Programming Assignment: Genetic Algorithm

Topics and references

- Evolutionary algorithm, fitness function, crossover, mutation
- Framework of Genetic Algorithm
- 8-Queens Problem

Task

In this assignment you have to implement the Genetic Algorithm to solve N-Bits problem and 8-Queens Puzzle.

You have to implement a generic version of the algorithm and a domain-specific code for the N-Bits and 8-Queens as test beds. You can find the complete list of structures and functions to implement in form of pseudocode on the Moodle web page and in the assignment framework that is provided and accessible on the SVN server.

Complete instruction to the assignment and explanation of the method **will be given in the class**.

You implementation must passes all given tests in order to get the full mark for the assignment.

Submission details

Please read the following details carefully and adhere to all requirements to avoid unnecessary deductions. Since submission details will remain the same for all programming assessments, this portion will be skipped in future documents detailing labs and assignments.

Source files

You have to submit the header `functions.h` and source file `functions.cpp`.

```
#ifndef FUNCTIONS_H
#define FUNCTIONS_H

#include <iostream>
#include <sstream>
#include <vector>

#include "data.h"

#define UNUSED(x) (void)x;

// Returns result of addition of all genes in a chromosome
template<typename Gene>
struct Fitness_Accumulate
{
    int operator()(const std::vector<Gene>& genes) const
    {
        UNUSED(genes);
```

```
        // Your code ...

        return 0;
    }
};

// Returns a measure in percentages that indicates the fitness of
// a particular chromosome into a particular solution.
// Fittest chromosome has all genes equal 1.
template<typename Gene>
struct Fitness_Nbits
{
    int operator()(const std::vector<Gene>& genes) const
    {
        UNUSED(genes);

        // Your code ...

        return 0;
    }
};

// Returns a measure in percentages that indicates the fitness of
// a particular chromosome into a particular solution of 8 queens
// problem
template<typename Gene>
struct Fitness_8queens
{
    int operator()(const std::vector<Gene>& genes) const
    {
        UNUSED(genes);

        // Your code ...

        return 0;
    }
};


namespace AI
{
    // Crossover methods for the genetic algorithm
    enum CrossoverMethod { Middle, Random  };

    // Simplest gene seeding class/function
    struct Seed
    {
        int operator()(int p = 0) const
        {
            return p;
        }
    };

    // Gene seeding class/function with a fixed value
    template<int Val = 0>
    struct Seed_Value
    {
```

```
        int operator()(int /* p */ = 0) const
        {
            return Val;
        }
    };

    // Gene random seeding class/function
    template<int Max>
    struct Seed_Random
    {
        int operator()(int /* p */ = 0) const
        {
            // Your code ...
            return 0;
        }
    };

    // Gene class
    template<typename T = int, typename S = Seed>
    class Gene
    {
        T value;

    public:
        Gene(int p = 0)
            : value{ S()(p) }
        {
        }

        T getValue() const
        {
            return value;
        }

        void setValue(T v)
        {
            value = v;
        }

        friend std::ostream& operator<<(std::ostream& os, const Gene& rhs)
        {
            os << rhs.value;
            return os;
        }
    };

    // Chromosome class
    template<typename Gene, typename Fitness, size_t Size>
    class Chromosome
    {
        std::vector<Gene> genes;
        int fitness;

    public:

        using gene_type = Gene;

        static const size_t size = Size;
```

```
    Chromosome()
        : genes(Size), fitness{ Fitness()(genes) }
    {
    }

    std::vector<Gene>& getGenes()
    {
        return genes;
    }

    void setGenes(const std::vector<Gene>& v)
    {
        genes = v;
        fitness = Fitness()(genes);
    }

    Gene getGene(size_t i) const
    {
        return genes[i];
    }

    void setGene(size_t i, const Gene& v)
    {
        genes[i] = v;
        fitness = Fitness()(genes);
    }

    int getFitness() const
    {
        return fitness;
    }

    // Select a random mutation point and change
    // gene at the mutation point
    void randomMutation()
    {
        setGene(std::rand() % Chromosome::size, Gene());
    }

    // Copy genes from a source
    void copyGenesFrom(Chromosome& src)
    {
        std::copy(src.genes.begin(), src.genes.end(), genes.begin());
        fitness = Fitness()(genes);
    }

    friend std::ostream& operator<<(std::ostream& os,
