An Artificial Neural Network Experiment

Robert Jay Brown III *

May 14, 2003

Abstract

This paper describes an experimental computer program that could serve as one component of a computer vision system. The program is an artificial neural network learning machine implemented as a committee network of threshold logic units to function as a trainable pattern recognizer for visual images composed of a rectangular array of pixels. Each pixel contains a single number representing the gray scale value of that region of the field of view.

Introduction

A robot vision system consists of many components:

Image capture, usually with a video camera, charge coupled device, laser or microwave radar, scanning sonar, etc., in which the image is converted from light or other radiation to an analog electrical signal;

Image digitization and signal processing, in which the image is divided into a set of picture elements, or pixels, each of which is assigned a value representing the brightness, color, etc. of that part of the image;

Region, edge, and boundary detection, in which the distinct visual elements of the image are separated;

Image scaling and alignment, in which the digitized image is rotated, translated, and otherwise transformed to place it into some standard position and size; and

Image recognition, in which the preprocessed image is submitted to a pattern recognition algorithm to allow the robot to categorize it, that is, recognize what the image represents.

Only the last of these problems will be considered in this paper.

Neurophysiology

`SILOAM` operates by modeling on the computer one possible organization of the actual neural structure of the brain. Consider a single nerve cell (figure 1). During World War II, a physiologist and a mathematician worked together to try to create a model of the brain based on anatomical and physiological experimental findings. The team of McCulloch and Pitts started by modeling the single nerve cell, or neuron. Their model is known today as the "McCulloch-Pitts neuron" (figure 2).

Microscopic studies reveal that the nerve cell is composed of a cell body, or cyton, many input fibers, or dendrites, and a single output fiber, or axon, which branches to send signals to the dendrites of other nerve cells. Physiological studies that selectively stimulate sets of dendrites while observing the axon yield the following result: some dendrites excite the neuron to produce an output (we say the neuron "fires"), and other dendrites inhibit the firing.

The cell either fires or does not fire. There is not any specific tie between which inhibitory input cancels an exitory input. Each input has a "weighting factor" associated with it: excitory inputs have a positive weight, and inhibitory inputs have a negative weight. The neuron, supposedly in the cyton, combines all these inputs to produce the output. Thus it appears that the neuron adds up all the weights associated with inputs that are stimulated, and if the result exceeds a certain threshold then the neuron fires.

*Mr. Brown is a computer scientist involved in the design of electronic survieliance, intercept, and cryptography systems, robotics, avionics, medical equipment, and artificial intelligence systems. He holds a BA in mathematics from Dowling College, and is a working on an MS in mathematics from Florida Atlantic University.

Threshold Logic

This model of a neuron is referred to as a threshold logic unit, or more simply TLU (figure 3). It is not difficult to construct a TLU out of readily available hardware components: a Schmidt trigger connected in series after an operational amplifier wired to operate as an analog summer will suffice (figure 4). The weights are the gains determined by the proportionality factors of the scaling resistors at the plus or minus inputs to the op-amp. Adjustable weights may be realized by using potentiometers with the wiper connected to the stimulus input, with one end of the resistance element connected to the plus input of the op-amp, and the other end connected to the minus input. In the present case, we choose to simulate the operation of a TLU by software. It's cheaper, and as we shall see, it allows the program to adjust its own weighting factors. This is necessary if the TLU is to be automatically trained, rather than "tweaked" into alignment by a skilled technician.

The TLU is a physical embodiment of a linear equation formed by setting an inner product, or metric (measure of distance), to zero. The solution set for the equation thus formed defines a cleaving plane that acts as the boundary between two half-spaces in the weight space of all possible weight sets that could make up the weights for the input of a TLU. The cleaving planes, being the solution to a homogeneous linear equation, must pass through the origin. When the equality is changed to an inequality, the sign of the inequality indicates on which side of the cleaving plane the weight point lies.

`SILOAM` contains many TLU's. Each has m times n inputs to "see" the m by n image array we shall present to it. In addition, each TLU has an additional input which is always excited. This extra input provides a "reference point" or "bias" (analogous to the DC component of a Fourier series) to set the threshold that determines the firing point of the TLU. This input is necessary to homogenize the linear equation formed by the inner product used to compute the metric. Without it, an input of all zeros would be degenerate, and the training algorithm would fail to converge.

Pattern-Space Geometry

The task of computing the output of a TLU is performed by the vector operation of taking the "dot product" of the stimulus vector and the weight vector associated with the TLU. The sign of the result determines the output: positive sign means the nerve has fired, negative means it hasn't. Each weight vector may be interpreted as a point in hyperspace, or weight space. If we look at the dot product formed between the input pattern vector and the TLU weight vector, we see that if the elements of the pattern vector are viewed as constants, and the weight vector elements are viewed as a variables, then if the dot product is set equal to zero, this dot product forms a linear homogeneous equation. The situation in 3-space is:

$Ax + By + Cz = 0$

A, B, and C are the components of the weight vector, and x, y, and z are the components of the pattern to be recognized. Remember that z is set to a constant value of one. The solution set defines a plane that passes through the origin (figure 5). The plane forms a pattern surface which cleaves weight space into 2 half-spaces: this places one set of possible TLU weights on one side of the pattern plane and one set on the other side. Any given weight point will be on either the negative or the positive side of the pattern plane. (The two's complement arithmetic of the computer makes it convenient to consider a point lying on the plane itself to be on the positive side of the plane.)

The absolute value of the dot product is proportional to the perpindicular distance from the weight point to the pattern plane. Thus a given pattern hyperplane divides weight hyperspace into 2 half-hyperspaces. The dot product of the pattern vector with the weight point returns the distance from the weight point to the pattern plane. These weight points are defined by the weights for each of the inputs to the TLU; the coordinates of the weight point are just the values of the weights for the TLU.

By this convention, we may visualize each TLU as being represented by a point in this weight space. The dot product of the weight point with the augmented pattern vector is the perpendicular distance from the weight point to the pattern plane. If this quantity is positive, then the TLU "recognizes" the pattern; if it is negative, then it does not recognize it (figure 6).

The TLU is a pattern dichotomizer: its corresponding weight point in weight space is on the positive side of some pattern hyperplanes, and on the negative side of the other pattern hyperplanes. Thus it divides all pattern planes in weight space into 2 classes or sets: those it is on the positive side of, which it recognizes, and those it is on the negative side of, which it does not recognize.

Artificial Neural Networks

Now, how about more difficult cases, where a simple linear function cannot separate the categories? We will first explore a "yes/no" decision from a single network. We will use one possible network of TLU's, called the committee network (figure 7). The committee network is a type of layered machine. A committee of TLU's is composed of

an odd number of TLU's, each presented with the same input pattern vector. Each TLU in the committee decides whether the pattern is one that it recognizes, and casts a vote accordingly. Then a chairman TLU counts the votes. The chairman's inputs are the outputs of the committee members. It's weights are fixed at 1 for all of its inputs, so it simply functions as a vote counter for the rest of the committee. By training multiple networks independently, we may increase the number of recognizable classes to any power of 2 (figure 8).

We have developed a democratic machine: how can such a thing work? We can show how a democratic committee with a majority rule voting system can make a substantial improvement in our pattern recognizer. The divisions formed by each of the TLU's in our committee can occur at different places. Through proper training (to be described later), the voting TLU's in the committee can be made to form point clusters in weight space such that a majority of TLU weight points will always be on the proper side of every presented pattern hyperplane (figure 9). Thus we can select the weight points such that a majority of TLU's will vote "yes", and this defines the set of patterns that the committee as a whole will recognize.

