McMaster University
MECHTRON 2MD3: Data Structures and Algorithms for Mechatronics

Midterm Test

Instructions
Plots and short answer should be submitted in a single pdf document with the file name "1234-genetic-programming.pdf" where 1234 is your student ID. Submit C++ source files as per the question instructions. All code should be formatted and commented. Please ensure your source files compile and run properly before submitting. Your code will be checked for memory leaks using valgrind (https://valgrind.org/).

Download (or pull) the code directory from:
https://gitlab.cas.mcmaster.ca/kellys32/2md3_2025/-/tree/main/midterm?ref_type=heads

This directory contains a near-complete implementation of a Genetic Programming experiment to solve the Cart Centering (isotropic rocket) problem. This application of trees was discussed in detail in lecture 2025-12-2MD3-TreeGP.pdf.
The genetic_programming_01.cpp program will evolve trees for 100 generations and print statistics on the best tree at each generation. Your task involves the following parts;

Part 1 [21 marks total]. Implement the following functions:

`void LinkedBinaryTree::printExpression(Node* v)`
This function must recursively print the expression tree with parentheses. For example, a tree representing postfix expression "1 2 + 3 *" is printed as "((1+2)*3)" **[3 marks]**

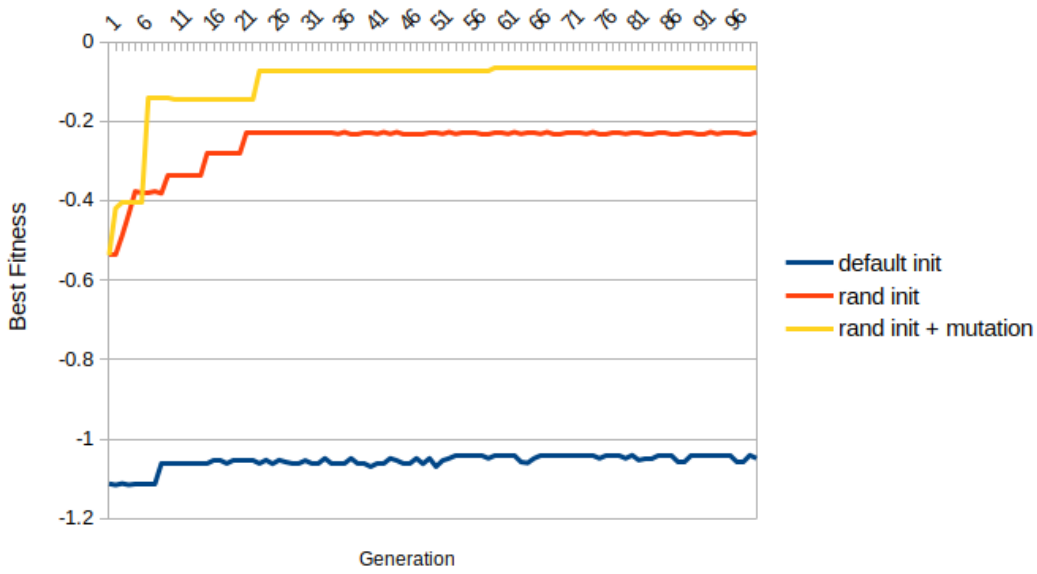
`LinkedBinaryTree createRandExpressionTree(int max_depth, mt19937& rng)`
This function should create and return a randomly generated expression tree with a depth no greater than max_depth. (The function currently simply returns the expression tree "*a* + *b*" where *a* and *b* are the X position and velocity of the cart, respectively.) **[6 marks]**

`void LinkedBinaryTree::deleteSubtreeMutator(mt19937& rng)`
This function should delete a randomly selected subtree from the tree. **[6 marks]**

`void LinkedBinaryTree::addSubtreeMutator(mt19937& rng)`
This function should add a randomly created subtree to the tree. **[6 marks]**

Place holders for these functions already exist. Your task is to fill in their code. You may reuse any code existing in the class. You should see a significant improvement in the best fitness after implementing `createRandExpressionTree`, and a further improvement after implementing the mutation operators. When plotted, the data should look something like the following plot. The blue line is the original code without modification. The red line shows results when the population is initialized with random trees. The yellow line shows

