// Arquivo: mensagens\_chat\_completo.txt

// Mensagem 1 (15:XX, 16/03/2025)

Grok, leia esta documentação mas nao faca nada, já te passo o código main e mais informacao sobre o testador para continuarmos.

DOCUMENTACAO ARBIS MVD

Objetivo Geral

Desenvolver um sistema modular e escalável de monitoramento e detecção de arbitragem triangular em tempo real, utilizando WebSockets para coletar cotações de cinco exchanges (Bybit, Binance, Deribit, Bitpreco, OKX), processando bids, asks e quantidades (amount) para identificar oportunidades viáveis com base em lucro mínimo (gatilho = 0.003) e liquidez mínima (valorMinimoArbitragem = 100 USDT). O sistema registra resultados em logs estruturados (CSV e snapshots) e está preparado para futura integração de execução de trades via Worker coordenador.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Descrição Geral

O sistema, implementado em Node.js com o script principal mainheap.js, gerencia a coleta de cotações em tempo real de três pares (ex.: BTCUSDT, BTCBRL, USDTBRL) definidos por argumentos de linha de comando (cripto, dolar, moeda) através de 15 workers (3 pares x 5 exchanges). Ele processa dados em estruturas otimizadas (cotacoes e bests com heaps), calcula lucros potenciais de arbitragem triangular (vendeDomesticamente e compraDomesticamente), valida liquidez, e registra oportunidades em arquivos CSV (Arbs[cripto][dolar][moeda].txt) e snapshots (saida.txt). O console exibe marcações de tempo (30s) e mensagens de arbitragem, preparando o terreno para a próxima etapa: execução de trades via Worker coordenador.

Adição (07/03/2025): Neste chat, o foco foi ajustar o script (renomeado internamente para novomainc.js em algumas discussões) para garantir que bests refletisse corretamente os melhores preços em cotacoes, preparando o sistema para a transição ao Worker coordenador no próximo chat.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

2. Considerações Fundamentais e Restrições

Performance:

• Uso de heaps (O(log n)) em bidHeaps e askHeaps para atualizar bests, priorizando eficiência em alta frequência.

• Escolha justificada: Comparado a index (O(n log n)), heaps reduzem latência em atualizações frequentes, essencial para cotações em tempo real.

Dados Coletados: Bid, ask, bidAmount, askAmount, bidTime, askTime, capturando preços e liquidez.

Formação Dos Pares:

Os pares de negociação, normalmente chamados ‘symbol’ seguem o padrão de moeda-base e moeda-de-negociacao. Ex. o Bitcoin negociado em Theter é BTCUSDT e negociado em Reais é BTCBRL.

Na arbitragem triangular que estamos programando vai ter três ativos/entidades e três ‘symbol’. Os ativos são: cripto, dólar e moeda, que são respectivamente: BTC, USDT e BRL. (poderiam ter sido ETH, USDC e EUR). Estas três entidades definem os pares da seguite maneira: cripto+dólar, dólar+moeda e cripto+moeda, que rendem os symbols da arbitragem BTCUSDT, USDTBRL e BTCBRL.

Workers:

• 15 workers (3 pares x 5 exchanges: Bybit, Binance, Deribit, Bitpreco, OKX), cada um com um WebSocket ativo.

• Razão: Modularidade e escalabilidade, evitando sobrecarga no main thread. Alternativas como um WebSocket por exchange foram descartadas por maior complexidade na gestão de pares.

Estruturas de Dados:

• exchanges:

const exchanges = [

{ exchange: 'Bybit', fees: 0.001, venderPode: true, comprarPode: true },

{ exchange: 'Binance', fees: 0.001, venderPode: true, comprarPode: true },

{ exchange: 'Deribit', fees: 0.001, venderPode: true, comprarPode: true },

{ exchange: 'bitpreco', fees: 0.001, venderPode: true, comprarPode: true },

{ exchange: 'OKX', fees: 0.001, venderPode: true, comprarPode: true }

];

o Descrição: Lista estática de exchanges com taxas (fees) e permissões (venderPode, comprarPode). Futuramente, será expandida com saldos dinâmicos (ex.: saldoBTCBRL, saldoBTCUSDT, saldoUSDTBRL) em SharedArrayBuffer.

o Propósito: Configuração inicial para workers e validação preliminar de arbitragem.

• cotacoes:

const cotacoes = {

"Binance": {

"BTCUSDT": { bid: "91967.25", bidAmount: "5.2198", bidTime: "2025-03-03 10:04:58.495", ask: "91967.26", askAmount: "0.6324", askTime: "2025-03-03 10:04:58.495" },

...

},

...

};

o Descrição: Buffer global com cotações por exchange e par, incluindo preços e quantidades (amount), atualizado em tempo real pelos workers.

o Mudança: Renomeado de cotacoesBuffer para cotacoes, enriquecido com amount para refletir liquidez.

• bests:

const bests = {

"BTCUSDT": {

"bid": { preco: 91967.25, amount: 5.2198, time: "2025-03-03 10:04:58.495", exchange: "Binance" },

"ask": { preco: 91959.73, amount: 0.0206, time: "2025-03-03 10:04:58.462", exchange: "Bybit" }

},

...

};

o Descrição: Melhores bid e ask por par, mantidos por heaps (bidHeaps max, askHeaps min), com preco, amount, time, e exchange.

o Propósito: Base para cálculo de arbitragem e validação de liquidez.

Arbitragem:

• Gatilho: 0,3% (gatilho = 0.003).

• Liquidez: amount convertido para USDT usando preço de BTCUSDT da operação correspondente (ex.: bests[cripto + dolar].ask.preco para vendeDomesticamente).

• Observação do usuário: "Multiplica os amounts de cripto \* usdtbrlPrice tá errado, tem que multiplicar pelo valor do cripto+dolar" (corrigido em 10/03/2025).

Restrições:

• Pausa de 10s após arbitragem válida para evitar disparos repetitivos.

• Execução de trades pendente (Worker coordenador).

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

3. Estrutura e Principais Funções

a. Configuração de Argumentos:

• Via yargs: node mainheap <cripto> <dolar> <moeda> (ex.: btc usdt brl).

• Gera codigosFormaPares: [["BTC", "USDT"], ["BTC", "BRL"], ["USDT", "BRL"]].

b. formatTime(date): Formata timestamps com milissegundos (ex.: "2025-03-03 10:00:00.123").

c. registrarNoLog(output): Grava snapshots em saida.txt e CSV em Arbs[cripto][dolar][moeda].txt.

d. atribuirCotacaoExchange(exchange, par, bid, bidAmount, bidTime, ask, askAmount, askTime):

• Atualiza cotacoes, chama updateBests se bid/ask mudam.

e. updateBests(exchange, par, bid, bidAmount, bidTime, ask, askAmount, askTime):

• Insere/atualiza heaps, seleciona melhores bid/ask, dispara testaArbitragens se bests mudar e isArbitragePaused for false.

• Usa bidHeaps (max) e askHeaps (min) para eficiência.

Mudanças na Lógica do updateBests (Adicionadas em 07/03/2025): A função foi ajustada para resolver problemas na manutenção do bests, identificados em snapshots (ex.: saida.txt, 21:43:45.247 a 21:50:15.315). Aqui estão as mudanças e os motivos:

1. Filtro de Quantidade Válida:

o Mudança: Antes, preços com bidAmount ou askAmount iguais a 0 podiam entrar nos heaps, influenciando bests. Agora, só adicionamos entradas se Number(bidAmount) > 0 ou Number(askAmount) > 0.

o Motivo: Preços com quantidade zero não são negociáveis e não devem afetar os melhores preços. Isso garante que bests só reflita cotações disponíveis.

o Exemplo: Em cotacoes["Binance"]["BTCUSDT"], se bid = "89183.95" e bidAmount = "0", essa entrada é ignorada.