To train such a committee, we present it with a pattern vector and observe the result. If the committee returns the correct answer, we present another pattern; if not, we must correct it. This is done by adjusting the weights to produce a more favorable vote (somewhat equivalent to lobbying in legislative processes, where the TLU plays the part of the politician). This does not insure that the correct decision will be obtained the next time the committee sees this pattern, but the vote will be closer. Since we insist on an odd number of TLU's in a committee (exclusive of the chairman), we can never have a tie. What we do is convert one TLU at a time to a more "enlightened" view. By repeating this process enough, the committee will return a favorable decision. When this occurs, we say that the network has been trained to recognize the pattern.

The Training Algorithm

How do we go about finding the correct TLU to adjust, and how do we perform the adjustment? We pick that TLU which voted wrong which was the least sure of his vote. This means that we pick the TLU that had the wrong sign for its dot product, but the magnitude of the dot product was the minimum of all the TLU's with the wrong output. This corresponds to selecting the weight point closest to the pattern hyperplane, but on the wrong side of it. Now we know which TLU to work on, but how do we adjust the weights to produce the desired effect? We move the weight point for the selected TLU along the perpendicular from the weight point to the pattern hyperplane, towards the pattern hyperplane, through to the other side of the pattern hyperplane, thereby changing the TLU's classification of the pattern.

We actually move the weight point by an amount determined by a constant, the correction fraction, times the distance from the weight point to the pattern hyperplane. This constant must lie between 1 and 2 for the training algorithm to converge (figure 10). If it is greater than 1, then the weight point will move to the other side of the pattern hyperplane; if it is less than 1, then the weight point will move towards the pattern hyperplane, but not through it. In this case, the training algorithm will not converge and training will never be accomplished because the weight point will always be on the wrong side of the pattern plane even though it gets constantly closer to it. This technique is called fractional correction. If the distance moved is the least integer such that the pattern plane will be crossed, this choice results in a training strategy known as absolute correction. The simplest technique is constant correction, where a constant distance is always moved. Such strategies allow for the use of integer arithmetic resulting in faster execution and simpler hardware.

Fixed increment correction with binary images using 8-bit signed integer weights lends itself to cheap parallelism. A separate Intel 80C51 microcomputer on a chip could be used for each TLU, taking weight points out of an array in on-board ROM. Using 6 committees of 7 voting TLUs each, plus a single 80C51 to count all the votes and act as central control results in 6*7+1=43 80C51 processor chips. These microcomputer chips cost less than three dollars in quantity. A system such as this should be able to perform real-time optical text scanning of printed literature. Using surface mount or hybrid packaging methods, the device should be as portable as a walkman radio. A speech synthesizer could serve as an output device, receiving ASCII text from the pattern recognizer. Voila! A reading device for the visually handicapped. To this end, the performance of an 8-bit integer network with fixed increment correction has been examined with encouraging results. The TMS-320 series of digital signal processors from Texas Instruments may prove less expensive than the 8051 array. Although the 320 is more expensive, it is much faster than the 8051, and the overall system cost may be less.

The Experiment

The SILOAM source file (Listing 1) is heavily commented, and should be easily understood when read in conjunction with this article. It was written with flexibility, portablilty, and readability in mind. There are actually 3 versions of SILOAM, a floating point version, a 16 bit integer version, and an 8 bit integer version. The symbol `ELTYPE` is defined on the compiler invocation line, and determines the type definition for an element of a weight point vector.

The pattern presentation order for training is selectable by the -o option. Initial conditions for the weight points are selected by the -r option. The correction method is selectable by the -a, -i, and -f options, which specify absolute, fixed increment, or fractional correction. The level of detail for logging is selectable by the -l option: -l0 displays only final results, -l3 displays the most detail.

The program has been run on a small pattern file, representing a binary image of each of the numbers 0 thru 9 (Listing 2). It has also been successfully taught the entire uppercase alphabet. The alphabet pattern file was generated by rasterizing characters from the Hershey character database of the National Bureau of Standards. The entire ASCII character set was generated in this fashion as a high resolution dot matrix raster, and was taught to `SILOAM`. An example of a character image from this file is shown (Listing 3). The output produced by a run of the program is shown (Listing 4).

It is interesting to observe that all of the binary images of the character sets could be learned by a network of only one TLU per committee. When this is the case, the pattern set is said to be "linearly seperable". A pattern file of random analog pixel values was generated, comprising 100 images. In this case, a single TLU could not learn the pattern set. Three TLUs were required, and different training methods produced radically differnt results. Fixed increment performed quite poorly. Absolute correction did somewhat better, but fractional correction did the best. Values of the correction fraction closer to 2 seemed to perform better and resulted in faster convergence. In fact, convergence was even achieved with values above 2, although when they got up to about 2.5, convergence failed.

`SILOAM` will actually get confused and forget things it has already learned in the process of trying to learn new things. This is similar to what every teacher or parent knows about the learning phenomenon in children. It is shown in Nilsson's book, however, that the training procedure will converge to the desired result, given that a suitable distance to move the weight point is chosen, and that the capacity of the machine (related to the number of TLUs per committee and the number of patterns to be recognized) is not exceeded. This is known as the Fundamental Training Theorem, or the Perceptron Convergence Theorem.

Despite the impressive performance of this simple network, it has some serious theoretical shortcomings. It cannot learn a simple exclusive OR function. That the network is deficient can be shown with a geometric proof. Since there are 2 variables input to an XOR, a 2 by 1 input pattern is needed. This results in 3 dimensions for the pattern space. Since there are 4 combinations of the inputs, there are four pattern planes. All four planes pass through the origin. If a sphere is imagined about the origin, these 4 planes intersect the sphere in 4 great circles. These 4 circles each intersect each other. If the south pole of the sphere is placed on the origin of a 2 dimensional graph, and the surface of the sphere projected onto the plane, as in a polar projection of the globe, 4 intersecting circles result, one centered in each quadrant of the plane. Their common intersection is around the origin of the graph. If we count the number of distinct regions these circles divide the plane into, we get 14, but there are 16 possible combinations of accepting and rejecting 4 distinct patterns composed of 2 independent variables. The exclusive OR and the exclusive NOR, or equivalence relation, are these missing regions. A similar argument in higher dimensions can be used to show that all such networks are likewise deficient in not being able to learn all possible pattern sets that they could be confronted with.

Rumelhart and McCleland show, however, that other network topologies, neural activation functions, and training algorithms, especially the gradient descent training method, are capable of producing all possible boolean switching functions. Their book is highly recommended for anyone interested in persuing the study of artificial neural networks in depth.

Topics for Further Investigation

One idea that needs to be explored is recognizing patterns over time, that is, sequences of patterns. One idea might be to incorporate some sort of feedback into the network to provide a memory capability. The output bit vector could be concatenated with the pattern input vector to provide an input to the network that was a function not only of the current input, but also of the previous output.

Additional exploration needs to be done with non-binary pattern elements. Remember that the recognition process does not require the elements of the pattern vector to be ones and zeroes; any real values will still satisfy the vector geometrical constraints and should be recognizable with essentially the same algorithm.

Geoffrey Hinton discussed some of his work with artificial neural networks at the AAAI-86 conference in Philadelphia last August. He showed how a real-valued, differentiable activation function could be trained by the method of gradient backflow to form its own independently developed internal abstractions. His experiment involved learning two similar family trees. The network formed, on its own and without being taught explicitly by the examples, the abstract concepts of generation and various family relations. It was also able to generalize its experience to determine the relations between individuals it had not been previously introduced to. It got better than 3 out of 4 new problems correct. Hinton shows that a statistical correlating recognizer would fail this test, and that true generalization of induced abstractions is being performed.

