$
* storage of (k+1)-valued patterns
* allowance of patterns with larger Hamming distances
* robust pattern recognition

CHAPTER 6
THE BASIC ALGORITHM AND ITS IMPLEMENTATION

Step 1. Generate the matrix (64 x 64) of orthogonal vectors R using the algorithm described in Chapter 2.

Step 2. Read one base character set and calculate the strings by adding up horizontal and vertical frequencies of 1's for all 26 characters in that character set. Place this result into a matrix A.

Step 3. Calculate the pseudo-inverse of the matrix A (A^+) using the technique described in Chapter 3.

Step 4. Calculate $W = RA^+$.

Step 5. Read input and convert it into a string as in step 2.

Step 6. Calculate output = W x input_string.

Step 7. Iteratively, generate $output_i = W^*$ x $output_{(i-1)}$. Do it twice.

Step 8. Compare the final output with each of the 26 orthogonal vectors used to represent the base character set.

Step 9. Read another base character set. Calculate the strings by adding up horizontal and vertical frequencies of 1's for the characters in the character set into matrix A. Go to step 3.

Step 10. Use voting and averaging to decide the character which best matches the input character.

CHAPTER 7
EXPERIMENTAL RESULTS & CONCLUSIONS OF THE ORTHOGONAL VECTORS METHOD

7.1 Experimental Results

The orthogonal vectors algorithm was tested with 2600 real samples (including normal and boldface upper and lower case characters). The algorithm has been implemented in 'C' in the SUN-OS environment. A recognition rate of 86% has been achieved.

7.2 Conclusions

A neural-based orthogonal vector with pseudo-inverse approach for pattern recognition has been presented. A method to generate N orthogonal vectors for a N-neuron network is also proposed. Using this approach, patterns with 20% error can be recognized, using English letters (both upper and lower case) for experiment. The robustness of this approach is obvious.

Also, the use of homogeneous, fully inter-connected higher-order multi-valued Hopfield neural network with the subnetworking technique is presented. The orthogonal prototype vectors and the pseudo-inverse method are used to construct the synaptic matrix of the network. Two important properties of this kind of neural networks, namely convergence time and memory capacity, have also been studied. It is also believed that with all the features of the advanced approaches proposed herein, the neural network is of high memory capacity, fast convergence rate, with error-recovery capability. Further work can be done on ways to improve the recognition rate.

CHAPTER 8
USE OF LEVENSHTEIN DISTANCE AND RECOGNITION RULES FOR CHARACTER RECOGNITION

8.1 Introduction

This chapter proposes a hybrid method --- structural-statistic aproach by using string matching and use of rules to recognize a character.

The common idea of structural matching is to compare an unknown pattern with a number of samples, or prototypes, aptterns using a distance or similarity measure. There is a well definition of the distance between unknown input string and prototype string, that is the Levenshtein distance, based on the cost of the edit operations.

There are two phases :

1. Learning phase
2. Recognition phase

Each phase involves three important steps:

1. Histogram string generation
2. Distance measurement
3. Application of rules

8.2 Structural Representation

Strings are used to represent the input image. The frequency of 1's (foreground) in each column and row are added and placed as a string of numbers. This string is compared with the strings representing the base character sets to recognise the correct character. Also, a set of recognition rules is used to determine the actual character.

8.3 String Matching

Syntactic and structural pattern recognition is based on discrete mathematical relations as the detailed descriptions of structure. The simplest structural description of a pattern is its representation as an ordered sequence of elementary components; the presence or absence of a component, and the relative positions of the components, characterize the pattern taken as a whole. Comparision of two such descriptions resemble one another.

The distance d(x,y) between x and y can be defined, according to Levenshtein algorithm as d(x,y) = min (s is a sequence of edit operations which transforms x into y)

So the distance between x and y is obtained by summing up the costs of all elementary operations of the sequence with minimum total costs among all sequences which transform a string x into another string y.

The distance between the generated string and the strings of the prototypes of classes are measured by teh algorithm --- the Levenshtein distance. This is a dynamic programming procedure, i.e. a particular breadth first searching. And it is an error-correcting string matching algorithm. The complexity of this algorithm is O(mn), with respect to both time and space, where m and n being the length of two strings.

Three edit operations, namely insertion, deletion, and substitution, are introduced in the measurement of the Levenshtein distance. In this step, the cost of three operations are defined as follows:

1. the cost of insertion : cost (a->b) = 1;
2. the cost of deletion : cost (a->e) = 1;
3. the cost of insertion : cost (e->b) = coefficient x difference of two symbols

8.4 Recognition Procedure

The basic idea of structural matching, i.e. recognition, is to match two strings, an unknown input pattern and the prototypes, in order to find the prototype which is most similar to an unknown input pattern. The advantages of using string matching are that it is very efficient. A well-known concept from statistical decision theory, nearest-neighbor classification (NN - classification), is applied in this procedure. Using the distance measure of the Levenshtein distance to classify the unknown string x.

1. Get an unknown input image and transform into a string as a pattern, to be compared with the prototypes of classes. Similarly, it is a histogram string generation procedure. Of course, the length of the string is not a constant due to the noise or distortion.

2. Before executing the basic algorithm, check the length of the transformed string if it is within a certain range. If the length exceeding the range, the algorithm consider that it does not belong to a certain class of prototypes and ignore it. Then match with another one to avoid exhaustive calculations of the algorithm. Otherwise calculate the distances between unknown input string and all possible candidates.

3. Repeat the above procedure for ten base sets of characters.

4. Then, using the recognition rules, match the input string with the prototype which is closest to the input string.

8.5 Recognition Rules

For given base sets of characters, execute the recognition algorithm and generate matching patterns and distances for each character to be recognized. Once these

rules are generated, for matching a character with each of the ten base sets, use these rules to recognise any unknown input string.

CHAPTER 9
EXPERIMENTAL RESULTS & CONCLUSIONS USING RECOGNITION RULES

9.1 Experimental Results

The Levenshtein algorithm was implemented in 'C' in the SUN-OS environment and the recognition rate was found to be 99.14%. Then, recognition rules were applied and the results were found to be perfect. The algorithm was implemented in 'C' in the SUN-OS environment. A recognition rate of 100% has been achieved. The algorithm was tested for 2600 real samples of characters.

9.2 Conclusions

In this work, error tolerance classification for character recognition is presented. Although the noise or distortion of character image is unavoidable, the recognition procedure with some knowledge, that is roughly distinguished the difference or cost instead of exactly difference or cost, will discard the effect of the noise and by judging the length of the image string to bypass obviously unneccessary calculations thus to reduce the executing time of the basic algorithm. The experimental result shows that as the performance becomes more and more satisfactory, this method is correctly applied in the experiment.

