Introduction to Decentralized Finance

Homework 4

1 Announcement

- The assignment includes one programming problem and six handwritten problems.
- If you encounter any issues with the homework, please first visit the discussion forum on NTU COOL.
- If you need further assistance, attend the TA hour so the teaching assistant can help you with your problem.
- If you do not receive a response in the forum, or if posting your question would reveal your answer, leave a comment in your pull request (PR), and the TA will reach out to you.
- If you encounter any issues with Homework 4 that cannot be resolved using the methods above, please reach out to `**@csie.ntu.edu.tw` following these guidelines:
  1. Title your email `[DeFi-HW4][Summary-Of-Your-Issue]`. Please note that we will **NOT** receive emails in other formats as we have applied filters to our email system.
  2. Provide detailed information about your computer, including the operating system.
  3. Outline the methods you attempted previously, the resources you consulted, the steps you followed, and the results of your efforts.
  4. Note that ambiguous requests, such as attaching screenshots without proper descriptions, will not be answered.
- There is **NO** late submission allowed. After the deadline, you will automatically lose write access to the repository. Please ensure you push your code changes to GitHub before the deadline.
- You are **NOT** allowed to modify any protected files, such as `.github/**/*`, and we will provide these paths for you. You will receive zero points if any protected file is modified.

2 Preliminaries

- Create a homework repository by clicking this link.
- **Important**: Select your name and link it to your GitHub account. Failure to complete this step may result in a penalty.
- Clone the repository, enter the project directory, and install dependencies using: `cd hw && forge install`.
- Add necessary packages, run `forge install OpenZeppelin/openzeppelin-contracts --no-commit`
- **Optional**: If you encounter any issues with package linking, run `forge remappings`.

3 Problem 1: LendSphere (70 pt)

LendSphere conveys the idea of a global, interconnected ecosystem for lending, combining innovation and accessibility in decentralized finance.

Professor Liao decided to develop his own lending protocol. After analyzing the structures of Aave and Compound, he found them overly complex. For the first iteration of his product, he aims to create a simplified and minimal lending service, without incorporating an interest rate model or liquidity farming mechanism.

3.1 Description

LendSphere is a simplified lending service, with the following configuration:

- LIQUIDATION_REWARD: 5%. This refers to the percentage bonus given to liquidators for successfully liquidating an undercollateralized position. For example, if a liquidator repays part of a borrower's debt, they can receive an additional 5% of the collateral as a reward. The reward token should be the collateral originally deposited by the borrower.
- LIQUIDATION_FACTOR: 80%. This defines the Loan-to-Value (LTV) ratio at which a position becomes eligible for liquidation. The LTV is calculated as the ratio of total borrowed asset value to total collateral asset value. If the liquidation factor multiplied by the current collateral value exceeds the borrowed amount, the loan becomes liquidatable.
- CLOSE_FACTOR: 50%. This limits the portion of the borrower's debt that can be repaid during a single liquidation event. For example: With a 50% close factor, only half of the outstanding debt can be liquidated at a time.
- MIN_HEALTH_FACTOR: 1e18. This value, scaled relative to 1e18, represents the safety of a borrower's position, with values below 1e18 indicating liquidation risk.

There are several operations you are required to implement. Here we list the operations commonly seen in lending services:

- supply: Allows users to deposit tokens as collateral.
- withdraw: Enables users to withdraw their supplied collateral, subject to maintaining a healthy position.
- borrow: Allows users to borrow tokens against their supplied collateral.
- repay: Enables users to repay borrowed tokens to reduce or clear their debt.
- liquidate: Allows third parties (liquidators) to repay a portion of a borrower's debt and claim collateral when their health factor drops below the threshold.

3.2 Specification

In the LendSphere design, there are several data structure you need to maintain:

- `allowedTokens`: A list of ERC20 tokens permitted for deposits, withdrawals, and borrowing within the protocol. It can only be configured by the protocol owner.
- `tokenToPriceFeed`: A mapping from token addresses to their corresponding price feed addresses, used for fetching real-time prices. LendSphere integrates Chainlink as price oracle.
- `accountToTokenDeposits`: A mapping of user accounts to the amount of each token they have deposited in the protocol.
- `accountToTokenBorrows`: A mapping of user accounts to the amount of each token they have borrowed from the protocol.

For each operation below, only tokens on the whitelist are allowed to be traded; otherwise, it should revert with the error string `Token Not Allowed`.

3.2.1 setAllowedToken

This operation is restricted to the owner, who provides two values: the token and its corresponding price oracle address. The newly added token is appended to the `allowedTokens` list, and the `tokenToPriceFeed` mapping is updated. The operation must emit the `AllowedTokenConfiguration` event upon successful execution and allows re-assignment of the oracle address.

