Digitálne meny a blockchain

ZADANIE 1 – MyBlockChain

Tomáš Golis ID: 110787

FIIT STU

Cvičenie: Streda 18:00

Cvičiaci : Kristián Košťál

19.3.2023

1.Fáza

V prvej fáze bolo za úlohu dopísať triedu HandleTxs tak aby zvalidovala prichádzajúce transakcie a následne vrátila pole validných Transakcií a popri tom udržovala UTXOPool.

Validná transakcia musela spĺňať 5 podmienok.

1) V prvej podmienke stačilo prejsť všetky inputy transakcií a zistiť či sa nachádzajú v príslušnom UTXOPoole

```go
func (h *HandleTxs) firstCondition(tx Transaction) bool {
    for _, input := range tx.Inputs {
        utxo := UTXO{
            TxHash: input.prevTxHash,
            Index:  input.outputIndex,
        }
        if !h.UTXOPool.Contains(utxo) {
            return true
        }
    }
    return true
}
```

2) V druhej podmienke bolo potrebné overiť podpisy na všetkých inputoch. Adresy sú uložené ako *rsa.PrivateKey/*rsa.PublicKey zložené zo 2056 bitoch, práve kvôli podpisovaniu a overeniu transakcií.

```go
func (h *HandleTxs) secondCondition(tx Transaction) bool {
    for i, input := range tx.Inputs {
        address := h.getAddress(input.outputIndex, input.prevTxHash)
        if address == nil {
            fmt.Println("nil address")
            return false
        }
        hashed := sha256.Sum256(tx.getDataToSign(i))
        err := rsa.VerifyPKCS1v15(address, crypto.SHA256, hashed[:], input.signature)
        if err != nil {
            fmt.Printf("error while verifying signature: %v", err)
            return false
        }
    }
    return true
}
```

3) V tretej podmienke je nutné overiť či nejaké UTXO nie je nárokované viackrát

```go
func (h *HandleTxs) thirdCondition() bool {
    utxos := h.UTXOPool.GetAllUTXO()
    for i := 0; i < len(utxos); i++ {
        for k, utxo := range utxos {
            if k == i {
                break
            }
            if utxo.equals(utxos[i]) {
                // if utxo.hashCode() == utxos[i].hashCode() {
                return false
            }
        }
    }
    return true
}
```

4) V štvrtej podmienke stačilo prejsť všetky outputy Transakcií a verifikovať či sú > 0

5) Pre overenie 5. podmienky bolo nutné spočítať všetky hodnoty a overiť či súčet inputov bol väčší ako súčet outputov

```go
func (h *HandleTxs) fifthCondition(tx Transaction) bool {
    sumOutputs := 0.0
    sumInputs := 0.0
    for i := 0; i < len(tx.Outputs); i++ {
        sumOutputs += tx.Outputs[i].value
    }
    for i := 0; i < len(tx.Inputs); i++ {
        value := h.getValue(tx.Inputs[i].outputIndex, tx.Inputs[i].prevTxHash)
        if value < 0 {
            return false
        }
        sumInputs += value
    }
    if sumOutputs > sumInputs {
        return false
    }
    return true
}
```

1.Fáza nebola náročná, len mi trvalo dlhšie kým som pochopil čo vlastne po mne chcete.

2.Fáza

2.Fázu som nerobil lebo sa neoplatila za jeden bod.

3.Fáza

V 3. Fázy bolo za úlohu implementovať štruktúru Blockchain ktorá je zodpovedná za udržiavanie blokového reťazcu.

```go
type Blockchain struct {
    // BlockNode BlockNode
    TransactionPool TransactionPool
    BlockNode       []BlockNode
}

type BlockNode struct {
    B        Block
    Parent   *BlockNode
    Children []*BlockNode
    Height   int
    UPool    UTXOPool
}
```

- array nespracovaných Txs
- vpodstate blockchain

- Block
- Rodicovský Block
- Deti
- Výška
- UTXOPool prislúchajúci danému Nodu

Func (b *Blockchain) GetBlockAtMaxHeight() **(Block, int)** prehľadáva array BlockNodov a hľadá ten s najväčšou výškou. Po nájdení vráti daný block a jeho výšku.

