<https://bybit-exchange.github.io/docs/v5/websocket/public/orderbook>

mais informacoes <https://www.okx.com/docs-v5/en/?shell#order-book-trading-market-data-ws-all-trades>

channel em tag WS/Order book channel

Mais info <https://bitypreco.com/api#web_socket>

Versao funcionar bests: main arb

Versao funcional Workers: main, workerBy, workerBin em dir botsgpt/workers

versão funcional: baseBy3

Voce esta me ajudando a desenvolver um programa de arbitragens de criptos. Vou te passar uma documentação e tres códigos do que já foi desenvolvido, mas não faça nada ainda, ok?

Ok, Agora vamos continuar agregando funções e/ou ajustes em etapas.

Um ultimo detalhe. Para facilitar o dialogo vc vai numerar as mensagens que recebe e as respostas que vc da, e vai mostrar essa numeração no inicio da tua resposta, assim:

Número da mensagem: 4

Numero da resposta: 4. Ok? Estamos listos?

Uma ultima consideração muito importante: Quando vc refaz, altera ou agrega algo no código, vc algumas vezes acaba tirando coisas, que depois faltam e cria problemas. Então, antes de refazer, alterar ou agregar algo ao código, verifica se vc colocou no código novo todo o que antes estava e não foi mencionado para refazer, alterar ou tirar.

**Documentação Atualizada do Código:**

**1. Descrição Geral:**

O código implementa uma aplicação para arbitragem de criptomoedas, utilizando cotações em tempo real de múltiplas exchanges. Atualmente, ele se conecta exclusivamente à (a) 11/02 – (As exchanges atuais são Bybit, OKX e Deribit) para acessar dados de WebSocket em pares de criptomoedas especificados, coletando bid e ask. Os dados são armazenados em um buffer otimizado, exibidos no console em tempo real e gravados em um arquivo de log para futura análise. A cada 15 segundos, o buffer completo é impresso no console.

**2. Considerações Fundamentais e Restrições:**

* A aplicação tem que rodar o mais rápido possível.
* O código coleta dos websockets somente depende bid e ask, por enquanto.
* A aplicação está configurada para funcionar com Worker threads e por enquanto tem apenas dois Workers, um para bybit e outro para binance. Outras exchanges irão sendo adicionadas no futuro e serão criados novos Workers para elas. (a) 11/02 – As exchanges atuais são Bybit, OKX e Deribit.
* O processo principal gerencia um buffer global e logs, enquanto os Workers são responsáveis pela coleta e envio de dados processados.
* Cada um destes workers maneja 3 websockets, um para cada par da Exchange. Ou seja que o total de Workers sera 3x total de exchanges. (a) 11/02 – São 12 workers.
* O buffer global se chama cotacoesBuffer e armazena a ultimas cotações de cada par e de cada Exchange. Exemplo:

{

"Bybit": {

"USDTBRL": {

"bid": "5.966",

"ask": "5.971",

"time": "2025-01-22 15:15:40.924"

},

"BTCBRL": {

"bid": "625652",

"ask": "627345",

"time": "2025-01-22 15:15:35.505"

},

"BTCUSDT": {

"bid": "104918.52",

"ask": "104918.53",

"time": "2025-01-22 15:15:42.445"

}

},

"OKX": {

"BTC-USDT": {

"bid": "104921.2",

"ask": "104921.3",

"time": "2025-01-22 15:15:42.411"

},

"BTC-BRL": {

"bid": "625890",

"ask": "625980",

"time": "2025-01-22 15:15:42.441"

},

"USDT-BRL": {

"bid": "5.964",

"ask": "5.965",

"time": "2025-01-22 15:15:42.029"

}

},

"Binance": {

"BTCUSDT": {

"bid": "104910.45000000",

"ask": "104910.46000000",

"time": "2025-01-22 15:15:42.414"

},

"USDTBRL": {

"bid": "5.96400000",

"ask": "5.96500000",

"time": "2025-01-22 15:15:42.163"

},

"BTCBRL": {

"bid": "625549.00000000",

"ask": "625684.00000000",

"time": "2025-01-22 15:15:41.995"

}

}

}

* Depois, existe uma outra estrutura chamada bests que armazena os melhores bids e asks de cada par, de todas as exchanges. Exemplo:

{

"BTCBRL": {

"bid": {

"valor": 622139,

"exchange": "Binance"

},

"ask": {

"valor": 621766,

"exchange": "OKX"

}

},

"USDTBRL": {

"bid": {

"valor": 5.973,

"exchange": "Bybit"

},

"ask": {

"valor": 5.968,

"exchange": "Binance"

}

},

"BTCUSDT": {

"bid": {

"valor": 104263.05,

"exchange": "Bybit"

},

"ask": {

"valor": 104229.4,

"exchange": "OKX"

}

}

}

* A mesma função que maneja cotaçõesBuffer também maneja bests.
* O arquivo de log cotacoes.log é criado automaticamente se não existir.
* A execução é contínua, necessitando de interrupção manual.

**3. Estrutura e Principais Funções:**

**a. Configuração de Argumentos de Linha de Comando:**

Os argumentos criptoDaArbitragem, dolarDaArbitragem e moedaDaArbitragem entram a travez do yargs e assim: .argv;

// Recebe os parâmetros da linha de comando

const cripto = argv.criptoDaArbitragem.toUpperCase(); // btc ou outro.

const dolar = argv.dolarDaArbitragem.toUpperCase(); // usdt ou outro.

const moeda = argv.moedaDaArbitragem.toUpperCase(); // brl ou outro.

e definem os pares para arbitragem. Exemplo: BTC, USDT, e BRL vão definir os pares BTCUSDT, BTCBRL E USDTBRL.

**b. Função formatTime(date):**

Formata a hora no formato ISO e inclui milissegundos, usado para logs e cotações.

**c. Função registrarNoLog:**

Grava uma linha formatada no arquivo cotacoes.log.

**d. Função atribuirCotacaoExchange(exchange, par, bid, ask):**

* Atualiza ou adiciona as cotações no buffer cotacoesBuffer.
* Atualiza apenas se houver mudança nos valores de bid/ask.
* Cria uma linha de log e registra no arquivo.

**e. Função setInterval():**

Imprime a estrutura completa do buffer cotacoesBuffer a cada 15 segundos.

**f. Função connectBybitWebSocket(pair):**

* Conecta ao WebSocket da exchange para um par específico.
* Envia uma mensagem para assinar o tópico especifico da exchange, por exemplo orderbook.1.<pair>.
* Recebe os dados do livro de ordens e chama atribuirCotacaoExchange para atualizar as cotações no buffer.

