Limit Order Book Notes

2025-08-11 00:54

Status: Ongoing
Tags: quant, python, cs

Main Content

Preface

The purpose of this is to document the purpose, process and method of creating a **limit order book** (in Python). This is not an end-to-end implementation to be used live, but more so a proof of concept to practice SWE concepts and system design principles.

OrderBook Overview

An OrderBook is the heart of a financial exchange, functioning as an electronic ledger of all outstanding buy and sell orders for a specific security. Its primary purpose is to match buyers with sellers, thus facilitating trades and discovering the market price.

A **Limit Order Book (LOB)** is a specific type of order book that only accepts **limit orders**. A limit order is an instruction to buy or sell a set quantity of a security at a specified price **or better**. For example, a buy limit order at $50.10 will only execute at a price of $50.10 or lower. A sell limit order at $50.15 will only execute at a price of $50.15 or higher. The book is a list of all such unfulfilled orders, waiting for a matching counterparty.

Price-Time Priority

Price-Time Priority is the method that ensures fairness in an exchange.

- **Price Priority**
  - The book will always give priority to the best price for any given transaction.
  - For **buy** orders, the **highest price** will be prioritised, and for **sell** orders, the **lowest price** will be prioritised.
  - For example, a buy order for $101 will be matched before a buy order for $100, as it maximises value for the **seller**
  - In the other scenario, a sell order for $100 will be matched before a sell order for $101, as it maximises value for the **buyer**
- **Time Priority**
  - In the scenario that we have multiple orders, all at the same price, the one that was placed first will receive priority
  - This means the OrderBook will follow a FIFO (First In First Out) approach

Efficiency

A key challenge in OrderBook design is to **efficiently** store and retrieve bids and asks for filling orders. The storage of these will be from highest to lowest **(bid)** and lowest to highest **(ask)** as described previously. Additions, deletions and lookups for the best price also need to be as efficient as possible. For this, we are going to use 3 main data structures.

- **Main Dictionary**
  - This will be used to store our dictionaries, `self.bids`, and `self.asks`.
  - The structure of this will be `{ price (float): collections.deque([Order, Order, ...]) }`.
  - This is the main data store, which will map a price level to a queue of all of the orders at that price. A `deque` (double-ended queue) will handle FIFO accurately through appending right whenever we have a new order, and popping left whenever we fill one.
- **Sorted Price Lists**
  - We will have 2 price lists, `self.bid_prices`, and `self.ask_prices`.
  - These will be structured as `[price_1 (float), price_2 (float)... ]`.
  - To sort these efficiently, we can use `bisect.insort(a, x)`, where `a` is our list and `x` is the element we are inserting into the

list. This function will insert the item into the list while preserving the order of the list.

- **Order Map**
  - This will be our method of cancelling and accessing order info efficiently.
  - It will be structured through `{ order_id (int): Order, ... }`
  - As these order objects are **persistent** through the program (including the order itself in the queue for its price), we can add a **cancelled flag** to the order so that when it hits the front of the queue, it will be **discarded**.

Methods

For this OrderBook implementation, we will have several methods for interacting and requesting information from the book itself externally. An overview of the methods is as follows:

- `add_order(side, price, quantity)`
  - Arguments:
    - `side` - Whether the order is a request to **buy** or **sell**.
    - `price` - The requested **price** to fill the order.
    - `quantity` - The requested **quantity** for the order.
  - This method is used to add an order to the book, which will then be added as an object to the data structures outlined above.
  - An `order_id` will be generated to correspond to this order as well to access its information, or **cancel** it.
- `cancel_order(order_id)`
  - Arguments:
    - `order_id` - The generated ID that will be provided to the user **upon creation** of their order.
  - By enacting this method, the order object corresponding to the given ID will have an `is_cancelled` **flag** changed to `True` within the object. This means that when the order hits the front of the queue, it is immediately discarded (this allows for **O(1)** time).
- `get_best_bid()`
  - Returns the best **bid** price currently available.
- `get_best_ask()`
  - Returns the best **ask** price currently available.

Order Types

An exchange can have a myriad of order types attached to it. Some of which are outlined below:

- **Limit**
  - An order filled at the specified price **or better**. A **buy** order executes at the limit price or **lower**. A **sell** order executes at the limit price or **higher**.
  - This order is typically considered **standard** and is what most simple order books base their order matching on.
- **Market**
  - An order to buy or sell **immediately** at the **best** currently available **price**. It prioritises speed of execution over a specific price.
- **Fill-Or-Kill (FOK)**
  - Requires the entire order quantity to be filled immediately. If the order cannot be filled at once, it is **cancelled** entirely (no partial fills).
- **Immediate-Or-Cancel (IOC)**
  - Executes as much of an order as possible immediately. Any portion of the order that **cannot** be filled instantly is **cancelled**.
- **Stop**
  - An order to buy or sell once a security reaches a specified "stop price." When triggered, it becomes a **market order**. It's often used to limit losses on a position.
- **Stop-Limit**
  - A two-part order that becomes a **limit order** once the specified "stop price" is reached. This provides more price control than a stop order but does not guarantee execution.

References

Coding Jesus' Limit Order Book in C++ series