APPENDIX A

PROGRAM LISTINGS

ORTHOGONAL VECTOR & PSEUDO-INVERSE METHOD

```
#include <stdio.h>
#include <math.h>
#define N 8
#define ep 0.001
extern int  makestring();
recog(narg,filename,arg1,arg2)
FILE *filename;
int *arg1,*narg;
char *arg2;
{
char ch;
int *A;
int arlen[27];
int inlen;
float de,w,y;
float ident[65];
int SIGMA[65][65],ORTHO[65][27];
int loop;
float b[27],c[27];
int z[27],mini;
int s[27];
int smin;
int M[N*N+1][N*N+1],Da[N*N+1][27],Dat[27][N*N+1];
float InM[27][27];
FILE *fp1,*fp2;
float IM[N*N+1][N*N+1];
int   In[N*N+1];
float W[N*N+1][N*N+1];
float hf1[N*N+1],hf[N*N+1];
int h[N*N+1];
int step;
int test,tt1,i,j,k,l,p,num;
char ts;
int cc[27];
FILE *fp,*inf;

A = (int *) malloc(sizeof(int)  *N*N*N*N);
step=(*narg);
     /* allocate space for A */
fp1=fopen("data.txt","r");
fp2=fopen("data","w");
inlen=makestring(fp1,fp2);
fclose(fp1);
fclose(fp2);

inf=fopen("data","r");
ortho(N*N,A);
     /* get the orthogonal vector */
cc[1]='a';
cc[2]='b';
cc[3]='c';
cc[4]='d';
cc[5]='e';
cc[6]='f';
```

```
cc[7]='g';
cc[8]='h';
cc[9]='i';
cc[10]='j';
cc[11]='k';
cc[12]='l';
cc[13]='m';
cc[14]='n';
cc[15]='o';
cc[16]='p';
cc[17]='q';
cc[18]='r';
cc[19]='s';
cc[20]='t';
cc[21]='u';
cc[22]='v';
cc[23]='w';
cc[24]='x';
cc[25]='y';
cc[26]='z';

     /* get the 26 characters */

fp1=fopen(filename,"r");
fp2=fopen("strings.dat","w");
for (i=1;i<=26;i++)
     arlen[i]=makestring(fp1,fp2);
fclose(fp1);
fclose(fp2);


if ((fp=fopen("strings.dat","r")) == NULL )
{
printf("error open file\n");
}

     /* get the 26 sample characters */

for(i=1;i<27;i++)
{
   for(j=1;j< N*N+1 ;j++)
     {
     fscanf(fp,"%d",&Dat[i][j]);
     }
}
     /* change the 0 value to -1 of sample characters */

fclose(fp);

     /* close the sample characters file "a88.dat" */

for(i=1;i<27;i++)
{
   for(j=1;j< N * N + 1;j++)
```

```
          Da[j][i] = Dat[i][j];
}

     /* transpose the sample matrix to Da    */

for(i=1;i<=N*N;i++)
     {
     for(j=1;j<=N*N;j++)
     {
     W[i][j]=0.0;
     IM[i][j]=0.0;
     }
     }


for(i=1;i<=26;i++)
     {
     for(j=1;j<=26;j++)
     M[i][j]=0;
     }

     /*  initial the matrix W and IM and M   */


for(k=1; k<=26; k++)
{
    for(j=1; j<=26; j++)
    {
     for(i=1; i<=64; i++)
     M[k][j] = M[k][j] +  Dat[k][i] * Da[i][j];
     }
}

       /*  get the value of M ( M = transpose(Da) * Da ) */

for (i=1; i<=26; i++)
{
     for (j=1; j<=26; j++)
     {
     InM[i][j] = (float) (M[i][j]);
     }
}

     /* transfer the integer date to float date in order for
*/
     /* calculate the inverse of matrix M
*/

     /* below is to inverse the matrix M */


for (j=1; j<=26; j++)
      {
      z[j]=j;
```

```
      }
for (i=1; i<=26; i++)
{
   k=i; y = InM[i][i]; l=i-1; p=i+1;
   for(j=p;j<=26;j++)
   {
   w = InM[i][j];
   if (fabs(w) > fabs(y))
       {
       k=j; y=w;
       }
   }

   if (fabs(y) < ep)
   {
   printf("no inverse exists\n");
   exit(1);
   }
   y= 1 / y;
   for (j=1;j<=26;j++)
       {
       c[j]=InM[j][k];
       InM[j][k] = InM[j][i];
       InM[j][i] =  -c[j] * y;
       b[j] = InM[i][j] * y;
       InM[i][j] = InM[i][j] * y;
       }

InM[i][i] = y;j = z[i];
z[i] = z[k]; z[k] = j;

for(k=1; k<=l; k++)
for(j=1; j<=l; j++)
InM[k][j] = InM[k][j] - b[j] * c[k];

for(k=1; k<=l; k++)
for(j=p; j<=26; j++)
InM[k][j] = InM[k][j] - b[j] * c[k];

for(k=p; k<=26; k++)
for(j=1; j<=l; j++)
InM[k][j] = InM[k][j] - b[j] * c[k];

for(k=p; k<=26; k++)
for(j=p; j<=26; j++)
InM[k][j] = InM[k][j] - b[j] * c[k];
}

for(i=1; i<=26; i++)
{
l1: k = z[i];
if (k != i)
   {
   for(j=1; j<=26; j++)
```

```
   {
   w = InM[i][j];
   InM[i][j] = InM[k][j];
   InM[k][j] = w;
   }
   p = z[i];
   z[i] = z[k];
   z[k] = p;
   goto l1;
   }
}


     /* got the inverse of matrix M in InM  */

for(i=1;i<=26;i++)
{
   for(j=1;j<=N*N;j++)
     {
     for(k=1;k<=26;k++)
     IM[i][j] = IM[i][j] + InM[i][k] * Dat[k][j];
     }
}

     /*  calculate matrix IM = Da+ (psuedo inverse of Da) */
     /*  (IM = inverse((transpose(Da)*Da)*transpose(Da)   */


for (i=1;i<N*N+1;i++)
     for (j=1;j<=26;j++)
          ORTHO[i][j]=A[(i-1)*N*N+j-1+step];

for (i=1;i<N*N+1;i++)
     for (j=1;j<N*N+1;j++)
          SIGMA[i][j]=0;

/* W* = calculate sum of outer products of 26 orthogonal
vectors in SIGMA*/

for (loop=1;loop<=26;loop++)
{
     for (i=1;i<N*N+1;i++)
          for (j=1;j<N*N+1;j++)
               SIGMA[i][j]=SIGMA[i][j]+ORTHO[i][loop]*ORTHO[
j][loop]/18;
}


for(i=1;i<=N*N;i++)
   for(j=1;j<=N*N;j++)
        for(k=1;k<=26;k++)
             W[i][j] = W[i][j] +
