COP 5536 Spring 2024

Programming Project

Due – Apr 3rd 2024, 11:59 pm EST

GatorGlide Delivery Co.

Problem Description:

In the heart of Florida, GatorGlide Delivery Co. emerged as a tech-savvy beacon of efficient logistics inspired by the swift movements of the alligator. It is planning to elevate its software infrastructure to meet the growing demands of customers. The new system aims to optimize order management, delivery routes, and enhance overall efficiency. They use AVL trees to intricately store the order priorities and ETAs to manage the orders efficiently.

Order management system:

- Order details are given whenever an order is created with the following fields: orderId, currentSystemTime, orderValue, deliveryTime
- We can perform operations like createOrder, cancelOrder, and searchOrder
- currentSystemTimes are relative to a fixed value. So, they can only be some integer values, representing time since an epoch (like 1, 2, 3, etc.).
- deliveryTime is the time required to process and deliver the order (from source to destination).

Orders are served according to their priorities. Priority of an order is calculated using the following equation:

priority = valueWeight * normalizedOrderValue - timeWeight * orderCreationTime

where valueWeight = 0.3, timeWeight = 0.7, and normalizedOrderValue = order.orderValue / 50
Note: Since the orders are in the order of 50s, like 100, 250, 300, etc.*

Why do we need normalization?
Normalization, achieved by dividing order values by 50, prevents larger values from dominating the priority calculation. Without it, the system might prioritize higher-order values disproportionately.

There is only one delivery agent, and one starting point for this entire operation. One order is delivered at a time. The delivery agent starts as soon as there is an order in the system.

Every time an order is created:

- The Estimated Time of Arrival (ETA) is printed. It is calculated by using the ETA of the order that needs to be delivered before it, and also the delivery time of the current order. To find the order that needs to be delivered right before it, an AVL tree with order priorities as keys is needed (there can be duplicates in this tree). And also, the ETAs of all the orders with a priority lower than the current order will be updated.

- All the orders that have ETA less than or equal to the order_created_at time are delivered and removed from the system and ETAs are printed. To perform this efficiently, a second AVL tree is used with ETAs as keys. (There won't be any duplicates). Whenever an order is canceled, the order is removed from the system, and the ETAs of all the orders that have a lower priority than it will be updated.

**Note that a delivery might be out for delivery when this order has been created, i.e., the ETA > order_creation_time but is in the process of delivery. In that case, ensure that even though the current orders' priority is greater, it must not move in front of that.*

The system should support the following operations:

1. print(orderId): Prints the order details of the given order_id.
Output format: [orderId, currentSystemTime, orderValue, deliveryTime, ETA]

2. print(time1, time2): Prints all the orders that will be delivered within the given times (including both times) and are undelivered.
Output format: if orders exist [orderId1, orderId2, ……..], There are no orders in that time period if none

3. getRankOfOrder(orderId): Takes the order_id and returns how many orders will be delivered before it.
Output format: Order {orderId} will be delivered after {numberOfOrders} orders

4. createOrder(order_id, current_system_time, orderValue, deliveryTime): Creates the order, prints the ETA, and also prints which previously unfulfilled orders have been delivered along with their delivery times.
Output format: Order {orderId} has been created - ETA: {ETA}
Updated ETAs: [orderId1: ETA, orderId2: ETA,.....] if any ETAs have been updated
Order {orderId} has been delivered at time {ETA} for all such orders if they exist, each in a new line

**Note that the deliveryTime is the time that is needed for the order to be delivered by the delivery agent, the time taken to get back from the delivery destination back to the source also needs to be accounted, which is the same as the deliveryTime.*

5. cancelOrder(order_id, current_system_time): Cancels the order and updates the ETAs of all the orders with lower priority.
Output format: Order {orderId} has been canceled
Updated ETAs: [orderId1: ETA, orderId2: ETA, ……..] if any ETAs have been updated
[or]
Cannot cancel. Order {orderId} has already been delivered if the order is out for delivery or is already delivered

6. updateTime(order_id, current_system_time, newDeliveryTime): Takes the current_system_time, order_id, and the new delivery time. It updates the ETAs of all the orders with lower priority.
Output format: Updated ETAs: [orderId1: ETA, orderId2: ETA, ……..] if any ETAs have been updated
[or]
Cannot update. Order {orderId} has already been delivered if the order is out for delivery or is already delivered

*[*Note: Everytime, while printing the updated ETAs, only the orders whose ETA has changed are printed]*

Programming Environment
You may use either Java, C++, or Python for this project. Your program will be tested using the Java or g++ compiler or python interpreter on the thunder.cise.ufl.edu server. So, you should verify that it compiles and runs as expected on this server, which may be accessed via the Internet. Your submission must include a makefile that creates an executable file named **gatorDelivery**.

