Cosc 1P03 Assignment 3

ADTs

(Due date Mar 11th 16:00 est, Late date Mar 14th 16:00 est)

Back Ground -- Genetic algorithms

From Wikipedia

In computer science and operations research, a **genetic algorithm** (**GA**) is a metaheuristic inspired by the process of natural selection that belongs to the larger class of evolutionary algorithms (EA). Genetic algorithms are commonly used to generate high-quality solutions to optimization and search problems by relying on biologically inspired operators such as mutation, crossover and selection.[1] Some examples of GA applications include optimizing decision trees for better performance, automatically solve sudoku puzzles,[2] hyperparameter optimization, etc.

The Assignment

Creating a genetic algorithm (GA) to solve a problem is beyond the scope of this course. However, a major part of a GA is to create the next population (generation) of chromosome or Genome. For example, if a GA has a population of say 100 Genome, then the next generation created from the current population will consist of 100 Genome, generated by way of inheritance, mutation, inversion, and crossover.

To aid in the management and creation of the new Genome you will write a package called GENE which supports the Genome type as an interface. In essence, a Genome consists of a string of items defined by an alphabet defined as a separate interface. For example if a Genome is to represent "aabdgba" then the alphabet consists of the characters "abdg" only.

Thus, an interface defined as:

```
public interface SomeAlphabet {

      final char[] alphabet={'a','b','d','g'};

}
```

Defines the alphabet for which the Genome must comply with. When creating the implementation class, the Genome interface and the above interface are implemented.

To hold the Genome you will use a String, where the contents of the string must belong to the defined alphabet. Each Genome has a length (Cardinality) and usually all Genome in a given GA will be the same length. To aid in the development of the package the interface is given:

```
package GENE;

public interface Genome{

   // Returns the cardinality of this.

   public int Card();

   // Creates a clone object of this

   public Genome Clone();

   // Concatenate this and g, this = this + g;

   public Genome ConCat(Genome g);

   // Returns true if this and g are the same

   public boolean Equal(Genome g);


   // Each element of this will randomly mutate to another element of the alphabet
   // if a random variable Pi<p where 0<=Pi<=1 & 0<=p<=1

   public Genome Mutate(double p);


   // The first i elements of this will be prepended to the tail elements of B which
   // contains the elements of B less the first i elements.
   //
   // E.g.  A = aaaaa & B=bbbbb  then A.OnePointCrossOver(B,2) => aaabb
   // Raises UsageException if A & B are not the same length or i>A.Card().

   public Genome OnePointCrossOver(Genome g, int i);


   // The elements of this which lye within i & j are substituted with the element
   // that lye within i & j of g.
   //
   // E.g. A=aaaaaa & B=bbbbbb then A.TwoPointCrossOver(B,2,4) => aabbba
   // Raises UsageException if A & B are not the same length or i>j>A.Card().


   public Genome TwoPointCrossOver(Genome g, int i, int j);


   // The Genome this is reversed. E.g. A=abcde then A.Inversion() => edcba

   public Genome Inversion();

}
```

It is important to note that objects of Genome are immutable, that is operations on *this* will create a new Genome object. When creating an implementation of Genome consider creating a class which describes the alphabet and restricts elements of the Genome to that alphabet. E.g.

```
public class MyGene implements Genome, AlphabetInterface{
```

will implement the Genome interface restricting the elements within the Genome to the alphabet defined in the interface AlphabetInterface. A valid instantiation of this implementation would be.

```
Genome a;
a = new MyGene("aaagada");
```

where

```
a = new MyGene("aaxyz");
```

would raise a UsageException.

This check should happen when a new object is created.

A UsageException is a very broad exception error. Enhance UsageException to pass a detail message which can then be displayed when UsageException is caught. See lecture example. There are many places where throwing a UsageException is appropriate.

The program package which you create should contain the interface, an implementation with an appropriate alphabet interface; and an UsageException class. To test your program, write a test harness which exercises all components of the program. You are responsible for creating a comprehensive test harness, which exercises all aspects of your implementation including exception cases.

Submission

Submit via Sakai.