Neurophysiology Revisited

What can we learn about the natural mind based on our model neural network? Psychiatrists treat mental illness with drugs such as phenathiazines and lithium, and electroshock therapy. Electroshock therapy is assumed to destroy connections, thereby altering weights by setting them to zero. Phenathiazines affect the release of neurotransmitters such as seratonin, norepinephrine, and dopamine. A change in these would have the same effect as altering the weights at the inputs to the neuron. Lithium is metabolized by the body's electrochemistry in the same manner as sodium, but behaves differently in nerve conduction and firing potential. Thus lithium acts as an inert place holder for the neuroactive sodium in the sodium-potasium complex. The presence of lithium would affect the threshold of the neuron, but this is just another weight in our model. Thus these psychoactive drugs are trying to counteract a medical problem that is manifested in a perterbation of the weights of the neuron. If the drugs are under prescribed, the desired effect will not be achieved, and if they are overprescribed, the wieghts may be totally scrambled, resulting in a worsening of symptoms.

Hardware Implementations

Recently, threshold logic has been receiving a lot of attention from the press. The front page of the Electronic Engineering Times carried an article on February 3, 1986, titled "Neural research yields computer that can learn" that described research into a speech learning program based on Hopfield networks that apparently makes use of time feedback techniques similar to those outlined above.

The very next week an article appeared in the same publication that touted threshold logic as the key to optical computers. This article gave some details on how electro- optical technology can implement threshold logic gates with an enormous number of inputs. The week after that, an article appeared giving details of a Gallium Arsenide TLU that uses the analog addition of the brightness of lightwaves to perform the summing operation, so that the device is essentially a threshold detector with a photo-resistor for an input. This TLU implementation does not increase in complexity no matter how many inputs it recieves: they are just lights shining on its single photo-resistor. The output of this device is a single solid state laser. This last article also described a holographic optical interconnect scheme that is very interesting. The output lasers reflect off of a hologram placed over the chip. The hologram acts as a phased array reflector that directs each output laser's light only to those photo-resistors that are supposed to be connected to that output. In this way, the logic signals travel at the speed of light without wasting any chip real estate on signal interconnect lines. The logic may now be placed closer together, and a three dimensional medium is available for interconnect wiring instead of the two dimensional masks of current day chips. Gallium Arsenide chips are available in production today that operate at speeds of 20 GHz. With more closely spaced circuits, and more flexible design rules for interconnects, a tremendous increase in speed should be shown by these new devices.

This kind of hardware is just what is needed to take these learning systems from the several seconds per iteration speed zone into the picoseconds per iteration arena. We may certainly expect to see more of threshold logic learning systems in the future if this hardware implementation effort succeeds.

Acknowledgements

Many people and organizations have helped make this work possible. My thanks go first to my wife Darlene for her patience and her proofreading assistance; however, any errors that remain must be my own. Paul Gothier of IBM in Boca Raton showed me how these networks could be viewed as digital signal processing systems. Dr. J. A. S. Kelso of the Center for Complex Systems at Florida Atlantic University has suggested the application of Chaotic Dynamics theory to the convergence of the training process. Dr. Angel Esteves of the Henderson Mental Health Clinic explained the action of the various neurotransmitters and their theraputic effects. Dr. Bahram Ravani of the Mechanical Engineering Department of the University of Wisconsin at Madison encouraged a previous version of this paper to be written. Dr. Frederick Hoffman of the Department of Mathematics of Florida Atlantic University has been a source of guidance and also proofread an earlier version of this paper. Danny Bennett of Computers For Business prompted me to write an earlier version of this paper to explain how neurological concepts could be embodied in a computer program. Dr. Tadeo Ogura of South Oaks Psychiatric Hospital helped me understand some of the aspects of these networks. Ken Moskowitz helped me debug the original Fortran version of this program. Dan Harmon introduced me to the concepts of neural networks by showing me a crumpled yellow teletype listing of a BASIC program that he recieved in the mail from a friend at the University of Western Austrailia.

Thanks go to Perry Offshore, Inc. of Riviera Beach, Florida, Associated Data Consultants, Inc. of Boca Raton, Florida, and Computers For Business, Inc. of Hollywood Florida for the use of computer equipment and software during the development of this program. Reuters News Service of New York provided the use of computer equipment and software during the development of the original FORTRAN version of this program.

Bibliography

Barron, Roger L. 1975. Learning Networks improve Computer-aided Prediction and Control. *Computer Design, August 1975.*

Boas, Mary L. 1966. Mathematical Methods In The Physical Sciences, Chapter 5. *John Wiley & Sons, Inc. New York.*

Brown, Chappell, 1986. Neural Research Yields Computer That Can Learn. *Electronic Engineering Times, Feb. 3, 1986, p 1.*

Brown, Chappell, 1986. Threshold Logic Gates: Key To Making Optical Computers Reality? *Electronic Engineering Times, Feb 10, 1986, p 51.*

Brown, Chappell, 1986. Researchers see lightwaves at end of VLSI tunnel. *Electronic Engineering Times, Feb 17, p 39.*

Brown, Chappell, 1986. CCDs, NMOS team to build neural net models on chip. *Electronic Engineering Times, Sept. 8, 1986 p 33.*

Buzan, Tony and Terence Dixon. 1977. The Evolving Brain. *Holt, Rinehart & Winston. New York.*

Hinton, Geoffrey E. 1986. Learning Distributed Representations of Concepts. *Proc. of the 8th annual conf. of the Cognitive Science Soc. Amherst Mass. Aug. 1986. Lawrence Erlbaum Assoc.*

Hopfield, J. J. 1982. Neural Networks and Physical Systems with Emergent Computational Abilities. *In "Proceedings of the National Academy of Sciences", Washington DC, Vol. 79 pp 2554-2558.*

Kosko, Bart. 1986. Artificial Neuron Systems Gain Support. *Applied Artificial Intelligence Reporter, Vol. 3 No. 9 p 14.*

Masland, Richard H. 1986. The Functional Architecture of the Retina. *Sci. Am. Vol. 255 No. 6 pp 102-111.*

McCorduck, Pamela. 1979. Machines Who Think: a personal inquiry into the history and prospects of artificial intelligence. *W. H. Freeman & Co. San Francisco.*

McCulloch, W. and W. Pitts, 1943. A Logical Calculus of the Ideas Immanent in Nervous Activity. *Bull. of Math. Biophysics, Vol. 5 pp 115-133.*

Miller, Richard K. 1986. Optical Computers: the next frontier in computing. *SEAI Technical Publications, Madison Georgia.*

Nilsson, Nils J. 1965. Learning Machines: foundations of trainable pattern-classifying systems. *McGraw-Hill New York.*

Rumelhart, David E., James L. McCleland, and the PDP Research Group. 1986. Parallel Distributed Processing. *MIT Press, Cambridge Mass.*

Schwartz, Tom J. 1986. IBM research yields artificial neural net workstation. *Electronic Engineering Times, Sept. 1, 1986 p 67.*

Silbar, Margaret L. 1971. In Quest of a Human-like Robot. *Analog. Vol. 88 Nov. 1971 pp 77-98.*