(float)(ORTHO[i][k])*IM[k][j];
```

```
     /* calculate W (W = 26 orth vector * Da+)   */


/* reading input matrix */
for(i=1;i<=N*N;i++)
       fscanf(inf,"%d",&h[i]);

/* initialise hf */
for(i=1;i<N*N+1;i++)
     hf[i]=0.0;


for(i=1;i<N*N+1;i++)
     for(j=1;j<N*N+1;j++)
          hf[i] = hf[i] + W[i][j]*(float)(h[j]);

/* ITERATIONS */

for (i=1;i<N*N+1;i++)
     hf1[i]=0;

for (j=1;j<=64;j++)
     for (k=1;k<=64;k++)
          hf1[j]=hf1[j]+SIGMA[j][k]*hf[k];
for (i=1;i<N*N+1;i++)
     hf[i]=hf1[i];
for (i=1;i<N*N+1;i++)
     hf1[i]=0;

for (j=1;j<=64;j++)
     for (k=1;k<=64;k++)
          hf1[j]=hf1[j]+SIGMA[j][k]*hf[k];


for (i=1;i<N*N+1;i++)
     {
     h[i]=(int)(hf[i]+0.5);
     if (h[i]<-18)
          h[i]=-18;
     if (h[i]>18)
          h[i]=18;
     }

     /* calculate hf = W * h          */



smin=2000;  mini=1;


for(i=1;i<=26;i++)
{
s[i]=0;
for(j=1;j<=N*N;j++)
```

```
        s[i] = s[i] + abs(ORTHO[j][i] - h[j]);
            if ((inlen-arlen[i])>4)
                 s[i]=2000;
        if ( smin >= s[i])
            {
            smin = s[i];
            mini = i;
            }
}

        (*arg1)=smin;
        (*arg2)=cc[mini];
        /* print out the character according to the position
*/
        /* if minimal distance > smin, recognition  fail
*/
fclose(inf);
}


#include <stdio.h>
#include <math.h>
#define N 8
#define ep 0.001
extern int  makestring();
RECOG(narg,filename,arg1,arg2)
FILE *filename;
int *arg1,*narg;
char *arg2;
{
char ch;
int *A;
int arlen[27];
int inlen;
float de,w,y;
float ident[65];
int SIGMA[65][65],ORTHO[65][27];
int loop;
float b[27],c[27];
int z[27],mini;
int s[27];
int smin;
int M[N*N+1][N*N+1],Da[N*N+1][27],Dat[27][N*N+1];
float InM[27][27];
FILE *fp1,*fp2;
float IM[N*N+1][N*N+1];
int   In[N*N+1];
float W[N*N+1][N*N+1];
float hf1[N*N+1],hf[N*N+1];
int h[N*N+1];
int step;
int test,tt1,i,j,k,l,p,num;
char ts;
int cc[27];
```

```
FILE *fp,*inf;

A = (int *) malloc(sizeof(int)  *N*N*N*N);
step=(*narg);
     /* allocate space for A */
fp1=fopen("data.txt","r");
fp2=fopen("data","w");
inlen=makestring(fp1,fp2);
fclose(fp1);
fclose(fp2);

inf=fopen("data","r");
ortho(N*N,A);
     /* get the orthogonal vector */
cc[1]='A';
cc[2]='B';
cc[3]='C';
cc[4]='D';
cc[5]='E';
cc[6]='F';
cc[7]='G';
cc[8]='H';
cc[9]='I';
cc[10]='J';
cc[11]='K';
cc[12]='L';
cc[13]='M';
cc[14]='N';
cc[15]='O';
cc[16]='P';
cc[17]='Q';
cc[18]='R';
cc[19]='S';
cc[20]='T';
cc[21]='U';
cc[22]='V';
cc[23]='W';
cc[24]='X';
cc[25]='Y';
cc[26]='Z';

     /* get the 26 characters */

fp1=fopen(filename,"r");
fp2=fopen("strings.dat","w");
for (i=1;i<=26;i++)
     arlen[i]=makestring(fp1,fp2);
fclose(fp1);
fclose(fp2);


if ((fp=fopen("strings.dat","r")) == NULL )
{
printf("error open file\n");
```

```
}
     /* get the 26 sample characters */
for(i=1;i<27;i++)
{
   for(j=1;j< N*N+1 ;j++)
     {
     fscanf(fp,"%d",&Dat[i][j]);
     }
}
     /* change the 0 value to -1 of sample characters */
fclose(fp);
     /* close the sample characters file "a88.dat" */
for(i=1;i<27;i++)
{
   for(j=1;j< N * N + 1;j++)
        Da[j][i] = Dat[i][j];
}
     /* transpose the sample matrix to Da    */
for(i=1;i<=N*N;i++)
     {
     for(j=1;j<=N*N;j++)
     {
     W[i][j]=0.0;
     IM[i][j]=0.0;
     }
     }

for(i=1;i<=26;i++)
     {
     for(j=1;j<=26;j++)
     M[i][j]=0;
     }
     /*  initial the matrix W and IM and M   */

for(k=1; k<=26; k++)
{
    for(j=1; j<=26; j++)
    {
     for(i=1; i<=64; i++)
     M[k][j] = M[k][j] +  Dat[k][i] * Da[i][j];
     }
}
       /*  get the value of M ( M = transpose(Da) * Da ) */
```

```
for (i=1; i<=26; i++)
{
     for (j=1; j<=26; j++)
     {
     InM[i][j] = (float) (M[i][j]);
     }
}

     /* transfer the integer date to float date in order for
*/
     /* calculate the inverse of matrix M
*/

     /* below is to inverse the matrix M */


for (j=1; j<=26; j++)
      {
      z[j]=j;
      }
for (i=1; i<=26; i++)
{
   k=i; y = InM[i][i]; l=i-1; p=i+1;
   for(j=p;j<=26;j++)
   {
   w = InM[i][j];
   if (fabs(w) > fabs(y))
       {
       k=j; y=w;
       }
   }

   if (fabs(y) < ep)
   {
   printf("no inverse exists\n");
   exit(1);
   }
   y= 1 / y;
   for (j=1;j<=26;j++)
       {
       c[j]=InM[j][k];
       InM[j][k] = InM[j][i];
       InM[j][i] =  -c[j] * y;
       b[j] = InM[i][j] * y;
       InM[i][j] = InM[i][j] * y;
       }

InM[i][i] = y;j = z[i];
z[i] = z[k]; z[k] = j;

for(k=1; k<=l; k++)
for(j=1; j<=l; j++)
InM[k][j] = InM[k][j] - b[j] * c[k];
```

```
for(k=1; k<=l; k++)
for(j=p; j<=26; j++)
InM[k][j] = InM[k][j] - b[j] * c[k];