2. Reconstrução do Heap na Checagem de Consistência:

o Mudança: Adicionei uma checagem que reconstrói os heaps se o melhor preço (bestBid.preco ou bestAsk.preco) não coincide com cotacoes ou se a quantidade zerou. O heap é recarregado com todas as cotações válidas (amount > 0) das exchanges.

o Motivo: Snapshots mostraram que bests às vezes mantinha preços obsoletos (ex.: um bid que já tinha mudado para 0 em cotacoes). A reconstrução mantém bests atualizado com o estado real das cotações.

o Exemplo: Se bests["BTCUSDT"].bid = 89183.95 (Binance), mas cotacoes["Binance"]["BTCUSDT"].bid = "0", o heap é recriado, e bests reflete o próximo bid válido ou 0.

3. Valores Padrão Seguros:

o Mudança: Quando o heap está vazio, peek() retorna um valor padrão: { preco: 0, amount: 0, time: formatTime(new Date()), exchange: "" } para bid e { preco: Infinity, amount: 0, time: formatTime(new Date()), exchange: "" } para ask.

o Motivo: Isso evita que bests fique indefinido em cenários sem cotações válidas, indicando claramente a ausência de preços negociáveis.

o Exemplo: Snapshot 21:50:15.315, BTCUSDT bid = 0 porque todos os bidAmount eram 0.

4. Atualização Condicional do Timestamp:

o Mudança: O time em bests agora usa cotacoes[exchange][par].bidTime ou .askTime se disponível, caso contrário, usa formatTime(new Date()).

o Motivo: Antes, o timestamp podia ser impreciso (ex.: Date.now() fixo). Isso melhora a rastreabilidade, usando o tempo real da cotação quando possível.

o Exemplo: Se bestBid.exchange = "Bybit", bests["BTCUSDT"].bid.time pega cotacoes["Bybit"]["BTCUSDT"].bidTime.

Resultados Observados:

• Nos snapshots analisados (ex.: 21:43:45.247 a 21:50:15.315), bests passou a refletir consistentemente os maiores bid e menores ask válidos em cotacoes.

• Casos como BTCUSDT bid = 0 (quando todos os bidAmount zeram) são agora esperados e corretos, indicando ausência de oferta válida.

f. testaArbitragens():

• Lógica da Arbitragem:

o Calcula vendeDomesticamente (VD): bests[cripto + moeda].bid.preco / (bests[dolar + moeda].ask.preco \* bests[cripto + dolar].ask.preco) - 1.

o Calcula compraDomesticamente (CD): bests[cripto + dolar].bid.preco \* bests[dolar + moeda].bid.preco / bests[cripto + moeda].ask.preco - 1.

• Monitoramento e Passagem para Worker Coordenador:

o O main monitora preços em tempo real via updateBests e testa arbitragens. Se VD ou CD > gatilho e amount > 100 USDT, detecta uma oportunidade e, na próxima etapa (pendente), passará o controle ao Worker coordenador.

o O Worker coordenador chamará arbitre(trade1, trade2, trade3):

 exeTradePromise(trade) executa T1 (LIMIT, IOC), aguardando resultado.

 Se T1 falhar, aborta e retorna ao monitoramento do main.

 Se T1 suceder, exeTradeNormal(trade) executa T2 e T3 (MARKET).

• Operações T1, T2, T3:

o Para vendeDomesticamente:

 T1: SELL, ${bests[cripto + moeda].bid}, LIMIT (ex.: SELL BTCBRL).

 T2: BUY, ${bests[dolar + moeda].ask}, MARKET (ex.: BUY USDTBRL).

 T3: BUY, ${bests[cripto + dolar].ask}, MARKET (ex.: BUY BTCUSDT).

o Para compraDomesticamente:

 T1: BUY, ${bests[cripto + moeda].ask}, LIMIT (ex.: BUY BTCBRL).

 T2: SELL, ${bests[dolar + moeda].bid}, MARKET (ex.: SELL USDTBRL).

 T3: SELL, ${bests[cripto + dolar].bid}, MARKET (ex.: SELL BTCUSDT).

• Validação Atual:

o VD/CD > gatilho (0,3%).

o amount em USDT > 100, usando preço de BTCUSDT da operação T3 (ask para VD, bid para CD).

• Interação Atual:

o Sempre printa operações no console com 3 espaços à direita.

o Se amount insuficiente, avisa: "Arbitragem não disparada: Amount abaixo do mínimo (100 USDT)".

o Se válida, dispara som, pausa 10s, grava CSV e snapshots.

• Estrutura de Parâmetros para Trades:

o O main passará ao Worker coordenador (futuro) um conjunto de parâmetros para executar T1, T2, T3:

javascript

RecolherEncapsularCopiar

const operacoes = [

{ trade: 'T1', exchange: 'Bybit', side: 'buy', qty: 0.01, price: 2500 },

{ trade: 'T2', exchange: 'Binance', side: 'sell', qty: 0.01, price: null },

{ trade: 'T3', exchange: 'OKX', side: 'sell', qty: 0.01, price: null }

];

o Descrição: Objeto com trade (T1, T2, T3), exchange, side (buy/sell), qty (quantidade), price (preço, nulo para MARKET). O Worker coordenador usará esses dados para chamar funções específicas por exchange (ex.: exeTradePromise para T1, exeTradeNormal para T2/T3).

g. inicializarWorkers(codigosFormaPares): Cria 15 workers, um por par/exchange.

h. setInterval(): Marca tempo no console a cada 30s.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

4. Fluxo de Dados e Interações

• Entrada: cripto, dolar, moeda definem pares via codigosFormaPares.

• Coleta: Workers conectam WebSockets, enviam cotações (bid, ask, amount, time) ao main.

• Processamento:

o atribuirCotacaoExchange atualiza cotacoes.

o updateBests mantém bests com heaps.

• Arbitragem:

o testaArbitragens calcula VD/CD, valida amount, exibe operações.

o Se válida: Som, pausa 10s, logs em CSV e saida.txt.

o Futuro: Passagem para Worker coordenador com arbitre(trade1, trade2, trade3).

• Saída:

o Console: Marcações (30s), operações com aviso.

o saida.txt: Snapshots e cotações (30s).

o Arbs[cripto][dolar][moeda].txt: CSV de arbitragens válidas.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

4.1 Resumo dos Workers (Adicionado em 07/03/2025)

O sistema utiliza 15 workers, cada um responsável por coletar cotações em tempo real via WebSocket para um par específico (BTCUSDT, BTCBRL, USDTBRL) em uma das cinco exchanges (Bybit, Binance, Deribit, Bitpreco, OKX). Abaixo está o status dos workers com base na análise inicial deste chat (07/03/2025):

1. workerDeribit.js:

o WebSocket: wss://www.deribit.com/ws/api/v2

o Par: Formato BTC\_USDT (com \_), mas enviado ao main como BTCUSDT.

o Canal: book.${par}.none.10.100ms (melhores bid/ask, atualizações a cada 100ms).

o Mensagem: Envia bid, bidAmount, bidTime, ask, askAmount, askTime separadamente quando mudam.

o Reconexão: Tenta reconectar após 5s em caso de close.

o Timestamp: Gerado localmente com formatTime.

