**SAGAR INSTITUTE OF RESEARCH AND TECHNOLOGY**



**LAB File**

**OF**

**Soft Computing IT-702(A)**

**Submitted To Submitted By**

Amit Jain Sir Name :-

SIRT - CSIT Enrollment No :-

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Department of Computer Science &Information Technology**

**IINDEX**

|  |  |  |  |
| --- | --- | --- | --- |
| **s.NO.** | **PRACTICALS** | **DATE OF COMPLITION** | **SIGN** |
| **1** |  |  |  |
| **2** |  |  |  |
| **3** |  |  |  |
| **4** |  |  |  |
| **5** |  |  |  |
| **6** |  |  |  |
| **7** |  |  |  |
| **8** |  |  |  |

**IT 701- Soft Computing Lab**

**Course Objective** : The objective of this course is to create a competitive industry required IoT skill in students.

**Course Outcomes** :

|  |  |  |
| --- | --- | --- |
| **S. No.** | **Description** | **Bloom’s Taxonomy Level** |
| CO1. | Understand concept of ANN and explain the XOR problem | (Level 1) |
| CO2. | Use supervised neural networks to classify given inputs | (Level 2, 3) |
| CO3. | Understand unsupervised neural networks for clustering data | (Level 4) |
| CO4. | Build Fuzzy inference system using concepts of fuzzy logic | (Level 5) |
| CO5. | Obtain an optimized solution to a given problem using genetic algorithm | (level 6) |

**List of Practical**

1. Implement Perception network with binary input and output. CO5

2. Using Adeline net, generate XOR function with bipolar inputs and COS targets. CO5

3. Calculation of new weights for a back propagation network, given the values of input pattern, output pattern, target output, learning rate and COS activation function. CO5

4. Use of ART algorithm to cluster vectors. CO5

5. Implement various laws associated with fuzzy sets. CO5

6. Implement various operations associated with fuzzy sets. CO5

7. Implement TSP using Genetic Algorithm. CO5

8. Implement word matching using Genetic Algorithm. CO5

1. **Implement Perception network with binary input and output.**

/\*PERCEPTRON\*/

#include<stdio.h>

#include<conio.h>

main()

{

signed int x[4][2],tar[4];

float w[2],wc[2],out=0;

int i,j,k=0,h=0;

float s=0,b=0,bc=0,alpha=0;

float theta;

clrscr();

printf("Enter the value of theta & alpha");

scanf("%f%f",&theta,&alpha);

for(i=0;i<=3;i++)

{

printf("Enter the value of %d Inputrow & Target",i);

for(j=0;j<=1;j++)

{

scanf("%d",&x[i][j]);

}

scanf("%d",&tar[i]);

w[i]=0;

wc[i]=0;

}

printf("\Net \t Target\tWeight changes\tNew weights\t Bias changes\tBias \n");

printf("-----------------------------------------------------------------------------\n");

mew:

printf("ITERATION %d\n",h);

printf("----------------------------------------------------------------------------\n");

for(i=0;i<=3;i++)

{

for(j=0;j<=1;j++)

{

s+=(float)x[i][j]\*w[j];

}

s+=b;

printf("%.2f\t",s);

if(s>theta)

out=1;

else if(s<-theta)

out=-1;

else

{

out=0;

}

printf("%d\t",tar[i]);

s=0;

if(out==tar[i])

{

for(j=0;j<=1;j++)

{

wc[j]=0;

bc=0;

printf("%.2f\t",wc[j]);

}

for(j=0;j<=1;j++)

printf("%.2f\t",w[j]);

k+=1;

b+=bc;

printf("%.2f\t\t",bc);

printf("%.2f\t",b);

}

else

{

for(j=0;j<=1;j++)

{

wc[j]=x[i][j]\*tar[i]\*alpha;

w[j]+=wc[j];

printf("%.2f\t",wc[j]);

wc[j]=0;

}

for(j=0;j<=1;j++)

printf("%.2f\t",w[j]);

bc=tar[i]\*alpha;

b+=bc;

printf("%.2f\t\t",bc);

printf("%.2f\t",b);

}

printf("\n");

}

if(k==4)

{

printf("\nFinal weights\n");

for(j=0;j<=1;j++)

{

printf("w[%d]=%.2f\t",j,w[j]);

}

printf("Bias b=%.2f",b);

}

else

{

k=0;

h=h+1;

getch();

goto mew;

}

getch();

