Data Structures and Algorithms

Programming Assignment: Genetic Algorithms

Open a command line, and navigate to the location where you have unzipped the assignment files. Navigte to the folder:

AssignmentCode/bin/
Run the following commands:
cmake ../GeneticAlgorithms/student/
make

1. The assignment goal is to identify the coefficients of a polynomial function f(x), given the x values and corresponding f(x) values. A polynomial function having “n” coefficients generally takes the following format:

$$f(x)=\sum_{i=0}^{i=n} a_i x^i \qquad \text{where } a_i, i=0-n, \text{ are the coefficients of } f(x).$$

2. The expected input and output files are explained below:
    a. 1_function_values: This is the only input file that will be supplied to your code. The code in line 25 of homework.cpp shows how this file is used.
    b. There are two output files expected:
        i. 1_result: The predicted coefficients are printed in this file. Check code on lines 41 and 64 of homework.cpp to see how this file is generated.
        ii. 1_memory_runtime: The amount of memory used and run-time is printed in this file. This file is generated by homework.cpp.
    c. 1_actual_coeff: This file contains the coefficients that were used to generate the values in 1_function_values. After using genetic algorithms to predict the coefficients, you may use this file to check if the coefficients that you predicted are close enough to the true value. However, this file must not be input to your code.

3. The assignment requires completing the code in the function: `std::string GAPolynomialSolver::solvePolynomial` of the file GAPolynomialSolver.cpp. You are welcome to implement other functions or classes in GAPolynomialSolver.h and GAPolynomialSolver.cpp. Some code has already been implemented in these files given to you. This code is intended to give hints of how to create a good solution, and generate the right output. It is strongly recommended that you do not make any changes to the functions for which code has already been written in GAPolynomialSolver.h and GAPolynomialSolver.cpp. However, if you feel that your solution will require changes to these functions, you may do so. Note the following:
    a. While testing, we will be expecting the result files to be in the exact same format as the result files given to you. If there is any change in the output format, your assignment will not be graded.

4. You cannot use std::template packages or external APIs. You cannot change the #include statements given in any of the files. You can reuse code, or classes that ONLY YOU have written.

5. Ensure that all your code or classes are in the two files GAPolynomialSolver.h and GAPolynomialSolver.cpp and you are not creating any other .h or .cpp files for submission.

6. How will we identify correctness of solution:
    a. We have a separate test set with the same file formats as you have been given.
    b. We will generate SSE (sse_student) between your predicted coefficients and the true coefficients.
    c. We have a set of coefficients generated by our implementation of Genetic algorithms. We will generate SSE (sse_ta) between our predicted coefficients and the true coefficients.
    d. The ratio sse_ta / sse_student will be computed for each test case. (sse_test)
    e. The TAs have been given multiple test cases. sse_test will be computed for each test case. An average value of the sse_test will be taken to get avg_sse.

f. Your correctness score will be avg_sse * maximum points allotted for correctness.

Submissions:

1. Submit only the files GAPolynomialSolver.h and GAPolynomialSolver.cpp on gradescope and canvas.
2. Any number of submissions are allowed. Your last submission will be used for grading.

Grading method:

1. If your code runs and generates an output file in the correct format for each input file, you get submission points.
2. If your method generates correct results for each test case, you get points for correctness.
3. We will measure the memory consumption in Bytes (submitted_memory). We will take the memory used by our implementation (our_memory). Memory score will be based on the ratio: (our_memory / submitted_memory) * max score for memory. Sometimes, a very good implementation can show a memory consumption of zero. In this case, you will get full points for memory consumption.
4. We will measure the run time in milliseconds (submitted_time). We will take the time used by our implementation (our_time). Your run-time score will be based on the ratio: (our_time / submitted_time) * max score for run-time.
5. The maximum scores for sbumission, correctness, memory and runtime is based on a percentage of total points. The percentage breakup is below:

Grading Rubrick

Task No	Output: Does the code run? Are the output files generated with the filenames and formats as specified in the problem?		Correctness: Is the right output generated?		Timing			Memory			Total points obtained	Total points for task
	Points obtained	Total points	Points obtained	Total points	Time taken to run (ms)	Comparison with other students (divide by min of others)	Timing score	Memory consumed (Bytes)	Comparison with other students (divide by min of others)	Memory score		
1		10.00%		50.00%			20.00%			20.00%		100.00%