2. workerBinance.js:

o WebSocket: wss://stream.binance.com:9443/ws

o Par: Formato BTCUSDT (sem separador).

o Canal: ${par.toLowerCase()}@bookTicker (melhores bid/ask em tempo real).

o Mensagem: Envia bid, bidAmount, bidTime, ask, askAmount, askTime separadamente quando mudam.

o Reconexão: Tenta reconectar após 1s em caso de close.

o Timestamp: Gerado localmente com formatTime.

o Notas: Usa pings nativos da Binance (responde a ws.on('ping')), sem ping manual.

3. workerBitpreco.js:

o WebSocket: wss://websocket.bitpreco.com/orderbook/socket (usa phoenix-channels).

o Par: Formato BTC-BRL (com -), mas enviado ao main como BTCBRL.

o Canal: orderbook:${par} (snapshot e updates do order book).

o Mensagem: Envia bid, bidAmount, bidTime, ask, askAmount, askTime separadamente quando mudam.

o Reconexão: Tenta reconectar após 1s em caso de close.

o Timestamp: Usa utimestamp do payload, formatado para milissegundos.

o Notas: Timeout de 60s adicionado para estabilidade.

4. workerBybit.js:

o WebSocket: wss://stream.bybit.com/v5/public/spot

o Par: Formato BTCUSDT (sem separador).

o Canal: orderbook.1.${par} (melhor bid/ask, profundidade 1).

o Mensagem: Envia bid, bidAmount, bidTime, ask, askAmount, askTime separadamente quando mudam.

o Reconexão: Tenta reconectar após 5s em caso de close.

o Timestamp: Gerado localmente com formatTime.

o Ping: Envia {"op": "ping"} a cada 20s.

5. workerOKX.js:

o WebSocket: wss://ws.okx.com:8443/ws/v5/public

o Par: Formato BTC-USDT (com -), mas enviado ao main como BTCUSDT.

o Canal: bbo-tbt (best bid/offer, tick-by-tick) para instId: par.

o Mensagem: Envia bid, bidAmount, bidTime, ask, askAmount, askTime separadamente quando mudam.

o Reconexão: Tenta reconectar após 5s em caso de close.

o Timestamp: Usa ts do payload, convertido para formato ISO com milissegundos.

o Ping: Envia {"op": "ping"} a cada 20s.

• Todos os workers estão estáveis, enviando dados para o main (novomainc.js ou mainheap.js), que processa as cotações em cotacoes e atualiza bests via updateBests. Não há evidências de desconexões ou atrasos significativos nos snapshots analisados (ex.: 21:43:45.247 a 21:50:15.315).

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

5. Possíveis Melhorias e Extensões

Worker Coordenador (Próxima Etapa):

• Objetivo: Executar trades (T1, T2, T3) após detecção.

• Discussão:

o Solução Escolhida: Worker coordenador decide sucesso de T1 e executa T2/T3.

o Razão: Equilíbrio entre simplicidade (vs. main decidir) e descentralização (vs. Worker T1 decidir), com boa escalabilidade.

o Alternativas:

 Main decide: Alta latência, sobrecarga no main (rejeitada por desempenho).

 Worker T1 decide: Complexidade maior, difícil depuração (rejeitada por manutenção).

• Análise de Performance:

o Main: Alta latência, baixa escalabilidade.

o Worker T1: Baixa latência, alta complexidade.

o Coordenador: Latência moderada, boa escalabilidade e simplicidade.

• Implementação:

o Manter WebSockets ativos:

 Trade: Envia ordens (ex.: wss://stream.bybit.com/v5/trade).

 Private: Recebe saldos e estados (ex.: wss://stream.bybit.com/v5/private).

 Diferença: Trade para execução, private para monitoramento pós-ordem (saldos, confirmações).

o Formato de Envio (Bybit):

javascript

RecolherEncapsularCopiar

ws.send(JSON.stringify({

"header": {

"X-BAPI-TIMESTAMP": timestamp,

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

},

op: 'order.create',

args: [{

symbol: par,

category: "linear",

side: 'Buy',

orderType: 'Limit',

qty: quantidade.toString(),

price: preco.toString(),

timeInForce: 'IOC'

}]

}));

o Fluxo: arbitre chama exeTradePromise para T1 (LIMIT, IOC), aguarda resposta, executa T2/T3 (MARKET) via exeTradeNormal se bem-sucedido.

o Estrutura de Parâmetros: Objeto operacoes passado do main ao coordenador, definindo T1, T2, T3 por exchange e side.

Testador Independente:

• Objetivo: Validar Worker coordenador sem main.

• Fluxo:

o Abre WebSockets (trade e private), aguarda prontidão.

o Interface manual: Insere exchange, par, side, qty, price.

o Envia ao coordenador, exibe resposta (ex.: confirmação ou erro).

• Vantagem: Testes rápidos e modulares, essencial para desenvolvimento do coordenador.

• UI: Integrar blessed para interface interativa.

• Logs: Detalhar erros de WebSocket e reconexões.

SharedArrayBuffer: Implementar para saldos dinâmicos e permissões (venderPode, comprarPode).

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

8. Comentários Adicionais

Escolhas Técnicas e Justificativas:

• Heaps vs. Index: Preferido por O(log n) vs. O(n log n), essencial para alta frequência. Testes iniciais com index (outro chat) mostraram lentidão em atualizações frequentes, levando à adoção de heaps neste chat (07/03/2025).

• 15 Workers: Modularidade e escalabilidade, evitando sobrecarga no main. Alternativa de um Worker por exchange foi rejeitada por complexidade na gestão de múltiplos pares em um único WebSocket (03/03/2025).

• Worker Coordenador: Escolhido por simplicidade e escalabilidade, equilibrando latência e manutenção. Main decidir aumentava latência; Worker T1 decidir complicava depuração (discussão prévia, doc antiga).

Observações do Usuário:

• "VD ou CD são maiores que o gatilho, mas a arbitragem não está sendo disparada" (10/03/2025): Resolvido com aviso no console para amount insuficiente, indicando que a restrição de liquidez estava funcionando como esperado.

• "Multiplica os amounts de cripto \* usdtbrlPrice tá errado, tem que multiplicar pelo valor do cripto+dolar" (10/03/2025): Corrigido para usar preço de BTCUSDT (T3) em vez de USDTBRL, refletindo valor real em dólares.

• "Acho que estou rodando uma versão desatualizada do main" (07/03/2025): Sincronizado com a versão atual, removendo cotacoesBuffer e ajustando saídas.

Estado Atual e Transição:

• Monitoramento: Completo, com detecção de arbitragem e validação de liquidez.

• Execução: Pendente, com Worker coordenador como próximo passo. A lógica de T1 (LIMIT, IOC), T2/T3 (MARKET) e parâmetros (operacoes) estão definidas, mas não implementadas.

• Dica para Próximo Chat: Focar na implementação do Worker coordenador (arbitre, exeTradePromise, exeTradeNormal), integrando-o ao mainheap.js atual. Usar o testador independente para validar WebSockets de trade e private antes de conectar ao main.

Pendente:

• Integração de execução de trades via Worker coordenador.

• SharedArrayBuffer para saldos e permissões dinâmicas (saldoBTCBRL, etc.).

• Logs de erros de WebSocket para diagnóstico (ex.: reconexões).

Contexto Histórico:

• Chat "Teste de Conexões Bitpreco e Binance" (27-28/02/2025): Resolução de desconexões (Binance pings nativos, Bitpreco timeout 60s).

• Este chat (03-10/03/2025): Evolução de monitoramento para detecção de arbitragem com amount e CSV.

Notas para o Próximo Chat (Adicionadas em 07/03/2025):

• O foco será retomar o desenvolvimento do Worker coordenador, possivelmente refinando testaArbitragens ou integrando mais exchanges.