for(k=p; k<=26; k++)
for(j=1; j<=l; j++)
InM[k][j] = InM[k][j] - b[j] * c[k];

for(k=p; k<=26; k++)
for(j=p; j<=26; j++)
InM[k][j] = InM[k][j] - b[j] * c[k];
}

for(i=1; i<=26; i++)
{
l1: k = z[i];
if (k != i)
   {
   for(j=1; j<=26; j++)
   {
   w = InM[i][j];
   InM[i][j] = InM[k][j];
   InM[k][j] = w;
   }
   p = z[i];
   z[i] = z[k];
   z[k] = p;
   goto l1;
   }
}


     /* got the inverse of matrix M in InM  */

for(i=1;i<=26;i++)
{
   for(j=1;j<=N*N;j++)
     {
     for(k=1;k<=26;k++)
     IM[i][j] = IM[i][j] + InM[i][k] * Dat[k][j];
     }
}

     /*  calculate matrix IM = Da+ (psuedo inverse of Da) */
     /*  (IM = inverse((transpose(Da)*Da)*transpose(Da)   */


for (i=1;i<N*N+1;i++)
     for (j=1;j<=26;j++)
          ORTHO[i][j]=A[(i-1)*N*N+j-1+step];

for (i=1;i<N*N+1;i++)
     for (j=1;j<N*N+1;j++)
```

```
          SIGMA[i][j]=0;

/* calculate sum of inner products of 26 orthogonal vectors
in SIGMA*/

for (loop=1;loop<=26;loop++)
{
     for (i=1;i<N*N+1;i++)
          for (j=1;j<N*N+1;j++)
               SIGMA[i][j]=SIGMA[i][j]+ORTHO[i][loop]*ORTHO[
j][loop]/18;
}


for(i=1;i<=N*N;i++)
   for(j=1;j<=N*N;j++)
        for(k=1;k<=26;k++)
             W[i][j] = W[i][j] +
(float)(ORTHO[i][k])*IM[k][j];

     /* calculate W (W = 26 orth vector * Da+)   */


/* reading input matrix */
for(i=1;i<=N*N;i++)
       fscanf(inf,"%d",&h[i]);

/* initialise hf */
for(i=1;i<N*N+1;i++)
     hf[i]=0.0;


for(i=1;i<N*N+1;i++)
     for(j=1;j<N*N+1;j++)
          hf[i] = hf[i] + W[i][j]*(float)(h[j]);

/* ITERATIONS */

for (i=1;i<N*N+1;i++)
     hf1[i]=0;

for (j=1;j<=64;j++)
     for (k=1;k<=64;k++)
          hf1[j]=hf1[j]+SIGMA[j][k]*hf[k];
for (i=1;i<N*N+1;i++)
     hf[i]=hf1[i];
for (i=1;i<N*N+1;i++)
     hf1[i]=0;

for (j=1;j<=64;j++)
     for (k=1;k<=64;k++)
          hf1[j]=hf1[j]+SIGMA[j][k]*hf[k];
```

```
for (i=1;i<N*N+1;i++)
     {
     h[i]=(int)(hf[i]+0.5);
     if (h[i]<-18)
          h[i]=-18;
     if (h[i]>18)
          h[i]=18;
          }

     /* calculate hf = W * h          */


smin=2000;  mini=1;


for(i=1;i<=26;i++)
{
s[i]=0;
for(j=1;j<=N*N;j++)
     s[i] = s[i] + abs(ORTHO[j][i] - h[j]);
         if ((inlen-arlen[i])>4)
              s[i]=2000;
     if ( smin >= s[i])
         {
         smin = s[i];
         mini = i;
         }
}

     (*arg1)=smin;
     (*arg2)=cc[mini];
     /* print out the character according to the position
*/
     /* if minimal distance > smin, recognition  fail
*/
fclose(inf);
}

main : main.c recog.o  RECOG.o
     cc -g main.c recog.o  RECOG.o makestring.o digit.o
ortho.o -lm -o main
recog.o : recog.c ortho.o makestring.o
     cc recog.c -c
RECOG.o : RECOG.c ortho.o makestring.o
     cc RECOG.c -c
ortho.o : ortho.c digit.o
     cc ortho.c -c
digit.o : digit.c
     cc digit.c -c
makestring.o : makestring.c
     cc makestring.c -c
```

```
#include <stdio.h>
#include <math.h>
void ortho(n,A)

     /*      Orthogonal vectors generation program
*/
     /*   (to generate n orthogonal vector and store it in
A)  */


int n;
int *A;
{ int *B;
     int dsize = (int)( (log((float) n))/log (2.0) + 0.5);
        int i, j, k;

        B = (int *) malloc(sizeof(int)*dsize);
        for (i = 0; i < n; i ++) {
                digit(i, B, dsize);
          for (j = 0; j < n; j ++)
          {
            A[i*n+j] = 1;
            for (k = 0; k < dsize; k ++)
            A[i*n+j] =
(B[k]*(j*(int)pow(2.0,(double)(k+1))/n)%2)?
                                        -1*A[i*n+j] : 1*A[i*n+j];
                }
        }
     for (i=0;i<n*n;i++)
          A[i]=18*A[i];
}

#include <stdio.h>
void digit(i,B,size)
int i;
int *B;
int size;
{
        int k = 0;

        for (k = 0; k < size; k ++) {
                B[k] = i > 0 ? i % 2 : 0;
                i = i / 2;
        }
}
#include <stdio.h>
#include <ctype.h>
#include <string.h>
int makestring(file1, file2)
FILE *file1, *file2;
{
int i,j,k;
```

```
char line[81];
int linestr;
int len;
int rowflag,colflag;
int ctr[64];

for (i=1;i<=64;i++)
     ctr[i]=0;
rowflag=1;
i=1;
while (rowflag)
{
     fgets(line,81,file1);
     if (strcmp(line,"\n")==0)
          rowflag=0;
     else
     {
     linestr=strlen(line)-1;
     for (j=1;j<=linestr;j++)
          if (line[j]=='1')
               {
               ctr[j]++;
               ctr[linestr+i]++;
               }
     i++;
     }
}
len=i+linestr;
for (j=linestr+i;j<=64;j++)
     ctr[j]=0;
for (i=1;i<=64;i++)
     {
     ctr[i]=ctr[i]-18;
     fprintf(file2,"%d ",ctr[i]);
     }
fprintf(file2,"\n");
return(len);
}
```