**4. Fluxo de Dados e Interações:**

1. **Entrada de Parâmetros:** Os três parâmetros de arbitragem são fornecidos via linha de comando. Eles geram três pares de arbitragem no formato <cripto><dolar>, <dolar><moeda>, e <cripto><moeda>.
2. **Conexão com WebSocket:** Para cada par gerado, a função connectBybitWebSocket é chamada para se conectar ao WebSocket e assinar o tópico do livro de ordens.
3. **Coleta de Cotações:** Ao receber os dados do WebSocket, a função atribuirCotacaoExchange atualiza o buffer global cotacoesBuffer e registra no arquivo de log.
4. **Exibição no Console:** As cotações são exibidas no console em tempo real conforme atualizadas.
5. **Impressão Periódica:** A estrutura completa de cotacoesBuffer é exibida no console a cada 15 segundos.

**5. Possíveis Melhoria e Extensões:**

1. **Adicionar mais exchanges:** Expandir o suporte para outras exchanges que oferecem WebSocket, com estrutura modular para integração.
2. **Melhorar a escalabilidade.**
3. **Interface de Usuário:** Integrar o blessed para criar uma tela que substitua as impressões do console.log.
4. **Melhorias de Log:** Adicionar logs detalhados para eventos como reconexão e erros no WebSocket.

* **Escalabilidade:** O buffer pode ficar sobrecarregado ao lidar com muitos pares e exchanges.

**8. Comentários Adicionais:**

Nenhum por enquanto

**Workers - Implementação com Modularização de WebSocket**

**Arquitetura Geral:**

A aplicação foi reorganizada para suportar um modelo modular e escalável, utilizando **Workers** para gerenciar cada par de WebSocket de forma independente.

**Componentes:**

**1. Processo Principal (main.js)**

**Funções principais:**

* Gerenciar o buffer global (cotacoesBuffer).
* Inicializar e monitorar os Workers.
* Receber dados processados dos Workers e chamar a função atribuirCotacaoExchange para atualizar o buffer.

**2. Worker (worker+codigoDaExchange.js)**

**Funções principais:**

* Conectar-se ao WebSocket para um par específico.
* Existe um WebSocket para cada par.
* Processar os dados recebidos do WebSocket.
* Enviar os dados processados de volta ao processo principal.

**Codigos de Pares:**

Na linha de comando se ingressa o cripto, dólar e moeda. Ie: BTC, USDT, BRL. Com isso se cria a array const codigosFormaPares = [

  [cripto,dolar],

  [dolar,moeda],

  [cripto,moeda]

];

Na hora de criar os Workers, se executa um forEach para esta array e se passa para o worker cada elemento desta, que são os codigosDePar que serão utilizados para formar cada par, para o acesso a websocket correto.

Já no worker se forma cada par = codigosDePar[0]+separador+ codigosDePar[1].

O separador depende de cada Exchange, em algumas é =”” então da ie: BTCUSDT, em outras, como okx é “-“ e resulta em BTC-USDT.

**Execusao e Controle de Arbitragens.**

**Estrutura:**

let exchanges = [

{ exchange: 'Binance', codigoDaExchange: 'Bin', saldoBTCBRL: 0.0002, saldoBTCUSDT: 0.0001, saldoUSDTBRL: 100, podeArbVenda: true, podeArbCompra: true },

{ exchange: 'Bybit', codigoDaExchange: 'By', saldoBTCBRL: 0.00, saldoBTCUSDT: 0.0003, saldoUSDTBRL: 100, podeArbVenda: false, podeArbCompra: true},

{ exchange: 'OKX', codigoDaExchange: 'OKX', saldoBTCBRL: 0.0002, saldoBTCUSDT: 0.000, saldoUSDTBRL: 100, podeArbVenda: true , podeArbCompra: false},

];

Esta matriz teria que estar no SharedArrayBuffer com acesso para todos os workers.

Aqui tem dois tipos de dados: 1=os saldos dos diferentes pares em cada broker, 2=permissoes. Assim: a arbitragem pode ocorrer em um sentito (por exemplo no sentido de Venda) ou em seu oposto (de Compra). Estes campos informan se é permitido ou nao fazer em cada sentido.

Um worker para saldos e dentro dele 1 socket para os saldos de cada um dos brokers. Os saldos se atualizariam automaticamente com as mensagens dos sockets.

Um worker para as arbitragens. Ele dispara as operacoes da arbitragem seguindo as permissoes e eventualmente as modifica. Aqui seria necessario 1 socket para cada broker e para cada par.

O processo seria o seguinte: Quando o main detecta uma posibilidade de arbitragem checa na array exchanges se os sentidos da arbitragem de cada exchange estao permitidos. Se estao permitidos, entao chama o worker de arbitragens e passa os parametros de: sentido da arbitragem, brokers envolvidos, e precos. O worker executa as operacoes e eventualmente atualiza as permissoes.

**COMENTARIOS GPT**

 **Estrutura exchanges no SharedArrayBuffer:**

* Você apresentou uma estrutura que contém informações de saldos e permissões para arbitragem em diferentes exchanges.
* Foi sugerido que esta matriz fosse armazenada em um SharedArrayBuffer para ser acessada por múltiplos Workers.

 **Processo Proposto:**

* Um Worker de Saldos (WS) atualiza os saldos automaticamente com mensagens recebidas de sockets.
* O Main detecta oportunidades de arbitragem e verifica permissões na estrutura exchanges.
* Se as permissões permitirem, o Main aciona o Worker de Arbitragens (WA) com os parâmetros necessários (sentido, brokers e preços).
* O WA executa as operações de arbitragem e atualiza as permissões e saldos.

 **Discussão de Dúvidas e Ajustes:**

* **Atualização de Saldos e Permissões:**
  + Concordamos que Atomics é necessário para evitar condições de corrida.
* **Comunicação entre Main e Workers:**
  + O método padrão postMessage foi considerado suficiente.
  + O próprio WA atualizará permissões diretamente, evitando ciclos desnecessários de comunicação.
* **Sockets e Escalabilidade:**
  + Inicialmente, a suposição é de que não há limitações no tráfego ou número de sockets. Problemas futuros podem ser tratados com pooling ou distribuição de carga.
* **Conflitos de Permissões e Atualizações Simultâneas:**
  + Foi assumido que conflitos e inconsistências são improváveis. Caso surjam, mutexes ou atrasos poderiam ser implementados.

**Limites Websockets:**

Quero saber se minha aplicacao esta dentro desse limites. Ela arbitra 5 criptos. Cada cripto maneja 3 pares, cada par necessita um websocket que recebe dados do orderbook desse par. Alem disso necessita outro ws com informacoes sobre os saldos da conta. E por fim necessita outros 3 websockets para mandar operacoes de cada par.

Bina, By e Okx ok. 28/1/2025

**Funções Que Executam as Operacoes Via WebSocket e Workers**

**Objetivo:**

Implementação da lógica de arbitragem, prototipado com a utilização de WebSocket de trades a Bybit.