}

1. **Using Adeline net, generate XOR function with bipolar inputs and COS targets.**

/\*ADALINE\*/

#include<stdio.h>

#include<conio.h>

main()

{

signed int x[4][4],tar[4];

float wc[4],w[4],e=0,er=0,yin=0,alp=0.5,b=0,bc=0,t=0;

int i,j,k,q=1;

clrscr();

for(i=0;i<=3;i++)

{

printf("\nEnter the %d row and target\t",i);

for(j=0;j<=3;j++)

{

scanf("%d",&x[i][j]);

}

scanf("%d",&tar[i]);

printf("%d",tar[i]);

w[i]=0.0;

wc[i]=0.0;

}

mew:

er=0;e=0;

yin=0;

printf("\n ITERATION%d",q);

printf("\n------------------");

for(i=0;i<=3;i++)

{

t=tar[i];

for(j=0;j<=3;j++)

{

yin=yin+x[i][j]\*w[j];

}

b=b+bc;

yin=yin+b;

bc=0.0;

printf("\nNet=%f\t",yin);

e=(float)tar[i]-yin;

yin=0.0;

printf("Error=%f\t",e);

printf("Target=%d\t\n",tar[i]);

er=er+e\*e;

for(k=0;k<=3;k++)

{

wc[k]=x[i][k]\*e\*alp;

w[k]+=wc[k];

wc[k]=0.0;

}

printf("Weights \t");

for(k=0;k<=3;k++)

{

printf("%f\t",w[k]);

}

bc=e\*alp;

printf("b=%.2f\t",b);

getch();

printf("\n Error Square=%f",er);

if(er<=1.000)

{

printf("\n");

for(k=0;k<=1;k++)

printf("%f\t",w[k]);

getch();

}

else

{

e=0;

er=0;

yin=0;

q=q+1;

goto mew;

}

getch();

}

}

1. **Calculation of new weights for a back propagation network, given the values of input pattern, output pattern, target output, learning rate and COS activation function.**

/\*BACK PROPAGATION NETWORK\*/

#include<stdio.h>

#include<conio.h>

#include<math.h>

#include<stdlib.h>

void main()

{

float v[2][4],w[4][1],vc[2][4],wc[4][1],de,del[4],bl,bia,bc[4],e=0;

float x[4][2],t[4],zin[4],delin[4],yin=0,y,dy,dz[4],b[4],z[4],es,alp=0.02;

int i,j,k=0,itr=0;

v[0][0]=0.1970;

v[0][1]=0.3191;

v[0][2]=-0.1448;

v[0][3]=0.3594;

v[1][0]=0.3099;

v[1][1]=0.1904;

v[1][2]=-0.0347;

v[1][3]=-0.4861;

w[0][0]=0.4919;

w[1][0]=-0.2913;

w[2][0]=-0.3979;

w[3][0]=0.3581;

b[0]=-0.3378;

b[1]=0.2771;

b[2]=0.2859;

b[3]=-0.3329;

bl=-0.141;

x[0][0]=-1;

x[0][1]=-1;

x[1][0]=-1;

x[1][1]=1;

x[2][0]=1;

x[2][1]=-1;

x[3][0]=1;

x[3][1]=1;

t[0]=0;

t[1]=1;

t[2]=1;

t[3]=0;

clrscr();

for(itr=0;itr<=387;itr++)

{

e=0;

es=0;

for(i=0;i<=3;i++)

{

do

{

for(j=0;j<=1;j++)

{

zin[k]+=x[i][j]\*v[j][k];

}

zin[k]+=b[k];

k+=1;

}while(k<=4);

for(j=0;j<=3;j++)

{

z[j]=(1-exp(-zin[j]))/(1+exp(-zin[j]));

dz[j]=((1+z[j])\*(1-z[j]))\*0.5;

}

for(j=0;j<=3;j++)

{

yin+=z[j]\*w[j][0];

}

yin+=bl;

y=(1-exp(-yin))/(1+exp(-yin));

dy=((1+y)\*(1-y))\*0.5;

de=(t[i]-y)\*dy;

e=t[i]-y;

es+=0.5\*(e\*e);

for(j=0;j<=3;j++)

{

wc[j][0]=alp\*de\*z[j];

delin[j]=de\*w[j][0];

del[j]=delin[j]\*dz[j];

}

bia=alp\*de;

for(k=0;k<=1;k++)

{

for(j=0;j<=3;j++)

