Summer Of Bitcoin Challenge Solution

Aim: To maximise the total fees of the valid block transactions from the given mempool.

Constrains: 1. Maximum weight should not be greater than 40,00,000

2. A transaction may only appear in a block if all of its parents appear earlier in the block.

Possible Outcomes:

1. If CT $\in$ Block $\Rightarrow$ PT $\in$ Block.
2. If PT $\in$ Block $\Rightarrow$ May or May not CT $\in$ Block
3. If PT $\notin$ Block $\Rightarrow$ CT $\notin$ Block.

Where CT: Child transaction PT: Parent transaction

Approach For Solution:

In the mempool there are two types of transactions:

1) **Transaction with Parent/Parents: Dependent on Parent**
2) **Transaction without Parent/Parents: Independent**

And selection of transactions in the valid block is based on two factors i.e.

I. Fees of Transaction

II. Weight Of Transaction

Now we can select the transaction based on these factors so if the transaction is parent dependent then the fee and weight of the parent transaction must also be taken into consideration and the independent transaction can be regulated independently.
Hence to solve this complexity instead of using the given fees and weight we should use a optimal fees and weight i.e.

Optimal fees= fees + parents optimal fees

Optimal weight= weight + parents optimal weight

Example:
Let's consider a instance
Max Weight: 90

Transaction ID	Fees	Weight	Parent ID
123	**300**	**20**	**113, 136**
113	**100**	**25**	**141**
141	**200**	**50**	
136	**150**	**35**	

Now here we can optimize the fee and weight as:

Transaction ID	Fees	Weight	Parent ID
123	**750 (300+(100+200)+150)**	**120 (20+(25+40)+35)**	**113, 136**
113	**300 (100+200)**	**65 (25+40)**	**141**
141	**200**	**40**	
136	**150**	**35**	

Now after obtaining the optimal fees we can find the list of transactions using the concept of knapsack problem. In the solution I used both approaches to solve it i.e. greedy (Main solution) and dynamic programming (Alt. Solution).

Note: SOB_Solution_MAIN.py is the main solution and it uses greedy methodology while ALT_SOB_Dynamic.py is an alternative solution with dynamic programming.

Working:

In the SOB_Solution_MAIN.py the final solution is obtained by finding a fees/weight ratio and then deciding which of the transactions is included into block while maximising the fees and taking constraints into considerations.

Note: The disadvantage of ALT_SOB_Dynamic.py as it is resource consuming due to the memoization property of dynamic programming.

Files Included:

1. **SOB_Solution_MAIN.py**: Main Solution Code
2. **block.txt**: The output from the program i.e. transactions separated by newlines which make a valid block.
3. **fee_obtained.txt**: fee of the block (block.txt)
4. **weight_obtained.txt**: weight of the block (block.txt)
5. **mempool.csv:** Data pool of transactions
6. **Problem Statement:** Problem of SOB code challenge
7. **ALT_SOB_Dynamic.py:** Alternate solution for the problem (my first approach but resource consuming, I also tried to used memmap to implement this but still execution time is very high)

Author:

Shashank Joshi
shashank002joshi@gmail.com