价格计算器

背景

1. 区块链上的数字资产用token形式存储，这些token之间可以通过去中心化交易所（DEX）相互兑换（swap）。
2. 去中心化交易所（DEX）是通过流动性池（pool）来完成token之间的swap操作的。一个pool里面存有两种不同类型的token。
3. 用户调用swap函数时，将一种token转移到pool中，DEX将根据智能合约代码中定义的计算方法，pool将另一种token转移给用户。
4. 目前有两类主流DEX的swap计算方法：uniswap v2、uniswap v3。其他DEX的计算都是直接照抄或者稍加修改的。

案例：
https://etherscan.io/tx/0xad7812f34af8392b5aa7154e774842fc077ec52ab9ee71b9deba78f670880e37#eventlog

目标

有两种token：A和B，如何将A换成B可以得到最多数量的B

输入：token A地址，token A数量，token B地址

输出：token B的最多数量和swap路径（经过了哪些pool）

要求提供两种计算方法选择：
1.基于公式：不用把合约跑起来，直接套公式算（速度快，适用型强）
2.基于复现：将合约计算过程在本地复现跑一遍（准确率高）

项目内容概况

入口文件 `main.rs`

输入示例

```
let token_a_str = "0x9Cb2f26A23b8d89973F08c957C4d7cdf75CD341c";
let token_b_str = "0xdAC17F958D2ee523a2206206994597C13D831ec7";
let token_a_amount: u64 = 100;
```

调用计算

```
let (all_paths, all_dex_paths )= get_best_path(client, chain_id, token_a_str, token_b_str, U256::from(token_a_amo

if all_paths.is_empty() {
    println!("未找到路径");
    return;
}

println!("所有路径:");
for (i, ((path, amount), dex)) in all_paths.iter().zip(all_dex_paths.iter()).enumerate() {
    println!("Index: {}", i);
    println!("Path: {:?}", path);
    println!("Dex: {:?}", dex);
    println!("Amount_out: {}", amount);
}
for (path, amount) in all_paths {
```

```
    println!("Path: {:?}, Amount_out: {}", path, amount);
}
```

入口函数`get_best_path`：寻找最优路径的算法已经写好，不用修改

关键函数`get_best_pool`：通过不同的计算方法计算在DEX不同的pool中swap token的结果（已完成部分，待完成）

基于公式的方法

1.uniswap v2类计算公式已完成（uniswap_v2，sushiswap）

```
fn get_amount_out_v2(amount_in: U256, reserve_in: U256, reserve_out: U256) -> U256 {
    // 检查输入和储备是否有效
    assert!(
        !amount_in.is_zero(),
        "UniswapV2Library: INSUFFICIENT_INPUT_AMOUNT"
    );
    assert!(
        !reserve_in.is_zero() && !reserve_out.is_zero(),
        "UniswapV2Library: INSUFFICIENT_LIQUIDITY"
    );

    // 计算有效输入金额（扣除手续费 0.3%）
    let fee_multiplier = U256::from(997); // 997 表示 0.3% 的手续费
    let fee_base = U256::from(1000); // 1000 是手续费基数
    let amount_in_with_fee = amount_in * fee_multiplier;

    // 计算分子 numerator = amountInWithFee * reserveOut
    let numerator = amount_in_with_fee * reserve_out;

    // 计算分母 denominator = reserveIn * 1000 + amountInWithFee
    let denominator = reserve_in * fee_base + amount_in_with_fee;

    // 计算最终的输出 amountOut = numerator / denominator
    numerator / denominator
}
```

2.**uniswap v3类未完成（Todo）**

基于复现的方法

现状：uniswap_v3，curve_v1，curve_v2，curve_registry已完成，其他未完成

总体思路：在本地用rust复现合约存储在sol_simple文件夹，然后在`get_best_pool`函数中创建并调用

```
// 解析 ABI 数据
let merged_abi = parse_abi(&serde_json::Value::Array(pool_abis.clone()))?;
// 创建合约实例
let pool_contract = Contract::new(pool_address, merged_abi, client.clone());
```

接下来的目标

优先完成基于公式的方法

开发

1. 仿照其他合约的写法，开发uniswap v2和sushiswap基于复现的方法
2. 学uniswap v3的公式，仿照`get_amount_out_v2`的格式，写一个`get_amount_out_v3`函数

测试

1. 写一些脚本从链上收集一些uniswap v2和sushiswap的交易数据，测试`get_amount_out_v2`函数计算的正确性。
2. 写一些脚本比对两种方法和链上真实数据之间的差异，反馈给开发改进代码。

参考资料：

uniswap_v2 router地址: https://etherscan.io/address/0x7a250d5630b4cf539739df2c5dacb4c659f2488d

sushiswap router地址: https://etherscan.io/address/0xd9e1cE17f2641f24aE83637ab66a2cca9C378B9F

uniswap_v2白皮书: https://app.uniswap.org/whitepaper.pdf

uniswap_v3白皮书: https://app.uniswap.org/whitepaper-v3.pdf

(访问历史数据)etherscan_api: https://docs.etherscan.io/etherscan-v2/api-endpoints/logs

(访问实时数据)以太坊RPC节点: https://eth-mainnet.g.alchemy.com/v2/quR8N0XVxV090J3xRxc43qxtcladUxnV

在线rust运行工具: https://play.rust-lang.org/