{

vc[k][j]=alp\*del[j]\*x[i][k];

v[k][j]+=vc[k][j];

}

}

for(j=0;j<=3;j++)

{

bc[j]=alp\*del[j];

w[j][0]+=wc[j][0];

b[j]+=bc[j];

}

bl+=bia;

for(j=0;j<=3;j++)

{

zin[j]=0;

z[j]=0;

dz[j]=0;

delin[j]=0;

del[j]=0;

bc[j]=0;

}

k=0;yin=0;y=0;

dy=0;bia=0;de=0;

}

printf("\nEpoch %d:\n",itr);

for(k=0;k<=1;k++)

{

for(j=0;j<=3;j++)

{

printf("%f\t",v[k][j]);

}

printf("\n");

}

printf("\n");

for(k=0;k<=3;k++)

{

printf("%f\t",w[k][0]);

}

printf("\n%f",bl);

printf("\t");

for(k=0;k<=3;k++)

{

printf("%f\t",b[k]);

}

getch();

}

getch();

}

1. **Use of ART algorithm to cluster vectors.**

getch();/\* ART1 NETWORK TO CLUSTER FOUR VECTORS \*/

#include<stdio.h>

#include<conio.h>

main()

{

float n=4.0,m=3.0,o=0.4,l=2.0;

float b[4][3],t[3][4],s[4],x[4],sin=0,y[3],xin=0;

int i,j,k=0,J,c=0;

y[0]=0,y[1]=0,y[2]=0;

clrscr();

for(i=0;i<=3;i++)

{

for(j=0;j<=2;j++)

{

b[i][j]=0.2;

}

}

for(i=0;i<=2;i++)

{

for(j=0;j<=3;j++)

{

t[i][j]=1.0;

}

}

mew:

printf("Enter the input value:\n");

for(i=0;i<=3;i++)

{

scanf("%f",&s[i]);

x[i]=s[i];

sin+=s[i];

}

for(i=0;i<=2;i++)

{

printf("\nY");

do

{

y[i]+=s[k]\*b[k][i];

k+=1;

}while(k<=3);

if(y[0]>=y[1])

{

if(y[0]>=y[2])

J=0;

else

J=2;

}

else

{

if(y[1]>=y[2])

J=1;

else

J=2;

}

for(i=0;i<=3;i++)

{

x[i]=s[i]\*t[J][i];

xin+=x[i];

}

if(xin/sin>=0.4)

{

for(i=0;i<=3;i++)

{

b[i][J]=(2\*x[i])/(1+xin);

t[J][i]=x[i];

}

}

else

{

y[J]=-1;

}

printf("\n");

for(i=0;i<=3;i++)

{

for(j=0;j<=2;j++)

{

printf("%f\t",b[i][j]);

}

printf("\n");

}

for(i=0;i<=2;i++)

{

for(j=0;j<=3;j++)

{

printf("%f\t",t[i][j]);

}

printf("\n");

}

getch();

y[0]=y[1]=y[2]=0;

sin=xin=0;

c+=1;

k=0;

if(c<=3)

goto mew;

}

getch();

}

1. **Implement various laws associated with fuzzy sets.**

#include<stdio.h>

#include<alloc.h>

#include<conio.h>

#include<stdlib.h>

struct SET

{

float nr[5];

float dr[5];

int n;

};

typedef struct SET fuzzy;

void getval(fuzzy \*m,char \*x)

{

int i;

float f;

clrscr();

printf("\n Enter the %s:\n",x);

for(i=0;i<m->n;i++)

{

printf(" Numerator Element %d :",i+1);

scanf("%f",&f);

m->nr[i]=f;

fflush(stdin);

printf(" Denominator Element %d:",i+1);

scanf("%f",&f);

m->dr[i]=f;

}

}

void printval(fuzzy \*m,char \*x)

{

int i;

printf("\n %s={",x);

for(i=0;i<m->n;i++)

{

printf(" %6.2f / %6.2f",m->nr[i],m->dr[i]);

if(i!=m->n-1) putch('+');

}

printf(" }");

}

fuzzy unionset(fuzzy a,fuzzy b)

{

fuzzy temp;

char ch;

int i;

temp.n=a.n;

for(i=0;i<a.n;i++)

{

if(a.dr[i]!=b.dr[i])

{

printf("\n Denominators not equal");

getch();

exit(0);