Your program should execute using the following
For C/C++:
$ ./ gatorDelivery file_name
For Java:
$ java gatorDelivery file_name
Where file_name is the name of the file that has the input test data.

Input and Output Requirements:

- **Read input from a text file where input_filename is specified as a command-line argument.**
- All Output should be written to a text file having filename as concatenation of **input_filename** + "_" + "**output_file.txt**".

(eg. inputFilename = '*test1.txt'* , outputFilename = '*test1_output_file.txt')*

- The program should print the delivery details of all remaining orders and terminate when the operation encountered in the input file is ***Quit()***.

Example 1:
Input

```
createOrder(1001, 1, 200, 3)
createOrder(1002, 3, 250, 6)
createOrder(1003, 8, 100, 3)
createOrder(1004, 13, 100, 5)
print(2, 15)
updateTime(1003, 15, 1)
createOrder(1005, 30, 300, 3)
Quit()
```

Output:

```
Order 1001 has been created - ETA: 4
Order 1002 has been created - ETA: 13
Order 1003 has been created - ETA: 22
Order 1001 has been delivered at time 4
Order 1004 has been created - ETA: 30
[1002]
Updated ETAs: [1003: 20, 1004: 26]
Order 1005 has been created - ETA: 34
Order 1002 has been delivered at time 13
Order 1003 has been delivered at time 20
Order 1004 has been delivered at time 26
Order 1005 has been delivered at time 34
```

Example 2:

Input

```
createOrder(101, 2, 300, 4)
createOrder(102, 3, 600, 3)
print(101)
createOrder(103, 7, 200, 2)
createOrder(104, 8, 500, 3)
cancelOrder(102, 9)
createOrder(105, 10, 300, 4)
getRankOfOrder(105)
Quit()
```

Output:

```
Order 101 has been created - ETA: 6
Order 102 has been created - ETA: 13
[101, 2, 300, 4, 6]
Order 103 has been created - ETA: 18
Order 101 has been delivered at time 6
Order 104 has been created - ETA: 19
Updated ETAs: [103: 24]
Order 102 has been canceled
Updated ETAs: [104: 13, 103: 18]
Order 105 has been created - ETA: 24
Order 105 will be delivered after 2 orders
Order 104 has been delivered at time 13
Order 103 has been delivered at time 18
Order 105 has been delivered at time 24
```

Submission Requirements:

- Include a makefile for easy compilation.
- Provide well-commented source code.
- Submit a PDF report that includes project details, function prototypes, and explanations.
- Follow the input/output and submission requirements as described in the reference project.

Do not use nested directories. All your files must be in the first directory that appears after unzipping. You must submit the following:

- Makefile: You must design your makefile such that ‘make’ command compiles the source code and produces an executable file. (For java class files that can be run with java command)

- Source Program: Provide comments.
- REPORT:
- The report should be in PDF format.
- The report should contain your basic info: Name, UFID and UF Email
- Present function prototypes showing the structure of your programs. Include the structure of your program.

To submit, please compress all your files together using a zip utility and submit it to the Canvas system.
Please look for the "Assignment Project" in Canvas for submission.
Your submission should be named LastName_FirstName.zip.
Please make sure the name you provided is the same as the name that appears on the Canvas system.
Please do not submit directly to a TA. *All email submissions will be ignored without further notification. Please note that the due date is a hard deadline. No late submission will be allowed. Any submission after the deadline will not be accepted.*

Grading Policy:
Grading will be based on the correctness and efficiency of algorithms. Below are some details of the grading policy.

- Correct implementation and execution: 70%
- Comments and readability: 15%
- Report: 15%

Important: Your program will be graded based on the produced output. You must make sure to produce the correct output to get points. There will be a threshold for the running time of your program. If your program runs slow, we will assume that you have not implemented the required data structures properly.
You will get ***negative points*** *if you do not follow the* ***input/output or submission requirements*** *above.* ***Following is the clear guidance on how your marks will be deducted.***

- Source files are not in a single directory after unzipping: -5 points
- Incorrect output file name: -5 points
- Error in make file: -5 points
- Make file does not produce an executable file that can be run with one of the following commands: -5 points

./ gatorDelivery file_name
java gatorDelivery file_name

- Hard coded input file name instead of taking as an argument from the command prompt: -5 points
- Not following the Output formatting specified in examples: -5 points
- Any other input/output or submission requirement not followed that is mentioned in the document: -3 points

Also, we may ask you to fix the above problems and demonstrate your projects.

Miscellaneous:

Implement AVL tree from scratch, **without using built-in libraries**.

Your implementation should be your own. You have to work by yourself for this assignment (discussion is allowed). **Your submission will be checked for plagiarism!!**