Program Source Code

The original program was written in 1985 and 1986. That makes it paleolithic C. This version has been cleaned up "just enough" to make it work on gcc with Red Hat Linux 8.0

```
#define PGM_ID "SILOAM Gnu gcc version for Linux 10/24/97"
/*-------------------------------------------------------------------------
 *              An Adaptive Template Matching Image Categorizer
 *              This Is An Experimental Computer Vision Program
 *.........................................................................
 *
 *                 SSSS  III  L       OOO     A    M   M
 *                S       I   L      O   O   A A   MM MM
 *                 SSS    I   L      O   O  A   A  M M M
 *                    S   I   L      O   O  AAAAA  M   M
 *                SSSS   III  LLLLL   OOO   A   A  M   M
 *                  i     m     e      n      d      a
 *                  m     a     a             a      c
 *                  p     g     r             p      h
 *                  l     e     n             t      i
 *                  e           i             i      n
 *                              n             v      e
 *                              g             e      r
```

```
 *                                                    y
 *
 *.....................................................................
 *
 *      This program implements a trainable pattern classifier as
 *  a committee network of threshold logic units.  It learns to
 *  recognize patterns by being trained from a set of prototype
 *  patterns presented in a training file.  The training file is
 *  organized as a set of visual images represented as an orthogonal
 *  array of picture elements, or pixels.  Each pixel is a number
 *  representing the gray-scale value of that point in the image.
 *  Associated with each pattern is a number, or tag, that represents
 *  the category to which that pattern belongs.
 *
 *.....................................................................
 */

/*.....................................................................
 *
 *     Written:  During the period 1975 thru 1986,
 *               in BASIC, FORTRAN, and C,
 *               on various computers including
 *               PDP-11/35, PDP-11/45,
 *               8080, 8085, and Z-80 CP/M 2.2 machines,
 *               8086, 8087, 8088, 80186, 80286, 80287,
 *               IBM-PC, XT, and AT and Clones.
 *  This program is an on-going experiment.
 *
 *          By:  R. J. Brown
 *               Elijah Laboratories Inc.
 *               P. O. Box 833
 *               Warsaw KY 41095
 *               1 606 567-4613
 *
 *   Ownership:  I hereby place this program in the public domain.
 *
 *      System:  Samsung 8 MHz 80286 IBM-PC/AT clone
 *               with 80287, DOS 3.3
 *
 *    Compiler:  Microsoft C Version 5.0
 *
 *.....................................................................
 *
 *      "And as Jesus passed by, he saw a man which was BLIND from
 *      his birth...  And said unto him, Go, wash in the pool of
 *      Siloam...  He went his way therefore, and washed, and came
 *      SEEING."
 *                  John 9:1-7
 *
 *      "Train up a child in the way he should go: and when he is
 *      old, he will not depart from it."
 *                  Proverbs 22:6
 *
 *      "And God said, let us make man in our image, after our
 *      likeness."
 *                  Genesis 1:26
 *
 *      "Thou shalt not make a machine in the image of man."
 *                  Frank Herbert -- Dune
 *
```

```
 *.....................................................................
 */

/************************************************************************
 *
 *        O p e r a t i n g    S y s t e m    I n t e r f a c e
 *
 ************************************************************************/

/* #include <sys/types.h> */
#include <stdio.h>              /* needed for stream input/output       */
#include <math.h>/* for square root routine */
#include <time.h>               /* for getting current time of day      */

#define index strchr /* ANSI name for MicroSoft C ver 5.0 */


/************************************************************************
 *
 *      T y p e    D e f i n i t i o n s    &    # d e f i n e ' s
 *
 ************************************************************************/

#define FALSE     0             /* boolean constant for 'false'         */
#define TRUE     !FALSE         /* boolean constant for 'true'          */

#define void                    /* function that returns no value       */

#define forall(index,limit)\
for((index)=0;(index)<(limit);(index)++)    /* looping word     */

#define kase(id,stmt) \
case(id): { \
    stmt; \
    break; \
    } /* shorthand form for case statement */

#define u(x) ((unsigned)(x))    /* shorthand for '(unsigned)' cast      */

typedef unsigned char  byte;    /* an 8-bit byte of storage             */
typedef unsigned int   word;    /* a 16-bit word of storage             */

typedef word     boolean;       /* a decision variable,
 * 'true' or 'false value only
 */

typedef ELTYPE     element;     /* an element is a real number          */
typedef DOTYPE     DOT; /* type of a dot product may be bigger! */
typedef element   *vector;      /* a vector is a set of elements        */

typedef vector    tlu;          /* a tlu is a vector                    */

typedef struct {                /* the collection of                    */
tlu     *wtpt;          /* a set of tlu weight points,          */
DOT     *dot;           /* and dot product save cells           */
}         committee;    /* is a committee                       */

typedef char      *pointer;     /* a general pointer to whatever...     */
```

```
/*****************************************************************
 *
 *        G l o b a l   V a r i a b l e   D e f i n i t i o n s
 *
 *****************************************************************/

FILE    *pat,           /* the input training pattern file             */
*fopen();       /* the file opener                              */

byte    patname[64],    /* ascii filename of input file                */
*index();       /* string search library function               */

int     ncom,           /* number of committees in the network         */
patwide,        /* pattern width in pixels                      */
pathite,        /* pattern height in pixels                     */
pats_so_far,    /* how many patterns in file so far             */
pats_missed,    /* how many patterns were mis-recognized so far */
missed,         /* # of patterns missed on this pass            */
tlus_trained,   /* how many tlu's have been adjusted so far     */
npass,          /* number of current pass thru pattern file     */
log_level,      /* level of detail for run-time logging         */
dim,            /* number of elements in a vector (dimension)   */
ntlu,           /* number of tlus per committee                 */
corr_incr,      /* fixed increment correction constant          */
*vote;          /* pointer to vote count array                  */

boolean goofed,         /* mis-recognition indicator for training loop  */
start_over,     /* select start over on error training strategy */
absolute,       /* flag for absolute correction training method */
*decsn,         /* pointer to network's decision array          */
*class;         /* pointer to class (category) array            */


DOT     patmag;         /* pattern magnitude (used for training)       */

element fraction,       /* correction fraction for training            */
maxel=0; /* maximum element in a weight point */
radius;         /* average radius (distance from origin)
 * of tlu weight point at initialization
 */

vector  pattern;        /* pointer to current input pattern            */

committee *net;         /* pointer to network as an array of committees */


/*****************************************************************
 *
 *               L i b r a r y   R o u t i n e s
 *
 *****************************************************************/

/*extern double  sqrt();  /* square root library function               */
/*extern pointer calloc();/* memory allocation library function         */
/*extern long    time();  /* benchmark timing routine                   */


/*****************************************************************
 *
 *      B A N N E R  --  D i s p l a y  P r o g r a m  I. D.
```

```
 *
 ************************************************************************/

void banner() { /* display program identification information          */

    printf("\n%s",PGM_ID);     /* Program Identification is #define'd
* at top of source file
*/

    printf("\nWritten by:  R. J. Brown, Elijah Laboratories Intn'l");
    printf("\nThis program is in the Public Domain.\n");
}


/************************************************************************
 *
 *              H E L P    D i s p l a y    S c r e e n
 *
 ************************************************************************/

void help() {   /* some user friendly help for the uninitiated !        */

    printf("Simple Image Learning On Adaptive Machinery\n");
    printf("An Adaptive Template Matching Image Categorizer\n");
    printf("\n");
    printf("    R. J. Brown, Elijah Laboratories International\n");
    printf("    5150 W. Copans Rd. Suite 1135, Margate FL 33063\n");
    printf("\n");
    printf("usage:       siloam <options> filename[.ext]\n\n");
    printf("where:  filename -- is the input pattern file.\n\n");
    printf("options:  -r##.# -- gives initialization radius.\n");
    printf("            -t## -- gives number of TLUs per committee.\n");
    printf("              -o -- start over on error,\n\n");
    printf("choose one: -i## -- fixed increment correction, ## = incr.\n");
