**PHỤ LỤC**

Hàm tạo gen

public class Genome

{

public double[] m\_genes;

public string[] m\_word\_genes;

private string s\_genes;

public string S\_genes

{

get { return s\_genes; }

set { s\_genes = value; }

}

private int m\_length;

private double m\_fitness;

static Random m\_random = new Random();

public Genome()

{

//

// TODO: Add constructor logic here

//

}

public Genome(int position, double stringLength)

{

//m\_length = length;

//m\_genes = new double[ length ];

//CreateGenes();

int bitLength = (int)Math.Log(stringLength, 2);

string binary = Convert.ToString(position, 2);

int i = 0;

string prefix = "";

while (i <= bitLength - binary.Length) {

prefix += "0";

i++;

}

s\_genes = prefix + binary;

}

public Genome(int length, bool createGenes)

{

m\_length = length;

m\_genes = new double[ length ];

if (createGenes)

CreateGenes();

}

public Genome(ref double[] genes)

{

m\_length = genes.GetLength(0);

m\_genes = new double[ m\_length ];

for (int i = 0 ; i < m\_length ; i++)

m\_genes[i] = genes[i];

}

public Genome(string gene)

{

m\_length = gene.Length;

m\_word\_genes = new string[m\_length];

CreateCharacterGenes(gene);

}

private void CreateGenes()

{

// DateTime d = DateTime.UtcNow;

for (int i = 0 ; i < m\_length ; i++)

m\_genes[i] = m\_random.NextDouble();

}

public void CreateCharacterGenes(string gene)

{

for (int i = 0; i < m\_length; i++)

{

m\_word\_genes[i] = gene[i].ToString();

}

}

public void Crossover(ref Genome genome2, out Genome child1, out Genome child2)

{

int pos = (int)(m\_random.NextDouble() \* (double)m\_length);

child1 = new Genome(m\_length, false);

child2 = new Genome(m\_length, false);

for(int i = 0 ; i < m\_length ; i++)

{

if (i < pos)

{

child1.m\_genes[i] = m\_genes[i];

child2.m\_genes[i] = genome2.m\_genes[i];

}

else

{

child1.m\_genes[i] = genome2.m\_genes[i];

child2.m\_genes[i] = m\_genes[i];

}

}

}

public void Crossover\_s(ref Genome genome2, out Genome child1, out Genome child2)

{

Random rd = new Random();

child1 = new Genome();

child2 = new Genome();

int pos = (int)rd.Next(0, genome2.s\_genes.Length - 1);

pos = (int)genome2.s\_genes.Length / 2;

int lenght = genome2.S\_genes.Length;

string genome\_t1 = s\_genes;

string genome\_t2 = genome2.s\_genes;

child1.S\_genes = genome\_t1.Substring(0, pos + 1) + genome\_t2.Substring(pos + 1, lenght - pos - 1);

genome\_t1 = s\_genes;

genome\_t2 = genome2.S\_genes;

child2.S\_genes = genome\_t2.Substring(0, pos + 1) + genome\_t1.Substring(pos + 1, lenght - pos - 1);

}

public void Mutate()

{

for (int pos = 0 ; pos < m\_length; pos++)

{

if (m\_random.NextDouble() < m\_mutationRate)

m\_genes[pos] = (m\_genes[pos] + m\_random.NextDouble())/2.0;

}

}

public void Mutate\_s()

{

Random rd = new Random();

int pos = (int)rd.Next(0, s\_genes.Length - 1);

StringBuilder sb = new StringBuilder(S\_genes);

if (m\_random.NextDouble() < m\_mutationRate)

{

if (S\_genes[pos] == '1')

{

sb[pos] = '0';

}

else

{

sb[pos] = '1';

}

}

S\_genes = sb.ToString();

}

public string Genes()

{

return s\_genes;

}

public void Output()

{

for (int i = 0 ; i < m\_length ; i++)

{

System.Console.WriteLine("{0:F4}", m\_genes[i]);

