CS 101 Spring 2019 Projects 2 and 3

Due Mar. 4 (Project 2) and March 13 (Project 3)

Suppose that we have a vehicle competing in a competition that receives a score both on performance and fuel efficiency. In order to maximize the total score, a simulation is being developed to determine the best strategy and vehicle parameters. Your job is to build a class around a data structure that will support efficient simulation.

The class will contain both a doubly linked list and an array. The list should contain elements that have the following two data elements. The first is a float representing the simulated time that the vehicle would need for that simulation run. The time will be a floating point value between 0 and 500. The second is a float representing the fuel usage for the same run. The fuel value will also be a float between 0 and 500. The doubly linked list will be kept in sorted order for both performance (time) and fuel efficiency, using the following observation:
Run A dominates run B if the time for A is less than B and the fuel usage of A is less than or equal to B. If run A dominates run B then run B can and should be removed from the list.

Another component of your class should be an array where the elements of the array point to entries in the list. This array exists to give O(1) access to elements of the list, if we know the index of the specific element that we are accessing.

We will call the class dList, for doubly ordered list. Here are the methods that the dList class should support:

dList(float[], float[],int) Constructor where the first parameter is an array of times, the second is an array of fuel usage, and the third is the number of elements in the arrays. This should create a dList from the two arrays. The initial arrays are unsorted.

void out(char) Output method. Traverses the dList in order. The parameter indicates the direction. The default is 'a' indicating increasing time. If the parameter is 'd', then the traversal should be slow to fast, or decreasing time.

void out(int i, char) The same as above, but only outputs the first i runs that appear in the order indicated.

int insert(float, float) Inserts the item with time and fuel usage as parameters into the linked list. For project 3 this will return the index of the item in the array, but for this part, you can keep a counter and return that value. The counter should start at 0, and be incremented for every insert operation and be set to the size-1 in the constructor.

For the first part of this project (project 2), only the operations above need to be implemented. This means that the doubly linked list is the only data structure that is needed. The array will be used in the second part (project 3). The sample project2 main file tests only the above methods of the class.

Notes:
The constructor should run in O(N lg N) time, this means that you need to implement a O(N lg N) sorting method for the constructor and should remove the dominated nodes in O(N) time.

The out methods should run in time O(# runs output).
Insert should run in O(Size of the list).

Project 3

For project 3, we add the following methods to the class:

void increase_time(int i, float t) Increases the time of the item at index i by adding t to its time. This may cause this item to be dominated.

void decrease_time(int I, float t)) Decreases the time of the item at index i by subtracting t from its time. This may cause it to dominate some entries in the list.

void decrease_fuel(int i, float f) Decreases the fuel usage of the item at index i by subtracting f from its fuel usage. This may cause it to dominate some entries in the list.

int index_before(int i) returns the index of the run that is immediately before the run at index i in the list. In other words, the index of the run that is a faster time, but the closest to run i in the list.

int index_after(int i) returns the index of the run that is immediately after the run at index i in the list. In other words, the index of run that uses less fuel, but the closest to run i in the list.

For these methods, the array of pointers to the list nodes needs to be added to the class. In the constructor, the position (index) of a run in the two arrays determines its index in the array of pointers to the linked list.

Because the dList class is performing dynamic allocation, the class implementation for project 3 should include a destructor, the copy constructor and the assignment operator.

Notes:
The constructor should run in O(N lg N) time. The out methods should run in time O(# runs output). Insert should run in O(Size of the list). Increase time should run in O(1) time, while decrease time and fuel should run in O(Number of items dominated). Index_before and index_after should run in O(1) time.

The array entries should not be removed when an item is dominated. This way, the array index at the time the item is created gives you O(1) access to the item in the list.

You may use dummy header and tail nodes in your linked list if you want.

Your code should be in a file named dList.cpp. You should not use any include statements, but you can expect that iostream has been included in the main.

An example P2main.cpp and P3main.cpp will be provided containing proper output. Grading will be accomplished by supplying other files named P2main.cpp and P3main.cpp.

Please submit a makefile with your class. The main program will be named P2main.cpp/P3main.cpp, and the executable should be named p2/p3.