3.2.2 getValueInETH

Given the token address and the token amount, return the total value measured in ETH. For example, if the input parameter is DAI and amount is 2000, and the current price of DAI is 0.005 ETH (1 ETH = 2000 DAI), then this operation should return 0.005 * 2000 = 2000 DAI * (0.0005 ETH / 1 DAI) = 1 ETH

3.2.3 getTokenValueFromEth

Given a total value in ETH and a token address, calculate the equivalent token amount for the specified value. For instance, if the total value is 30 ETH (`totalValueInETH`) and the token is `token0`, with a price of 2 ETH per `token0`, the function should return $30 \times 1\,\text{ether}/2 = 15\,\text{ether}$ units of `token0`.

3.2.4 viewCollateral

Returns the total value of supplied collateral, measured in ETH.

3.2.5 viewDebt

Returns the total value of borrowed asset, measured in ETH.

3.2.6 healthFactor

The metric used in lending and borrowing protocols to represent the safety of a borrower's collateral relative to their loan. It is calculated as:

$$\text{Health Factor} = \frac{\text{Total Collateral Value} \times \text{Liquidation Factor}}{\text{Total Borrowed Value}} \times 1e18$$

where the total collateral value is the worth of assets supplied as collateral, adjusted by their Liquidation Factor ratio to account for risk and volatility, and the total borrowed value represents the amount borrowed. A health factor greater than 1 indicates the loan is safe, while a health factor less than or equal to 1e18 suggests the loan is at risk of liquidation, as the collateral may no longer cover the borrowed amount. For example, if a borrower supplies $1,000 in collateral with an 80% Liquidation Factor ratio (adjusted value: $800) and borrows $400, the health factor is as follows, indicating a safe loan with a buffer against liquidation.

$$\text{Health Factor} = \frac{1000 \times 80\%}{400} \times 1e18 = 2e18$$

If no assets are borrowed, the health factor should return a value of 100e18.

3.2.7 Supply

This operation updates the user's supply balance, transfers tokens to the protocol through `transferFrom`, reverts with `TransferFailed` error if the transfer fails. If operation succeed, it should emit an event, `TokenSupply` to log the deposit details.

3.2.8 Withdraw

Allows users to withdraw their supplied collateral while ensuring their position remains healthy. If sufficient funds are unavailable, the function reverts with `Insufficient Funds` error string. If the withdrawal results in an unhealthy position, it reverts with `Insolvency Risk` error string. Upon successful withdrawal, the function emits a `TokenWithdraw` event.

3.2.9 Borrow

Allows users to lend by supplying collateral while maintaining a healthy position. If lendSphere lacks sufficient tokens to lend, the transaction should revert with `Insufficient borrowable tokens` If the position is unhealthy, it should revert with `Insolvency Risk`. If the transfer operation fails, it should revert with a `TransferFailed` custom error.

3.2.10 Repay

Allows users to repay borrowed tokens to reduce or clear their debt. The specified token is transferred from the user's account to the protocol, updating the `accountToTokenBorrows` mapping. Emits a `TokenRepay` event upon successful repayment. If a user attempts to repay more than the borrowed amount, the transaction should revert with the error string `Excessive Repayment`.

3.2.11 Liquidation

Enables liquidators to repay a portion of a borrower's debt when the borrower's health factor falls below the minimum threshold (`MIN_HEALTH_FACTOR`), making their position eligible for liquidation. The function requires the repay token (`repayToken`) and reward token (`rewardToken`) to be in the whitelist (`allowedTokens`). It calculates the amount to repay (half the debt adjusted by `CLOSE_FACTOR`) and the liquidation reward in the reward token (`LIQUIDATION_REWARD`). The repay token is transferred to the protocol, reducing the borrower's debt in `accountToTokenBorrows`, and the reward token is transferred to the liquidator. Emits a `Liquidate` event with details of the liquidation upon successful liquidation.

liquidators cannot specify the amount to liquidate. Each liquidation automatically repays the maximum liquidable amount determined by the `CLOSE_FACTOR`.

3.3 Example

The protocol owner has configured three tokens: `token0`, `token1`, and `token2`. The prices for these tokens are set at `1 ETH`, `2 ETH`, and `3 ETH` respectively. A user holds 50 ether unit of each token.

3.3.1 Case 1

The user supplies 50 ether unit of `token2` as collateral and borrows 50 ether unit of `token0` and 35 ether uint of `token1`. The total collateral value is $50 \times 3 = 150$ ETH, while the total borrowed value is $50 \times 1 + 35 \times 2 = 120$ ETH. The current health factor is calculated as $150 \times 80\%/120 \times 1e18 = 1e18$, indicating a healthy position.

3.3.2 Case 2

The user supplies 50 ether unit of `token2` as collateral and borrows 50 ether unit of `token0` and 50 ether unit of `token1`. The total collateral value is $50 \times 3 = 150$ ETH, while the total borrowed value is $50 \times 1 + 50 \times 2 = 150$ ETH. The current health factor is calculated as $150 \times 80\%/150 \times 1e18 = 0.8e18$, indicating an unhealthy position, this borrow operation will be reverted.