    printf("              -a -- absolute correction.\n");
    printf("          -f##.# -- fractional correction, ##.# is lambda.\n");
    printf("             -l# -- logging level: 0=least; 3=most.\n");
    exit(0);
}


/************************************************************************
 *
 *      S I G N  --  T h e   S i g n   O f   A n   E l e m e n t   +/- 1
 *
 ************************************************************************/

int sign(x)     /* return the sign of a number as plus or minus one     */
element x;      /* argument is an element                               */
{
    return( x<(element)0 ? -1   /* if number is negative, return -1     */
                 :  1 );/* else return +1                       */
}


/************************************************************************
 *
 *    I S I G N  --  T h e   S i g n   O f   A n   I n t e g e r   +/- 1
 *
 ************************************************************************/
```

```
int isign(x)    /* return the sign of a number as plus or minus one     */
int x;          /* argument is an integer                               */
{
    return( x<0  ? -1           /* if number is negative, return -1     */
 :  1 );        /* else return +1                       */
}


/*********************************************************************
 *
 *   A B S  --  A b s o l u t e   V a l u e   Of   A n   E l e m e n t
 *
 *********************************************************************/

element eabs(x)     /* the absolute value of an element                */
element x;          /* argument is an element                          */
{
    return( x<0 ? -x      /* if number is negative, make it positive    */
        :  x );   /* else return it like it is                 */
}


/*********************************************************************
 *
 *  I A B S  --  A b s o l u t e   V a l u e   O f   A n   I n t e g e r
 *
 *********************************************************************/

int iabs(x)          /* the absolute value of an integer               */
int x;               /* argument is an integer                         */
{
    return( x<0 ? -x        /* if number is negative, make it positive  */
:  x );     /* else return it like it is               */
}

/*********************************************************************
 *
 *              A L P H A  --  S t e p   F u n c t i o n
 *
 *********************************************************************/

int alpha(x)        /* step function return zero or one                */
int x;              /* argument is an integer (in this program...)     */
{
    return( x>0 ? 1        /* if argument strictly positive, return one */
: 0 );     /* else return zero                         */
}


/*********************************************************************
 *
 *       M O V E  --  S t r i n g   M o v e   F u n c t i o n
 *
 *********************************************************************/

char *move(src,dst)     /* move a string returning ptr to end of result */
char *src,*dst;         /* pointers to source & destination strings     */
{
    while(0!=((*dst++)=(*src++)));  /* copy bytes until end of source   */
```

```
    return(--dst);                  /* return ptr to end of destination */
}


/*****************************************************************
 *
 *    R A D I U S    S T A T I S T I C S  --  S u m m a r y    I n f o
 *
 *****************************************************************/

void radius_statistics() {   /* show how weight points are distributed  */

    element r,          /* current radius accumulator                   */
    *pe;        /* pointer to current element                   */
    float   mu=0,       /* mean of radii                                */
    sigma=0;    /* standard deviation of radii                  */

    committee *pc=net;  /* pointer to current committee                 */

    vector  *pt;        /* pointer to current tlu                       */

    int     c,              /* committee loop counter                   */
    t,              /* tlu loop counter                             */
    e,              /* element loop counter                         */
    n=ncom*ntlu;    /* number of tlu's altogether                   */

    forall(c,ncom) {            /* for all committees...                */
pt=pc++->wtpt;              /* point to first tlu                       */
forall(t,ntlu) {            /* for all tlu's...                         */
    pe=*pt++;                   /* point to first element               */
    r=0.;                       /* initialize radius tally              */
    forall(e,dim) {             /* for all elements...                  */
r+=(*pe)*(*pe);             /* accumulate radius sqr'd  */
pe++;                       /* point to next element    */
    }
    mu+=sqrt((float)r);         /* accumulate sum of radii              */
    sigma+=(float)r;            /* accumulate variance variable */
}
    }
    mu/=(float)n;               /* divide to get overall average radius */
    sigma-=mu*mu*n;             /* compute variance                     */
    sigma=sqrt(sigma)/mu;       /* compute standard deviation           */

    printf("\nmean of the radii:  %f",mu);      /* print statistical    */
    printf("\nstandard deviation: %f",sigma);   /* summary of weight    */
    printf("\n");                               /* point distribution   */
}


/*****************************************************************
 *
 *      R E A D   H E A D E R  --  R e a d    F i l e    H e a d e r
 *
 *****************************************************************/

void read_header()          /* read training file header information    */
{

    rewind(pat);                    /* rewind pattern file              */
    pats_so_far=0;                  /* reset pattern sequence counter   */
```

```
    fscanf(pat,                     /* header comes from pattern file   */

   "hdr %d %d %d \n",       /* header must start with 'hdr'
     * then read header information
     * composed of three numbers
     */

   /* put this information into the following global variables  */

   &ncom,                   /* number of committees in network  */
   &patwide,                /* pattern width in pixels          */
   &pathite);               /* pattern height in pixels         */
}


/*********************************************************************
 *
 *   R A N D O M   --   R a n d o m    N u m b e r   G e n e r a t o r
 *
 *********************************************************************/

element random() {      /* generate a uniformly distributed
 * random number from the open interval (0...1)
 */

    return(rand()/16384.);       /* return scaled random integer        */
}


/*********************************************************************
 *
 *      I N I T    V A L  --  I n i t i a l   E l e m e n t   V a l u e
 *
 *********************************************************************/

element init_val(radius)    /* generate init'l value for element a tlu  */
element radius;             /* the avarage radius of a weight point     */
{

    return(                                 /* return the           */
(radius*sqrt(3.))/(sqrt((float)dim))    /* average weight value */
* (2.*random()-1)                       /* scaled randomly by a */
    );                                      /* uniform distribution */
}


/*********************************************************************
 *
 *   I N I T I A L I Z E  --  A l l o c a t e    S t o r a g e,  E t c.
 *
 *********************************************************************/

void initialize() {     /* allocate & initialize network array storage  */

    committee *pc;      /* pointer to current committee of network      */
    tlu       *pt;      /* pointer to current tlu of committee          */
    element   *pe,      /* pointer to current element of tlu            */
       x; /* current initialization weight value */
```

```
    int c,              /* committee index in network                   */
t,              /* tlu index in committee                       */
e;              /* element index in tlu                         */

    printf("\ninitializing");   /* say what's taking so long !          */

    dim=patwide*pathite+1;      /* number of elements in a tlu          */

    pattern=(vector)calloc(u(dim),          /* allocate the pattern     */
   u(sizeof(element)));     /* vector                   */

    class=(boolean *)calloc(u(ncom),    /* allocate the class array     */
 u(sizeof(boolean)));   /* which will contain the
 * desired decision bits
 * from the committees,
 * as read from the training
 * file.  the actual verdict
 * of the network will be
 * compared with this to see
 * if training is required.
 */

    vote=(int *)calloc(u(ncom),     /* allocate the votes array         */
u(sizeof(int)));    /* which will contain the
     * count of votes for each
     * committee.
     */

    decsn=(boolean *)calloc(u(ncom),    /* allocate the decision        */
 u(sizeof(boolean)));   /* array which will contain
 * the bits of the answer,
 * on bit per committee.
 */

    pc=net=(committee *)calloc(u(ncom),    /* allocate the network      */
  u(sizeof(committee)));   /* as an array of committees */

    forall(c,ncom) {            /* for all committees in the network... */

pc->wtpt=pt=(tlu *)calloc(u(ntlu),  /* allocate a committee     */
   u(sizeof(tlu))); /* as an array of tlu's     */

pc++->dot=(DOTYPE *)calloc(u(ntlu),     /* together with dot    */
 u(sizeof(DOT)));       /* product save cells   */

forall(t,ntlu) {        /* for all tlu's in the committee...    */