}

System.Console.Write("\n");

}

public void GetValues(ref double[] values)

{

for (int i = 0 ; i < m\_length ; i++)

values[i] = m\_genes[i];

}

private static double m\_mutationRate;

private string p;

public double Fitness

{

get

{

return m\_fitness;

}

set

{

m\_fitness = value;

}

}

public static double MutationRate

{

get

{

return m\_mutationRate;

}

set

{

m\_mutationRate = value;

}

}

public int Length

{

get

{

return m\_length;

}

}

}

Lớp so sánh các gen

GenomComparer.cs

public sealed class GenomeComparer : IComparer

{

public GenomeComparer()

{

}

public int Compare( object x, object y)

{

if ( !(x is Genome) || !(y is Genome))

throw new ArgumentException("Not of type Genome");

if (((Genome) x).Fitness > ((Genome) y).Fitness)

return 1;

else if (((Genome) x).Fitness == ((Genome) y).Fitness)

return 0;

else

return -1;

}

}

Lớp thuật toán di truyền (gồm các hàm tính các thế hệ lai tiếp theo và đột biến)

GA.cs

public delegate double GAFunction(string gene, string[] S, string[] Sm, double hsA, double hsB);

/// <summary>

/// Genetic Algorithm class

/// </summary>

public class GA

{

/// <summary>

/// Default constructor sets mutation rate to 5%, crossover to 80%, population to 100,

/// and generations to 2000.

/// </summary>

///

private double m\_mutationRate;

private double m\_crossoverRate;

private int m\_populationSize;

private int m\_generationSize;

private int m\_genomeSize;

private double m\_totalFitness;

private string m\_strFitness;

private bool m\_elitism;

double hsA;

double hsB;

private ArrayList m\_thisGeneration;

public ArrayList ThisGeneration

{

get { return m\_thisGeneration; }

set { m\_thisGeneration = value; }

}

private ArrayList m\_nextGeneration;

private ArrayList m\_fitnessTable;

static Random m\_random = new Random();

static private GAFunction getFitness;

private double xacSuatLai;

private System.Windows.Forms.TextBox txtXSDotBien;

private int p;

private int soTheHe;

public GAFunction FitnessFunction

{

get

{

return getFitness;

}

set

{

getFitness = value;

}

}

// Properties

public int PopulationSize

{

get

{

return m\_populationSize;

}

set

{

m\_populationSize = value;

}

}

public int Generations

{

get

{

return m\_generationSize;

}

set

{

m\_generationSize = value;

}

}

public int GenomeSize

{

get

{

return m\_genomeSize;

}

set

{

m\_genomeSize = value;

}

}

public double CrossoverRate

{

get

{

return m\_crossoverRate;

}

set

{

m\_crossoverRate = value;

}

}

public double MutationRate

{

get

{

return m\_mutationRate;

}

set

{

m\_mutationRate = value;

}

}

public string FitnessFile

{

get

{

return m\_strFitness;

}

set

{

m\_strFitness = value;

}

}

/// <summary>

/// Keep previous generation's fittest individual in place of worst in current

/// </summary>

public bool Elitism

{

get

{

return m\_elitism;

}

set

{

m\_elitism = value;

}

}

public GA()

{

InitialValues();

m\_mutationRate = 0.05;

m\_crossoverRate = 0.80;

m\_populationSize = 100;

m\_generationSize = 2000;

m\_strFitness = "";

}

public GA(double crossoverRate, double mutationRate, int populationSize, int generationSize, int genomeSize)

{

InitialValues();

m\_mutationRate = mutationRate;

m\_crossoverRate = crossoverRate;

m\_populationSize = populationSize;

m\_generationSize = generationSize;

m\_genomeSize = genomeSize;

m\_strFitness = "";

}

public GA(double crossoverRate, double mutationRate, int populationSize, int generationSize, double a, double b)

{

InitialValues();

m\_mutationRate = mutationRate;

m\_crossoverRate = crossoverRate;

m\_populationSize = populationSize;

m\_generationSize = generationSize;

this.hsA = a;

this.hsB = b;

m\_strFitness = "";

}

public GA(int genomeSize)

{

InitialValues();

m\_genomeSize = genomeSize;

}

public void InitialValues()

{

m\_elitism = false;

}

/// <summary>

/// Method which starts the GA executing.

