Тестовое задание

16 марта 2021 г.

Тестовое задание состоит в реализации относительно простого алгоритма торговли. Можно выбрать один из двух уровней сложности (попроще и посложнее). Очень рекомендуется хотя бы попробовать сделать сложный вариант, так как полученные знания пригодятся в дальнейшем.

1 Описание алгоритма

1. Робот выставляет ордер #1 на покупку по цене **buy price** = **current price - gap** / **2**.
2. (a) Если цена уменьшается до **buy price**, то ордер #1, скорее всего, будет исполнен. В этом случае перейти к пункту 3.
   (b) Если цена увеличивается до такого значения, что становится истинным условие **current price** > **buy price** + **gap** + **gap ignore**, то робот должен отменить ордер #1. Далее, вернуться к пункту 1.
3. Робот выставляет ордер #2 на продажу по цене **sell price** = **current price** + **gap**.
4. (a) Если цена увеличивается до **sell price**, то ордер #2, скорее всего, будет исполнен. В этом случае вернуться к пункту 1.
   (b) Если цена уменьшается до такого значения, что становится истинным условие **current price** < **sell price - gap - gap ignore**, то робот должен отменить ордер #2. После этого следует вернуться к пункту 3.

2 Требования к реализации (от наиболее важных до наименее)

1. **Язык программирования**

   Разработка должна обязательно вестись на Python 3.

2. **Конфигурация робота**

   Робот должен иметь возможность конфигурироваться с помощью файла. Тип файла может быть произвольным, например, **.yaml**, **.json**, **.txt**, **.xml**. Пример конфига в **.yaml**-формате (про **client id** и **client secret** - ниже):

   ```
   robot:
       gap: 100.0
       gap_ignore: 50.0
   exchange:
       client_id: ...
       client_secret: ...
   ```

   Можно добавить какие-то еще поля и изменить структуру файла. Единственным обязательным условием является наличие параметров **gap** и **gap ignore**.

3. **Запуск робота**

   Запуск робота нужно производить через Docker и docker-compose.

4. **Poetry**

   Для создания проекта и управления зависимостями в виртуальном окружении использовать poetry.

5. **Unit-тестирование**

   Покрыть тестами какую-нибудь часть кода на свое усмотрение (две-три функции).

6. **База данных**

   Нужно создать базу данных, в которую записывать информацию об ордерах (размер, статус, покупка/продажа и т.д.). Желательно поднимать ее через Docker и docker-compose. Желательно использовать MySQL 8.0, но можно выбрать любую другую.

3 Сложный вариант реализации

В сложном варианте реализации нужно использовать реальную биржу Deribit и реализовать ее API для авторизации и размещения ордеров. Deribit - это криптовалютная деривативная биржа, которая имеет testnet.

1. **Testnet**

   На нем можно тестировать стратегии (**обратите внимание на префикс test в доменном имени в ссылке!**), не тратя настоящие деньги. Необходимо создать аккаунт. После успешной аутентификации в верхнем правом углу будет отображен текущий баланс. Если он равен нулю, то его нужно пополнить. Для этого перейдите в настройки аккаунта (см. рисунок 1), на вкладку Deposit. Получите новый BTC Deposit Address и скопируйте его (например, 2NBe6ZzLMkkqHXMVmpX...). Перейдите по ссылке над BTC адресом (Dericoins), вставьте адрес в поле Deposit Address, укажите количество биткоинов и нажмите кнопку Make deposit. Через несколько минут ваш баланс должен пополниться.

2. **Интерфейс**

   Требуется изучить интерфейс биржи (что такое красные и зеленые свечи на графике, как состоит Order Book и зачем он нужен, форму отправки ордеров, различные типы ордеров Market, Limit, Stop-Market, Stop-Limit).

   Так как Deribit - деривативная биржа, то торговать вы будете не парой инструментов BTC/USD, а контрактами BTC-PERPETUAL, да еще и с плечом (max leverage в форме отправки ордеров). В это для начала можно не вникать, просто имейте ввиду, что если вы заплейсите ордер в интерфейсе на 1 BTC, то при плече = 100х можно считать, что вы заплейсили 100 BTC (хотя в балансе отображается, что у вас всего, допустим, 1 BTC).

3. **API**

   Пробежаться и изучить документацию API Deribit. Реализовать возможность плейсинга и отмены ордера (для отмены ордера можно использовать метод для отмены BCEX ордеров). Эти методы являются приватными, поэтому потребуется реализовать также метод для аутентификации. А для аутентификации потребуется сгенерировать два ключа **Client Id** и **Client Secret**. Сделать это можно в настройках аккаунта на вкладке **Api**.

   Также потребуется получать текущую цену. В API Deribit, как и в интерфейсе, есть несколько цен (index price, last price, mark price). Можно использовать mark price, которая в интерфейсе отображается в середине Order Book.

   Обратите внимание, что на странице API Deribit есть примеры запросов на Python, которые можно использовать.

4. **Python asyncio**

   API Deribit в основном работает через websocket, хотя некоторые методы можно вызвать через http (например, метод аутентификации). Следовательно, если вы будете реализовывать полностью сложный вариант, то вам придется использовать asyncio.

4 Простой вариант реализации

В простом варианте реализации можно не реализовывать API Deribit, а сделать заглушку (mock) необходимых методов. Для симуляции выполнения ордеров можно использовать **time.sleep()** с рандомным аргументом. Также вы можете только частично реализовать API Deribit (например, только метод аутентификации через http). Тогда вам не понадобится использовать asyncio, так как не будет никаких websocket.

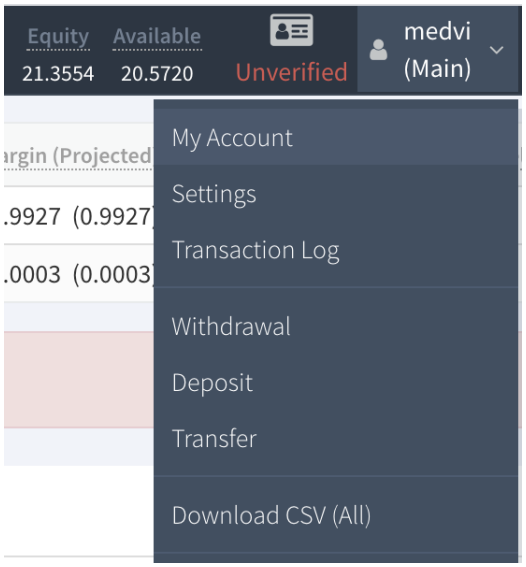

Рис. 1: Настройки аккаунта