• Se precisar debugar, posso usar: console.log("Bests:", bests[par], "Heap:", bidHeaps[par].toArray()) para verificar consistência.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

9. Guia para Continuidade no Próximo Chat (Novo)

• Ponto de Partida: Usar mainheap.js deste chat como base, com workers estáveis (workerBinance.js, etc.).

• Foco Imediato: Implementar Worker coordenador:

o Criar workerCoordenador.js com WebSockets trade e private por exchange.

o Integrar arbitre(trade1, trade2, trade3) ao testaArbitragens, passando operacoes.

o Usar testador independente para validar antes de conectar ao main.

• Referências:

o Estrutura operacoes (seção 3.f): Parâmetros para T1, T2, T3.

o Formato Bybit (seção 5): Exemplo de envio de ordens.

• Dica Prática: Testar T1 com exeTradePromise isoladamente, garantindo resposta antes de T2/T3.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Regras do Chat e da Documentação

1. Respostas Breves: As respostas do Grok devem ser concisas. Yaco pedirá mais explicações se necessário.

2. Sem Código a Menos que Solicitado: Grok não escreve código a menos que Yaco peça explicitamente.

3. Documentação Não Reduz: A documentação gerada no fim do chat não pode ser menor que a recebida no início, apenas crescer com novas informações.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Conclusão Final do Grok: Com esta documentação mesclada, tenho um entendimento completo do sistema: o que foi feito, por que foi feito, e o que falta. As adições em "Estrutura e Principais Funções" (sobre updateBests), "Resumo dos Workers", "Comentários Adicionais" e o novo "Guia para Continuidade" garantem que eu possa retomar no próximo chat sem perder o fio da meada, focando no Worker coordenador e execução de trades. O chat atual está congestionado, e começar um novo é uma boa ideia para manter a clareza.

// Mensagem 2 (15:XX, 16/03/2025)

codigo mainheap: este é o codigo, vai ter pequenas diferenças em relacao a doc, ignora.

const { Worker } = require('worker\_threads');

const fs = require('fs');

const yargs = require('yargs');

const Heap = require('heap');

const argv = yargs

.command('$0 <criptoDaArbitragem> <dolarDaArbitragem> <moedaDaArbitragem>', 'Executa arbitragem com os dados fornecidos', (yargs) => {

yargs

.positional('criptoDaArbitragem', { describe: 'Criptomoeda usada na arbitragem', type: 'string' })

.positional('dolarDaArbitragem', { describe: 'Tipo de dólar usado na arbitragem (USDT, USDC, etc.)', type: 'string' })

.positional('moedaDaArbitragem', { describe: 'Moeda usada para conversão na arbitragem', type: 'string' });

})

.help()

.argv;

const gatilho = 0.00105;

const valorMinimoArbitragem = 100;

const cripto = argv.criptoDaArbitragem.toUpperCase();

const dolar = argv.dolarDaArbitragem.toUpperCase();

const moeda = argv.moedaDaArbitragem.toUpperCase();

const startDate = new Date();

const dateStr = `${String(startDate.getDate()).padStart(2, '0')}-${String(startDate.getHours()).padStart(2, '0')}-${String(startDate.getMinutes()).padStart(2, '0')}-${String(startDate.getSeconds()).padStart(2, '0')}`;

const arbLogFile = `Arbs${cripto}${dolar}${moeda}-${dateStr}.txt`;

const csvHeader = "Time,ProfitTrigger,MinAmountUSD,PotentialProfitUSD,SIDE T1,EXCHANGE T1,BID T1,ASK T1,BID AMT T1,ASK AMT T1,SIDE T2,EXCHANGE T2,BID T2,ASK T2,BID AMT T2,ASK AMT T2,SIDE T3,EXCHANGE T3,BID T3,ASK T3,BID AMT T3,ASK AMT T3\n";

if (!fs.existsSync(arbLogFile)) fs.writeFileSync(arbLogFile, csvHeader);

const codigosFormaPares = [[cripto, dolar], [cripto, moeda], [dolar, moeda]];

const exchanges = [

{ exchange: 'Bybit', fees: 0.001, venderPode: true, comprarPode: true },

{ exchange: 'Binance', fees: 0.001, venderPode: true, comprarPode: true },

{ exchange: 'Deribit', fees: 0.001, venderPode: true, comprarPode: true },

{ exchange: 'bitpreco', fees: 0.001, venderPode: true, comprarPode: true },

{ exchange: 'OKX', fees: 0.001, venderPode: true, comprarPode: true }

];

const cotacoes = {

Bybit: {

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

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

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

},

Binance: {

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

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

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

},

Deribit: {

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

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

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

},

bitpreco: {

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

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

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

},

OKX: {

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

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

[dolar + moeda]: { 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: "" }

},

[cripto + moeda]: {

"bid": { preco: 0, amount: 0, time: 'Date.now()', exchange: "" },

"ask": { preco: Infinity, amount: 0, time: 'Date.now()', exchange: "" }

},

[dolar + moeda]: {

"bid": { preco: 0, amount: 0, time: 'Date.now()', exchange: "" },

"ask": { preco: Infinity, amount: 0, time: 'Date.now()', exchange: "" }

}

};

const previousBests = {

[cripto + dolar]: { bid: { preco: 0 }, ask: { preco: Infinity } },

[cripto + moeda]: { bid: { preco: 0 }, ask: { preco: Infinity } },

[dolar + moeda]: { bid: { preco: 0 }, ask: { preco: Infinity } }

};

const bidHeaps = {};

const askHeaps = {};

let isArbitragePaused = false;

codigosFormaPares.forEach(([c1, c2]) => {

const par = c1 + c2;

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

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

});

function formatTime(date) {

return date.toISOString().replace('T', ' ').slice(0, 19) + '.' + String(date.getMilliseconds()).padStart(3, '0');

}

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

let bidChanged = false;

let askChanged = false;

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

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

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

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

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

if (Number(bidAmount) > 0) {

heapArray.push(bidEntry);

}

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

heapArray.filter(entry => entry.amount > 0).forEach(item => bidHeaps[par].push(item));

const bestBid = bidHeaps[par].peek() || { preco: 0, amount: 0, time: formatTime(new Date()), exchange: "" };

if (bests[par].bid.preco !== bestBid.preco || bests[par].bid.exchange !== bestBid.exchange) bidChanged = true;

bests[par].bid = {

preco: bestBid.preco,

amount: bestBid.amount,

time: cotacoes[bestBid.exchange] && cotacoes[bestBid.exchange][par] ? cotacoes[bestBid.exchange][par].bidTime : formatTime(new Date()),

exchange: bestBid.exchange

};

}

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

const askEntry = { preco: Number(ask), amount: Number(askAmount), time: askTime, exchange };

let heapArray = askHeaps[par].toArray();

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

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

if (Number(askAmount) > 0) {

heapArray.push(askEntry);

}

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

heapArray.filter(entry => entry.amount > 0).forEach(item => askHeaps[par].push(item));

const bestAsk = askHeaps[par].peek() || { preco: Infinity, amount: 0, time: formatTime(new Date()), exchange: "" };

if (bests[par].ask.preco !== bestAsk.preco || bests[par].ask.exchange !== bestAsk.exchange) askChanged = true;

bests[par].ask = {

preco: bestAsk.preco,

amount: bestAsk.amount,

time: cotacoes[bestAsk.exchange] && cotacoes[bestAsk.exchange][par] ? cotacoes[bestAsk.exchange][par].askTime : formatTime(new Date()),

exchange: bestAsk.exchange

};