APPENDIX B
PROGRAM LISTINGS
LEVENSHTEIN DISTANCE AND RECOGNITION RULES

```
#include <stdio.h>
#include <ctype.h>
#include <string.h>
int makestring(file1, file2)
FILE *file1, *file2;
{
int i,j,k;
char line[81];
int linestr;
int len;
int rowflag,colflag;
int ctr[64];

for (i=1;i<=64;i++)
     ctr[i]=0;
rowflag=1;
i=1;
while (rowflag)
{
     fgets(line,81,file1);
     if (strcmp(line,"\n")==0)
          rowflag=0;
     else
     {
     linestr=strlen(line)-1;
     for (j=1;j<=linestr;j++)
          if (line[j]=='1')
               {
               ctr[j]++;
               ctr[linestr+i]++;
               }
     i++;
     }
}
len=i+linestr;
for (j=linestr+i;j<=64;j++)
     ctr[j]=0;
for (i=1;i<=64;i++)
     {
     ctr[i]=ctr[i]-18;
     fprintf(file2,"%d ",ctr[i]);
     }
fprintf(file2,"\n");
return(len);
}
/* Method 1 */
#include <stdio.h>
#define MAXLEN 90
#define const 0.5
#define del 1
#define ins 1
extern void makestring1();
extern float min();
extern float diff();
```

```
recog1(filename,arg1,arg2)
FILE *filename;
float *arg1;
char *arg2;
{
int i,j,k,m;
int cc[27];
FILE *fp;
char img_str[81];
int img_len;
char sam_str[27][81];
int sam_len[27];
float DELTA[MAXLEN][MAXLEN];
float mindis;
int minidx;

makestring1("data.txt","data1",1);
fp=fopen("data1","r");
fgets(img_str,81,fp);
img_len=strlen(img_str);
fclose(fp);

cc[1]='a';
cc[2]='b';
cc[3]='c';
cc[4]='d';
cc[5]='e';
cc[6]='f';
cc[7]='g';
cc[8]='h';
cc[9]='i';
cc[10]='j';
cc[11]='k';
cc[12]='l';
cc[13]='m';
cc[14]='n';
cc[15]='o';
cc[16]='p';
cc[17]='q';
cc[18]='r';
cc[19]='s';
cc[20]='t';
cc[21]='u';
cc[22]='v';
cc[23]='w';
cc[24]='x';
cc[25]='y';
cc[26]='z';

for ( i=0; i<MAXLEN; i++ )
     DELTA[0][i] = i;

     /* get the 26 characters */
```

```
makestring1(filename,"strings1.dat",26);
if ((fp=fopen("strings1.dat","r")) == NULL )
{
     printf("error open file\n");
}
for ( i=1; i<=26; i++ )
{
     fgets(sam_str[i],81,fp);
     sam_len[i]= strlen(sam_str[i]);
}
fclose(fp);

mindis = 10000;
minidx = 0;
for( j=1; j<=26; j++ )
{
     if ( abs(img_len-sam_len[j]) > 4 ) continue;
     for( k=1; k<img_len ; k++ )
          for( m=1; m<sam_len[j]; m++ )

          DELTA[k][m] = min(min(DELTA[k-1][m]+ins,DELTA[k][m-
1]+del),
          min(DELTA[k][m-1]+del,
           DELTA[k-1][m-
1]+const*diff(img_str,sam_str,k,j,m)));

     k--; m--;
     if ( DELTA[k][m] < mindis )
     {
          mindis = DELTA[k][m];
          minidx = j;
     }
}
(*arg1) = mindis;
(*arg2) = cc[minidx];
}

/* Method 2 */
#include <stdio.h>
#define MAXLEN 90
#define const 0.5
#define del 1
#define ins 1
extern void makestring2();
extern float min();
extern float diff();

recog2(filename,arg1,arg2)
FILE *filename;
float *arg1;
char *arg2;
{
int i,j,k,m;
```

```
int cc[27];
FILE *fp;
char img_str[81];
int img_len;
char sam_str[27][81];
int sam_len[27];
float DELTA[MAXLEN][MAXLEN];
float mindis;
int minidx;

makestring2("data.txt","data2",1);
fp=fopen("data2","r");
fgets(img_str,81,fp);
img_len=strlen(img_str);
fclose(fp);

cc[1]='a';
cc[2]='b';
cc[3]='c';
cc[4]='d';
cc[5]='e';
cc[6]='f';
cc[7]='g';
cc[8]='h';
cc[9]='i';
cc[10]='j';
cc[11]='k';
cc[12]='l';
cc[13]='m';
cc[14]='n';
cc[15]='o';
cc[16]='p';
cc[17]='q';
cc[18]='r';
cc[19]='s';
cc[20]='t';
cc[21]='u';
cc[22]='v';
cc[23]='w';
cc[24]='x';
cc[25]='y';
cc[26]='z';

for ( i=0; i<MAXLEN; i++ )
     DELTA[0][i] = i;

     /* get the 26 characters */

makestring2(filename,"strings2.dat",26);
if ((fp=fopen("strings2.dat" "r")) == NULL )
{
     printf("error open file\n");
}
for ( i=1; i<=26; i++ )
```

```
{
     fgets(sam_str[i],81,fp);
     sam_len[i]= strlen(sam_str[i]);
}
fclose(fp);

mindis = 10000;
minidx = 0;
for( j=1; j<=26; j++ )
{
     if ( abs(img_len-sam_len[j]) > 4 ) continue;
     for( k=1; k<img_len ; k++ )
          for( m=1; m<sam_len[j]; m++ )

         DELTA[k][m] = min(min(DELTA[k-1][m]+ins,DELTA[k][m-
1]+del),
          min(DELTA[k][m-1]+del,
           DELTA[k-1][m-
1]+const*diff(img_str,sam_str,k,j,m)));

     k--; m--;
     if ( DELTA[k][m] < mindis )
     {
          mindis = DELTA[k][m];
          minidx = j;
     }
}
(*arg1) = mindis;
(*arg2) = cc[minidx];
}

/* Method 4 */
#include <stdio.h>
#define MAXLEN 90
#define const 0.5
#define del 1
#define ins 1
extern void makestring3();
extern float min();
extern float diff();

recog3(filename,arg1,arg2)
FILE *filename;
float *arg1;
char *arg2;
{
int i,j,k,m;
int cc[27];
FILE *fp;
char img_str[81];
int img_len;
char sam_str[27][81];
int sam_len[27];
float DELTA[MAXLEN][MAXLEN];
```

```
float mindis;
int minidx;

makestring3("data.txt","data3",1);
fp=fopen("data3","r");
fgets(img_str,81,fp);
img_len=strlen(img_str);
fclose(fp);

cc[1]='a';
cc[2]='b';
cc[3]='c';
cc[4]='d';
cc[5]='e';
cc[6]='f';
cc[7]='g';
cc[8]='h';
cc[9]='i';
cc[10]='j';
cc[11]='k';
cc[12]='l';
cc[13]='m';
cc[14]='n';
cc[15]='o';
cc[16]='p';
cc[17]='q';
cc[18]='r';
cc[19]='s';
cc[20]='t';
cc[21]='u';
cc[22]='v';
cc[23]='w';
cc[24]='x';
cc[25]='y';
cc[26]='z';

for ( i=0; i<MAXLEN; i++ )
     DELTA[0][i] = i;

     /* get the 26 characters */

makestring3(filename,"strings3.dat",26);
if ((fp=fopen("strings3.dat","r")) == NULL )
{
     printf("error open file\n");
}
for ( i=1; i<=26; i++ )
{
     fgets(sam_str[i],81,fp);
     sam_len[i]= strlen(sam_str[i]);
}
fclose(fp);

mindis = 10000;
```

```
minidx = 0;
for( j=1; j<=26; j++ )
{
     if ( abs(img_len-sam_len[j]) > 4 ) continue;
     for( k=1; k<img_len ; k++ )
          for( m=1; m<sam_len[j]; m++ )

         DELTA[k][m] = min(min(DELTA[k-1][m]+ins,DELTA[k][m-
1]+del),
         min(DELTA[k][m-1]+del,
          DELTA[k-1][m-
1]+const*diff(img_str,sam_str,k,j,m)));

     k--; m--;
     if ( DELTA[k][m] < mindis )
     {
          mindis = DELTA[k][m];
          minidx = j;
     }
}
(*arg1) = mindis;
(*arg2) = cc[minidx];
}

/* Method 4 */
#include <stdio.h>
#define MAXLEN 90
#define const 0.5
#define del 1
#define ins 1
extern void makestring4();
extern float min();
extern float diff();

recog4(filename,arg1,arg2)
FILE *filename;
float *arg1;
char *arg2;
{
int i,j,k,m;
int cc[27];
FILE *fp;
char img_str[81];
int img_len;
char sam_str[27][81];
int sam_len[27];
float DELTA[MAXLEN][MAXLEN];
float mindis;
int minidx;

makestring4("data.txt","data4",1);
fp=fopen("data4","r");
fgets(img_str,81,fp);
img_len=strlen(img_str);
```

```
fclose(fp);

cc[1]='a';
cc[2]='b';
cc[3]='c';
cc[4]='d';
cc[5]='e';
cc[6]='f';
cc[7]='g';
cc[8]='h';
cc[9]='i';
cc[10]='j';
cc[11]='k';
cc[12]='l';
cc[13]='m';
cc[14]='n';
cc[15]='o';
cc[16]='p';
cc[17]='q';
cc[18]='r';