**Logica da Arbitragem.**

O main monitora os preços e testa as arbitragens. Se detectar uma oportunidade ela passa o controle para o worker coordenador. Ele chama a função arbitre(trade1, trade2, trade3). Esta chama a função exeTradePromise(trade) para executar o T1, que sera uma ***operação Limit e IOC***. Se a operação falhar a arbitragem é abortada e retorna ao monitoramento do main. Se a operação der certo chama a função exeTradeNormal(trade) para os T2 e T3. Estas são ***operações Market***.

**1. Função connectaWebSockets:**

A função connectWebSockets é responsável por **estabelecer a conexão com os WebSockets** de trades e orders, enviar a autenticação com a chave de API e monitorar a resposta para verificar se a autenticação foi bem-sucedida.

Os códigos estão na pasta coordenador e tem para bybit e okx.

**2. Função coordena :**

A detecção da arbitragem é feita no Main. Ele passa para o worker coordenador as informações necessárias para sua execução. São 3 operações. T1, T2 e T3. Todas elas podem ser tanto compras quanto vendas. O função coordena o envio das ordens. Primeiro envia T1 e espera o resultado. Se ele foi executado corretamente, então envia T2 e T3. Mas se T1 falhar a arbitragem é abortada.

**3. Estrutura de parâmetros para trades.**

O main tem que passar para a coordenadora um conjunto de parâmetros para que esta execute as três operações. Estes parâmetros são: trade (T1, T2 ou T3), Exchange, side(buy/sell), qty (quantidade), price (preço). Os parâmetros são passados como um objeto no seguinte formato:

const operacoes = [

trade: 'T1',

exchange: 'Bybit',

side: 'buy',

qty: 0.01,

price: 2500

},

{

trade: 'T2',

… etc.

}

Depois, em função de trade, Exchange e side a coordenadora vai definir quais as funções especificas de cada Exchange ela chama.

**4. Função executaTrade1.**

Esta função envia o T1 e depois tem que esperar o resultado. Tem que ser montado algum tipo de processo/código para que o programa pare e não execute nem rejeite os trades T2 e T3 sem que tenha chegado a resposta do T1.

As T1 são sempre limit e IOC (inmediate or cancel).

**5. Função executaTrade2.**

Esta função executa os **trades 2 e 3** que são sempre operações de mercado. Não há necessidade de aguardar resposta, pois esses trades não dependem de validação.

**Descrição:**

* Envia os **trades 2 e 3** para o WebSocket.
* Assume-se que os trades são executados com sucesso e não há necessidade de validação.

function exeTradeNormal(trade) {

const ws = new WebSocket(bybitPrivateWsUrl);

ws.on('open', () => {

console.log('Conectado ao WebSocket para Trade:', trade);

ws.send(JSON.stringify(trade)); // Envia o trade para o WebSocket

});

ws.on('message', (msg) => {

console.log('Mensagem de resposta para Trade:', msg); // Apenas para logging

});

ws.on('error', (err) => {

console.error('Erro no WebSocket para Trade:', err); // Erro no WebSocket

});

}

**5. Resumo do Processo de Arbitragem:**

1. **Conexão WebSocket**: A função connectBybitPrivateWebSocket é chamada para estabelecer a conexão com a Bybit e autenticar a chave de API.
2. **Execução de Trades**: A função arbitre envia os três trades em sequência:
   * **Trade 1** é executado com exeTradePromise. A função aguarda a resposta e, se for bem-sucedida, prossegue com os outros dois trades.
   * **Trade 2 e 3** são executados com exeTradeNormal sem a necessidade de confirmação.
3. **Controle de Erros**: Se **Trade 1** falhar, os trades 2 e 3 não são executados, abortando a arbitragem.

**Formato mandar operaciones ws Bybit.**

const timestamp = Date.now().toString(); // Timestamp atual em milissegundos

ws.send(

JSON.stringify({

"header": {

"X-BAPI-TIMESTAMP": timestamp,

"X-BAPI-RECV-WINDOW": "2000",

// "Referer": "bot-001" // for api broker

},

op: 'order.create',

args: [{

symbol: par,

category: "linear",

side: 'Buy',

orderType: 'Limit',

qty: quantidade.toString(),

price: preco.toString(),

timeInForce: 'GTC',

timestamp: timestamp // Adiciona o timestamp aqui

}]

})

);

**Conclusão:**

A implementação utiliza WebSocket para interagir com a Bybit, organizando a execução dos trades de forma assíncrona. O processo é dividido em funções distintas, cada uma com responsabilidades claras, e o fluxo de trabalho é controlado de forma simples e eficiente, utilizando **Promises** para garantir a execução correta e síncrona do **trade 1**. A solução é escalável e modular, permitindo que os trades subsequentes sejam executados sem sobrecarregar o sistema.

**Como se distribuem os trades entre workers/exchanges**

**Objetivo:**

**Implementar um sistema de arbitragem de criptomoedas que utilize workers para delegar a execução dos trades, garantindo simplicidade no código, boa escalabilidade e desempenho.**

1. Problema Inicial:

Trata-se de um sistema para executar arbitragens de criptomoedas entre diferentes exchanges, montado a partir do Workers threads. Existe uma função no main que monitora os preços dos ativos e aciona a função de arbitragem quando uma oportunidade é detectada. A arbitragem envolve a execução de três trades, onde:

* O trade 1 (T1) é o mais importante. Se ele falhar, trade 2 (T2) e trade 3 (T3) não são executados consequentemente a operação de arbitragem aborta.
* T2 e T3 são operações de mercado e sempre executam após a confirmação do sucesso de T1.

A dúvida era sobre como organizar a distribuição do trabalho entre o main thread e os workers responsáveis pelas diferentes exchanges e pares de criptomoedas.

2. Soluções Consideradas:

Solução 1: Main decide se T1 foi bem-sucedido

* O main thread é responsável por receber a resposta de T1 e decidir se os trades 2 e 3 serão executados.
* O main thread também gerencia a execução de todos os trades.

Vantagens:

* Simplicidade na implementação.

Desvantagens:

* Latência maior: o main precisa aguardar as respostas para decidir a execução dos próximos trades.
* Sobrecarga do main: o main fica responsável por gerenciar a execução de todos os trades, o que pode limitar a escalabilidade.

Solução 2: Worker T1 decide se foi bem-sucedido

* O worker do trade 1 (T1) é responsável por decidir se T1 foi bem-sucedido e, em caso afirmativo, enviar os trades 2 e 3.

Vantagens:

* Menor latência: a decisão é tomada de forma mais rápida, diretamente no worker, sem precisar esperar a comunicação com o main thread.
* Descentralização do trabalho: o main thread não precisa processar nada depois de iniciar a arbitragem.

Desvantagens:

* Aumento na complexidade de implementação e depuração, pois a lógica de decisão é descentralizada nos workers.

Solução 3: Worker Coordenador decide

* O main thread apenas inicia a arbitragem e delega a execução dos trades aos workers específicos de cada exchange e par.
* Um worker coordenador recebe os resultados dos workers responsáveis pelos trades, decide se T1 foi bem-sucedido e executa os trades subsequentes.

