Evoluiu em 28/04/2025 para Documentacao Geral Arbis 28042025 – Antes Integracao.

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:

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.

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