/// </summary>

public void Go(String[] populationInput, string[] Sm)

{

if (getFitness == null)

throw new ArgumentNullException("Need to supply fitness function");

//if (m\_genomeSize == 0)

// throw new IndexOutOfRangeException("Genome size not set");

// Create the fitness table.

m\_fitnessTable = new ArrayList();

m\_thisGeneration = new ArrayList(populationInput.Length);

m\_nextGeneration = new ArrayList(populationInput.Length);

Genome.MutationRate = m\_mutationRate;

CreateGenomes(populationInput);

RankPopulation(populationInput, Sm);

StreamWriter outputFitness = null;

bool write = false;

if (m\_strFitness != "")

{

write = true;

outputFitness = new StreamWriter(m\_strFitness);

}

for (int i = 0; i < m\_generationSize; i++)

{

CreateNextGeneration();

RankPopulation(populationInput, Sm);

}

if (outputFitness != null)

outputFitness.Close();

}

/// <summary>

/// After ranking all the genomes by fitness, use a 'roulette wheel' selection

/// method. This allocates a large probability of selection to those with the

/// highest fitness.

/// </summary>

/// <returns>Random individual biased towards highest fitness</returns>

private int RouletteSelection()

{

double randomFitness = m\_random.NextDouble() \* m\_totalFitness;

int idx = -1;

int mid;

int first = 0;

int last = m\_populationSize -1;

mid = (last - first)/2;

// ArrayList's BinarySearch is for exact values only

// so do this by hand.

while (idx == -1 && first <= last)

{

if (randomFitness < (double)m\_fitnessTable[mid])

{

last = mid;

}

else if (randomFitness > (double)m\_fitnessTable[mid])

{

first = mid;

}

mid = (first + last)/2;

// lies between i and i+1

if ((last - first) == 1)

idx = last;

}

return idx;

}

/// <summary>

/// Rank population and sort in order of fitness.

/// </summary>

private void RankPopulation(string[] S, string[] Sm)

{

m\_totalFitness = 0;

for (int i = 0; i < S.Length; i++)

{

Genome g = ((Genome) m\_thisGeneration[i]);

g.Fitness = FitnessFunction(g.Genes(), S, Sm, hsA, hsB);

m\_totalFitness += g.Fitness;

}

m\_thisGeneration.Sort(new GenomeComparer());

// now sorted in order of fitness.

double fitness = 0.0;

m\_fitnessTable.Clear();

for (int i = 0; i < S.Length; i++)

{

fitness += ((Genome)m\_thisGeneration[i]).Fitness;

m\_fitnessTable.Add((double)fitness);

}

}

/// <summary>

/// Create the \*initial\* genomes by repeated calling the supplied fitness function

/// </summary>

private void CreateGenomes(String[] populationInput)

{

for (int i = 0; i < populationInput.Length; i++)

{

Genome g = new Genome(i, populationInput.Length);

m\_thisGeneration.Add(g);

//if (populationInput[i].Length > 0)

//

// Genome g = new Genome(populationInput[i]);

// m\_thisGeneration.Add(g);

//}

}

}

private void CreateNextGeneration()

{

m\_nextGeneration.Clear();

Genome g = null;

if (m\_elitism)

g = (Genome)m\_thisGeneration[m\_populationSize - 1];

for (int i = 0; i < m\_populationSize; i += 2)

{

int pidx1 = RouletteSelection();

int pidx2 = RouletteSelection();