Vantagens:

* Simplicidade: a lógica centralizada do coordenador facilita a implementação e depuração.
* Boa escalabilidade: a comunicação entre workers e o coordenador permite maior flexibilidade para adicionar mais exchanges e pares sem sobrecarregar o main thread.

Desvantagens:

* Latência um pouco maior: o coordenador precisa processar os resultados antes de executar os trades subsequentes, mas a diferença não é grande.

3. Análise de Performance:

|  |  |  |  |
| --- | --- | --- | --- |
| Solução | Latência de Comunicação | Carga no Main Thread | Escalabilidade |
| Solução 1 (Main decide) | Alta (espera pela resposta de T1) | Alta (Main gerencia tudo) | Limitada, o main pode travar com muitas arbitragens |
| Solução 2 (Worker T1 decide) | Baixa (decisão rápida no Worker T1) | Nenhuma (Main apenas delega) | Ilimitada, depende apenas da capacidade dos workers |
| Solução 3 (Worker Coordenador decide) | Moderada (decisão no coordenador) | Baixa (Main apenas delega) | Boa, o coordenador gerencia as decisões sem sobrecarregar o Main |

4. Escolha da Solução:

Optamos pela Solução 3 (Worker Coordenador decide) por várias razões:

* Simplicidade: a implementação é mais direta e o código permanece mais simples.
* Escalabilidade razoável: a comunicação entre o main thread, coordenador e workers oferece uma boa flexibilidade sem causar grandes problemas de desempenho.
* Menor complexidade de depuração: a decisão sobre a execução dos trades fica centralizada no coordenador, facilitando o rastreamento de erros e problemas.

Considerações de implementação:

* O main thread inicia o processo de arbitragem e delega os trades a workers específicos de cada exchange.
* O worker coordenador recebe os resultados dos workers e decide se T1 foi bem-sucedido. Se T1 for bem-sucedido, o coordenador envia T2 e T3. Caso contrário, a arbitragem é abortada.

5. Conclusão:

A Solução 3 foi escolhida como a mais adequada para este cenário. Ela equilibra simples implementações, boa performance e escalabilidade. Além disso, a comunicação entre o main, o coordenador e os workers permite que a arbitragem seja executada de forma eficiente e sem sobrecarregar o sistema.

**-------------**

**Extensão do Worker Coordenador e Implementação do Testador**

Para otimizar a implementação inicial e facilitar os testes do sistema de arbitragem, decidimos modificar a estrutura do worker coordenador e adicionar um módulo de testes interativo.

**1. Modificações no Worker Coordenador**

* O worker coordenador será responsável por abrir e manter um WebSocket para cada exchange.
* Esses WebSockets permanecerão sempre ativos, mesmo quando não houver operações.
  + Para evitar desconexões, será implementado um mecanismo de ping/pong e reconexão automática.
* O worker receberá chamadas de execução de ordens do main e realizará as operações correspondentes.

**2. Implementação do Testador Independente**

* Para testar sem depender do main, criamos um testador interativo.
* Ele permitirá o envio manual de ordens para o worker coordenador, simulando chamadas reais.
* O testador exibirá a resposta completa de cada operação e continuará disponível para novas interações.

**Fluxo do Testador**

1. O worker coordenador abre os WebSockets e avisa quando todos estiverem prontos.
2. O testador permite ao usuário inserir exchange, par, side (buy/sell), tipo (market/limit), quantidade e preço (se necessário).
3. A ordem é enviada ao worker coordenador e executada.
4. A resposta da exchange é exibida no testador.

**3. Vantagens da Abordagem**

* Menos complexidade inicial: Um único worker gerencia todas as exchanges.
* WebSockets sempre ativos: Redução do tempo de resposta nas operações.
* Testes mais fáceis: Permite validar rapidamente o funcionamento dos WebSockets e das ordens sem precisar esperar arbitragem real.
* Expansível no futuro: Poderemos adicionar logs detalhados, simulações e automações conforme necessário.

**CAPITULO DA MUDANÇA DE ESTRUTURAS.**

São as mudanças que vão ter que ser integradas no main geral e Workers para substituir a estrutura de index pela de heap, além de agregar as amount das ofertas.

**Documentação Atualizada do Código de Arbitragem de Criptomoedas**

**Data Atual**: 28 de fevereiro de 2025  
**Objetivo Geral**: Desenvolver um sistema de monitoramento e execução de arbitragem de criptomoedas em tempo real, utilizando WebSockets para coletar cotações de múltiplas exchanges, processando bids, asks e quantidades (amount) para identificar e aproveitar oportunidades de arbitragem.

**Visão Geral**

Existem dois códigos principais sendo desenvolvidos em paralelo:

1. **Código 1 (Chat Atual - Heap)**: Focado na otimização da estrutura de dados, migrando de index para heap pra melhorar desempenho e precisão na rastreamento dos melhores bids e asks, incluindo amount pra refletir liquidez. Este código coleta cotações em tempo real e mantém uma estrutura de bests atualizada.
2. **Código 2 (Outro Chat - Index e Avaliação)**: Focado na avaliação de oportunidades de arbitragem, com cálculo de lucros potenciais e logs detalhados, mas usa uma estrutura de index mais simples e não considera amount.

**1. Código 1: Modelo com Heap (Chat Atual)**

**Descrição**: Este código implementa um sistema de monitoramento de cotações em tempo real, substituindo a estrutura de index por heap (max-heap pra bid, min-heap pra ask) pra otimizar a atualização dos melhores preços. Inclui amount nas cotações pra capturar liquidez e suporta múltiplas exchanges (Bybit, Binance, Deribit, Bitpreco).

**Estrutura e Principais Componentes**

* **Argumentos de Linha de Comando**: Recebe cripto, dolar e moeda (ex.: BTC, USDT, BRL) via yargs, gerando os pares BTCUSDT, BTCBRL, USDTBRL.
* **Exchanges**: Lista configurável com { exchange, fees, venderPode, comprarPode }. Atualmente inclui Bybit, Binance, Deribit e Bitpreco.
* **Estrutura de Dados**:
  + **cotacoesBuffer**: Armazena cotações por exchange e par, com bid, bidAmount, bidTime, ask, askAmount, askTime.

javascript

WrapCopy

{

"Bybit": {

"BTCUSDT": { bid: "86751.64", bidAmount: "0.7287", bidTime: "2025-02-27 10:54:47.685", ask: "86751.65", askAmount: "0.0307", askTime: "2025-02-27 10:54:48.075" },

...

},

...

}

* + **bests**: Mantém os melhores bid e ask por par, com preco, amount, time e exchange, usando heaps.

javascript

WrapCopy

{

"BTCUSDT": {

"bid": { preco: 86777.03, amount: 7.3911, time: "2025-02-27 10:54:46.952", exchange: "Binance" },

"ask": { preco: 86751.65, amount: 0.0307, time: "2025-02-27 10:54:48.075", exchange: "Bybit" }

},

...