cc[19]='s';
cc[20]='t';
cc[21]='u';
cc[22]='v';
cc[23]='w';
cc[24]='x';
cc[25]='y';
cc[26]='z';

for ( i=0; i<MAXLEN; i++ )
     DELTA[0][i] = i;

     /* get the 26 characters */

makestring4(filename,"strings4.dat",26);
if ((fp=fopen("strings4.dat","r")) == NULL )
{
     printf("error open file\n");
}
for ( i=1; i<=26; i++ )
{
     fgets(sam_str[i],81,fp);
     sam_len[i]= strlen(sam_str[i]);
}
fclose(fp);

mindis = 10000;
minidx = 0;
for( j=1; j<=26; j++ )
{
     if ( abs(img_len-sam_len[j]) > 4 ) continue;
     for( k=1; k<img_len ; k++ )
          for( m=1; m<sam_len[j]; m++ )
```

```
          DELTA[k][m] = min(min(DELTA[k-1][m]+ins,DELTA[k][m-
1]+del),
          min(DELTA[k][m-1]+del,
           DELTA[k-1][m-
1]+const*diff(img_str,sam_str,k,j,m)));

     k--; m--;
     if ( DELTA[k][m] < mindis )
     {
          mindis = DELTA[k][m];
          minidx = j;
     }
}
(*arg1) = mindis;
(*arg2) = cc[minidx];
}

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
extern void recog1();
extern void recog2();
extern void recog3();
extern void recog4();
extern void RECOG1();
extern void RECOG2();
extern void RECOG3();
extern void RECOG4();

main()
{
int i,j;
float min_dist;
float min1_dist,min2_dist,min3_dist,min4_dist;
char res_chr,chr[10];
char res1_chr,res2_chr,res3_chr,res4_chr;
char avchr[10];
int  cnt[10];
float dist[10],avdis[10];
/* Method 1 of recognition */
for (i=0;i<10;i++)
{
     dist[i]=0;
     chr[i]=' ';
}
recog1("lower0.txt",&dist[0],&chr[0]);
recog1("lower1.txt",&dist[1],&chr[1]);
recog1("lower2.txt",&dist[2],&chr[2]);
recog1("lower3.txt",&dist[3],&chr[3]);
recog1("lower4.txt",&dist[4],&chr[4]);
RECOG1("upper0.txt",&dist[5],&chr[5]);
RECOG1("upper1.txt",&dist[6],&chr[6]);
RECOG1("upper2.txt",&dist[7],&chr[7]);
RECOG1("upper3.txt",&dist[8],&chr[8]);
```

```
RECOG1("upper4.txt",&dist[9],&chr[9]);

min1_dist=dist[0];
res1_chr=chr[0];
for (i=0;i<10;i++)
{
     if (dist[i]<min1_dist)
     {
          min1_dist=dist[i];
          res1_chr=chr[i];
     }
}
/* Method 2 of recognition */

for (i=0;i<10;i++)
{
     dist[i]=0;
     chr[i]=' ';
}
recog2("lower0.txt",&dist[0],&chr[0]);
recog2("lower1.txt",&dist[1],&chr[1]);
recog2("lower2.txt",&dist[2],&chr[2]);
recog2("lower3.txt",&dist[3],&chr[3]);
recog2("lower4.txt",&dist[4],&chr[4]);
RECOG2("upper0.txt",&dist[5],&chr[5]);
RECOG2("upper1.txt",&dist[6],&chr[6]);
RECOG2("upper2.txt",&dist[7],&chr[7]);
RECOG2("upper3.txt",&dist[8],&chr[8]);
RECOG2("upper4.txt",&dist[9],&chr[9]);

min2_dist=dist[0];
res2_chr=chr[0];
for (i=0;i<10;i++)
{
     if (dist[i]<min2_dist)
     {
          min2_dist=dist[i];
          res2_chr=chr[i];
     }
}
/* Method 3 of recognition */

for (i=0;i<10;i++)
{
     dist[i]=0;
     chr[i]=' ';
}
recog3("lower0.txt",&dist[0],&chr[0]);
recog3("lower1.txt",&dist[1],&chr[1]);
recog3("lower2.txt",&dist[2],&chr[2]);
recog3("lower3.txt",&dist[3],&chr[3]);
recog3("lower4.txt",&dist[4],&chr[4]);
RECOG3("upper0.txt",&dist[5],&chr[5]);
RECOG3("upper1.txt",&dist[6],&chr[6]);
```

```
RECOG3("upper2.txt",&dist[7],&chr[7]);
RECOG3("upper3.txt",&dist[8],&chr[8]);
RECOG3("upper4.txt",&dist[9],&chr[9]);

min3_dist=dist[0];
res3_chr=chr[0];
for (i=0;i<10;i++)
{
     if (dist[i]<min3_dist)
     {
          min3_dist=dist[i];
          res3_chr=chr[i];
     }
}
/* Method 4 of recognition */

for (i=0;i<10;i++)
{
     dist[i]=0;
     chr[i]=' ';
}
recog4("lower0.txt",&dist[0],&chr[0]);
recog4("lower1.txt",&dist[1],&chr[1]);
recog4("lower2.txt",&dist[2],&chr[2]);
recog4("lower3.txt",&dist[3],&chr[3]);
recog4("lower4.txt",&dist[4],&chr[4]);
RECOG4("upper0.txt",&dist[5],&chr[5]);
RECOG4("upper1.txt",&dist[6],&chr[6]);
RECOG4("upper2.txt",&dist[7],&chr[7]);
RECOG4("upper3.txt",&dist[8],&chr[8]);
RECOG4("upper4.txt",&dist[9],&chr[9]);

min4_dist=dist[0];
res4_chr=chr[0];
for (i=0;i<10;i++)
{
     if (dist[i]<min4_dist)
     {
          min4_dist=dist[i];
          res4_chr=chr[i];
     }
}
/* Recognition of character from experimental analysis */
if ((res1_chr==res2_chr)&&(res1_chr != res3_chr))
{
    res_chr=res2_chr;
    if (min1_dist < min2_dist)
      min_dist=min1_dist;
    else
      min_dist=min2_dist;
}
if ((res1_chr==res3_chr)&&(res1_chr != res2_chr))
{
    res_chr=res3_chr;
```

```
    if (min1_dist < min3_dist)
      min_dist=min1_dist;
    else
      min_dist=min3_dist;
}
if ((res2_chr==res3_chr)&&(res2_chr != res1_chr))
{
    res_chr=res2_chr;
    if (min2_dist < min3_dist)
      min_dist=min2_dist;
    else
      min_dist=min3_dist;
}
if ((res1_chr==res2_chr)&&(res2_chr==res3_chr))
{
    res_chr=res2_chr;
    if ((min1_dist <= min2_dist) && (min1_dist <=
min3_dist))
      min_dist=min1_dist;
    if ((min2_dist <= min3_dist) && (min2_dist <=
min1_dist))
        min_dist=min2_dist;
    if ((min3_dist <= min2_dist) && (min3_dist <=
min1_dist))
        min_dist=min3_dist;
}
if (res3_chr=='H')
{
  res_chr='H';
  min_dist=min3_dist;
}
if (res2_chr=='E')
{
  res_chr='E';
  min_dist=min2_dist;
}
if (res1_chr=='a')
{
  res_chr='a';
  min_dist=min1_dist;
}
if ((res1_chr=='b') || (res3_chr=='b'))
{
  res_chr='b';
  min_dist=min1_dist;
}
if ((res1_chr=='b') && (res2_chr=='h') && (res3_chr=='h'))
{
  res_chr='h';
  if (min2_dist < min3_dist)
    min_dist=min2_dist;
  else
    min_dist=min3_dist;
}
```

```
if ((res1_chr=='b')&&(res2_chr=='s')&&(res3_chr=='h'))
{
  res_chr='b';
  min_dist=min1_dist;
}
if ((res1_chr=='c')&&(res2_chr=='t')&&(res3_chr=='c'))
{
  res_chr='c';
  if (min1_dist < min3_dist)
    min_dist=min1_dist;
  else
    min_dist=min3_dist;
}
if (res1_chr=='e')
{ res_chr='e';
  min_dist=min1_dist;
}
if
((res1_chr=='c')&&(res2_chr=='c')&&(res3_chr=='c')&&(res4_ch
r=='e'))
{
  res_chr='e';
  min_dist=min4_dist;
}
if (res1_chr=='m')
{
  res_chr='m';
  min_dist=min1_dist;
}
if (res1_chr=='u')
{
  res_chr='u';
  min_dist=min1_dist;
}
if (res3_chr=='u')
{
  res_chr='u';
  min_dist=min3_dist;
}
if (res2_chr=='n')
{
  res_chr='n';
  min_dist=min2_dist;
}
if (res2_chr=='o')
{
  res_chr='o';
  min_dist=min2_dist;
}
if ((res1_chr=='s')&&(res2_chr=='l')&&(res3_chr=='l'))
{
  res_chr='o';
  if ((min1_dist <= min2_dist) && ( min1_dist <= min3_dist))
    min_dist=min1_dist;
```

```
  if ((min2_dist<=min1_dist)&&(min2_dist<=min3_dist))
    min_dist=min2_dist;
  if ((min3_dist<=min1_dist)&&(min3_dist<=min2_dist))
    min_dist=min3_dist;
}
if ((res1_chr=='u')&&(res3_chr=='u'))
{
  res_chr='u';
  if (min1_dist < min3_dist)
    min_dist=min1_dist;
  else
    min_dist=min3_dist;
}
if (res3_chr=='w')
{
  res_chr='w';
  min_dist=min3_dist;
}
if ((res2_chr=='x')&&(res3_chr=='x'))