Genome parent1, parent2, child1, child2;

parent1 = ((Genome)m\_thisGeneration[pidx1]);

parent2 = ((Genome)m\_thisGeneration[pidx2]);

if (m\_random.NextDouble() < m\_crossoverRate)

{

parent1.Crossover\_s(ref parent2, out child1, out child2);

}

else

{

child1 = parent1;

child2 = parent2;

}

child1.Mutate\_s();

child2.Mutate\_s();

m\_nextGeneration.Add(child1);

m\_nextGeneration.Add(child2);

}

if (m\_elitism && g != null)

m\_nextGeneration[0] = g;

m\_thisGeneration.Clear();

for (int i = 0 ; i < m\_populationSize; i++)

m\_thisGeneration.Add(m\_nextGeneration[i]);

}

public void GetBest(out int values, out double fitness)

{

Genome g = ((Genome)m\_thisGeneration[m\_populationSize-1]);

//values = new double[g.Length];

//g.GetValues(ref values);

values = int.Parse(Convert.ToInt32(g.S\_genes, 2).ToString());

fitness = (double)g.Fitness;

}

public void GetWorst(out double[] values, out double fitness)

{

GetNthGenome(0, out values, out fitness);

}

public void GetNthGenome(int n, out double[] values, out double fitness)

{

if (n < 0 || n > m\_populationSize-1)

throw new ArgumentOutOfRangeException("n too large, or too small");

Genome g = ((Genome)m\_thisGeneration[n]);

values = new double[g.Length];

g.GetValues(ref values);

fitness = (double)g.Fitness;

}

//public int maxLengthString(string S, string Sm)

//{

// char[] delimiterChars = { ' ', ',', '.', ':', '\t' };

// string[] arrayS = S.Split(delimiterChars);

// for (int i = 0; i < S.Length; i++)

// {

// }

//}

}

Lớp giao diện (gồm các hàm tính độ thích nghi và xử lý trên giao diện)

Form.cs

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

string[] words;

string[] words\_search;

private ArrayList words\_result;

private void Form1\_Load(object sender, EventArgs e)

{

System.Windows.Forms.ToolTip ToolTip1 = new System.Windows.Forms.ToolTip();

ToolTip1.SetToolTip(this.btnMoVB, "Mở văn bản");

System.Windows.Forms.ToolTip ToolTip2 = new System.Windows.Forms.ToolTip();

ToolTip2.SetToolTip(this.btnTim, "Tìm");

txtSoTheHe.Text = "10";

txtXSLai.Text = "0.8";

txtXSDotBien.Text = "0.05";

txtHeSoA.Text = "0.7";

txtHeSoB.Text = "0.3";

}

private void btnMoVB\_Click(object sender, EventArgs e)

{

//DialogResult result = openFileDialog1.ShowDialog();

//if (result == DialogResult.OK) // Test result.

//{

//Do whatever you want

//openFileDialog1.FileName .....

//}

OpenFileDialog dialog = new OpenFileDialog();

dialog.Filter = "Text files | \*.pdf"; // file types, that will be allowed to upload

dialog.Multiselect = false; // allow/deny user to upload more than one file at a time

if (dialog.ShowDialog() == DialogResult.OK) // if user clicked OK

{

String path = dialog.FileName; // get name of file

//string content;

PdfReader reader = new PdfReader(path);

string text = string.Empty;

for (int page = 1; page <= reader.NumberOfPages; page++)

{

text += PdfTextExtractor.GetTextFromPage(reader, page);

}

reader.Close();

txtVanBan.Text = text;

//using (StreamReader reader = new StreamReader(new FileStream(path, FileMode.Open), new UTF8Encoding())) // do anything you want, e.g. read it

//{

txtDuongDan.Text = path;

// content = reader.ReadToEnd();

// txtVanBan.Text = content;

//char[] delimiterChars = { ' ', ',', '.', ':', '\t' };

char[] delimiterChars = { ' ', '\t' };

// //string text = "one\ttwo three:four,five six seven";