}

* + **bidHeaps e askHeaps**: Heaps por par pra otimizar a seleção dos melhores preços (O(log n) por atualização).
* **Workers**: Um por exchange e par (ex.: workerBybit.js), coletam dados via WebSocket e enviam pro main.
* **Funções Principais**:
  + atribuirCotacaoExchange: Atualiza cotacoesBuffer e chama updateBests pra manter bests atualizado.
  + updateBests: Insere/atualiza entradas nos heaps e seleciona o melhor bid/ask.
  + inicializarWorkers: Cria workers pra cada par e exchange.
  + formatNumber: Formata preços e quantidades (ex.: BTCUSDT preço com 2 decimais, amount com 4).
* **Saída**: Exibe cotacoesBuffer e bests no console a cada 10 segundos e grava em saida.txt (sobrescrevendo no início).

**Mudanças em Relação ao Código Original**

* **De index pra heap**: Substituímos o index (O(n log n) pra ordenação) por heap (O(log n) por atualização), resolvendo problemas de atualização em quedas de bid e subidas de ask.
* **Inclusão de amount**: Adicionamos bidAmount e askAmount em cotacoesBuffer e bests, refletindo liquidez, ausente no outro código.
* **Mais Exchanges**: Expandimos de Bybit/OKX pra incluir Binance, Deribit e Bitpreco.
* **Horários Precisos**: Usamos timestamps das exchanges (ex.: utimestamp da Bitpreco) quando disponíveis, em vez de new Date() local.

**Considerações**

* **Desempenho**: O uso de heaps melhora a eficiência em cenários de alta frequência.
* **Escalabilidade**: Suporta múltiplas exchanges e pares, mas ainda não implementa lógica completa de arbitragem.
* **Pendências**: Falta calcular e executar arbitragens baseadas nos bests.

**2. Código 2: Modelo com Index e Avaliação (Outro Chat)**

**Descrição**: Este código monitora cotações, avalia oportunidades de arbitragem e grava logs detalhados, usando uma estrutura de index pra rastrear bids e asks. Não inclui amount e foca em análise e registro.

**Estrutura e Principais Componentes**

* **Argumentos de Linha de Comando**: Igual ao Código 1, gera pares como BTCUSDT, BTCBRL, USDTBRL.
* **Exchanges**: Lista com { exchange, fees, venderPode, comprarPode }, atualmente Bybit, Binance, OKX, Deribit.
* **Estrutura de Dados**:
  + **cotacoesBuffer**: Similar ao Código 1, mas só com bid, ask e time, sem amount.

javascript

WrapCopy

{

"Binance": {

"BTCUSDT": { bid: "104910.45", ask: "104910.46", time: "2025-01-22 15:15:42.414" },

...

},

...

}

* + **bests**: Melhores bid/ask por par, com valor e exchange, sem amount.

javascript

WrapCopy

{

"BTCUSDT": {

"bid": { valor: 104263.05, exchange: "Bybit" },

"ask": { valor: 104229.4, exchange: "OKX" }

},

...

}

* + **index**: Lista ordenada de bids e asks por par e exchange.
* **Workers**: Um por exchange e par, similares ao Código 1.
* **Funções Principais**:
  + atribuirCotacaoExchange: Atualiza cotacoesBuffer e chama processUpdateAndLog.
  + updateBests: Atualiza bests com base em bid/ask, sem usar heaps.
  + updateIndex: Mantém o index ordenado (O(n log n)).
  + testaArbitragens: Calcula lucros potenciais de venda/compra doméstica e loga se ultrapassar o gatilho (0.003).
  + registrarNoLog: Grava logs em cotacoes<cripto>.txt com cotações e resultados.
* **Saída**: Logs detalhados em arquivo CSV e console a cada 60 segundos.

**Considerações**

* **Foco em Avaliação**: Calcula arbitragens (vendeDomesticamente, compraDomesticamente) e loga resultados.
* **Limitação**: Não considera liquidez (amount) e usa index, menos eficiente que heap.

**Diferenças Entre os Códigos**

| **Aspecto** | **Código 1 (Heap)** | **Código 2 (Index)** |
| --- | --- | --- |
| **Estrutura de Dados** | Heap (max/min) | Index (lista ordenada) |
| **Complexidade** | O(log n) por atualização | O(n log n) por ordenação |
| **Amount** | Inclui bidAmount e askAmount | Não inclui |
| **Exchanges** | Bybit, Binance, Deribit, Bitpreco | Bybit, Binance, OKX, Deribit |
| **Arbitragem** | Monitora bests, sem execução ainda | Avalia e loga oportunidades |
| **Saída** | Console e saida.txt | Console e cotacoes<cripto>.txt |

**Mudanças Necessárias no Código 2 pra Adaptação**

Pra alinhar o Código 2 com as melhorias do Código 1, aqui estão as adaptações necessárias:

1. **Substituir index por heap**:
   * Remover index, buildIndices, updateIndex e processUpdateAndLog.
   * Adicionar bidHeaps e askHeaps como no Código 1.
   * Atualizar updateBests pra usar heaps:

javascript

WrapCopy

function updateBests(exchange, par, bid, bidAmount, bidTime, ask, askAmount, askTime) {

if (bid !== null && bid !== undefined) {

const bidEntry = { preco: Number(bid), amount: Number(bidAmount), time: bidTime, exchange };

const heapArray = bidHeaps[par].toArray();

const existingIndex = heapArray.findIndex(e => e.exchange === exchange);

if (existingIndex !== -1) heapArray.splice(existingIndex, 1);

heapArray.push(bidEntry);

bidHeaps[par] = new Heap((a, b) => b.preco - a.preco);

heapArray.forEach(item => bidHeaps[par].push(item));

bests[par].bid = bidHeaps[par].peek() || { preco: 0, amount: 0, time: 'Date.now()', exchange: "" };

}

*// Similar pra ask com min-heap*

}

* + Adicionar Heap como dependência (const Heap = require('heap');).

1. **Incluir amount**:
   * Atualizar cotacoesBuffer e bests pra incluir bidAmount, askAmount:

javascript

WrapCopy

const cotacoesBuffer = {

Binance: {

[cripto+dolar]: { bid: '0', bidAmount: '0', bidTime: 'Date.now()', ask: '9007199254740991', askAmount: '0', askTime: 'Date.now()' },

...

},

...

};

const bests = {

[cripto+dolar]: { "bid": { preco: 0, amount: 0, time: 'Date.now()', exchange: "" }, "ask": { preco: Infinity, amount: 0, time: 'Date.now()', exchange: "" } },

...

