  AddrToken["UNI"]="0x1f9840a85d5aF5bf1D1762F925BDADdC4201F984";

    AddrToken["WETH"] = "0xB4FBF271143F4FBf7B91A5ded31805e42b2208d6";

    AddrToken["USD//C"] = "0x07865c6E87B9F70255377e024ace6630C1Eaa37F";

    AddrToken["DAI"] = "0xdc31Ee1784292379Fbb2964b3B9C4124D8F89C60";

* Variables

PERIOD: dùng để chỉ khoảng thời gian tích lũy tối thiểu sau mỗi lần Update

Pair: Địa chỉ của cặp token0,token1

Token0, token1: địa chỉ của cặp token muốn swap

price0CumulativeLast: giá trị Cumulative lần cập nhật gần nhất của token0 so với token1

price1CumulativeLast: giá trị Cumulative lần cập nhật gần nhất của token1 so với token0

blocktimestampLast: thời gian của block được gọi ở lần update gần nhất

price0Average giá trị trung bình của token0 so với token1

price1Average giá trị trung bình của token1 so với token0

* Function update()

price0Average = FixedPoint.uq112x112(uint224((price0Cumulative - price0CumulativeLast) / timeElapsed));

price1Average = FixedPoint.uq112x112(uint224((price1Cumulative - price1CumulativeLast) / timeElapsed));

price0CumulativeLast = price0Cumulative;

price1CumulativeLast = price1Cumulative;

blockTimestampLast = blockTimestamp;

* Consult(address token,uint amountIn)
* Nhập địa chỉ của token muốn đổi và số amount muốn đổi.
* Xuất ra số amount đổi ra

pragma solidity 0.6.6;

import '@uniswap/v2-core/contracts/interfaces/IUniswapV2Factory.sol';

import '@uniswap/v2-core/contracts/interfaces/IUniswapV2Pair.sol';

import '@uniswap/lib/contracts/libraries/FixedPoint.sol';

import '@uniswap/v2-periphery/contracts/libraries/UniswapV2OracleLibrary.sol';

import '@uniswap/v2-periphery/contracts/libraries/UniswapV2Library.sol';

// fixed window oracle that recomputes the average price for the entire period once every period

// note that the price average is only guaranteed to be over at least 1 period, but may be over a longer period

contract PriceOracle {

    using FixedPoint for \*;

    uint public constant PERIOD = 1;

    IUniswapV2Pair immutable pair;

    address public immutable token0;

    address public immutable token1;

    uint    public price0CumulativeLast;

    uint    public price1CumulativeLast;

    uint32  public blockTimestampLast;

    FixedPoint.uq112x112 public price0Average;

    FixedPoint.uq112x112 public price1Average;

    //uint8 public constant decimals = 18;

    constructor(address factory, address tokenA, address tokenB) public {

        IUniswapV2Pair \_pair = IUniswapV2Pair(UniswapV2Library.pairFor(factory, tokenA, tokenB));

        pair = \_pair;

        token0 = \_pair.token0();

        token1 = \_pair.token1();

        price0CumulativeLast = \_pair.price0CumulativeLast(); // fetch the current accumulated price value (1 / 0)

        price1CumulativeLast = \_pair.price1CumulativeLast(); // fetch the current accumulated price value (0 / 1)

        uint112 reserve0;

        uint112 reserve1;

        (reserve0, reserve1, blockTimestampLast) = \_pair.getReserves();

        require(reserve0 != 0 && reserve1 != 0, 'ExampleOracleSimple: NO\_RESERVES'); // ensure that there's liquidity in the pair

    }

    function update() external {

        (uint price0Cumulative, uint price1Cumulative, uint32 blockTimestamp) =

            UniswapV2OracleLibrary.currentCumulativePrices(address(pair));

        uint32 timeElapsed = blockTimestamp - blockTimestampLast; // overflow is desired

        // ensure that at least one full period has passed since the last update

        require(timeElapsed >= PERIOD, 'ExampleOracleSimple: PERIOD\_NOT\_ELAPSED');

        // overflow is desired, casting never truncates

        // cumulative price is in (uq112x112 price \* seconds) units so we simply wrap it after division by time elapsed

        price0Average = FixedPoint.uq112x112(uint224((price0Cumulative - price0CumulativeLast) / timeElapsed));

        price1Average = FixedPoint.uq112x112(uint224((price1Cumulative - price1CumulativeLast) / timeElapsed));

        price0CumulativeLast = price0Cumulative;

        price1CumulativeLast = price1Cumulative;

        blockTimestampLast = blockTimestamp;

    }

    // note this will always return 0 before update has been called successfully for the first time.

    function consult(address token, uint amountIn) external view returns (uint amountOut) {

        if (token == token0) {

            amountOut = price0Average.mul(amountIn).decode144();

        }

        else {

            require(token == token1, 'ExampleOracleSimple: INVALID\_TOKEN');

            amountOut = price1Average.mul(amountIn).decode144();

        }

    }

}