        const Chromosome& rhs)
    {
        os << '[';
        for (auto it = rhs.genes.begin(); it != rhs.genes.end(); ++it)
            os << *it << (it + 1 != rhs.genes.end() ? "," : "");
        os << "]=" << rhs.fitness;
        return os;
    }
};
```

```cpp
// Individual class
template<typename Chromosome>
class Individual
{
    Chromosome chromosome;

public:

    using chromosome_type = Chromosome;
    using gene_type = typename Chromosome::gene_type;

    Individual()
        : chromosome{ }
    {
    }

    Chromosome& getChromosome()
    {
        return chromosome;
    }

    std::vector<gene_type>& getGenes()
    {
        return chromosome.getGenes();
    }

    void setGenes(const std::vector<gene_type>& v)
    {
        chromosome.setGenes(v);
    }

    gene_type getGene(size_t i) const
    {
        return chromosome.getGene(i);
    }

    void copyGenesFrom(Individual& individual)
    {
        chromosome.copyGenesFrom(individual.chromosome);
    }

    void setGene(size_t i, gene_type gene)
    {
        chromosome.setGene(i, gene);
    }

    int getFitness() const
    {
        return chromosome.getFitness();
    }

    friend std::ostream& operator<<(std::ostream& os, Individual& rhs)
    {
        os << rhs.chromosome;
        return os;
    }
};
```

```cpp
// Population class
template<typename Individual>
class Population
{
    std::vector<Individual> individuals;
    Individual* fittest;

public:
    Population(size_t size = 0)
        : individuals{ }, fittest{ nullptr }
    {
        if (size)
        {
            individuals.resize(size);
            updateFittest();
        }
    }

    size_t getSize() const
    {
        return individuals.size();
    }

    Individual& getIndividual(size_t i)
    {
        return individuals[i];
    }

    Individual* getFittest() const
    {
        return fittest;
    }

    void updateFittest()
    {
        // Your code ...
    }

    friend std::ostream& operator<<(std::ostream& os, Population& rhs)
    {
        os << " = " << rhs.getFittest()->getFitness() << std::endl;
        for (size_t i = 0; i < rhs.getSize(); ++i)
            os << "  " << i << ':' << rhs.getIndividual(i) << std::endl;
        return os;
    }
};

// Genetic Algorithm class
template<typename Individual>
class GeneticAlgorithm
{
    Population<Individual>* population;
    int generation;

public:
    GeneticAlgorithm()
        : population{ nullptr }, generation{ 0 }
    {
```

```
        }

        ~GeneticAlgorithm()
        {
            // Your code ...
        }

        Individual* getFittest() const
        {

            // Your code ...

            return nullptr;
        }

        // Implementation of the Roulette Wheel Selection. The probability of an
        // individual to be selected is directly proportional to its fitness.
        Population<Individual>* selection(size_t sizeOfPopulation)
        {
            UNUSED(sizeOfPopulation);

            // Your code ...

            return nullptr;
        }

        // Crossover parents genes
        void crossover(Population<Individual>* newGeneration,
            CrossoverMethod crossoverMethod)
        {
            UNUSED(newGeneration);
            UNUSED(crossoverMethod);

            // Your code ...
        }

        // Do mutation of genes under a random probability
        void mutation(Population<Individual>* newGeneration,
            int mutationProbability)
        {
            UNUSED(newGeneration);
            UNUSED(mutationProbability);

            // Your code ...
        }

        // Replace existing population if any with a new generation
        void setPopulation(Population<Individual>* newGeneration)
        {
            UNUSED(newGeneration);

            // Your code ...
        }

        // Start the search
        void run(size_t sizeOfPopulation = 100, int mutationProbability = 70,
                    CrossoverMethod crossoverMethod = CrossoverMethod::Middle,
                    std::ostringstream* os = nullptr)
```

```
        {
            UNUSED(sizeOfPopulation);
            UNUSED(mutationProbability);
            UNUSED(crossoverMethod);
            UNUSED(os);

            // Your code ...
        }

        // Continue the search
        bool next(int mutationProbability, CrossoverMethod crossoverMethod,
                  std::ostringstream* os)
        {
            UNUSED(mutationProbability);
            UNUSED(crossoverMethod);
            UNUSED(os);

            // Your code ...

            return true;
        }
    };

} // end namespace

#endif
```

```
#include "functions.h"

namespace AI
{

} // end namespace
```

