A study on

RANDOM STRING SOLUTION

Using Genetic Algorithm

Department of Computer Science
ANANDA CHANDRA COLLEGE
Jalpaiguri, West Bengal 735224

Second Semester, 2022

Department of Computer Science

ANANDA CHANDRA COLLEGE

Certificate

This is to certify that this is a bonafide record of the report presented by the students whose names are given below during Second Semester 2021-22 in fulfilment of the requirement of the course.

Name of Students	Roll No	Contributions
Anirban Routh	21DSH0278	Report, Slides, Presentation
Sayan Das	21DSH0124	Slides, Presentation
Biswarup Roy	21DSH0055	Slides, Presentation

Guide

Kaniska Sarkar.
(Professor)

Date: 30/06/2022

Abstract

A genetic algorithm is used to convert a randomly generated string into a specific string as asked by the user.

Contents

1 Introduction

1.1 Description of "Random to Particular String Converter"

This report concerns the development of a genetic algorithm that can turn a randomly generated string into a particular string asked by the user.

1.2 Description of Genetic Algorithm

The solution created for this report is a genetic algorithm. Candidate solutions are represented as a Chromosomes. Chromosomes are then evaluated using a fitness function and only those which are near similar to the desired string are kept. Those Chromosomes those found fit are the randomly mutated to form new strings and the process repeats until the desired string is achieved.

1.3 Uses of Genetic Algorithm

Genetic Algorithm is used mainly in optimization problems like in:
1. DNA Analysis
2. Machine Learning
3. Vehicle Routing Problems
4. Robot Trajectory Generation
5. Cyber Security

1.4 Pseudo Code of Genetic Algorithm

```
GA()
    initialize population
    find fitness of population

    while (termination criteria is reached) do
        parent selection
        crossover with probability pc
        mutation with probability pm
        decode and fitness calculation
        survivor selection
        find best
    return best
```

2 Random String Solution using Genetic Algorithm

2.1 Random String Generation

To begin the solution we first need to generate a random string which then can be gradually mutated to form the desired string of the user.

Code to generate Random String:

```
private static Random random = new Random();
private static String randomString(final String allowedCharacters, final int length)
{
    return IntStream.generate(allowedCharacters::length).limit(length)
            .map(random::nextInt).mapToObj(allowedCharacters::charAt)
            .map(Object::toString).collect(Collectors.joining());
}
```

2.2 Fitness Function

From the entire Population we need to measure how well a candidate solution (chromosome) fits their environment. In this case, we want to know, how similar a given string is to a certain template, character by character. Therefore we use the following code to find the fitness value of the chromosome:

```
private static int fitness(final String testedString, final String perfectString)
{
    return IntStream.range(0, min(testedString.length(), perfectString.length()))
            .map(index -> abs(testedString.charAt(index) - perfectString.charAt(index))).sum();
}
```

This code is to output fitness value of each individual chromosome from each mutated population:

```
private static int fitness(final String testedString, final String perfectString)
{
    return IntStream.range(0, min(testedString.length(), perfectString.length()))
            .map(index -> abs(testedString.charAt(index) - perfectString.charAt(index))).sum();
}
private static void outputGeneration(final List<String> generation, final String perfectString)
{
    generation.stream().sorted(Comparator.comparingInt(string -> fitness(string, perfectString)))
            .peek(string -> System.out.format("%s - ", string))
            .map(string -> fitness(string, perfectString))
            .forEach(System.out::println);
    System.out.println("----- ----- ----- ----- ----- ----- ----- -----");
}
```

2.3 Survivor Selection

Fitness Population Selection is one of the most common way of parent selection, so next to remove all the chromosomes that doesn't match the required fitness function value.

```
private static List<String> unleashPredators
    (
        final List<String> generation,
        final String perfectString,
        final int survivors
    )
    {
    return generation.stream().sorted(Comparator.comparingInt
```

```
    (string -> fitness(string, perfectString)))
            .limit(survivors).collect(Collectors.toList());
    }
```

2.4 Crossover

We create new chromosomes, which share roughly equal amounts of both parent's(chromosomes) Gene.

```
 private static String breed(final String mom, final String dad) {
    return IntStream.range(0, min(mom.length(), dad.length()))
            .mapToObj(index -> random.nextBoolean() ? mom.charAt(index) : dad.charAt(index))
            .map(Object::toString).collect(Collectors.joining());
}
```

2.5 Mutation

We will select new chromosomes to mutate again completely randomly:

```
private static List<String> repopulate(
    final List<String> generation, final int offspring
) {
    return Stream.generate(() -> breed(
            generation.get(random.nextInt(generation.size())),
            generation.get(random.nextInt(generation.size()))
    )).limit(offspring).collect(Collectors.toList());
}

private static String mutate(final String individual)
{
        final int mutationLocation = random.nextInt(individual.length());
        final char newChar = (char) (individual.charAt(mutationLocation) +
        (random.nextBoolean() ? 1 : -1));
        final StringBuilder builder = new StringBuilder(individual);
        builder.setCharAt(mutationLocation, newChar);
        return builder.toString();
}
private static List<String> mutateGeneration(final List<String> generation,final double mutationChanc
    {
        return generation.stream()
            .map(individual -> random.nextDouble() < mutationChance ?
            mutate(individual) : individual)
            .collect(Collectors.toList());
    }
```

2.6 Termination Conditation

The termination condition is necessary to determine when the Genetic Algorithm run will end. The termination condition may be set to generate a certain number of generations or to terminate after a desired fitness value is achieved or it also can be a combination of both.