    pe=*pt++=(element *)calloc(u(dim),      /* allocate a tlu   */
    u(sizeof(element)));    /* as an array
     * of elements
     */

    forall(e,dim) { /* for each weight... */
if(radius==0) *pe++=(e!=0); /* grow connections? */
else {  /* or adjust weights? */
    x=eabs(*pe++=init_val( /* adjust, get initial */
      (element)radius));/* weight value */
    if(x>maxel) maxel=x; /* update max magnitude */
}
    }
```

```
    /*
     * initialize each element to a random value such that the
     * average radius, or distance from the origin, of each
     * weight point is 'radius'.  this will produce a
     * distribution of weight points clustered near the
     * surface of a hyper-sphere as the starting condition.
     * If the radius is zero, the all weights will be set
     * to zero except for the threshold setting weight.  This
     * is analogous to forcing the program to grow new
     * interneural connections on an as needed basis,
     * supposedly just like the real brain does!
     */
}
    }
    printf("\n");       /* perform new-line when initialize is done     */
}


/*********************************************************************
 *
 *      D O T P R O D   --   F o r m   A   D o t   P r o d u c t
 *
 *********************************************************************/

DOT dotprod(x,y)            /* form the scalar product of two vectors   */
vector  x,y;                /* both arguments are vectors               */
{
    DOT z=0;                /* result accumulator, initialized to zero  */
    int i;                  /* element index, used as loop counter      */

    forall(i,dim)           /* for all elements in each vector...       */
z+=(*x++)*(*y++);           /* compute the dot product          */

    return(z);              /* return it to the caller                  */
}


/*********************************************************************
 *
 *  R E A D   C L A S S  --  R e a d   T h e   C l a s s   T a g
 *
 *********************************************************************/

boolean read_class() {      /* read the class tag number for the image  */

    int       i,            /* loop counter for index in class array    */
      tmp;          /* temp cell to hold decimal category       */
    boolean  *pcl=class;    /* pointer to class (category) array        */

    if(fscanf(pat,"%d",&tmp)!=1)    /* read the pattern category        */
return(FALSE);              /* return FALSE for end of file     */

    forall(i,ncom) {                /* for each committee in network    */
*pcl++=tmp&1;               /* extract desired committee output */
tmp>>=1;                    /* advance to next committee        */
    }

    *pcl=1;                 /* augment with a 1 to prevent singularity  */
    pats_so_far++;          /* update pattern sequence counter          */
```

```
    return(TRUE);           /* return TRUE if class read successfully   */
}


/*********************************************************************
 *
 *   R E A D   P A T T E R N  --  R e a d   N e x t   P a t t e r n
 *
 *********************************************************************/

boolean read_pattern() {    /* read next pattern from training file     */

    int       i,j;          /* loop counters for row & collumn of image */
    element  *pe=pattern;   /* pointer to element of pattern vector     */
    float     tmp;          /* temp cell for input conversion           */

    forall(i,patwide)                   /* for each row in the image,   */
forall(j,pathite)               /* for each pixel in that row,  */
    if( fscanf(pat,"%f",&tmp)   /* input value of pixel         */
!=1 )  return(FALSE);   /* return FALSE if end-of-file  */
    else *pe++=(element)tmp;    /* convert to type element      */

    return( read_class() );     /* read in it's class as an array
 * of correct decisions for each
 * committee in the network.
 *
 * if the entire pattern is read,
 * together with its class,
 * then return TRUE
 */
}


/*********************************************************************
 *
 *      C O U N T   V O T E S  --  C o u n t   T h e   V o t e s
 *
 *********************************************************************/

int count_votes(pc) /* count the votes for each tlu in a committee      */
committee  *pc;     /* second parameter is a pointer to committee       */
{
    DOT *pd=pc->dot;        /* dot product save cell pointer            */
    tlu *pt=pc->wtpt;       /* tlu pointer                              */

    int     ti,             /* tlu index (loop counter)                 */
    count=0;        /* the count of votes for the committee     */

    forall(ti,ntlu)                       /* forall tlus in committee */
count+=sign(                        /* count votes as + or -    */
    *pd++=                          /* & save dot product as    */
dotprod(*pt++,pattern)      /* weight point dotted with */
);                                  /* pattern vector           */
    return(count);          /* return tally                             */
}


/*********************************************************************
 *
 *     R E C O G N I Z E  --  R e c o g n i z e   A   P a t t e r n
```

```
 *
 ************************************************************************/

void recognize() {  /* recognize a pattern by taking the decision
     * of each committee to be a bit in the category
     * number for the pattern
     */

    int     i,          /* loop counter                                 */
    *pv=vote;   /* pointer to vote count array                 */

    boolean *pdec=decsn;/* pointer to decision array.
 * this holds the decision bits for each
 * of the committees in the network.
 */

    committee *pc=net;  /* pointer to current committee in network      */

    forall(i,ncom)       /* for all committees in the network...         */
*pdec++=alpha(*pv++=count_votes(pc++)); /* how many votes ?     */
}


/************************************************************************
 *
 *   $ G E T   W E A K   T L U  --  S w a y   W h i c h   O n e  ?
 *
 ************************************************************************/

int get_weak_tlu(ci)    /* choose tlu most vulnerable to be swayed      */
int ci;                 /* argument is committee index                  */
{
    int     weak=0,             /* index of weakest tlu so far          */
    sv=isign(vote[ci]), /* sign of committee's vote             */
    ti;                 /* tlu index                            */

    DOT *pd=(&net[ci])->dot,        /* pointer to dot product array     */
        conviction=INFINITY,        /* lowest conviction so far         */
        d;                          /* saved dot product value          */

    forall(ti,ntlu) {   /* for all of the tlu's in this committee...    */

d=pd[ti];            /* get the saved dot product value         */

if(sign(d)==sv) {              /* if tlu voted incorrectly      */

    if(eabs(d)<conviction) {         /* and if this tlu has the
      * least conviction of any
      * that have been examined
      * so far,
      */

weak=ti;    /* then remember it as the best one so
     * far to adjust to sway the vote of this
     * committee
     */

conviction=eabs(d);     /* update lowest conviction     */
    }
}
```

```
    }
    return(weak);       /* return subscript of weakest tlu in committee */
}


/*********************************************************************
 *
 *  A D J U S T M E N T  --  C o r r e c t i o n   C o e f f i c i e n t
 *
 *********************************************************************/

element adjustment(ci,ti)   /* compute correction coefficient          */
int     ci,                 /* committee index                         */
ti;                 /* tlu index                                  */
{
    DOT d=(&net[ci])->dot[ti];          /* saved dot product           */

    if(corr_incr)                       /* fixed increment correction  */
return(corr_incr*sign(d));

    if(absolute)                        /* absolute correction         */
return((int)(d/patmag)+sign(d));

    if(fraction)                        /* fractional correction       */
return(d*fraction/patmag);

    abort("No correction method specified."); /* nobody told us
       * what kind of
       * correction to
       * perform !
       */
    return(0);  /* to keep compiler happy :-) */
}


/*********************************************************************
 *
 *      A D J U S T  --  C h a n g e   T L U ' s   W e i g h t s
 *
 *********************************************************************/

void adjust(ci,ti)  /* adjust the weights of a single tlu              */
int ci,             /* committee index                                 */
    ti;             /* tlu index                                       */
{
    vector pw=(&net[ci])->wtpt[ti],     /* pointer to a weight         */
   pp=pattern;                  /* pointer to a pixel           */

    element lambda=adjustment(ci,ti),   /* the correction coefficient  */
    wt,awt; /* temps for max weight point */

    int    i;                           /* element index & loop counter */

    tlus_trained++;                     /* count adjustment of tlu     */

    forall(i,dim) { /* for each coefficient */
wt=(*pw++)-=lambda*(*pp++); /* adjust weights */
awt=eabs(wt); /* save magnitude */
if(maxel<awt) {             /* new maximum ??? */
    maxel=awt; /* yes, update max elem */
```

```
    if(log_level) { /* if any logging,*/
printf("\nmaxel=%f",/* then display the */
 (float)maxel);   /* new maximum value */
    }
}
    }

    if(log_level>=3)
printf("\n        com=%d    tlu=%d    lambda=%g",
          ci,ti,(float)lambda);
}


/*****************************************************************