Compiling, executing, and testing

Run `make` with the default rule to bring program executable `main.out` up to date:

```
$ make
```

Or, directly test your implementation by running `make` with target `test`:

```
$ make test
```

If the `diff` command in the `test` rule is not silent, then one or more of your function definitions is incorrect and will require further work.

To make a memory leak test, run `make` with target `leak`:

```
$ make leak
```

Make sure that the output does not show any memory related issue.

File-level documentation

Every **edited by student** source and header file *must* begin with a *file-level* documentation block. This documentation serves the purpose of providing a reader the raison d'être of this source file at some later point of time (could be days or weeks or months or even years later). This module will use Doxygen to tag source and header files for generating html-based documentation. An introduction to Doxygen and a configuration file is provided on the module web page. Here is a sample for a C++ source file:

```
/*!*****************************************************************************
\file    functions.cpp
\author  Vadim Surov, <Your Name>
\par     DP email: vsurov\@digipen.edu, <Your Email>
\par     Course: CS380
\par     Section: A
\par     Programming Assignment 12
\date    04-30-2021

\brief
  This file has declarations and definitions that are required for submission
*******************************************************************************/
```

Function-level documentation

Every function that you declare and define and submit for assessment must contain *function-level documentation*. This documentation should consist of a description of the function, the inputs, and return value. In team-based projects, this information is crucial for every team member to quickly grasp the details necessary to efficiently use, maintain, and debug the function. Certain details that programmers find useful include: what does the function take as input, what is the output, a sample output for some example input data, how the function implements its task, and importantly any special considerations that the author has taken into account in implementing the function. Although beginner programmers might feel that these details are unnecessary and are an overkill for assignments, they have been shown to save considerable time and effort in both academic and professional settings. Humans are prone to quickly forget details and good function-level documentation provides continuity for developers by acting as a repository for information related to the function. Otherwise, the developer will have to unnecessarily invest time in recalling and remembering undocumented gotcha details and assumptions each time the function is debugged or extended to incorporate additional features. Here is a sample for function `substitute_char`:

```
/*!*****************************************************************************
\brief
   Replaces each instance of a given character in a string with
   other given characters.

\param string
  The string to walk through and replace characters in.

\param old_char
  The original character that will be replaced in the string.

\param new_char
  The character used to replace the old characters
```

```
\return
  The number of characters changed in the string.
*****************************************************************************/
```

Submission and automatic evaluation

1. In the course web page, click on the appropriate submission page to submit `functions.cpp` and `functions.h`.
2. Please read the following rubrics to maximize your grade. Your submission will receive:
    - `F` grade if your `functions.cpp` doesn't compile with the full suite of `g++` options.
    - `F` grade if your `functions.cpp` doesn't link to create an executable.
    - Your implementation's output doesn't match correct output of the grader (you can see the inputs and outputs of the auto grader's tests). The auto grader will provide a proportional grade based on how many incorrect results were generated by your submission. `A+` grade if output of function matches correct output of auto grader.
    - A deduction of one letter grade for each missing documentation block in `functions.cpp` and `functions.h`. Your submission `functions.*` (if it was edited by you) must have **one** file-level documentation block and **one** function-level documentation blocks. A teaching assistant will physically read submitted source files to ensure that these documentation blocks are authored correctly. Each missing or incomplete or copy-pasted (with irrelevant information from some previous assessment) block will result in a deduction of a letter grade. For example, if the automatic grader gave your submission an `A+` grade and one documentation block is missing, your grade will be later reduced from `A+` to `B+`. Another example: if the automatic grade gave your submission a `C` grade and the two documentation blocks are missing, your grade will be later reduced from `C` to `E`. Likewise, your submission `functions.h` must have **one** file-level documentation block and **one** function-level documentation block.