};

* + Modificar atribuirCotacaoExchange pra lidar com bidAmount e askAmount:

javascript

WrapCopy

function atribuirCotacaoExchange(exchange, par, bid = null, bidAmount = null, bidTime = null, ask = null, askAmount = null, askTime = null) {

const time = formatTime(new Date());

if (!cotacoesBuffer[exchange]) cotacoesBuffer[exchange] = {};

if (!cotacoesBuffer[exchange][par]) cotacoesBuffer[exchange][par] = { bid: null, bidAmount: null, bidTime: null, ask: null, askAmount: null, askTime: null };

if (bid !== null) cotacoesBuffer[exchange][par].bid = bid;

if (bidAmount !== null) cotacoesBuffer[exchange][par].bidAmount = bidAmount;

if (bidTime !== null) cotacoesBuffer[exchange][par].bidTime = bidTime || time;

if (ask !== null) cotacoesBuffer[exchange][par].ask = ask;

if (askAmount !== null) cotacoesBuffer[exchange][par].askAmount = askAmount;

if (askTime !== null) cotacoesBuffer[exchange][par].askTime = askTime || time;

if (bid > bests[par].bid.preco || ask < bests[par].ask.preco) updateBests(exchange, par, bid, bidAmount, bidTime, ask, askAmount, askTime);

testaArbitragens();

}

* + Ajustar workers pra enviar bidAmount e askAmount junto com bid e ask.

1. **Atualizar testaArbitragens**:
   * Incluir amount na lógica de arbitragem, verificando liquidez disponível antes de calcular lucros.

javascript

WrapCopy

function testaArbitragens() {

const time = formatTime(new Date());

const vendeDomesticamente = (bests[cripto+moeda].bid.preco / (bests[dolar+moeda].ask.preco \* bests[cripto+dolar].ask.preco) - 1).toFixed(5);

const compraDomesticamente = (bests[cripto+dolar].bid.preco \* bests[dolar+moeda].bid.preco / bests[cripto+moeda].ask.preco - 1).toFixed(5);

if (vendeDomesticamente > gatilho && bests[cripto+moeda].bid.amount > 0 && bests[dolar+moeda].ask.amount > 0 && bests[cripto+dolar].ask.amount > 0) {

console.log(time, '---', cripto, 'VD:', vendeDomesticamente, bests[cripto+moeda].bid.exchange);

registrarNoLog(vendeDomesticamente, compraDomesticamente);

}

if (compraDomesticamente > gatilho && bests[cripto+dolar].bid.amount > 0 && bests[dolar+moeda].bid.amount > 0 && bests[cripto+moeda].ask.amount > 0) {

console.log(time, '---', cripto, 'CD:', compraDomesticamente, bests[cripto+moeda].ask.exchange);

registrarNoLog(vendeDomesticamente, compraDomesticamente);

}

}

**Próximo Passo Aqui**

Com a documentação atualizada, podemos avançar no modelo de arbitragem com heap. Sugiro começar a calcular oportunidades de arbitragem baseadas nos bests e simular trades, aproveitando o amount pra verificar liquidez. O que acha? Me avisa como quer prosseguir!

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* PECULIARIDADES IMPORTANTES \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

Na Bybit as operações se postam no websocket /trade mas se visualizam no ws /order

[**WebSocket Order Entry**](https://bybit-exchange.github.io/docs/v5/websocket/trade/guideline)**: Mainnet:** wss://stream.bybit.com/v5/trade

[**WebSocket private stream**](https://bybit-exchange.github.io/docs/v5/websocket/private/order)**: Mainnet:** wss://stream.bybit.com/v5/private

e depois /order.

O sucesso nas respostas se avalia diferente nos dois:

No /trade: Mensagem recebida: {

retCode: 0,

retMsg: 'OK',

op: 'auth',

connId: 'cths17eg28s8ddq8p2p0-alu6q'

} **Sucesso significa** **que msg.retCode === 0.**

No /order: Mensagem recebida: {

success: true,

ret\_msg: '',

op: 'auth',

conn\_id: 'cthrpi5hv25dalqcc6v0-17e0if'

} **Sucesso significa que msg.success === true.**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* -------------------------- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Para definir qual é T1 tem que agregar uma rotina que veja qual é o preço que esta desencaixado da media que esta provocando a arbitragem. Essa operação tem que ser a T1.** Isso se poderia fazer calculando a média das cotações na matriz de cotações e vendo o maior desvio. Inclusive, poderia ser feito junto com o calculo dos bests.

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* -------------------------- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Instrucoes Para Gpt Criar Worker Dados para NOVA EXCHANGE.**

Vou te passar o códigos de worker que recupera o primeiro bid e o primeiro ask do orderbook da binance de um determinado instrumento que chamamos par. Observe que o worker recebe do main um objeto com dois códigos, e com esses códigos se forma a constante par que depois vai definir qual é o ws que o worker devera se conectar para obter os dados de bid e ask.

Eu preciso que vc faça o código do worker da Exchange ....................... Ou seja, vc vai tomar como modelo o worker da binance e criar o outro worker. Repare que, como cada Exchange define uma estrutura de dados e procedimentos peculiar, vc vai ter que consultar a documentação da nova exchange para ajustar a ela o código do modelo.

Alem disso, e antes que mais nada, vc tem que checar que o canal para esse par existe. Porque, por exemplo, o canal para BTC\_USDT existe, mas para BTC\_BRL nem sempre. Se o canal não existir finaliza o worker.

Também, tem que colocar um warning alertando para ajustar o separador entre os códigos recebidos do main usado pelo exange: tipo assim:

\*\*\*\*\*\*\*\*\*\*\*\*\*\* NÃO ESQUEÇA DE AJUSTAR O SEPARADOR DO PAR \*\*\*\*\*\*\*\*\*\*\*\*\*\*

Voce entendeu o que quero? Não faça nada ainda, so me responde e me explica o que vc vai fazer. Seguem os códigos.

Codigo do worker da binance:

\*\*\*\*\*\*\*\*\*\*!!!!!!!!!!!!! INCLUIR UM WORKER ATUALIZADO.