}

if(a.nr[i]<b.nr[i])

temp.nr[i]=b.nr[i];

else

temp.nr[i]=a.nr[i];

temp.dr[i]=a.dr[i];

}

return temp;

}

fuzzy intersect(fuzzy a,fuzzy b)

{

fuzzy temp;

int i;

temp.n=a.n;

for(i=0;i<a.n;i++)

{

if(a.dr[i]!=b.dr[i])

{

printf("\n Denominators not equal");

getch();

exit(0);

}

if(a.nr[i]>b.nr[i])

temp.nr[i]=b.nr[i];

else

temp.nr[i]=a.nr[i];

temp.dr[i]=a.dr[i];

}

return temp;

}

fuzzy complement(fuzzy a)

{

fuzzy temp;

int i;

temp.n=a.n;

for(i=0;i<a.n;i++)

{

temp.nr[i]=1-a.nr[i];

temp.dr[i]=a.dr[i];

}

return temp;

}

void main()

{

fuzzy a,b,ans;

char ch;

clrscr();

printf("\n Enter the no of componets:");

scanf("%d",&a.n);

b.n=a.n;

getval(&a,"A");

getval(&b,"B");

clrscr();

printval(&a,"A");

printval(&b,"B");

getch();

while(1)

{

clrscr();

printf("\n Menu:\n 1.AUB\n 2.A^B\n 3.A~\n 4.B~ \n 5.Print S,A,B\n 6.Exit");

switch((ch=getch()))

{

case '1':

ans=unionset(a,b);

printval(&ans,"AUB");

getch();

break;

case '2':

ans=intersect(a,b);

printval(&ans,"A^B");

getch();

break;

case '3':

ans=complement(a);

printval(&ans,"A~");

getch();

break;

case '4':

ans=complement(b);

printval(&ans,"B~");

getch();

break;

case '5':

printval(&a,"A");

printval(&b,"B");

getch();

break;

case '6':

exit(0);

}

}

}

1. **Implement various operations associated with fuzzy sets.**

#include<stdio.h>

#include<alloc.h>

#include<conio.h>

#include<stdlib.h>

struct SET

{

float nr[5];

float dr[5];

int n;

};

typedef struct SET fuzzy;

void printval(fuzzy \*m,char \*x)

{

int i;

printf("\n %s={",x);

for(i=0;i<m->n;i++)

{

printf(" %6.2f / %6.2f",m->nr[i],m->dr[i]);

if(i!=m->n-1) putch('+');

}

printf(" }");

}

fuzzy unionset(fuzzy a,fuzzy b)

{

fuzzy temp;

char ch;

int i;

temp.n=a.n;

for(i=0;i<a.n;i++)

{

if(a.dr[i]!=b.dr[i])

{

printf("\n Denominators not equal");

getch();

exit(0);

}

if(a.nr[i]<b.nr[i])

temp.nr[i]=b.nr[i];

else

temp.nr[i]=a.nr[i];

temp.dr[i]=a.dr[i];

}

return temp;

}

fuzzy intersect(fuzzy a,fuzzy b)

{

fuzzy temp;

int i;

temp.n=a.n;

for(i=0;i<a.n;i++)

{

if(a.dr[i]!=b.dr[i])

{

printf("\n Denominators not equal");

getch();

exit(0);

}

if(a.nr[i]>b.nr[i])

temp.nr[i]=b.nr[i];

else

temp.nr[i]=a.nr[i];

temp.dr[i]=a.dr[i];

}

return temp;

}

fuzzy complement(fuzzy a)

{

fuzzy temp;

int i;

temp.n=a.n;

for(i=0;i<a.n;i++)

{

temp.nr[i]=1-a.nr[i];

temp.dr[i]=a.dr[i];

}

return temp;

}

void main()

{

fuzzy a,b,ans;

char ch;

clrscr();

a.n=b.n=3;

a.nr[0]=0.1; a.dr[0]=1;

a.nr[1]=0.2; a.dr[1]=2;

a.nr[2]=0.3; a.dr[2]=3;

b.nr[0]=0.4; b.dr[0]=1;

b.nr[1]=0.3; b.dr[1]=2;

b.nr[2]=0.2; b.dr[2]=3;

printval(&a,"A");

printval(&b,"B");

printf("\n Menu:\n 1.AUB\n 2.A^B\n 3.A~\n 4.B~ \n 5.Print S,A,B\n 6.Exit");

while(1)

{

switch((ch=getch()))