3.3.3 Case 3

Based on `Case 1`, the user currently supplies 50 ether units of `token2` and borrows 50 ether units of `token0` and 35 ether units of `token1`. If market volatility causes the price of `token2` to drop to 2 ETH, the health factor will fall below threshold, making the position liquidable. The liquidator decides to repay `token0` and receive `token2` as the reward token. The liquidator must repay 25 ether units of `token0` (determined by the borrow amount and the close factor) and will receive a 5% reward in `token2`. Considering the price of `token2` is 2 ETH, this is equivalent to 13.125 ether units of `token2` as the reward.

3.4 Judge

The tests are in `hw/test/Lending/Lender.t.sol`.

3.4.1 setAllowedToken (5 pt)

- `forge test --mc LenderTest --mt testSetMultipleAllowedToken`
- `forge test --mc LenderTest --mt testSetAllowedTokenNotOwner`

3.4.2 getValueInETH (5 pt)

- `forge test --mc LenderTest --mt testgetValueInETH`

3.4.3 viewCollateral (5 pt)

- `forge test --mc LenderTest --mt testViewCollateral`

3.4.4 getTokenValueFromEth (5 pt)

- `forge test --mc LenderTest --mt testGetTokenValueFromEth`

3.4.5 Supply (5 pt)

- `forge test --mc LenderTest --mt testSupplyForMultipleTokens`
- `forge test --mc LenderTest --mt testSupplyNotAllowedTokens`

3.4.6 Borrow (10 pt)

- `forge test --mc LenderTest --mt testBorrowForMultipleTokens`
- `forge test --mc LenderTest --mt testBorrowForMultipleTokensAmountNotEnough`
- `forge test --mc LenderTest --mt testBorrowForMultipleTokensUnhealthyPosition`
- `forge test --mc LenderTest --mt testBorrowNotAllowedTokens`

3.4.7 Withdraw (10 pt)

- `forge test --mc LenderTest --mt testWithdrawMultipleTokens`
- `forge test --mc LenderTest --mt testWithdrawMultipleTokensAmountNotEnough`
- `forge test --mc LenderTest --mt testWithdrawMultipleTokensUnhealthyPosition`
- `forge test --mc LenderTest --mt testWithdrawNotAllowedTokens`

3.4.8 Repay (10 pt)

- `forge test --mc LenderTest --mt testRepayMultipleTokens`
- `forge test --mc LenderTest --mt testRepayNotAllowedTokens`

3.4.9 Liquidation (15 pt)

- `forge test --mc LenderTest --mt testLiquidateMultipleTokens`

3.5 Protected Files

- `hw/test/Lending/Lender.t.sol`
- `hw/test/Lending/LenderBase.t.sol`

4 Problem 2: Multichain Ecosystem (30 pt)

The rise of innovative Web3 technologies in DeFi, SocialFi, and GameFi has increased the demand for blockchain solutions, highlighting challenges like low speed, high gas fees, and MEV issues on Ethereum, the dominant blockchain with hundreds of billions in TVL. While new Layer 1 blockchains like Solana and Hyperliquid aim to address these issues with higher throughput and lower fees, Ethereum introduces Layer 2 solutions to enhance scalability without compromising decentralization and security. These solutions, including ZK Rollups with zero-knowledge proofs for faster finality and Optimistic Rollups with fraud proofs for higher throughput, improve performance and address the blockchain trilemma.

4.1 Description

Please provide answers to the questions below and include them in the README file of your homework repository.

4.2 Questions

- Please explain the inner workings of ZK Rollups and Optimistic Rollups and compare the differences between these rollup structures. Additionally, explain the differences between ZK Rollups, Volition, and Validium modes. (5 pt)
- According to L2 Beat, there are multiple stages for a Layer 2. What are the criteria for each stage, and what is the current status of the Layer 2 solutions in use? (5pt)
- Layer 2 solutions aim to address scaling issues in the Ethereum ecosystem, but they introduce liquidity fragmentation, which leads to interoperability challenges. Cross-chain bridges can help address these issues. Modern cross-chain bridges can be categorized into burn-and-mint, lock-and-mint, and lock-and-unlock mechanisms. Please analyze two cross-chain bridge structures (e.g., LayerZero, Wormhole and more) with distinct underlying mechanisms. (5 pt)
- Cross-chain bridges have suffered significant losses due to security breaches. For example, in 2021, PolyNetwork experienced a $611 million exploit, BNB Bridge faced a $586 million exploit, and Wormhole was attacked for $326 million. Please analyze the vulnerabilities in a cross-chain bridge structure, providing code examples to illustrate the issues. (5 pt)
- User experience is key to the mass adoption of blockchain. Currently, intent-based solutions are popular. What is an intent, and how does it differ from a cross-chain bridge? (5 pt)
- The ongoing bull market is marked by an increase in both the number and total losses from phishing scams. Please explain how `permit` and `permit2` work, along with examples of phishing scams related to these operaitons. (5 pt)