**Formatos dos Dados dos OrderBooks.**

binance: {

u: 61654272714,

s: 'BTCUSDT',

b: '93413.16000000', <--------------bid preco

B: '0.88285000', <--------------bid quantidade

a: '93413.17000000', <--------------Ask preço

A: '0.16641000' <--------------Ask quantidade

}

mais detalhes em https://developers.binance.com/docs/binance-spot-api-docs/web-socket-streams#individual-symbol-book-ticker-streams.

bybit:

{

"topic": "orderbook.50.BTCUSDT",

"type": "snapshot",

"ts": 1672304484978,

"data": {

"s": "BTCUSDT",

"b": [ //BID

...,

[

"16493.50", // Preco

"0.006" // Quantidade

],

[

"16493.00",

"0.100"

]

],

"a": [ // ASK

[

"16611.00",

"0.029"

],

mais informacao https://bybit-exchange.github.io/docs/v5/websocket/public/orderbook

okx

{

arg: { channel: 'bbo-tbt', instId: 'ETH-USDT' },

data: [

{

asks: [Array],

bids: [Array],

ts: '1740482559354',

seqId: 43126538467

}

]

}

asks e bids arrays: [ [ '2381.61', '0.2764', '0', '1' ] ] // preco, quantidade, depreciated, quantidade de ofertas

mais informacoes https://www.okx.com/docs-v5/en/?shell#order-book-trading-market-data-ws-all-trades-channel em tag WS/Order book channel

Bitpreco:

{

asks: [

{ amount: 0.03499, exchange: [], id: '1577771568', price: 500274 },

{

amount: 0.01472,

exchange: [Object],

id: '1577771546',

price: 500621

},

{ amount: 0.028619, exchange: [], id: '1577771848', price: 500885 },

{

amount: 0.5038485,

exchange: [],

id: '1577771793',

price: 500911

…

…

],

bids: [

{

amount: 0.50783199,

exchange: [],

id: '1577771839',

price: 499809

},

{ amount: 0.002622, exchange: [], id: '1577771846', price: 499808 },

{

amount: 0.00040015,

exchange: [],

id: '1577771739',

price: 499807

},

{

amount: 0.00009,

exchange: [Object],

id: '1577771645',

price: 499504

},

{

amount: 0.05,

exchange: [Object],

id: '1577771715',

price: 499388.47

},

{

amount: 0.23495,

exchange: [Object],

id: '1577771784',

price: 498801

},

…

…

Lista todos as ofertas de bids e todas as ofertas de asks do orderbook desse par.

Mais info <https://bitypreco.com/api#web_socket>

**PARAMETROS OKX**

tdMode

Trade Mode, when placing an order, you need to specify the trade mode.

Spot mode:

- SPOT and OPTION buyer: cash

Spot and futures mode:

- Isolated MARGIN: isolated

- Cross MARGIN: cross

- SPOT: cash

- Cross FUTURES/SWAP/OPTION: cross

- Isolated FUTURES/SWAP/OPTION: isolated

Multi-currency margin:

- Isolated MARGIN: isolated

- Cross SPOT: cross

- Cross FUTURES/SWAP/OPTION: cross

- Isolated FUTURES/SWAP/OPTION: isolated

Portfolio margin:

- Isolated MARGIN: isolated

- Cross SPOT: cross

- Cross FUTURES/SWAP/OPTION: cross

- Isolated FUTURES/SWAP/OPTION: isolated

ordType

Order type. When creating a new order, you must specify the order type. The order type you specify will affect: 1) what order parameters are required, and 2) how the matching system executes your order. The following are valid order types:

limit: Limit order, which requires specified sz and px.

market: Market order. For SPOT and MARGIN, market order will be filled with market price (by swiping opposite order book). For Expiry Futures and Perpetual Futures, market order will be placed to order book with most aggressive price allowed by Price Limit Mechanism. For OPTION, market order is not supported yet. As the filled price for market orders cannot be determined in advance, OKX reserves/freezes your quote currency by an additional 5% for risk check.

post\_only: Post-only order, which the order can only provide liquidity to the market and be a maker. If the order would have executed on placement, it will be canceled instead.

fok: Fill or kill order. If the order cannot be fully filled, the order will be canceled. The order would not be partially filled.

ioc: Immediate or cancel order. Immediately execute the transaction at the order price, cancel the remaining unfilled quantity of the order, and the order quantity will not be displayed in the order book.

optimal\_limit\_ioc: Market order with ioc (immediate or cancel). Immediately execute the transaction of this market order, cancel the remaining unfilled quantity of the order, and the order quantity will not be displayed in the order book. Only applicable to Expiry Futures and Perpetual Futures.

sz

Quantity to buy or sell.

For SPOT/MARGIN Buy and Sell Limit Orders, it refers to the quantity in base currency.

For MARGIN Buy Market Orders, it refers to the quantity in quote currency.

For MARGIN Sell Market Orders, it refers to the quantity in base currency.

For SPOT Market Orders, it is set by tgtCcy.

For FUTURES/SWAP/OPTION orders, it refers to the number of contracts.

tgtCcy

This parameter is used to specify the order quantity in the order request is denominated in the quantity of base or quote currency. This is applicable to SPOT Market Orders only.

Base currency: base\_ccy

Quote currency: quote\_ccy

If you use the Base Currency quantity for buy market orders or the Quote Currency for sell market orders, please note:

1. If the quantity you enter is greater than what you can buy or sell, the system will execute the order according to your maximum buyable or sellable quantity. If you want to trade according to the specified quantity, you should use Limit orders.

2. When the market price is too volatile, the locked balance may not be sufficient to buy the Base Currency quantity or sell to receive the Quote Currency that you specified. We will change the quantity of the order to execute the order based on best effort principle based on your account balance. In addition, we will try to over lock a fraction of your balance to avoid changing the order quantity.

2.1 Example of base currency buy market order:

Taking the market order to buy 10 LTCs as an example, and the user can buy 11 LTC. At this time, if 10 < 11, the order is accepted. When the LTC-USDT market price is 200, and the locked balance of the user is 3,000 USDT, as 200\*10 < 3,000, the market order of 10 LTC is fully executed; If the market is too volatile and the LTC-USDT market price becomes 400, 400\*10 > 3,000, the user's locked balance is not sufficient to buy using the specified amount of base currency, the user's maximum locked balance of 3,000 USDT will be used to settle the trade. Final transaction quantity becomes 3,000/400 = 7.5 LTC.

2.2 Example of quote currency sell market order:

Taking the market order to sell 1,000 USDT as an example, and the user can sell 1,200 USDT, 1,000 < 1,200, the order is accepted. When the LTC-USDT market price is 200, and the locked balance of the user is 6 LTC, as 1,000/200 < 6, the market order of 1,000 USDT is fully executed; If the market is too volatile and the LTC-USDT market price becomes 100, 100\*6 < 1,000, the user's locked balance is not sufficient to sell using the specified amount of quote currency, the user's maximum locked balance of 6 LTC will be used to settle the trade. Final transaction quantity becomes 6 \* 100 = 600 USDT.

**TESTES PARA CONECTAR WS BINANCE**

C:\Users\Yaco\Desktop\Cryptos\Nodeprojects\bot\BotsGPT\Trades>node tst

Conexão estabelecida

Recebida resposta (JSON): {

id: 1,

status: 400,

error: {

code: -1100,

msg: "Illegal characters found in parameter 'apiKey'; legal range is '^[a-zA-Z0-9]{36,64}$'."

},

rateLimits: [

{

rateLimitType: 'REQUEST\_WEIGHT',

interval: 'MINUTE',

intervalNum: 1,

limit: 6000,

count: 8

}

]

}

^C

C:\Users\Yaco\Desktop\Cryptos\Nodeprojects\bot\BotsGPT\Trades>node tst

Conexão estabelecida

Recebida resposta (JSON): {

id: 1,

status: 401,

error: {

code: -2015,

msg: 'Invalid API-key, IP, or permissions for action.'