2.7 Code Implementation

Java language was used for implementing the code. The complete code was composed of basic functions of genetic algorithm.

3 Code

```
import java.util.Comparator;
import java.util.List;
import java.util.Random;
import java.util.stream.Collectors;
import java.util.stream.IntStream;
import java.util.stream.Stream;

import static java.lang.StrictMath.abs;
import static java.lang.StrictMath.min;

public class GeneticHelloWorld {
    private static Random random = new Random();

    private static String randomString(final String allowedCharacters, final int length)
    {
        return IntStream.generate(allowedCharacters::length).limit(length).map(random::nextInt)
            .mapToObj(allowedCharacters::charAt).map(Object::toString)
            .collect(Collectors.joining());
    }

    private static List<String> firstGeneration(final String genes, final intdnaLength,
    final int count)
    {
        return Stream.generate(() -> randomString(genes, dnaLength)).limit(count)
        .collect(Collectors.toList());
    }

    private static int fitness(final String testedString, final String perfectString)
    {
        return IntStream.range(0, min(testedString.length(), perfectString.length()))
                .map(index -> abs(testedString.charAt(index) - perfectString.charAt(index))).sum();
    }

    private static void outputGeneration(final List<String> generation, final String perfectString)
    {
        generation.stream().sorted(Comparator.comparingInt(string -> fitness
        (string, perfectString)))
                .peek(string -> System.out.format("%s - ", string)).map(string ->
                fitness(string, perfectString))
                .forEach(System.out::println);
        System.out.println("----- ----- ----- ----- ----- ----- ----- -----");
    }

    private static List<String> unleashPredators(
            final List<String> generation,
            final String perfectString,
            final int survivors
    ) {
        return generation.stream().sorted(Comparator.comparingInt(string ->
        fitness(string, perfectString)))
                .limit(survivors).collect(Collectors.toList());
    }

    private static String breed(final String mom, final String dad) {
        return IntStream.range(0, min(mom.length(), dad.length()))
```

```
                .mapToObj(index -> random.nextBoolean() ? mom.charAt(index) : dad.charAt(index))
                .map(Object::toString).collect(Collectors.joining());
    }

    private static List<String> repopulate(final List<String> generation, final int offspring) {
        return Stream.generate(() -> breed(
                generation.get(random.nextInt(generation.size())),
                generation.get(random.nextInt(generation.size()))
                )).limit(offspring).collect(Collectors.toList());
    }

    private static String mutate(final String individual) {
        final int mutationLocation = random.nextInt(individual.length());
        final char newChar = (char) (individual.charAt(mutationLocation) +
        (random.nextBoolean() ? 1 : -1));
        final StringBuilder builder = new StringBuilder(individual);
        builder.setCharAt(mutationLocation, newChar);
        return builder.toString();
    }

    private static List<String> mutateGeneration(final List<String> generation,
    final double mutationChance) {
        return generation.stream()
                .map(individual -> random.nextDouble() < mutationChance ?
                mutate(individual) : individual)
                .collect(Collectors.toList());
    }

    private static final String PERFECT_STRING = "Hello World";
    private static final int GENERATION_SIZE = 40;
    private static final int SURVIVOR_SIZE = 10;
    private static final double MUTATION_CHANCE = 0.05;
    private static final int GENERATION_COUNT = 1000;

    public static void main(final String... args) {
        List<String> generation = firstGeneration("abcdefghijklmnopqrstuvwxyz",
        PERFECT_STRING.length(), GENERATION_SIZE);
        outputGeneration(generation, PERFECT_STRING);
        for (int index = 0; index < GENERATION_COUNT; index++) {
            generation = unleashPredators(generation,
            PERFECT_STRING, SURVIVOR_SIZE);
            generation = repopulate(generation, GENERATION_SIZE);
            generation = mutateGeneration(generation, MUTATION_CHANCE);
            outputGeneration(generation, PERFECT_STRING);
        }
    }
}
```

References

[htta] https://en.wikipedia.org/wiki/Genetic$_a lgorithm$.

[httb] https://www.geeksforgeeks.org/.

[Jon17] Paulina Jonušaitė. Hello world. *Quora*, 2017.

[Jon17] [httb] [htta]