// //System.Console.WriteLine("Original text: '{0}'", text);

string[] splitText = text.Split(delimiterChars);

List<string> result = new List<string>();

foreach (string item in splitText)

{

if (item != "")

result.Add(item);

}

words = result.ToArray();

txtSoKyTu.Text = words.Count().ToString();

// //System.Console.WriteLine("{0} words in text:", words.Length);

// //for (int i = 0; i < words.Length; i++)

// //{

// // txtVanBan.AppendText(words[i].ToString() + "\n");

// //}

// //txt\_Result.Text = content;

//}

}

}

private void btn\_Search\_Click(object sender, EventArgs e)

{

executeSearch();

}

private void executeSearch()

{

if (checkInput() == "")

{

char[] delimiterChars = { ' ', ',', '.', ':', '\t' };

//char[] delimiterChars = { ' ', '\t' };

words\_search = txtTim.Text.Split(delimiterChars);

int soTheHe = 0;

int.TryParse(txtSoTheHe.Text, out soTheHe);

double xacSuatLai = 0.8;

double.TryParse(txtXSLai.Text, out xacSuatLai);

double xacSuatDotBien = 0.05;

double.TryParse(txtXSDotBien.Text, out xacSuatDotBien);

double hsA = 0.7;

double.TryParse(txtHeSoA.Text, out hsA);

double hsB = 0.3;

double.TryParse(txtHeSoB.Text, out hsB);

GA ga = new GA(xacSuatLai, xacSuatDotBien, words.Length, soTheHe, hsA, hsB);

ga.FitnessFunction = new GAFunction(theActualFunction);

//ga.FitnessFile = @"H:\fitness.csv";

ga.Elitism = true;

ga.Go(words, words\_search);

//int values;

//double fitness;

//ga.GetBest(out values, out fitness);

////System.Console.WriteLine("Best ({0}):", fitness);

////for (int i = 0; i < values.Length; i++)

//// System.Console.WriteLine("{0} ", values[i]);

////label2.Text = fitness.ToString();

//txtViTriXuatHien.Clear();

//txtViTriXuatHien.AppendText("vi tri van ban " + (1 + values).ToString() + "\n");

//txtViTriXuatHien.AppendText("do thich nghi " + fitness.ToString() + "\n");

//string textResult = "";

//int i = 0;

//while (i < 20)

//{

// textResult += words[values + i] + " ";

// i++;

//}

//txtResult.Text = textResult;

this.words\_result = ga.ThisGeneration;

List<StoreResult> list = getResultPosition();

txtViTriXuatHien.Clear();

txtResult.Clear();

foreach (StoreResult item in list)

{

txtViTriXuatHien.AppendText(item.position + " ");

int tmpPos = item.position;

bool flag = false;

int i = tmpPos;

if (tmpPos == 0)

{

while (flag == false)

{

if (words[i].Contains(".") || words[i].Contains("?") || words[i].Contains("!"))

{

txtResult.AppendText(words[i]);

flag = true;

}

else

{

txtResult.AppendText(words[i] + " ");

i++;

}

}

}

else

{

i = tmpPos;

int startSen = i;

while (flag == false)

{

if (words[i].Contains(".") || i == 0 || words[i].Contains("?") || words[i].Contains("!"))

{

startSen = i == 0? 0 : i + 1;

flag = true;

}

else

{

i--;

}

}

flag = false;

while (flag == false)

{

if (words[startSen].Contains(".") || words[i].Contains("?") || words[i].Contains("!"))