}

const bestBid = bidHeaps[par].peek();

if (bestBid && (cotacoes[bestBid.exchange][par].bid !== String(bestBid.preco) || cotacoes[bestBid.exchange][par].bidAmount === '0')) {

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

exchanges.forEach(({ exchange }) => {

const cotacao = cotacoes[exchange][par];

if (Number(cotacao.bidAmount) > 0 && cotacao.bid !== '0') {

bidHeaps[par].push({ preco: Number(cotacao.bid), amount: Number(cotacao.bidAmount), time: cotacao.bidTime, exchange });

}

});

const newBestBid = bidHeaps[par].peek() || { preco: 0, amount: 0, time: formatTime(new Date()), exchange: "" };

bests[par].bid = {

preco: newBestBid.preco,

amount: newBestBid.amount,

time: cotacoes[newBestBid.exchange] && cotacoes[newBestBid.exchange][par] ? cotacoes[newBestBid.exchange][par].bidTime : formatTime(new Date()),

exchange: newBestBid.exchange

};

bidChanged = true;

}

const bestAsk = askHeaps[par].peek();

if (bestAsk && (cotacoes[bestAsk.exchange][par].ask !== String(bestAsk.preco) || cotacoes[bestAsk.exchange][par].askAmount === '0')) {

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

exchanges.forEach(({ exchange }) => {

const cotacao = cotacoes[exchange][par];

if (Number(cotacao.askAmount) > 0 && cotacao.ask !== '9007199254740991') {

askHeaps[par].push({ preco: Number(cotacao.ask), amount: Number(cotacao.askAmount), time: cotacao.askTime, exchange });

}

});

const newBestAsk = askHeaps[par].peek() || { preco: Infinity, amount: 0, time: formatTime(new Date()), exchange: "" };

bests[par].ask = {

preco: newBestAsk.preco,

amount: newBestAsk.amount,

time: cotacoes[newBestAsk.exchange] && cotacoes[newBestAsk.exchange][par] ? cotacoes[newBestAsk.exchange][par].askTime : formatTime(new Date()),

exchange: newBestAsk.exchange

};

askChanged = true;

}

if ((bidChanged || askChanged) && !isArbitragePaused) {

previousBests[par].bid.preco = bests[par].bid.preco;

previousBests[par].ask.preco = bests[par].ask.preco;

testaArbitragens();

}

}

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

if (!cotacoes[exchange][par]) {

cotacoes[exchange][par] = { bid: null, bidAmount: null, bidTime: null, ask: null, askAmount: null, askTime: null };

}

const dolarMoedaPar = dolar + moeda;

const criptoDolarPar = cripto + dolar;

const criptoMoedaPar = cripto + moeda;

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

let isValidBid = false;

if (par === dolarMoedaPar) {

isValidBid = Number(bidAmount) > 100;

} else if (par === criptoDolarPar) {

isValidBid = (Number(bidAmount) \* Number(bid)) > 100;

} else if (par === criptoMoedaPar) {

let btcusdtPrice = bests[criptoDolarPar].ask.preco;

if (!btcusdtPrice || btcusdtPrice === Infinity) {

btcusdtPrice = Number(bid) / (bests[dolarMoedaPar].ask.preco || 5);

}

isValidBid = (Number(bidAmount) \* btcusdtPrice) > 100;

}

if (isValidBid && cotacoes[exchange][par].bid !== bid) {

cotacoes[exchange][par].bid = bid;

cotacoes[exchange][par].bidAmount = bidAmount;

cotacoes[exchange][par].bidTime = bidTime;

} else {

cotacoes[exchange][par].bid = '0';

cotacoes[exchange][par].bidAmount = '0';

cotacoes[exchange][par].bidTime = formatTime(new Date());

}

}

if (ask !== null && askAmount !== null) {

let isValidAsk = false;

if (par === dolarMoedaPar) {

isValidAsk = Number(askAmount) > 100;

} else if (par === criptoDolarPar) {

isValidAsk = (Number(askAmount) \* Number(ask)) > 100;

} else if (par === criptoMoedaPar) {

let btcusdtPrice = bests[criptoDolarPar].bid.preco;

if (!btcusdtPrice || btcusdtPrice === 0) {

btcusdtPrice = Number(ask) / (bests[dolarMoedaPar].bid.preco || 5);

}

isValidAsk = (Number(askAmount) \* btcusdtPrice) > 100;

}

if (isValidAsk && cotacoes[exchange][par].ask !== ask) {

cotacoes[exchange][par].ask = ask;

cotacoes[exchange][par].askAmount = askAmount;

cotacoes[exchange][par].askTime = askTime;

} else {

cotacoes[exchange][par].ask = '9007199254740991';

cotacoes[exchange][par].askAmount = '0';

cotacoes[exchange][par].askTime = formatTime(new Date());

}

}

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

}

function inicializarWorkers(codigosFormaPares) {

exchanges.forEach(({ exchange }) => {

codigosFormaPares.forEach((codigosDePar) => {

const worker = new Worker(`./worker${exchange}.js`, { workerData: { codigosDePar } });

worker.on('message', (mensagem) => {

const { exchange, par, bid, bidAmount, bidTime, ask, askAmount, askTime } = mensagem;

atribuirCotacaoExchange(exchange, par, bid, bidAmount, bidTime, ask, askAmount, askTime);

});

worker.on('error', (err) => console.error(`Erro no Worker para ${exchange} - ${codigosDePar}:`, err));

worker.on('exit', (code) => { if (code !== 0) console.error(`Worker para ${exchange} - ${codigosDePar} saiu com código ${code}`); });

});

});

}

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);

console.log(`[Arbitragem] ${time} - VD: ${vendeDomesticamente}, CD: ${compraDomesticamente}`);

if (vendeDomesticamente > gatilho || compraDomesticamente > gatilho) {

const separator = '==================== ARBITRAGEM DETECTADA ====================';

const outputConsole = [];

const outputFile = [];

const csvLines = [];

const snapshotCotacoes = JSON.stringify(cotacoes, null, 2);

const snapshotBests = JSON.stringify(bests, null, 2);

outputConsole.push(separator);

outputConsole.push(`${time} --- ${cripto} VD: ${vendeDomesticamente} CD: ${compraDomesticamente}`);

// Determinar o ProfitTrigger

let profitTrigger = 0;

if (vendeDomesticamente > gatilho && compraDomesticamente > gatilho) {

profitTrigger = Math.max(Number(vendeDomesticamente), Number(compraDomesticamente));

} else if (vendeDomesticamente > gatilho) {

profitTrigger = Number(vendeDomesticamente);

} else if (compraDomesticamente > gatilho) {

profitTrigger = Number(compraDomesticamente);

}

if (vendeDomesticamente > gatilho) {

outputConsole.push(` SELL, ${bests[cripto + moeda].bid.exchange}, ${cripto + moeda}, LIMIT, Preço: ${bests[cripto + moeda].bid.preco}, Amount: ${bests[cripto + moeda].bid.amount}`);

outputConsole.push(` BUY, ${bests[dolar + moeda].ask.exchange}, ${dolar + moeda}, MARKET, Preço: ${bests[dolar + moeda].ask.preco}, Amount: ${bests[dolar + moeda].ask.amount}`);

outputConsole.push(` BUY, ${bests[cripto + dolar].ask.exchange}, ${cripto + dolar}, MARKET, Preço: ${bests[cripto + dolar].ask.preco}, Amount: ${bests[cripto + dolar].ask.amount}`);