{
  res_chr='x';
  if (min2_dist < min3_dist)
    min_dist=min2_dist;
  else
    min_dist=min3_dist;
}

printf("MIN DISTANCE = %6.1f;\tCHARACTER=
%c\n",min_dist,res_chr);
/*
printf("MIN DISTANCE1= %6.1f;\tCHARACTER=
%c\n",min1_dist,res1_chr);

printf("MIN DISTANCE2= %6.1f;\tCHARACTER=
%c\n",min2_dist,res2_chr);

printf("MIN DISTANCE3= %6.1f;\tCHARACTER=
%c\n",min3_dist,res3_chr);

printf("MIN DISTANCE4= %6.1f;\tCHARACTER=
%c\n",min4_dist,res4_chr);
*/
}
/* Method 1 of makestring */
#include <stdio.h>
#include <ctype.h>
#include <string.h>
void makestring1(file1, file2, lim)
char *file1,*file2;
int lim;
{
FILE *fp1, *fp2;
int i,j,k;
char line[81];
```

```
int linestr;
int rowflag,colflag;
int ctr[81];
char code[81];
char Table[36] = { '0', '1', '2', '3', '4', '5', '6', '7',
'8', '9',
                   'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h',
'i', 'j',
                   'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r',
's', 't',
                   'u', 'v', 'w', 'x', 'y', 'z' };

fp1=fopen(file1,"r");
fp2=fopen(file2,"w");

for (k=0;k<lim;k++)
{
     for (i=1;i<=81;i++)
          ctr[i]=0;
     rowflag=1;
     i=1;
     while (rowflag)
     {
          fgets(line,81,fp1);
          if (strcmp(line,"\n")==0)
               rowflag=0;
          else
          {
               linestr=strlen(line)-1;
               for (j=1;j<=linestr;j++)
                    if (line[j-1]=='1')
                    {
                         ctr[j]++;
                         ctr[linestr+i]++;
                    }
               i++;
          }
     }
     linestr+=i;
     for (i=1;i<linestr;i++)
     {
          if ( ctr[i] > 35 ) ctr[i] = 35;
          code[i]=Table[ctr[i]];
          fprintf(fp2,"%c",code[i]);
     }
     fprintf(fp2,"\n");
}
fclose(fp1);
fclose(fp2);
}
/* Make string for Diagonal method(Method 2) */
/* Left to Right direction */
#include <stdio.h>
#include <ctype.h>
```

```
#include <string.h>
void makestring2(file1, file2, lim)
char *file1,*file2;
int lim;
{
FILE *fp1, *fp2;
int i,j,k,index;
char line[81];
int linestr;
int rowflag,colflag;
int ctr[81];
char code[81];
char Table[36] = { '0', '1', '2', '3', '4', '5', '6', '7',
'8', '9',
                   'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h',
'i', 'j',
                   'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r',
's', 't',
                   'u', 'v', 'w', 'x', 'y', 'z' };

fp1=fopen(file1,"r");
fp2=fopen(file2,"w");

for (k=0;k<lim;k++)
{
        index=0;
     for (i=1;i<=81;i++)
          ctr[i]=0;
     rowflag=1;
     while (rowflag)
     {
          fgets(line,81,fp1);
          if ( strcmp(line,"\n") == NULL )
               rowflag=0;
          else
          {
               linestr=strlen(line)-1 ;
               for (j=1;j<=linestr;j++)
                    if (line[j-1]=='1')
                         ctr[index+j]++;
          }
          index++;
     }
        linestr+=(index-1);
     for (i=1;i<linestr;i++)
     {
          if ( ctr[i] > 35 ) ctr[i] = 35;
          code[i]=Table[ctr[i]];
          fprintf(fp2,"%c",code[i]);
     }
     fprintf(fp2,"\n");
}
fclose(fp1);
fclose(fp2);
```

```
}

/* Make string for Diagonal method(Method 3) */
/* Right to Left direction */
#include <stdio.h>
#include <ctype.h>
#include <string.h>
void makestring3(file1, file2, lim)
char *file1,*file2;
int lim;
{
FILE *fp1, *fp2;
int i,j,k,index;
char line[81];
char m;
int  n;
int linestr;
int rowflag,colflag;
int ctr[81];
char code[81];
char Table[36] = { '0', '1', '2', '3', '4', '5', '6', '7',
'8', '9',
                   'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h',
'i', 'j',
                   'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r',
's', 't',
                   'u', 'v', 'w', 'x', 'y', 'z' };

fp1=fopen(file1,"r");
fp2=fopen(file2,"w");

for (k=0;k<lim;k++)
{
        index=0;
     for (i=1;i<=81;i++)
          ctr[i]=0;
     rowflag=1;
     while (rowflag)
     {
          fgets(line,81,fp1);
          if ( strcmp(line,"\n") == NULL )
               rowflag=0;
          else
          {
               linestr=strlen(line)-1 ;
               n=linestr/2;
                        for (j=1;j<=n;j++)
                        {
                            m=line[j-1];
                    line[j-1]=line[linestr-j];
                    line[linestr-j]=m;
               }
               for (j=1;j<=linestr;j++)
                    if (line[j-1]=='1')
```

```
                              ctr[index+j]++;
                    }
                    index++;
          }
             linestr+=(index-1);
          for (i=1;i<linestr;i++)
          {
                    if ( ctr[i] > 35 ) ctr[i] = 35;
                    code[i]=Table[ctr[i]];
                    fprintf(fp2,"%c",code[i]);
          }
          fprintf(fp2,"\n");
}
fclose(fp1);
fclose(fp2);
}

/* Method 4 of makestring */
#include <stdio.h>
#include <ctype.h>
#include <string.h>
void makestring4(file1, file2, lim)
char *file1,*file2;
int lim;
{
FILE *fp1, *fp2;
int i,j,k;
char line[81];
char m;
int n;
int linestr;
int rowflag,colflag;
int ctr[81];
char code[81];
char Table[36] = { '0', '1', '2', '3', '4', '5', '6', '7',
'8', '9',
                   'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h',
'i', 'j',
                   'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r',
's', 't',
                   'u', 'v', 'w', 'x', 'y', 'z' };

fp1=fopen(file1,"r");
fp2=fopen(file2,"w");

for (k=0;k<lim;k++)
{
     for (i=1;i<=81;i++)
          ctr[i]=0;
     rowflag=1;
     i=1;
     while (rowflag)
     {
          fgets(line,81,fp1);
```

```
            if (strcmp(line,"\n")==0)
                 rowflag=0;
            else
            {
                 linestr=strlen(line)-1;
                 n=linestr/2;
                 for (j=1;j<=n;j++)
                 {
                   m=line[j-1];
                   line[j-1]=line[linestr-j];
                   line[linestr-j]=m;
                        }
                 for (j=1;j<=linestr;j++)
                      if (line[j-1]=='1')
                      {
                           ctr[j]++;
                           ctr[linestr+i]++;
                      }
                 i++;
            }
       }
       linestr+=i;
       for (i=1;i<linestr;i++)
       {
            if ( ctr[i] > 35 ) ctr[i] = 35;
            code[i]=Table[ctr[i]];
            fprintf(fp2,"%c",code[i]);
       }
       fprintf(fp2,"\n");
}
fclose(fp1);
fclose(fp2);
}
```