{

case '1':

ans=unionset(a,b);

printval(&ans,"1.AUB");

break;

case '2':

ans=intersect(a,b);

printval(&ans,"2.A^B");

break;

case '3':

ans=complement(a);

printval(&ans,"3.A~");

break;

case '4':

ans=complement(b);

printval(&ans,"4.B~");

break;

case '5':

printval(&a,"A");

printval(&b,"B");

break;

case '6':

exit(0);

}

}

}

1. **Implement TSP using Genetic Algorithm.**

#include<stdio.h>

#include<conio.h>

int tsp[10][10]={{999,10,3,2,5,6,7,2,5,4},

{20,999,3,5,10,2,8,1,15,6},

{10,5,999,7,8,3,11,12,3,2},

{3,4,5,999,6,4,10,6,1,8},

{1,2,3,4,999,5,10,20,11,2},

{8,5,3,10,2,999,6,9,20,1},

{3,8,5,2,20,21,999,3,5,6},

{5,2,1,25,15,10,6,999,8,1},

{10,11,6,8,3,4,2,15,999,1},

{5,10,6,4,15,1,3,5,2,999}

};

int pa[1000][10]= {{0,1,2,3,4,5,6,7,8,9},

{9,8,6,3,2,1,0,4,5,7},

{2,3,5,0,1,4,9,8,6,7},

{4,8,9,0,1,3,2,5,6,7}

};

int i,j,k,l,m,y,loc,flag,row,col,it,x=3,y=3;

int count,row=0,res[1][10],row1,col1,z;

int numoff=4;

int offspring[1000][10];

int mincost=9999,mc;

main()

{

int gen;

clrscr();

printf("Number of Generation : ");

scanf("%d",&gen);

offcal1(pa);

offcal2(pa);

printf(" \n\t\t First Generation\n");

for(i=0;i<count;i++)

{

for(j=0;j<10;j++)

printf("%d ",offspring[i][j]);

printf("\n");

}

for(y=1;y<=gen-1;y++)

{

getch();

clrscr();

for(i=0;i<count;i++)

for(j=0;j<10;j++)

pa[i][j]=offspring[i][j];

numoff=count;

offcal1(pa);

offcal2(pa);

printf(" \n\t\t %d Generation\n",y+1);

for(i=0;i<count;i++)

{

for(j=0;j<10;j++)

printf("%d ",offspring[i][j]);

printf("\n");

}

getch();

clrscr();

}

printf("\n\nMinimum Cost Path\n");

for(z=0;z<10;z++)

printf("%d ",res[0][z]);

printf("\nMinimum Cost %d \n",mincost);

}

/\* finding the offspring using cyclic crossover \*/

offcal1(pa)

int pa[1000][10];

{

count=0;

for(i=0;i<1000;i++)

for(j=0;j<10;j++)

offspring[i][j]=-1;

for(k=0;k<numoff;k++)

{

for(l=k+1;l<numoff;l++)

{

offspring[row][0]=pa[k][0];

loc=pa[l][0];

flag=1;

while(flag != 0)

{

for(j=0;j<10;j++)

{

if(pa[k][j] == loc )

{

if (offspring[row][j]==-1)

{

offspring[row][j]=loc;

loc=pa[l][j];

}

else

flag=0;

}

}

}/\* end while\*/

for(m=0;m<10;m++)

{

if(offspring[row][m] == -1)

offspring[row][m]=pa[l][m];

}

for(z=0;z<10;z++)

{

if(z<9)

{

row1=offspring[row][z];

col1=offspring[row][z+1];

mc=mc+tsp[row1][col1];

}

else

{

row1=offspring[row][z];

col1=offspring[row][0];

mc=mc+tsp[row1][col1];

}

}

if(mc < mincost)

{

for(z=0;z<10;z++)

res[0][z]=offspring[row][z];

mincost=mc;

}

count++;

row++;

}/\* end l\*/

}

}

offcal2(pa)

int pa[1000][10];

{

for(k=0;k<numoff;k++)

{

for(l=k+1;l<numoff;l++)

{

offspring[row][0]=pa[l][0];

loc=pa[k][0];

flag=1;

while(flag != 0)

{

for(j=0;j<10;j++)

{

if(pa[l][j] == loc )

{

if (offspring[row][j]==-1)

{

offspring[row][j]=loc;

loc=pa[k][j];

}

else

flag=0;

}

}

}/\* end while\*/

for(m=0;m<10;m++)

