Part 1: Generating order book data from raw updates

You will be given a set of csv files. Each csv represents order updates for single date. Each row contains a single order update in the market.

Your task is to write a program which reads an input csv, builds/updates the aggregate order book for each order update, and outputs a new csv. The output csv should contain one row per input row and, at a minimum, have the timestamp of each update, the price and total quantity of the best 5 levels with non-zero quantity on each side of the book, and the price and side of the update. See the next section for a description of aggregate order book updating.

For empty levels, quantity should be set to 0, and any price may be output.

For this part, c++ is preferred, but you may use python if you don't know c++.

Each input csv contains the following fields:

- timestamp - the timestamp in microseconds since open of that day
- side - the side of the order - either b for bid/buy or a for ask/sell
- action - the action taken by the update. Possible options are a (for add), d (for delete), m (for modify)
- id - the id of the order, unique per (day, side) pair
- price - the price of the order
- quantity - the quantity of the order

To make it easier for us to communicate with you about your code, please name your output csv fields in the following way. You are free to add any fields you think might be useful on top of these (see part 2).

- timestamp - the timestamp of the original update
- price - the price of the input row
- side - the side of the input row
- bp0 - the price of the top / highest level willing to buy
- bq0 - the total quantity of the top/highest level willing to buy
- bp1 - the price of the second highest level willing to buy
- bq1 - the total quantity of the second highest level willing to buy
- .. up to bp4, bq4
- ap0 - the price of the lowest level willing to sell
- aq0 - the total quantity of the lowest level willing to sell
- ... up to ap4, aq4

For any book level that is not available, the price will be ignored and your program should fill in the quantity as 0.

This program should be run on each input file to give one output file per day. Each day should be run separately, and assume the order book begins empty at the start of each day/input file. You will use the output from this in part 2.

Order Book Updating

The aggregate order book should, at each (price, side), keep track of the summed quantity of all open orders with that side and price. A "price level" represents the information for a single (side, price) pair. Bid/buy side price levels are sorted by price from highest price to lowest price, and ask/sell side price levels are sorted with lowest price first. The "best" N levels on each side are the first N levels on each side according to this sorting/ordering.

An order update with action of "add" means that a new order was entered, "delete" means an order was removed, and "modify" means that an order previously in the market had its quantity or price updated, and must be moved.

Example update scenario:

- The book is initially empty.
- An order add update comes in (action="a", id=1, side="b", price=15, quantity=3). The resulting book has a single buy price level at price=15 with qty=3. In this case, we would expect the output row to have bp0=15 bq0=3, and all other book quantity fields (bq1, aq1, etc) to be = 0.
- An order add update comes in (action="a", id=2, side="b", price=15, quantity=5). The resulting book has a single buy price level at price=15 with qty=8. The resulting book fields in your output should be updated to have bq0=8
- An order modify update comes in (action="m", id=1, side="b", price=20, quantity=3). The resulting book now has 2 buy price levels, one at price=20 with quantity 3 and one at 15, with quantity=5. The resulting book fields in your output should be bp0=20, bq0=3, bp1=15, bq1=5, with all other book quantity fields (bq2, aq2, etc) being 0
- An order add update comes in with (action="a", id=3, side="a", price=30, quantity=1). The resulting book has 2 buy price levels as before and a single ask level. The output row should have the same book fields as the last row except with ap0=30 and aq0=1
- An order delete update comes in (action="d", id=2, side="b", price=15, quantity=5). The resulting book now has a single buy and single sell price level. The output should have bp0=20, bq0=3, ap0=30, aq0=1

Part 2: Modelling and signal generation

In python, using the data you output in part 1, you will be creating a predictive model. You should write a program which, at a minimum, should:

- Load the data your program output from part 1
- Calculate a collection of predictive features. A very simple example might be "bq0 - aq0" as some metric of how many contracts there are to buy vs sell at the top prices.
- Choose and generate data for a prediction target in the future that you think would be useful for trading the product. Document what options you considered and why you chose what you did.
- Optionally subsample the data - document why you decide to do/not do this
- Use Lasso to predict your chosen target and combine the features into a final predictive signal

Please document the choices you make and why you make them.

We'd like you to demonstrate how you would write reusable library code if you were building your own research library. We'll be evaluating how you organize things, structure your code for reusability, and follow general good coding practices.

Please submit your code as a single archive (.zip or .tar.gz) file, with documentation on how to run