BIBLIOGRAPHY

Abu-Mostafa, Y.S. and St. Jacques, J.M., "Information Capacity of the Hopfield Model, IEEE Trans. Infa. Th. 31(4), 461-464.

Abu-Mostafa, Y.S., "Neural Networks for Computing", AIP Conf. Proc. 151, Snowbird, Utah, April 13-16, 1986, pp. 1-6.

Baldi, P., "Neural Network, Orientations of the Hypercube and Algebraic Threshold Functions", IEEE Tran Information Theory, vol. IT-34, pp. 523-530, 1988

Bunke, H., "String Matching For Structural Pattern Recognition", in "Syntactic and Structural Pattern Recognition Theory and Applications", ed. H. Bunke and A. Sanfeliu, World Scientific, Singapore, 1990.

Chao, D.Y. and Wang, D.T., "Proof of Some Higher Order Neural Network Properties", Intelligent Engineering Systems Through Artificial Neural Netowrks, ed. C.H. Dagli, S.R.T. Kumara, and Y.C. Shin, ASME Press, pp. 211-216, 1991

Chao, D.Y. and Wang, D.T., "Enhancement of Memory Capacity of Neural Networks", to appear in Proc. IEEE/RSJ Int'l Conf Intel. Rob. Syst. (refereed), July, 1992

Chao, D.Y. and Wang, D.T., "Pseudo-Inverse with Increasing Threshold : an Error-Recovery Pattern Recognition", submitted

Hall, P.A.V., and Dowling, G.R., "Approximate String Matching", Computing Surveys, Vol 12, pp. 381-402

Chen, H.H., Lee, Y.C., Sun, G.Z., Lee, H.Y., Maxwell, T., Giles, C.L., "High Order Correlation Model for Associative Memory", AIP Conf. Proc. 151, Snowbird, Utah, April 13-16, 1986, pp.86-89.

Hopfield J.J., Neural Networks and Physical Systems with Emergent Collective Computational Abilities", Proc Nat'l Academy Sci., USA, 79, 2554-2558

Kamp, Y and Hasler, M., "Recursive Neural Networks for Associative Memory", Wiley, New York, New York.

Kohonen, T., "Representation of Associated Data by Matrix Operators", IEEE Tran Comp. Vol C-22, pp. 701-702, 1973.

Personnaz, L., Guyon, I., Drefus, G., "Information Storage and Retrival in Spin-Glass Neural Networks", J Phys Lett., Vol, 46, pp.359-365, 1985.

Personnaz, L., Guyon, I., and Drefus, G., "Neural Networks for Associative Memory Design, "In H. Haken, ed. "Computation system - Natural and Artificial", pp. 142-151, Springer Verlag, Berlin, 1987.

Wagner, R.A., and Fischer, M.J., "The String-to-String Correcting Problem", JACM, Vol 21, pp.168-173, 1974.