**Documentação do Programa mainEstudoLiquidezMais e Workers**

**1. Visão Geral**

Este projeto consiste em um sistema principal (mainEstudoLiquidezMais) que monitora a liquidez (quantidades ofertadas) de pares de criptomoedas em tempo real, utilizando múltiplos workers para se conectar a diferentes exchanges (Binance, Bybit, OKX, bitpreco) via WebSockets ou canais Phoenix. O objetivo é coletar e registrar as quantidades (bid e ask) dos melhores preços de cada par em cada exchange, gerando um arquivo de log e exibindo os dados na tela a cada 20 segundos. O foco está nos pares fixos BTC-USDT, BTC-BRL e USDT-BRL, mas o sistema é dinâmico, permitindo ajustes para outras moedas ou exchanges.

**2. Estrutura do Programa**

**2.1. mainEstudoLiquidezMais**

* **Propósito**: Gerenciar a inicialização dos workers, processar os dados recebidos (quantidades bid/ask), armazená-los em um buffer, e registrar os resultados em um arquivo .txt e na saída do console a cada 20 segundos.
* **Entrada**: Não utiliza parâmetros de linha de comando; define pares fixos (BTC, USDT, BRL) no código para criar os pares cripto+dolar (BTCUSDT), dolar+moeda (USDTBRL) e cripto+moeda (BTCBRL).
* **Saída**:
  + Arquivo liquidez\_BTC\_USDT\_BRL.txt no formato CSV, com colunas para hora e bid/ask de cada par por exchange.
  + Saída na tela formatada, com cores verdes para Binance e OKX, mostrando as quantidades de BTC (4 decimais) e USDT (0 decimais).
* **Funcionamento**:
  + Mantém um buffer (cotacoesBuffer) para armazenar as quantidades de cada par por exchange.
  + Cria workers para cada combinação de exchange e par, passando codigosDePar (ex.: [BTC, USDT]).
  + Processa mensagens dos workers, atualiza o buffer, e registra os dados periodicamente.
  + Valida pares para bitpreco, aceitando apenas aqueles terminando em BRL.

**2.2. Workers (Binance, Bybit, OKX, bitpreco)**

* **Propósito**: Cada worker se conecta a uma exchange específica via WebSocket ou canal Phoenix, monitora um único par de criptomoedas (recebido via codigosDePar), captura as quantidades (bid/ask) do melhor preço no livro de ordens, e envia os dados pro main.
* **Entrada**: Recebe codigosDePar (um array com dois códigos, ex.: [BTC, BRL]) do main via workerData.
* **Saída**: Envia um objeto { exchange, par, bid, bidAmount, ask, askAmount } pro main, onde bid e ask são as quantidades (liquidez), e bidAmount/askAmount duplicam isso por consistência.
* **Funcionamento**:
  + Cria o par usando codigosDePar[0] + constSeparadorParaFormarPar + codigosDePar[1], com constSeparadorParaFormarPar específico pra cada exchange (vazio pra Binance/Bybit/bitpreco, "-" pra OKX).
  + Estabelece uma conexão WebSocket (Binance, Bybit, OKX) ou canal Phoenix (bitpreco) pro par.
  + Monitora eventos de atualização do livro de ordens (snapshot/update) e envia as quantidades ao main.
  + Inclui ping/pong periódico (a cada 20 segundos) pra evitar desconexões.
  + Lida com erros, reconexões automáticas e encerramento.

**2.2.1. Worker Binance**

* **Exchange**: Binance.
* **Conexão**: WebSocket via wss://stream.binance.com:9443/ws/{par.toLowerCase()}@bookTicker.
* **Formato do Par**: BTCUSDT, BTCBRL, USDTBRL (sem separador).
* **Dados**: Captura bidAmount (B) e askAmount (A) do stream @bookTicker, ignorando preços. Mais detalhes na documentação oficial: [Binance WebSocket Streams - Individual Symbol Book Ticker Streams](https://developers.binance.com/docs/binance-spot-api-docs/web-socket-streams#individual-symbol-book-ticker-streams).