{

if(offspring[row][m] == -1)

offspring[row][m]=pa[k][m];

}

for(z=0;z<10;z++)

{

if(z<9)

{

row1=offspring[row][z];

col1=offspring[row][z+1];

mc=mc+tsp[row1][col1];

}

else

{

row1=offspring[row][z];

col1=offspring[row][0];

mc=mc+tsp[row1][col1];

}

}

row++;

if(mc < mincost)

{

for(z=0;z<10;z++)

res[0][z]=offspring[row][z];

mincost=mc;

}

count++;

}/\* end l\*/

}

}

**8. Implement word matching using Genetic Algorithm.**

#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

#include<dos.h>

char input[15],parent[50][15],child[50][15],mating\_pool[105][15],mutant[05][15];

int pfit[50],cfit[50],fit[105],mfit[05],gen=0;

void get\_input()

{

int i;

clrscr();

printf("\n\n\n\t\tWORD MATCHING PROBLEM - GENETIC ALGORITHMS ASSIGNMENT");

printf("\n\t \*\*\*\*\*\*\*\*\*\*\*\*\*\*");

printf("\n\n\n\n\t\tENTER THE WORD TO BE MATCHED : ");

scanf("%s",input);

printf("\n\n\n\t THE ASCII EQUIVALENT OF THE LETTERS IN THE ENTERED WORD");

printf("\n\t--------------------------------------------------------------");

printf("\n\n LETTERS :");

for(i=0;i<strlen(input);i++)

{

printf(" %c ",input[i]);

}

printf("\n ASCII :");

for(i=0;i<strlen(input);i++)

{

printf(" %3d",input[i]);

}

getch();

}

void initial\_pop()

{

int i,j;

randomize();

for(i=0;i<50;i++)

{

for(j=0;j<strlen(input);j++)

{

parent[i][j]=random(26)+97;

if(parent[i][j]==input[j])

{

pfit[i]++;

}

}

}

}

void display()

{

int i,j,nexti;

clrscr();

printf("\n\n\t\t THE CHROMOSOMES OF PARENTS AND CHILDREN");

printf("\n\t --------------------------------------------\n");

printf("\n\t\t PREVIOUS GENERATION CHILDREN CHROMOSOMES\n\n");

for(i=0;i<50;i++)

{

if(((i)%4)==0) printf("\n");

for(j=0;j<strlen(input);j++)

{

printf("%c",child[i][j]);

}

printf("% 2d ",cfit[i]);

}

printf("\n\t\t\tMUTANTS OF THIS GENERATION\n");

for(i=0;i<05;i++)

{

if (i==3) printf("\n");

for(j=0;j<strlen(input);j++)

{

printf("%c",mutant[i][j]);

}

printf("% 2d ",mfit[i]);

}

getch();

clrscr();

printf("\n\n\t\t THE CHROMOSOMES OF PARENTS AND CHILDREN");

printf("\n\t --------------------------------------------\n");

printf("\n\t\t NEXT GENERATION PARENTS CHROMOSOMES\n\n");

for(i=0;i<50;i++)

{

if(((i)%4)==0) printf("\n");

for(j=0;j<strlen(input);j++)

{

printf("%c",parent[i][j]);

}

printf("% 2d ",pfit[i]);

}

getch();

}

void reproduction() //sorting\_based\_on\_fitness()

{

char tempc;

int temp;

int i,j,k;

for(i=0;i<50;i++)

{

for(j=0;j<strlen(input);j++)

{

mating\_pool[i][j]=parent[i][j];

fit[i]=pfit[i];

}

}

for(i=50;i<100;i++)

{

for(j=0;j<strlen(input);j++)

{

mating\_pool[i][j]=child[i-50][j];

fit[i]=cfit[i-50];

}

}

for(i=100;i<105;i++)

{

for(j=0;j<strlen(input);j++)

{

mating\_pool[i][j]=mutant[i-100][j];

fit[i]=mfit[i-100];

}

}

//sorting

for(i=0;i<105;i++)

{

for(j=i+1;j<105;j++)

{

if(fit[i]<fit[j])

{

for(k=0;k<strlen(input);k++)

{

tempc=mating\_pool[i][k];

mating\_pool[i][k]=mating\_pool[j][k];

mating\_pool[j][k]=tempc;

temp=fit[i];

fit[i]=fit[j];

fit[j]=temp;

}

}

}

}

for(i=0;i<50;i++)