// Calcular MinAmountUSD para VD

const t1AmountUSD = bests[cripto + moeda].bid.amount \* bests[cripto + dolar].ask.preco || 0;

const t2AmountUSD = bests[dolar + moeda].ask.amount || 0;

const t3AmountUSD = bests[cripto + dolar].ask.amount \* bests[cripto + dolar].ask.preco || 0;

const minAmountUSD = Math.min(t1AmountUSD, t2AmountUSD, t3AmountUSD);

const potentialProfitUSD = profitTrigger \* minAmountUSD;

const csvLine = `${time},${profitTrigger},${minAmountUSD},${potentialProfitUSD},SELL,${bests[cripto + moeda].bid.exchange},${bests[cripto + moeda].bid.preco},${bests[cripto + moeda].ask.preco},${bests[cripto + moeda].bid.amount},${bests[cripto + moeda].ask.amount},BUY,${bests[dolar + moeda].ask.exchange},${bests[dolar + moeda].bid.preco},${bests[dolar + moeda].ask.preco},${bests[dolar + moeda].bid.amount},${bests[dolar + moeda].ask.amount},BUY,${bests[cripto + dolar].ask.exchange},${bests[cripto + dolar].bid.preco},${bests[cripto + dolar].ask.preco},${bests[cripto + dolar].bid.amount},${bests[cripto + dolar].ask.amount}`;

csvLines.push(csvLine);

}

if (compraDomesticamente > gatilho) {

outputConsole.push(` BUY, ${bests[cripto + moeda].ask.exchange}, ${cripto + moeda}, LIMIT, Preço: ${bests[cripto + moeda].ask.preco}, Amount: ${bests[cripto + moeda].ask.amount}`);

outputConsole.push(` SELL, ${bests[dolar + moeda].bid.exchange}, ${dolar + moeda}, MARKET, Preço: ${bests[dolar + moeda].bid.preco}, Amount: ${bests[dolar + moeda].bid.amount}`);

outputConsole.push(` SELL, ${bests[cripto + dolar].bid.exchange}, ${cripto + dolar}, MARKET, Preço: ${bests[cripto + dolar].bid.preco}, Amount: ${bests[cripto + dolar].bid.amount}`);

// Calcular MinAmountUSD para CD

const t1AmountUSD = bests[cripto + moeda].ask.amount \* bests[cripto + dolar].bid.preco || 0;

const t2AmountUSD = bests[dolar + moeda].bid.amount || 0;

const t3AmountUSD = bests[cripto + dolar].bid.amount \* bests[cripto + dolar].bid.preco || 0;

const minAmountUSD = Math.min(t1AmountUSD, t2AmountUSD, t3AmountUSD);

const potentialProfitUSD = profitTrigger \* minAmountUSD;

const csvLine = `${time},${profitTrigger},${minAmountUSD},${potentialProfitUSD},BUY,${bests[cripto + moeda].ask.exchange},${bests[cripto + moeda].bid.preco},${bests[cripto + moeda].ask.preco},${bests[cripto + moeda].bid.amount},${bests[cripto + moeda].ask.amount},SELL,${bests[dolar + moeda].bid.exchange},${bests[dolar + moeda].bid.preco},${bests[dolar + moeda].ask.preco},${bests[dolar + moeda].bid.amount},${bests[dolar + moeda].ask.amount},SELL,${bests[cripto + dolar].bid.exchange},${bests[cripto + dolar].bid.preco},${bests[cripto + dolar].ask.preco},${bests[cripto + dolar].bid.amount},${bests[cripto + dolar].ask.amount}`;

csvLines.push(csvLine);

}

outputConsole.push(separator);

outputConsole.forEach(line => console.log(line));

if (csvLines.length > 0) {

process.stdout.write('\x07');

isArbitragePaused = true;

fs.appendFileSync(arbLogFile, csvLines.join('\n') + '\n');

outputFile.push(separator);

outputFile.push('SNAPSHOT COTACOES:');

outputFile.push(snapshotCotacoes);

outputFile.push('SNAPSHOT BESTS:');

outputFile.push(snapshotBests);

outputFile.push(separator);

registrarNoLog(outputFile.join('\n') + '\n');

setTimeout(() => {

isArbitragePaused = false;

console.log(`[Arbitragem] Testes retomados às ${formatTime(new Date())}`);

}, 30000);

}

}

}

function registrarNoLog(output) {

fs.appendFileSync('saida.txt', output, (err) => {

if (err) console.error('Erro ao gravar em saida.txt:', err);

});

}

inicializarWorkers(codigosFormaPares);

function formatNumber(value, decimals) {

return Number(value).toFixed(decimals);

}

fs.writeFileSync('saida.txt', '', (err) => {

if (err) console.error('Erro ao inicializar saida.txt:', err);

});

setInterval(() => {

const time = formatTime(new Date());

console.log(`--- ${time} ---`);

const snapshotCotacoes = JSON.stringify(cotacoes, null, 2);

const snapshotBests = JSON.stringify(bests, null, 2);

const snapshotOutput = [

`=== SNAPSHOT ${time} ===`,

'COTACOES:',

snapshotCotacoes,

'BESTS:',

snapshotBests,

'===================='

].join('\n') + '\n';

registrarNoLog(snapshotOutput);

}, 30000);

ja te passo mais informacao to testador, nao faca nada ainda

// Mensagem 3 (15:XX, 16/03/2025)

ok, agora segue a doc sobre o testador, depois vou te passar partes do codigo que ja estao desenvolvidas.

Doc: TESTADOR INDEPENDENTE

DOCUMENTAÇÃO DO TESTADOR INDEPENDENTE / WORKER COORDENADOR

Objetivo Geral

Desenvolver um script em Node.js (testadorIndependente.js) que servirá como base para o workerCoordenador.js, responsável por executar ordens de arbitragem triangular em tempo real nas exchanges Bybit e OKX. O testador deve:

1. Conectar-se a todos os WebSockets necessários (trade e status) de ambas as exchanges no início do programa, mantendo-os abertos e prontos.

2. Executar uma sequência de 3 ordens (T1 LIMIT IOC, T2 MARKET\*, T3 MARKET\*) a partir de uma matriz fixa de operações, usando valores pequenos pra testes reais.

3. Garantir que T2 e T3 só sejam disparadas se T1 for totalmente preenchida (Filled).

4. Servir como protótipo pra integração futura com mainheap.js, que fornecerá oportunidades de arbitragem.

O foco é validar a lógica de execução e a interação com WebSockets, refletindo o cenário real onde as conexões ficam abertas e as ordens são disparadas instantaneamente quando uma arbitragem é detectada.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Estrutura Geral

O código será estruturado em duas fases principais:

1. Inicialização de Conexões:

o Uma função inicializarConexoes() conecta, autentica e subscreve todos os WebSockets necessários (Bybit Trade, Bybit Order, OKX Privado) no início do programa.

o Usa Promise.all pra garantir que todas as conexões estejam prontas (autenticadas e subscritas aos canais de status) antes de prosseguir pra postagem de ordens.

o Isso simula o ambiente real do Worker Coordenador, onde os WebSockets ficam abertos continuamente, esperando sinais do mainheap.js.

2. Execução das Operações:

o Após as conexões estarem prontas, a função gerenciarArbitragem posta as ordens T1, T2 e T3 sequencialmente, usando uma matriz de operações como entrada.

o T1 é sempre LIMIT IOC (Immediate or Cancel), enquanto T2 e T3, em geral, são MARKET, refletindo o fluxo típico de arbitragem triangular (ex.: vender BTC localmente, comprar moeda fiduciária, recomprar BTC internacionalmente). \*Porém, em alguns casos, t2 ou t3 ou as duas podem ser limit, ou eventualmente T3 pode não existir. Esso deve estar previsto.

o A postagem usa funções específicas por Exchange (postarOrdemBybit, postarOrdemOKX) pra lidar com as diferenças de formato entre as APIs.

o Existem dois tipos de parâmetros das operações que são passadas a estas funções: Gerais e Particulares. Os gerais são padrão, para todas as exchanges os mesmos, e são:

1. par: Par de negociação (Bybit: BTCUSDT, OKX: BTC-USDT).