Func (b *Blockchain) GetUTXOPoolAtMaxHeight() UTXOPool funguje rovnako ako GetBlockAtMaxHeight ale vráti jeho UTXOPool namiesto Blocku a výšky

Func NewBlockchain(genesisBlock Block) *Blockchain vytvorí blockchain, pridá doňho genesisBlock a vráti novovytvorený Blockchain

```go
func NewBlockchain(genesisBlock Block) *Blockchain {
    coinbaseTx := genesisBlock.Coinbase
    index := 0
    utxoPool := NewUTXOPool()
    utxo := UTXO{
        TxHash: genesisBlock.Coinbase.Hash,
        Index:  index,
    }
    utxoPool.AddUTXO(utxo, coinbaseTx.Outputs[index])
    blockNode := NewBlockNode(genesisBlock, nil, *utxoPool)
    return &Blockchain{
        BlockNode:       []BlockNode{*blockNode},
        TransactionPool: TransactionPool{},
    }
}
```

func (b *Blockchain) BlockAdd(block Block) bool funkcia do daného blockchainu pridá nový block

1. Skontroluje sa či prevBlockHash daného bloku nie je nil
2. Získa z blockchainu rodiča, ak nemá **return false**
3. Získa max height a overí či parent.Height + 1 <= maxHeight-CUT_OFF_AGE
4. Overi validáciu transakcií
5. Pridá coinbase transakciu do UTXOPoolu pridávaného blocku
6. Odstráni transakcie v blocku z TransactionPoolu
7. Vytvorí sa nový BlockNode
8. Rodičovi sa pridá novovytvorený BlockNode ako dieťa
9. Do blockchainu sa pridá novovytvorený BlockNode
10. Vymažu sa staré bloky z blockchainu

```go
func (b *Blockchain) BlockAdd(block Block) bool {
    // if prevBlockHash is nil or his hash length is 0 (genesis) returns false
    if block.PrevBlockHash == nil || len(block.PrevBlockHash) == 0 {
        fmt.Println("previousBlockHash is incorrect")
        return false
    }

    // gets the father Node
    parent, err := b.getParent(block)
    if err != nil {
        fmt.Println("did not get parent")
        return false
    }

    // this checks whether the height of potentional new block
    // is not heigher then maxHeight - CUT_OFF_AGE
    _, maxHeight := b.GetBlockAtMaxHeight()
    if parent.Height+1 <= maxHeight-CUT_OFF_AGE {
        fmt.Println("did not meet the condition parent.Height+1 <= maxHeight-CUT_OFF_AGE")
        return false
    }

    // this checks whether all of the transactions are valid
    handler := HandleTxs{
        UTXOPool: parent.UPool,
    }

    validTxs := handler.Handler(block.Txs)
    if len(validTxs) != len(block.Txs) {
        fmt.Println("not validated")
        return false
    }
```

```go
    // adds a coinbase tx on index 0 to a handlers UTXOPool
    utxoPool := handler.UTXOPool
    utxo := UTXO{
        TxHash: block.Coinbase.Hash,
        Index:  0,
    }
    utxoPool.AddUTXO(utxo, block.Coinbase.Outputs[0])

    // removes Transactions from transaction pool
    for _, tx := range validTxs {
        b.TransactionPool.RemoveTransaction(tx.Hash)
    }

    node := BlockNode{
        B:      block,
        Parent: &parent,
        UPool:  utxoPool,
    }
    // pointer to children node is added to the parent node Children array
    parent.Children = append(parent.Children, &node)
    b.BlockNode = append(b.BlockNode, node)

    b.cleanOldBlocks(maxHeight)
    return true
}
```

Uživateľská príručka

Pre pridanie unitTestov stačí pridať funkciu do Tests.go (buď v priečinku firstPhase alebo thirdPhase) a následne ju zavolať v firstPhase_test.go alebo thirdPhase_test.go
Pre spustenie stačí napísať **go test ./firstPhase** alebo **go test ./thirdPhase**

Bohužiaľ som myslel že testovanie nebude mocť byt v rovnakom packagy preto som väčšinu mien funkcií začal s veľkým písmenom a teraz už je moc neskoro to zmeniť

Celé je to napísane v Golangu lebo nemám rád Javu a chcel som si to tam vyskúšať, hlavne keď za to boli bonusové body

Záver

Nakoniec by som skonštatoval že v rámci práce na tomto zadaní som sa veľa naučil, aj napriek tomu, že to prvotne bolo celkom náročné. Najväčším problémom bolo pochopenie, ako vlastne blockchain funguje, čo sa mi v konečnom dôsledku, dovoľujem si trvdiť, vďaka vypracovaniu tejto úlohy podarilo. Až na ten konsenzus tam furt netuším o čo ide.