Cryptocurrencies (WS2021/22) Prof. Matteo Maffei

Project 1 – Analysing the Blockchain

TU Wien

Group 04

Our group consists of the following members:

Tobias Eidelpes, Ege Mehmet Demirsoy, Nejra Komic
01527193, XXXXXXXX, XXXXXXXX, same order as the names

Exercise A: Finding invalid blocks

For this exercise all invalid blocks contained in the database provided to us had to be found. While there is an official[1] algorithm which allows network participants to verify whether a block is invalid or not, the stripped-down version of the blockchain we received does not require all the steps. This stripped-down version of the algorithm thus specifies which constraints the data must satisfy:

1. All blocks which do not have the coinbase transaction as their first transaction are invalid. This will be achieved by creating a view which lists all coinbase transactions. Then we query the database for all first transactions of each block and check if that transaction is in the view of all coinbase transactions. If it is not, we reject the block and add it to the invalid list.

2. All blocks which contain transactions which do not have inputs or outputs are invalid. We split this task into two queries, one for checking if a block contains transactions with zero inputs and another one for checking if a block contains transactions with zero outputs.

3. All blocks which have transactions with an invalid output value or where the sum of all output values exceeds the legal money range are invalid. This task is split into two queries as well. One for checking if individual output values are outside of the legal money range and a second one for checking if the sum of all output values per transaction is outside of the legal money range.

4. Reject all blocks which have transactions with inputs that do not have a corresponding output. For this task we first create a view which finds all non coinbase transactions. The output of that query is then filtered for all inputs which are not part of a coinbase transaction (so the non coinbase inputs). Finally, the non coinbase inputs are joined with the outputs and rows containing `NULL` as their `value` indicate an invalid block.

5. All blocks which contain transactions where the input's `sig_id` field is not the same as the output's `pk_id` field are invalid. Since we are not interested in the coinbase transactions, the

[1] `https://en.bitcoin.it/wiki/Protocol_rules#.22block.22_messages`

query uses the non coinbase inputs again to join them with the outputs. If the two fields do not match, the block is invalid.

6. All blocks which have inputs for which there exist outputs which have already been spent are invalid. This task is split into three queries. First, we find all outputs which have more than one input. Second, for all the outputs found, we find the corresponding inputs where the output was first spent. Third, the two tables are combined such that blocks with outputs which have corresponding inputs that are not listed as the first spending occurrence, are marked as invalid.

7. All blocks containing inputs which are not in the legal money range are invalid. First, we construct a view which gathers all transactions and their corresponding sum of value for all inputs. All blocks containing input sums which are outside of the legal money range are marked as invalid. Second, we reuse the view of all non coinbase inputs and filter them for the ones which have an output value outside of the legal money range.

8. All blocks where the sum of input values is smaller than the sum of output values are invalid. This task allows us to reuse the view created earlier of all input sums. Additionally, the sum of output values is obtained similarly to the input sums. After joining both input sums and output sums, we can filter for blocks which have smaller input sums than output sums. Those blocks are invalid.

9. All blocks where the coinbase value is larger than the sum of the block creation fee and all transaction fees are invalid. This task is split into four queries. First, we create a view which shows all block ids and their coinbase values. Second, we need to know the sum of all input values per block. Third, we repeat that query for the sum of the output values per block. Lastly, these three tables are joined and all blocks which satisfy the constraint are invalid.

Finally, the invalid blocks are written to the `invalid_blocks` table and all duplicates are removed.

Exercise B: UTXOs

Exercise C: De-anonymization

Work distribution

Tobias Eidelpes Code and report for Exercise A.

Ege Mehmet Demirsoy

Nejra Komic