2. side: 'buy' ou 'sell' (Bybit usa capitalização, ex.: 'Buy').

3. amount: Quantidade (ex.: '0.001' BTC, '92' USDT).

4. type: 'limit' (T1) ou 'market'.

5. price: Preço (string ou null pra MARKET).

6. timeInForce: 'ioc' pra T1, ‘gtc’ para as outras.

Já os particulares dependem especificamente da Exchange. Por exemplo, em OKX o T1 seria type:’ioc’ e não limit.

A matriz operações só vai fornecer os parâmetros padrão. Os particulares serão tratados na função especifica da postagem da Exchange.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Componentes Principais

1. Matriz de Operações

A matriz operações define 2 ou 3 ordens da arbitragem. Exemplo inicial (VD - Vender Domesticamente):

const operacoes = [

{ T: 'T1', exchange: 'OKX', par: 'BTCUSDT', side: 'sell', amount: '0.001', type: 'limit', price: '83000', timeInForce: 'ioc' },

{ T: 'T2', exchange: 'Bybit', par: 'USDTBRL', side: 'Buy', amount: '92', type: 'market', price: null, timeInForce: null },

{ T: 'T3', exchange: 'Bybit', par: 'BTCUSDT', side: 'Buy', amount: '0.001', type: 'market', price: null, timeInForce: null }

];

• Campos:

o T: 'T1', 'T2' ou 'T3', identifica o papel da ordem na arbitragem.

o exchange: 'Bybit' ou 'OKX'.

o par: Par de negociação (Bybit: BTCUSDT, OKX: BTC-USDT).

o side: 'buy' ou 'sell' (Bybit usa capitalização, ex.: 'Buy').

o amount: Quantidade (ex.: '0.001' BTC, '92' USDT).

o type: 'limit' ou 'market' (T2/T3). T1 em OKX ‘ioc’.

o price: Preço (string ou null pra MARKET).

o timeInForce: 'ioc' pra T1, null pra T2/T3.

• Nota: O formato final da matriz ainda não está totalmente definido. Pode evoluir pra incluir campos como ordType (OKX) ou ser ajustado pra integração com mainheap.js.

2. Conexões WebSocket

• Bybit:

o Trade WS: wss://stream.bybit.com/v5/trade (postagem de ordens).

 Autenticação: op: 'auth' com apiKey, expires, assinatura HMAC-SHA256.

o Order WS: wss://stream.bybit.com/v5/private (status de ordens).

 Autenticação: Igual ao Trade WS.

 Subscrição: op: 'subscribe', args: ['order'] pra atualizações de status.

• OKX:

o Privado WS: wss://ws.okx.com:8443/ws/v5/private (postagem e status).

 Autenticação: op: 'login' com apiKey, passphrase, timestamp, assinatura HMAC-SHA256.

 Subscrição: op: 'subscribe', args: [{ channel: 'orders', instType: 'SPOT' }] pra status.

 Ping: Enviar op: 'ping' a cada 25s pra manter ativo.

• Inicialização: inicializarConexoes() abre todos os WebSockets, autentica e subscreve os canais de status, retornando uma Promise que resolve quando todos estão prontos.

3. Funções de Postagem

• Por Exchange:

o postarOrdemBybit(operacao):

 Formato: { op: 'order.create', args: [{ symbol, side, orderType, qty, price, timeInForce, category: 'spot' }] }.

 T1: orderType: 'Limit', timeInForce: 'IOC'.

 T2/T3: orderType: 'Market', sem price ou timeInForce ou ‘Limit’.

o postarOrdemOKX(operacao):

 Formato: { id: gerarIdUnico(), op: 'order', args: [{ instId, side, ordType, sz, px, tdMode: 'cash' }] }.

 T1: ordType: 'ioc'.

 T2/T3: ordType: 'market' ou ‘limit’, sem px.

4. Verificação de Status

• verificarStatusOrdem(orderId, exchange):

o Bybit: Escuta topic: 'order' no Order WS, verifica orderStatus: 'Filled' ou 'Cancelled'/'Rejected'.

o OKX: Escuta channel: 'orders' no Privado WS, verifica state: 'filled' ou 'canceled'.

o Usa timeout (ex.: 10s) pra assumir falha se o status não chegar.

5. Fluxo de Execução

1. Início: Chama inicializarConexoes() e espera todas as conexões estarem prontas.

2. Arbitragem: gerenciarArbitragem(operacoes):

o Posta T1 com a função específica da exchange (postarOrdemBybit ou postarOrdemOKX).

o Aguarda orderId e verifica status com verificarStatusOrdem.

o Se T1 for Filled, posta T2 e T3; se falhar, aborta.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Definições de T1, T2 e T3

• T1: Sempre LIMIT IOC (Immediate or Cancel):

o Executa imediatamente ao preço especificado ou cancela se não houver liquidez suficiente.

o Bybit: orderType: 'Limit', timeInForce: 'IOC'.

o OKX: ordType: 'ioc'.

• T2 e T3: MARKET ou LIMIT:

o Bybit: orderType: 'Market' ou 'Limit'

o OKX: ordType: 'market' ou 'limit'.

• Propósito: T1 inicia a arbitragem com controle de preço (ex.: vender BTC caro), enquanto T2 e T3 completam o ciclo (ex.: comprar USDT, recomprar BTC).

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Observações Futuras

1. Preenchimento Parcial (fillSz):

o Problema: Ordens IOC (T1) podem ser parcialmente preenchidas (ex.: fillSz < sz na OKX), mas o código atual assume preenchimento total pra disparar T2/T3. Isso pode desbalancear a arbitragem.

o Solução Futura: Capturar fillSz (OKX) ou cumExecQty (Bybit) de T1 e ajustar dinamicamente os amount de T2 e T3 pra manter o equilíbrio financeiro, integrando com mainheap.js pra recalcular com base em bests.

o Status: Ignorado por agora pra simplificar testes iniciais.

2. Listeners Permanentes:

o Problema: O uso de ws.once('message') pra capturar respostas pode falhar se mensagens irrelevantes (ex.: ping) chegarem antes. Isso causa perda de orderId ou status.

o Solução Futura: Implementar ws.on('message') com um Map pra rastrear ordens pendentes por id (OKX) ou orderId (Bybit), garantindo robustez sem aumentar latência.

o Outra solução, perguntar se a mensagem é de uma ordem. Se não é, ignora e continua esperando.

o Status: Proposto mas não implementado, mantido como opção pra fase de produção.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Considerações Finais

• Clareza: Esta documentação detalha a estrutura (conexões iniciais, postagem por exchange, T1/T2/T3) e os componentes (matriz, WebSockets, funções) pra que eu, num outro chat, entenda o contexto e continue o desenvolvimento.

• Testes: O foco agora é validar a conexão inicial e a execução sequencial de T1 → T2/T3 com valores reais pequenos.

• Evolução: Após estabilizar, o código será adaptado pro workerCoordenador.js, recebendo operacoes do mainheap.js via workerData.