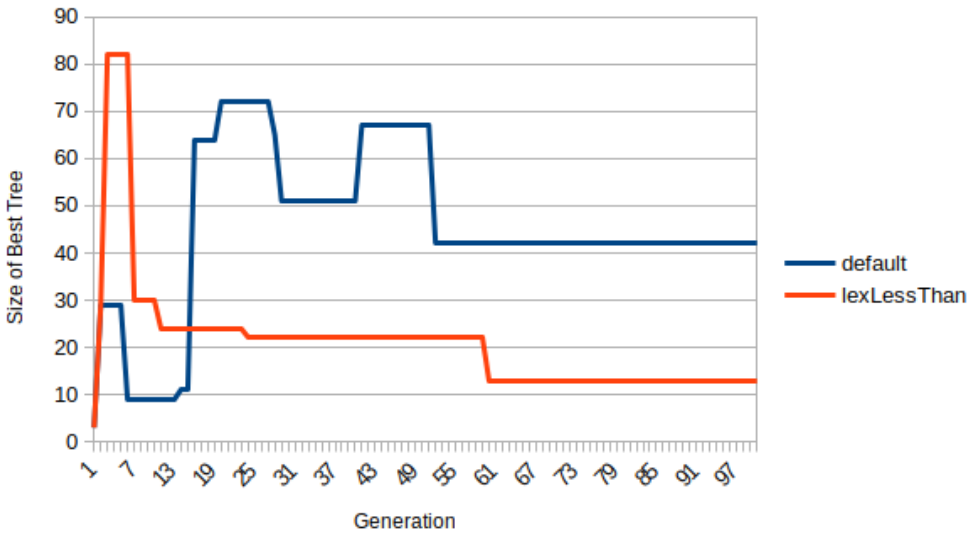
results when the population is initialized with random trees and both mutation operators are implemented.



Plot your results in the above format after implementing your functions. I used Excel but any plotting tool is fine. Your submission for part 1 must include your plot and the printout of the best individual evolved using mutation. Here's mine:

Best tree:
(((a > a) + (((b > b) * abs(((abs((b / abs(a))) * (((a / b) / a) / b)) > ((abs((a * (b / b))) - (abs(a) / a)) + a)))) * a)) - (b + a))
Generation: 99
Size: 42
Depth: 11
Fitness: -0.0889936

Part 2 [5 marks]. Implement a comparator ADT called LexLessThan which performs a lexicographic comparison of two trees T_A and T_B as follows: If the scores of T_A and T_B differ by less than 0.01, then T_A is "less than" T_B if T_A has more nodes than T_B. Otherwise, compare the trees by their score. The goal of using this comparator is to favour simpler trees only when their scores are similar. Repeat the experiment from part 1 using the new comparator. To do so, you must uncomment the line under "sort using comparator class". After doing so, does evolution produce simpler trees? A **complexity** line plot comparing the size of the best tree during evolution with and without the LexLessThan comparator might look like the following:



Part 3 [6 marks]. Try adding a crossover operator in which you sample two parents and create two children by swapping random subtrees from each parent. See lecture 2025-12-2MD3-TreeGP.pdf slides for an example. Note there is no placeholder code for crossover. You must determine where and how to add this feature. When results are ready, add a line to your **fitness** plot labelled "rand init + crossover + mutation" to show the results of this experiment.

Part 4 [6 marks]. The Cart Centering simulator has a "partially observable" mode in which the velocity state variable is not observable to the agent. In this mode, the expression trees will need some form of temporal memory to store multiple X observations over time and integrate them to predict the velocity internally. Different options were discussed in lecture on March 7. The main idea is to provide trees with an auxiliary data structure and add a custom operation to read and write from this memory. Do not modify cartCentering.h and do not simply change the input state to include X(t) and X(t-1). This is known as "autoregressive state" and has several disadvantages. In partially observable mode, the input to your expression trees MUST be X(t) and 0.0 (as in the starter code). The standard way to approach this problem is to create two new operations for your function set (in addition to the default + - * / abs >). You should design "read" and "write" operations which allow your trees to read and write from/to a data structure of your choosing. This will allow your trees to evolve their own approximation of velocity. With your memory mechanism implemented, repeat your last experiment in partially observable mode by setting `PARTIALLY_OBSERVABLE = true.` Note that evolving a tree with memory might require more generations or a larger population size. Show the results of this experiment by adding a line to your plot labelled "partially observable". Please comment your code to briefly describe your approach to adding memory.

Your submission for must include:

1. For parts 1, 3, and 4, a **fitness** plot like the one above (in part 1) showing the effect of your `createRandExpressionTree` function, mutation operators, crossover operator, and partially observable experiment. Also include a printout of the best tree from any experiment its statistics (Generation, Size, Depth, Fitness).

2. For part 2, a printout of the best tree evolved using LexLessThan comparator ADT and a **complexity** line plot comparing the size of the best tree during evolution with and without the LexLessThan comparator (See part 2 example plot).

3. Your modified code named “1234-gp.cpp” where 1234 is your student ID.

Animation: If you uncomment the lines below “Evaluate best tree with animation“, the program will display a simple animation of the best tree interacting with the Cart Centering simulator. This can be very useful for debugging. However, note that animating the best tree will reset its score. For best results, do not animate the tree prior to generating its printout.

The End.