},

rateLimits: [

{

rateLimitType: 'REQUEST\_WEIGHT',

interval: 'MINUTE',

intervalNum: 1,

limit: 6000,

count: 4

}

]

}

Conexão fechada

C:\Users\Yaco\Desktop\Cryptos\Nodeprojects\bot\BotsGPT\Trades>node tst

Conexão estabelecida

Recebida resposta (JSON): {

id: 1,

status: 400,

error: { code: -1022, msg: 'Signature for this request is not valid.' },

rateLimits: [

{

rateLimitType: 'REQUEST\_WEIGHT',

interval: 'MINUTE',

intervalNum: 1,

limit: 6000,

count: 4

}

]

}

Conexão fechada

C:\Users\Yaco\Desktop\Cryptos\Nodeprojects\bot\BotsGPT\Trades>node tst

Conexão estabelecida

Timestamp: 1738679417418

Method: session.logon

Params: {}

Message: 1738679417418session.logon{}

Key: KeyPair {

ec: EC {

curve: EdwardsCurve {

twisted: true,

mOneA: true,

extended: true,

type: 'edwards',

p: [BN],

red: [Red],

zero: [BN],

one: [BN],

two: [BN],

n: [BN],

g: [Point],

\_wnafT1: [Array],

\_wnafT2: [Array],

\_wnafT3: [Array],

\_wnafT4: [Array],

\_bitLength: 253,

\_maxwellTrick: true,

redN: [BN],

a: [BN],

c: [BN],

c2: [BN],

d: [BN],

dd: [BN],

oneC: true

},

n: BN { negative: 0, words: [Array], length: 10, red: null },

nh: BN { negative: 0, words: [Array], length: 10, red: null },

g: Point {

curve: [EdwardsCurve],

type: 'projective',

precomputed: [Object],

x: [BN],

y: [BN],

z: [BN],

t: [BN],

zOne: true

},

hash: [Function: SHA256] {

blockSize: 512,

outSize: 256,

hmacStrength: 192,

padLength: 64

}

},

priv: BN {

negative: 0,

words: [

54708268, 48361339,

46090246, 22077414,

59019031, 37681826,

32108347, 46767858,

27767197, 201512,

0

],

length: 10,

red: null

},

pub: null

}

Signature: 304402200b385a670d2b35241867766e941d9c4161227f196ea843a63ecee988e3f6c2cd02200b3e0a371bb5f19ad5f8209c4b1b9985b42f0818f404c7592eef5d1ee7c7a777

Recebida resposta (JSON): {

id: 1,

status: 400,

error: { code: -1022, msg: 'Signature for this request is not valid.' },

rateLimits: [

{

rateLimitType: 'REQUEST\_WEIGHT',

interval: 'MINUTE',

intervalNum: 1,

limit: 6000,

count: 4

}

]

}

Conexão fechada

C:\Users\Yaco\Desktop\Cryptos\Nodeprojects\bot\BotsGPT\Trades>openssl pkey -in private\_key.txt -outform der | openssl enc -inform der -outform hex

'openssl' não é reconhecido como um comando interno

ou externo, um programa operável ou um arquivo em lotes.

C:\Users\Yaco\Desktop\Cryptos\Nodeprojects\bot\BotsGPT\Trades>node tst1

node:internal/crypto/cipher:119

this[kHandle].init(cipher, credential, authTagLength);

^

Error: Unknown cipher

at Decipher.createCipherBase (node:internal/crypto/cipher:119:19)

at Decipher.createCipher (node:internal/crypto/cipher:132:3)

at new Decipher (node:internal/crypto/cipher:273:3)

at Object.createDecipher (node:crypto:150:10)

at Object.deprecated (node:internal/util:177:12)

at Object.<anonymous> (C:\Users\Yaco\Desktop\Cryptos\Nodeprojects\bot\BotsGPT\Trades\tst1.js:11:30)

at Module.\_compile (node:internal/modules/cjs/loader:1376:14)

at Module.\_extensions..js (node:internal/modules/cjs/loader:1435:10)

at Module.load (node:internal/modules/cjs/loader:1207:32)

at Module.\_load (node:internal/modules/cjs/loader:1023:12) {

code: 'ERR\_CRYPTO\_UNKNOWN\_CIPHER'

}

Node.js v20.11.0

C:\Users\Yaco\Desktop\Cryptos\Nodeprojects\bot\BotsGPT\Trades>node tst1

Conexão estabelecida

Recebida resposta (JSON): {

id: 1,

status: 400,

error: { code: -1022, msg: 'Signature for this request is not valid.' },

rateLimits: [

{

rateLimitType: 'REQUEST\_WEIGHT',

interval: 'MINUTE',

intervalNum: 1,

limit: 6000,

count: 4

}

]

}

Conexão fechada

C:\Users\Yaco\Desktop\Cryptos\Nodeprojects\bot\BotsGPT\Trades>node tst1

Conexão estabelecida

C:\Users\Yaco\Desktop\Cryptos\Nodeprojects\bot\BotsGPT\Trades\tst1.js:16

const signature = key.sign(message).toHex().slice(0, 128);

^

TypeError: key.sign(...).toHex is not a function

at generateSignature (C:\Users\Yaco\Desktop\Cryptos\Nodeprojects\bot\BotsGPT\Trades\tst1.js:16:39)

at WebSocket.<anonymous> (C:\Users\Yaco\Desktop\Cryptos\Nodeprojects\bot\BotsGPT\Trades\tst1.js:31:16)

at WebSocket.emit (node:events:518:28)

at WebSocket.setSocket (C:\Users\Yaco\Desktop\Cryptos\Nodeprojects\bot\node\_modules\ws\lib\websocket.js:251:10)

at ClientRequest.<anonymous> (C:\Users\Yaco\Desktop\Cryptos\Nodeprojects\bot\node\_modules\ws\lib\websocket.js:1012:15)

at ClientRequest.emit (node:events:518:28)

at TLSSocket.socketOnData (node:\_http\_client:575:11)

at TLSSocket.emit (node:events:518:28)

at addChunk (node:internal/streams/readable:559:12)

at readableAddChunkPushByteMode (node:internal/streams/readable:510:3)

Node.js v20.11.0

C:\Users\Yaco\Desktop\Cryptos\Nodeprojects\bot\BotsGPT\Trades>node tst1

Conexão estabelecida

Recebida resposta (JSON): {

id: 1,

status: 400,

error: { code: -1022, msg: 'Signature for this request is not valid.' },

rateLimits: [

{

rateLimitType: 'REQUEST\_WEIGHT',

interval: 'MINUTE',

intervalNum: 1,

limit: 6000,

count: 4

}

]

}

Conexão fechada

C:\Users\Yaco\Desktop\Cryptos\Nodeprojects\bot\BotsGPT\Trades>