 *
 * S W A Y   T L U S  --  S w a y   T L U s   T o   C h a n g e   V o t e
 *
 *****************************************************************/

void sway_tlus(ci)  /* sway enough tlu's to change the vote            */
int ci;             /* parameter is committee index                    */
{
    int i,                          /* loop counter                    */
lost_by=iabs(vote[ci]/2)+1, /* how many votes we lost by        */
weak_tlu;                   /* weakest wrong tlu in committee   */

    DOT *pd=(&net[ci])->dot;        /* pointer to dot product array    */

    forall(i,lost_by) {   /* do this enough times to sway the vote...  */

weak_tlu=get_weak_tlu(ci);  /* find most vulnerable tlu         */

adjust(ci,weak_tlu);        /* adjust its weights to change
     * its mind about the pattern
     */
pd[weak_tlu]=-sign(pd[weak_tlu]); /* flip sign of dot product
   * so this tlu won't be
   * considered again in this
   * loop
          */
    }
}


/*****************************************************************
 *
 *      S H O W   B I T S  --  D i s p l a y   B i t s   O n   C R T
 *
 *****************************************************************/

void show_bits(ps,pb)   /* display a bit vector on the screen          */
char    *ps;            /* the label for the bit vector                */
boolean *pb;            /* the pointer to the bit vector               */
{
    int i,              /* loop counter                                */
k=1,            /* power of two                                */
v=0;            /* value accumulator                           */

    forall(i,ncom) {        /* for all committees                      */
if(*pb++) v+=k;     /* convert binary to decimal                */
```

```
k<<=1;              /* advance to next bit                      */
    }
    printf("    %s %d",ps,v);     /* display label and value            */
}


/*********************************************************************
 *
 *      T R A I N  --  T r a i n    T h e   N e t w o r k
 *
 *********************************************************************/

train() {       /* train the network to recognize the pattern           */

    int     ci;     /* committee index                                  */

    goofed=FALSE;       /* give benefit of doubt -- assume didn't goof  */

    patmag=dotprod(pattern,pattern);    /* find pattern magnitude       */

    forall(ci,ncom)     /* for all the committees in the network...     */

if(decsn[ci]!=class[ci]) {      /* if the committee goofed up,  */
    goofed=TRUE;                /* then say so,                 */
    pats_missed++;              /* count misrecognized pattern, */
    sway_tlus(ci);              /* and change enough tlu's
 * so it won't goof up on this
 * pattern next time !
 */
}

    if(goofed) {                            /* did we goof?             */
missed++;                           /* yes, count the boo boo!  */
if(log_level>=2) {                  /* if detail requested,     */
    printf("\n");     /* start a new line */
    show_bits("siloam ",decsn);     /* show machine's decision  */
    show_bits("really ",class);     /* display what really is   */
}
    }
}

/*********************************************************************
 *
 *    T O T C O N S  --  T o t a l    N u m b e r   O f   C o n n e c t s
 *
 *********************************************************************/

int totcons() { /* count total # of connections */

    committee  *n=net; /* neural network pointer */
    tlu        *c,/* committee pointer */
     t; /* tlu pointer */
    int  i,j,k,/* loop indices */
no=0; /* totalizer accumulator */

    forall(i,ncom) { c=n++->wtpt; /* for each committee... */
forall(j,ntlu) { t=*c++; /* for each tlu in the committee*/
    forall(k,dim-1)  /* for each element in the tlu */
if(*t++!=0) no++; /* count it if it is connected */
}
```

```
    }
    return(no); /* return the count */
}

/*********************************************************************
 *
 *   S I L O A M   O u t s i d e   C o n t r o l   S t r u c t u r e
 *
 *********************************************************************/

void siloam() {     /* outside control structure for pattern recognizer */

    long start,stop;    /* timer value cells for benchmarking          */
    int  cons,new,old=0;/* connection counters */

    read_header();      /* read header information in the training file */

    initialize();       /* allocate the committees of TLUs and
 * initialize the weight points randomly
 */

    radius_statistics();/* print starting radius statistics            */

    npass=0;            /* initialize pass counter                     */
    start=time(0);   /* remember start time                        */

    do {                /* start over in training file,
 * we made a mistake...
 */

missed=0;       /* reset misrecognition counter                */

read_header();      /* rewind training file
     * and skip over header information...
     */

while(read_pattern()) { /* keep reading patterns until we've
 * done the entire training file and
 * recognized them all successfully
 */

    recognize();        /* attempt to recognize the pattern     */

    train();            /* adjust any weights necessary to get
 * the correct recognition if we goofed
 */

    if(goofed&&start_over) break;   /* select training strategy */

}                   /* end of while loop to read next pattern   */

npass++;                        /* increment pass counter       */
if(log_level>=1) {              /* give pass summary report     */
    cons=totcons(); /* count the connections */
    new=cons-old; /* compute how many new ones */
    old=cons; /* remember for next time */
    printf("\npass # %d    missed %d   cons=%d   new=%d",
             npass,        missed,     cons,     new);
}
```

```
    } while(missed);        /* end of do loop to train network          */

    stop=time(0);        /* get stop time                             */

    /***************** print end of run summary ***********************/

    printf("\n");
    printf("\ntraining completed in %ld seconds.\n",stop-start);
    printf("\nnumber of committees:  %d",ncom             );
    printf("\nnumber of tlus total:  %d",ncom*ntlu        );
    printf("\nnumber of elements:    %d",ncom*ntlu*dim    );
    printf("\nnumber of connections: %d",totcons()        );
    printf("\n");

    printf("\nnumber of passes thru file: %d",npass);
    printf("\nnumber of patterns in file: %d",pats_so_far );
    printf("\nnumber of mis-recognitions: %d",pats_missed );
    printf("\nnumber of tlu adjustments:  %d",tlus_trained);
    printf("\nmaximum element magnitude:  %f",(float)maxel);
    printf("\n");

    radius_statistics();    /* print ending radius statistics           */

}


/*********************************************************************
 *
 *       M A I N    P r o g r a m    S t a r t s    H e r e
 *
 *********************************************************************/

main(paramct,params)    /********** main program entry point ************/

int  paramct;           /* number of parameters on command line         */
char *params[];         /* array of pointers to strings for each param  */

{
    int i;              /* array index variable                         */

    /********************** identify the program ***********************/

    banner();           /* print program name, version, & release date  */

    printf("\nInvoked By:"); /* show how the program */
    for(i=1;i<=paramct;i++) printf(" %s",params[i]); /* was started up! */
    printf("\nelement type is %s",eltype); /* show arithmetic used */
    printf("\n");

    /********************** parse the command line *********************/

    if(paramct==1) help();  /* if no params, then give help and quit !  */
    patname[0]=0;           /* else set pattern filename to null string */

    for (i=1;i<paramct;i++) {           /* for each parameter...        */

if('-'==params[i][0])               /* is it an option ?        */

    switch(toupper(params[i][1])) { /* yes, which one ?         */
```

```
kase('O',start_over=TRUE)               /* strategy     */
kase('L',log_level=atoi(&params[i][2])) /* log detail   */
kase('T',ntlu=atoi(&params[i][2]))      /* # of TLUs    */
kase('R',radius=(element)atof(&params[i][2]))    /* init radius  */
kase('I',corr_incr=atoi(&params[i][2])) /* fixed incr   */
kase('A',absolute=TRUE)                 /* absolute  */
kase('F',fraction=atof(&params[i][2]))  /* fractional   */
    }

/****************** parse filename ***************************/

else if(index(&params[i][0],'.'))   /* is '.' in it?            */
    move(&params[i][0],patname);    /* yes, pattern file        */

else move(".PAT",                   /* no, default extension is */
    move(&params[i][0],patname));   /* '.pat' for pattern file  */
    }


    /**************** check for command line errors ********************/

    if(patname[0]==0)       /* check for missing pattern file name      */
abort(
"pattern filename not specified!");

    if(ntlu==0)             /* check for missing number of TLUs         */
abort(
"number of TLUs per committee not specified!");


    /************************ open pattern file ***********************/

    if(!(pat=fopen(patname,"r")))           /* if open fails, abort     */
abort(
"can't open pattern file!");


    /********* perform the training and recognition algorithm ***********/

/*  srand(1);  */   /* make random number generator repeatable --
     * ...this may be removed, if desired, after the
     * debug phase is complete!
     */

    siloam();       /* call the outside control structure for the
     * trainable pattern recognizer.
     */
}
```