{

for(j=0;j<strlen(input);j++)

{

parent[i][j]=mating\_pool[i][j];

pfit[i]=fit[i];

}

}

for(i=50;i<100;i++)

{

for(j=0;j<strlen(input);j++)

{

child[i-50][j]=mating\_pool[i][j];

cfit[i-50]=fit[i];

}

}

}

void crossover()

{

int xover\_pt;

int i,j,k;

for(i=0;i<50;i++)

{

xover\_pt=random(strlen(input));

cfit[i]=0;

cfit[i+1]=0;

for(j=0;j<xover\_pt;j++)

{

child[i][j]=parent[i][j];

if (input[j]==child[i][j])

cfit[i]++;

child[i+1][j]=parent[i+1][j];

if(input[j]==child[i+1][j])

cfit[i+1]++;

}

for(j=xover\_pt;j<strlen(input);j++)

{

child[i][j]=parent[i+1][j];

if(input[j]==child[i][j])

cfit[i]++;

child[i+1][j]=parent[i][j];

if(input[j]==child[i+1][j])

cfit[i+1]++;

}

i++;

}

}

void mutation()

{

int i,mut\_pt,j;

char mut\_val;

randomize();

for(i=0;i<05;i++)

{

mut\_pt=random(strlen(input));

mut\_val=random(26)+97;

mfit[i]=0;

for(j=0;j<mut\_pt;j++)

{

mutant[i][j]=parent[1][j];

if (mutant[i][j]==input[j])

{

mfit[i]++;

}

}

mut\_val=input[j];

mutant[i][mut\_pt]=mut\_val;

if (mutant[i][j]==input[j])

{

mfit[i]++;

}

for(j=mut\_pt+1;j<strlen(input);j++)

{

mutant[i][j]=parent[1][j];

if (mutant[i][j]==input[j])

{

mfit[i]++;

}

}

}

}

void results()

{

int i;

clrscr();

printf("\n\n\n\t\tWORD MATCHING PROBLEM - GENETIC ALGORITHM ASSIGNMENT");

printf("\n\t \*\*\*\*\*\*\*\*\*\*\*\*\*\*");

printf("\n\n\n\t\t THE MATCHING WORD FOR THE GIVEN INPUT WORD");

printf("\n\n\t\t OBTAINED USING GENETIC ALGORITHM");

printf("\n\n\t\t\t ");

for(i=0;i<strlen(input);i++)

{

printf("%c",parent[0][i]);

}

printf("\n\t\t\t --");

for(i=0;i<strlen(input);i++)

{

printf("-");

}

printf("--\n\n\n\t\t USER INPUT : %s",input);

printf("\n\n\n\t THE FITNESS OF THE GA GENERATED WORD AND THE USER'S INPUT");

printf("\n\n\t\t\t\t %2d/%d",pfit[0],strlen(input));

printf("\n\n\n\t\t\t GENERATIONS COUNT : %d",gen);

}

int input\_choice()

{

int choice,i;

clrscr();

printf("\n\n\n\n\t\t\t GENEREATION NUMBER : %d",gen);

printf("\n\t\t ------------------------------");

printf("\n\n\n\t\tTHE FITTEST INDIVIDUAL TILL THE PREVIOUS GENERATION\n\n\n\t\t\t\t");

for(i=0;i<strlen(input);i++)

{

printf("%c",parent[0][i]);

}

printf(" / ");

for(i=0;i<strlen(input);i++)

{

printf("%c",input[i]);

}

printf("\n\n\n\t\t\t WITH A FITNESS OF %d/%d",pfit[0],strlen(input));

printf("\n\n\n\n\t\tENTER YOUR CHOICE (TO CONTINUE 1 TO EXIT 0) : ");

scanf("%d",&choice);

return choice;

}

void main()

{

int i,choice;

clrscr();

get\_input();

initial\_pop();

//display();

reproduction(); //sorting\_based\_on\_fitness();

display();

printf("\nENTER YOUR CHOICE (TO CONTINUE 1 TO EXIT 0) : ");

scanf("%d",&choice);

while((choice==1)&&(pfit[0]!=strlen(input)))

{

crossover();

gen++;

mutation();

reproduction(); //sorting\_based\_on\_fitness();

display();

choice=input\_choice();

}

sound(1000);

delay(200);

nosound();

delay(200);

results();

getch();

sound(1000);

delay(200);

nosound();

}