{

txtResult.AppendText(words[startSen]);

flag = true;

}

else

{

txtResult.AppendText(words[startSen] + " ");

startSen++;

}

}

}

txtResult.AppendText("\n");

//txtResult.Text = textResult;

}

//ga.GetWorst(out values, out fitness);

//txtVanBan.AppendText(fitness.ToString() + "\n");

//label3.Text = fitness.ToString();

//System.Console.WriteLine("\nWorst ({0}):", fitness);

//for (int i = 0; i < values.Length; i++)

// System.Console.WriteLine("{0} ", values[i]);

//Console.ReadLine();

}

else

MessageBox.Show(checkInput());

}

private List<StoreResult> getResultPosition()

{

List<StoreResult> list = new List<StoreResult>();

int searchTextLength = words\_search.Length;

Genome g\_last = ((Genome)words\_result[words\_result.Count - 1]);

int position\_last = int.Parse(Convert.ToInt32(g\_last.S\_genes, 2).ToString());

StoreResult result\_last = new StoreResult();

result\_last.position = position\_last;

result\_last.fitness = g\_last.Fitness;

list.Add(result\_last);

for (int i = words\_result.Count - 2; i > 0; i--)

{

Genome g = ((Genome)words\_result[i]);

int position = int.Parse(Convert.ToInt32(g.S\_genes, 2).ToString());

if ((list[list.Count - 1].position + searchTextLength) < position && g.Fitness > 0)

{

StoreResult sr = new StoreResult();

sr.position = position;

sr.fitness = g.Fitness;

list.Add(sr);

}

}

return list;

}

private string checkInput()

{

string result = "";

if (words == null)

result += "Vui lòng chọn tệp tin văn bản\n";

if (txtTim.Text == "")

result += "Vui lòng nhập văn bản cần tìm\n";

return result;

}

// optimal solution for this is (0.5,0.5)

public static double theActualFunction(string gene, string[] S, string[] Sm, double hsA, double hsB)

{

//if (values.GetLength(0) != 2)

// throw new ArgumentOutOfRangeException("should only have 2 args");

//double x = values[0];

//double y = values[1];

//double n = 9; // should be an int, but I don't want to waste time casting.

//double f1 = Math.Pow(15\*x\*y\*(1-x)\*(1-y)\*Math.Sin(n\*Math.PI\*x)\*Math.Sin(n\*Math.PI\*y),2);

//return f1;

double a = hsA;

double b = hsB;

//F(x) = a.G(x) + b.H(x)

int position = int.Parse(Convert.ToInt32(gene, 2).ToString());

double test = a \* G(position, S, Sm) + b \* H(position, S, Sm);

return a \* G(position, S, Sm) + b \* H(position, S, Sm);

}

public static int G(int x, string[] X, string[] Y)

{

int[,] L = new int[Y.Count(), Y.Count()];

for (int i = 0; i < Y.Count(); i++)

{

L[i, 0] = 0;

}

for (int j = 0; j < Y.Count(); j++)

{

L[0, j] = 0;

}

//if (x == 0)

// x = 1;

for (int i = 1; i < Y.Count() && (i + x - 1) < X.Count(); i++)

{

for (int j = 1; j < Y.Count(); j++)

{

string tmp = Regex.Replace(X[i + x - 1], "[^0-9a-zA-Z]+", "");

if (tmp == Y[j])

{

L[i, j] = L[i - 1, j - 1] + 1;

}

else

{

L[i, j] = Math.Max(L[i - 1, j], L[i, j - 1]);

}

}

}

return L[Y.Count() - 1, Y.Count() - 1];

}

public static int H(int positionX, string[] S, string[] Sm)

{

int count = 0;

for (int i = 0; i < Sm.Length; i++)

{

for (int j = positionX; j < (positionX + Sm.Length) && j < S.Length; j++)

{

string tmp = Regex.Replace(S[j], "[^0-9a-zA-Z]+", "");

if (Sm[i] == tmp)

count++;

}

}

return count;

}

private void txtDuongDan\_TextChanged(object sender, EventArgs e)

{

}

private void txtTim\_KeyDown(object sender, KeyEventArgs e)

{

if (e.KeyValue == 13)

{

executeSearch();

}

}

private void groupBox1\_Enter(object sender, EventArgs e)

{

}

private void label10\_Click(object sender, EventArgs e)

{

}

class StoreResult {

public int position { get; set; }

public double fitness { get; set; }

}

}