Alphabet Pattern Training Set

```
hdr 5 15 28


0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.
```

```
0. 0. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 0. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0.
0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.
0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.
0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.
0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0.
1. 1. 1. 0. 0. 0. 0. 0. 1. 1. 1. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.

1


1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.

2
```

```
0. 0. 0. 1. 1. 1. 0. 0. 1. 0. 0. 0. 0. 0. 0.
0. 0. 1. 0. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0.
0. 0. 1. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.
1. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.
1. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.
1. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.
1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
1. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.

3

1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.
1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
```

```
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
```

4

```
1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
```

5

```
1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
```

```
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
```

```
6
```

```
0. 0. 0. 0. 1. 1. 0. 0. 1. 0. 0. 0. 0. 0. 0.
0. 0. 0. 1. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0.
0. 0. 1. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.
1. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.
1. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.
1. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.
1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
1. 0. 0. 0. 0. 0. 1. 1. 1. 1. 0. 0. 0. 0. 0.
1. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.
1. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0.
0. 0. 1. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0.
0. 0. 0. 1. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 1. 1. 0. 0. 1. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
```

```
7
```

```
1. 1. 1. 0. 0. 0. 1. 1. 1. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
```

```
0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
1. 1. 1. 0. 0. 0. 1. 1. 1. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
```

8

```
1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
```

9

```
0. 0. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
```

```
0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
1. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
1. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
```

10

```
1. 1. 1. 0. 0. 0. 0. 1. 1. 1. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 1. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 1. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.
1. 1. 1. 0. 0. 0. 0. 1. 1. 1. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
```

11

```
1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
```

```
0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
```

12

```
1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 0.
0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0.
0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0.
0. 0. 1. 1. 0. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0.
0. 0. 1. 1. 0. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0.
0. 0. 1. 1. 0. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0.
0. 0. 1. 1. 0. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0.
0. 0. 1. 0. 1. 0. 0. 0. 0. 1. 0. 1. 0. 0. 0.
0. 0. 1. 0. 1. 0. 0. 0. 0. 1. 0. 1. 0. 0. 0.
0. 0. 1. 0. 1. 0. 0. 0. 0. 1. 0. 1. 0. 0. 0.
0. 0. 1. 0. 1. 0. 0. 0. 0. 1. 0. 1. 0. 0. 0.
0. 0. 1. 0. 1. 0. 0. 0. 0. 1. 0. 1. 0. 0. 0.
0. 0. 1. 0. 0. 1. 0. 0. 1. 0. 0. 1. 0. 0. 0.
0. 0. 1. 0. 0. 1. 0. 0. 1. 0. 0. 1. 0. 0. 0.
0. 0. 1. 0. 0. 1. 0. 0. 1. 0. 0. 1. 0. 0. 0.
0. 0. 1. 0. 0. 1. 0. 0. 1. 0. 0. 1. 0. 0. 0.
0. 0. 1. 0. 0. 0. 1. 1. 0. 0. 0. 1. 0. 0. 0.
0. 0. 1. 0. 0. 0. 1. 1. 0. 0. 0. 1. 0. 0. 0.
0. 0. 1. 0. 0. 0. 1. 1. 0. 0. 0. 1. 0. 0. 0.
0. 0. 1. 0. 0. 0. 1. 1. 0. 0. 0. 1. 0. 0. 0.
1. 1. 1. 1. 1. 0. 0. 1. 0. 1. 1. 1. 1. 1. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
```

13

```
1. 1. 0. 0. 0. 0. 0. 1. 1. 1. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.
0. 1. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.
0. 1. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.
0. 1. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.
0. 1. 0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.
0. 1. 0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.
```

```
0. 1. 0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.
1. 1. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
```

14

```
0. 0. 0. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.
1. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0.
1. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0.
1. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0.
1. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0.
1. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0.
1. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0.
1. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0.
1. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0.
1. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.
0. 0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
```

15

```
1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.
```

```
0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
```

16

```
0. 0. 0. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.
1. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0.
1. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0.
1. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0.
1. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0.
1. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0.
1. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0.
1. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0.
1. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0.
1. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.
0. 1. 0. 1. 1. 1. 1. 0. 1. 0. 0. 0. 0. 0. 0.
0. 1. 0. 1. 0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0.
0. 0. 1. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 1. 1. 1. 0. 0. 1. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 1. 1. 1. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
```

17

```
1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0. 0.
1. 1. 1. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
```

18

```
0. 0. 1. 1. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.
1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.
1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.
1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.
1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.
1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.
1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.
1. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.
1. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.
1. 0. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
```

```
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
```

19

```
1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0.
1. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.
1. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.
1. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.
1. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.
1. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.
1. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
```

20

```
1. 1. 1. 0. 0. 0. 1. 1. 1. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
```

```
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
```

21

```
1. 1. 1. 0. 0. 0. 0. 0. 1. 1. 1. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0.
0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.
0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.
0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.
0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.
0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.
0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
```

22

```
1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1.
0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0.
0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0.
0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0.
0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0.
0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0.
0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0.
0. 0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 1. 0. 0.
0. 0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 1. 0. 0.
0. 0. 1. 0. 0. 0. 1. 0. 1. 0. 0. 0. 1. 0. 0.
0. 0. 0. 1. 0. 0. 1. 0. 1. 0. 0. 0. 1. 0. 0.
0. 0. 0. 1. 0. 0. 1. 0. 1. 0. 0. 1. 0. 0. 0.
0. 0. 0. 1. 0. 0. 1. 0. 1. 0. 0. 1. 0. 0. 0.
0. 0. 0. 1. 0. 0. 1. 0. 1. 0. 0. 1. 0. 0. 0.
0. 0. 0. 1. 0. 1. 0. 0. 0. 1. 0. 1. 0. 0. 0.
0. 0. 0. 1. 0. 1. 0. 0. 0. 1. 0. 1. 0. 0. 0.
0. 0. 0. 1. 0. 1. 0. 0. 0. 1. 0. 1. 0. 0. 0.
0. 0. 0. 0. 1. 1. 0. 0. 0. 1. 1. 0. 0. 0. 0.
0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0.
```

```
0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0.
0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
```

23

```
1. 1. 1. 0. 0. 0. 0. 0. 1. 1. 1. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 0.
0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.
0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.
0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.
0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.
0. 1. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0.
1. 1. 1. 0. 0. 0. 0. 0. 1. 1. 1. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
```

24

```
1. 1. 1. 0. 0. 0. 0. 0. 1. 1. 1. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 0.
0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.
0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.
0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.
```

```
0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
```

25

```
1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0.
1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.
1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.
1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.
1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.
1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.
0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.
1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.
1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
```

26