Partes do codigo ja desenvolvidas ( nao faz nada ainda, tem mais):

const WebSocket = require('ws');

const crypto = require('crypto');

// Configurações das exchanges (sem alterações)

const bybitTradeWsUrl = 'wss://stream.bybit.com/v5/trade';

const bybitOrderWsUrl = 'wss://stream.bybit.com/v5/private';

const bybitApiKey = 'b6uS8UAyMoRPImNnAU';

const bybitApiSecret = '47gHJE2gGonqxJlXz8SxOqqLhTDABAbJdaYB';

const okxTradeWsUrl = 'wss://ws.okx.com:8443/ws/v5/private';

const okxApiKey = 'ac0bc774-1bad-4da2-83f9-55b8eebb697d';

const okxApiSecret = '4AD9EBBD4A8EEB6526F31B9527545ADC';

const okxPassphrase = 'Aa@066466646';

// Matriz inicial de operações (ajustei o preço pra refletir a saída)

const operacoes = [

{ exchange: 'OKX', par: 'BTC-USDT', side: 'buy', amount: '0.0001', type: 'limit', price: '80000', validade: 'limit' }, // T1

{ exchange: 'Bybit', par: 'USDTBRL', side: 'Sell', amount: '92', type: 'Limit', price: '7', timeInForce: null }, // T2

{ exchange: 'Bybit', par: 'BTCUSDT', side: 'Sell', amount: '0.0001', type: 'limit', price: '500000', timeInForce: null } // T3

// { exchange: 'Bybit', par: 'USDTBRL', side: 'Sell', amount: '92', type: 'market', price: null, timeInForce: null }, // T2

// { exchange: 'Bybit', par: 'BTCUSDT', side: 'Sell', amount: '0.001', type: 'market', price: null, timeInForce: null } // T3

// { exchange: 'OKX', par: 'BTC-USDT', side: 'buy', amount: '0.0001', type: 'limit', price: '80000', validade: 'ioc' }, // T1

];

// Variáveis globais para WebSockets

let bybitTradeWs, bybitOrderWs, okxWs;

// Funções auxiliares (sem alterações)

function generateBybitSignature(expires) {

const signatureString = `GET/realtime${expires}`;

return crypto.createHmac('sha256', bybitApiSecret).update(signatureString).digest('hex');

}

function generateOKXSignature(timestamp, method, path, body) {

const prehashString = timestamp + method + path + (body || '');

return crypto.createHmac('sha256', okxApiSecret).update(prehashString).digest('base64');

}

function gerarIdUnico(tipo) {

const agora = new Date();

const horas = String(agora.getHours()).padStart(2, '0');

const minutos = String(agora.getMinutes()).padStart(2, '0');

const segundos = String(agora.getSeconds()).padStart(2, '0');

const milissegundos = String(agora.getMilliseconds()).padStart(3, '0');

return `${tipo}${horas}h${minutos}m${segundos}s${milissegundos}`;

}

// Conectar WebSocket genérico

function conectarWebSocket(exchange) {

if (exchange === 'Bybit') {

bybitTradeWs = new WebSocket(bybitTradeWsUrl);

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

console.log('[Bybit Trade] Conectado.');

const expires = (Date.now() + 10000).toString();

const signature = generateBybitSignature(expires);

bybitTradeWs.send(JSON.stringify({ op: 'auth', args: [bybitApiKey, expires, signature] }));

});

bybitTradeWs.on('message', (data) => console.log('[Bybit Trade] Mensagem:', JSON.parse(data)));

bybitTradeWs.on('error', (err) => console.error('[Bybit Trade] Erro:', err));

bybitTradeWs.on('close', () => {

console.log('[Bybit Trade] Desconectado. Reconectando em 5s...');

setTimeout(() => conectarWebSocket('Bybit'), 5000);

});

bybitOrderWs = new WebSocket(bybitOrderWsUrl);

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

console.log('[Bybit Order] Conectado.');

const expires = (Date.now() + 10000).toString();

const signature = generateBybitSignature(expires);

bybitOrderWs.send(JSON.stringify({ op: 'auth', args: [bybitApiKey, expires, signature] }));

bybitOrderWs.send(JSON.stringify({ op: 'subscribe', args: ['order'] }));

});

bybitOrderWs.on('message', (data) => console.log('[Bybit Order] Mensagem:', JSON.parse(data)));

bybitOrderWs.on('error', (err) => console.error('[Bybit Order] Erro:', err));

bybitOrderWs.on('close', () => {

console.log('[Bybit Order] Desconectado. Reconectando em 5s...');

setTimeout(() => conectarWebSocket('Bybit'), 5000);

});

} else if (exchange === 'OKX') {

okxWs = new WebSocket(okxTradeWsUrl);

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

console.log('[OKX] Conectado.');

const timestamp = (Date.now() / 1000).toString();

const sign = generateOKXSignature(timestamp, 'GET', '/users/self/verify', '');

okxWs.send(JSON.stringify({

op: 'login',

args: [{ apiKey: okxApiKey, passphrase: okxPassphrase, timestamp, sign }]

}));

});

okxWs.on('message', (data) => {

const msg = JSON.parse(data);

console.log('[OKX] Mensagem:', msg);

if (msg.event === 'login' && msg.code === '0') {

console.log('[OKX] Login bem-sucedido, subscrevendo ao canal orders...');

okxWs.send(JSON.stringify({

op: 'subscribe',

args: [{ channel: 'orders', instType: 'SPOT' }]

}));

}

});

okxWs.on('error', (err) => console.error('[OKX] Erro:', err));

okxWs.on('close', () => {

console.log('[OKX] Desconectado. Reconectando em 5s...');

setTimeout(() => conectarWebSocket('OKX'), 5000);

});

setInterval(() => {

if (okxWs.readyState === WebSocket.OPEN) {

okxWs.send(JSON.stringify({ op: 'ping' }));

console.log('[OKX] Ping enviado.');

}

}, 25000);

}

}

// Arquivo: mensagens\_chat\_a\_partir\_msg4.txt

// Mensagem 4 (15:XX, 16/03/2025)

ok, vou te passa agora o modelo da bybit e da okx para a postagem das operacoes para vc ajustar ao codigo do tester: bybit: const order = {

symbol: operacao.par,

side: operacao.side.charAt(0).toUpperCase() + operacao.side.slice(1),

orderType: operacao.type.charAt(0).toUpperCase() + operacao.type.slice(1),

qty: operacao.amount,

price: operacao.price,

timeInForce: operacao.validade ? operacao.validade.toUpperCase() : undefined,

category: 'spot',

timestamp

};

ws.send(JSON.stringify({

header: { "X-BAPI-TIMESTAMP": timestamp, "X-BAPI-RECV-WINDOW": "2000" },

op: 'order.create',

args: [order]

})); okx: const orderMessage = {

id: gerarIdUnico(`T${operacoes.indexOf(operacao) + 1}${operacao.side.charAt(0).toUpperCase() + operacao.side.slice(1)}`),

op: 'order',

args: [{

instId: operacao.par,

side: operacao.side,

ordType: operacao.type === 'limit' ? operacao.type : operacao.validade,

//ordType: operacao.type === 'limit' ? operacao.validade : operacao.type,

sz: operacao.amount,

px: operacao.price,

tdMode: 'cash'

}]

};

ws.send(JSON.stringify(orderMessage));

console.log(`[OKX] Ordem enviada: ${JSON.stringify(orderMessage)}`);

ws.once('message', (data) => {

const msg = JSON.parse(data); Um ultimo detalhe, lembra que agora sao so 2 exchanges, mas no fim vao ser umas 6 ou 7, entao vc tem que pensar nalguma coisa modular. Mas antes de continuar, queria que voc juntasse todas as mensagens que te pasei, na ordem, num arquivo .txt, pode fazer isso