**2.2.2. Worker Bybit**

* **Exchange**: Bybit.
* **Conexão**: WebSocket via wss://stream.bybit.com/v5/public/spot, inscrito em orderbook.1.{par}.
* **Formato do Par**: BTCUSDT, BTCBRL, USDTBRL (sem separador).
* **Dados**: Captura bidAmount e askAmount do primeiro nível do orderbook, ignorando preços. Mais informações na documentação oficial: [Bybit WebSocket API - Public Orderbook](https://bybit-exchange.github.io/docs/v5/websocket/public/orderbook).

**2.2.3. Worker OKX**

* **Exchange**: OKX.
* **Conexão**: WebSocket via wss://ws.okx.com:8443/ws/v5/public, inscrito em bbo-tbt (nível 1).
* **Formato do Par**: BTC-USDT, BTC-BRL, USDT-BRL (com "-").
* **Dados**: Captura bidAmount e askAmount do primeiro nível do orderbook, ignorando preços. Mais informações na documentação oficial: [OKX WebSocket API - Order Book Channel](https://www.okx.com/docs-v5/en/?shell#order-book-trading-market-data-ws-all-trades-channel).

**2.2.4. Worker bitpreco**

* **Exchange**: bitpreco.
* **Conexão**: Canal Phoenix via wss://websocket.bitpreco.com/orderbook/socket, inscrito em orderbook:{par}.
* **Formato do Par**: BTC-BRL, USDT-BRL (com "-"), apenas pares terminando em BRL.
* **Dados**: Captura bidAmount e askAmount (e opcionalmente bidPrice, askPrice) do primeiro nível de bids e asks, ignorando o resto do orderbook.

**3. Detalhes Técnicos**

**3.1. mainEstudoLiquidezMais**

* **Tecnologias**: Node.js, worker\_threads (pra paralelismo), fs (pra manipulação de arquivos).
* **Buffer**: Usa cotacoesBuffer como objeto para armazenar dados temporários, com chaves como exchange[par][bid/ask/time].
* **Saída no Arquivo**: Gera liquidez\_BTC\_USDT\_BRL.txt no formato CSV, com cabeçalho fixo e linhas atualizadas a cada 20 segundos.
* **Saída na Tela**: Usa códigos ANSI pra cores (verde pra Binance/OKX), formatando quantidades (4 decimais pra BTC, 0 pra USDT).
* **Validações**: Filtra pares inválidos pra bitpreco, aceitando apenas BTC-BRL e USDT-BRL.

**3.2. Workers**

* **Tecnologias**: Node.js, worker\_threads, ws (WebSocket pra Binance/Bybit/OKX), phoenix-channels (pra bitpreco).
* **Conexão**: Cada worker usa o protocolo específico da exchange (WebSocket ou Phoenix) pra se conectar e subscrever ao canal do par.
* **Ping/Pong**: Envia pings periódicos (20s) pra manter a conexão ativa, com reconexão automática em caso de falhas.
* **Erros**: Trata erros com logs e reconexão, garantindo robustez.
* **Dinamismo**: Os workers são genéricos, recebendo codigosDePar do main pra criar o par dinamicamente.

**4. Considerações e Notas**

* **Escalabilidade**: O sistema é escalável pra outras exchanges ou pares, bastando ajustar o main pra incluir novas exchanges/workers e os workers pra suportar novos formatos de par/conexão.
* **Performance**: Usa worker\_threads pra processar dados em paralelo, minimizando bloqueios no main.
* **Dependências**: Requer ws e phoenix-channels instalados via npm/yarn.
* **Manutenção**: O main e os workers são modulares, permitindo ajustes fáceis (ex.: intervalo de atualização, formato de saída, novas exchanges).
* **Debug**: Inclui logs de conexão, erros e mensagens recebidas, facilitando a depuração.