```
MODULE BitcoinChain
EXTENDS Naturals, FiniteSets, Sequences

CONSTANTS
    Nodes,          Set of nodes in the network
    MaxBlocks,      Maximum number of blocks that can be created. We need this to cap the model run.
    GenesisHash     Hash of the genesis block

VARIABLES
    blocksByNode,       blocksByNode[n] is the set of blocks known by node n
    chainsByNode,       chainsByNode[n] is the DAG of blocks for node n
    tipsByNode,         tipsByNode[n] is the current tip of the chain for node n
    workByNode,         workByNode[n] is a function mapping each block to its cumulative work
    confirmedByNode,    confirmedByNode[n] is the set of confirmed blocks for node n
    network,            network[n] is the set of blocks in transit to node n
    nextBlockId         Counter used to generate unique block IDs

vars ≜ ⟨blocksByNode, chainsByNode, tipsByNode, workByNode, confirmedByNode, network, nextBlockId⟩

 Type definitions
Block ≜ [id : Nat, prevHash : Nat, nonce : Nat, timestamp : Nat]
Chain ≜ [blocks : SUBSET Nat, edges : SUBSET (Nat × Nat)]

 Helper functions
WorkFor(b) ≜ 1   Simplified work calculation, each block contributes 1 unit of work

 Initial state
Init ≜
    ∧ blocksByNode  = [n ∈ Nodes ↦ {[id ↦ 0, prevHash ↦ GenesisHash, nonce ↦ 0, timestamp ↦ 0]}]
    ∧ chainsByNode = [n ∈ Nodes ↦ [blocks ↦ {0}, edges ↦ {}]]
    ∧ tipsByNode   = [n ∈ Nodes ↦ 0]
    ∧ workByNode = [n ∈ Nodes ↦ [i ∈ {0} ↦ 1]]   Initialize work for genesis block to 1
    ∧ confirmedByNode = [n ∈ Nodes ↦ {}]
    ∧ network = [n ∈ Nodes ↦ {}]
    ∧ nextBlockId = 1

 Action: Create a new block
CreateBlock(n) ≜
    ∧ nextBlockId < MaxBlocks
    ∧ LET
          newBlock ≜ [id ↦ nextBlockId,
                      prevHash ↦ tipsByNode[n],
                      nonce ↦ nextBlockId,      Simplified nonce
                      timestamp ↦ nextBlockId]   Simplified timestamp
          updatedChain ≜ [
              blocks ↦ chainsByNode[n].blocks ∪ {nextBlockId},
              edges ↦ chainsByNode[n].edges ∪ {⟨tipsByNode[n], nextBlockId⟩}
```

```
        ]
        newWork ≜ workByNode[n][tipsByNode[n]] + WorkFor(newBlock)
        updatedWork ≜ [workByNode[n] EXCEPT ![nextBlockId] = newWork]
        prevTip ≜ tipsByNode[n]  Previous tip before creating the new block
    IN
     Update the blocks known by the local node
    ∧ blocksByNode′ = [blocksByNode EXCEPT ![n] = @ ∪ {newBlock}]
     Update the chain for the local node
    ∧ chainsByNode′ = [chainsByNode EXCEPT ![n] = updatedChain]
     Update the tip for the local node
    ∧ tipsByNode′ = [tipsByNode EXCEPT ![n] = nextBlockId]
     Track total work for the new block at this node, even if it is a constant for now
    ∧ LET
        newWorkMap ≜ [b ∈ DOMAIN workByNode[n] ∪ {nextBlockId} ↦
            IF b = nextBlockId THEN newWork ELSE workByNode[n][b]]
        IN
            workByNode′ = [workByNode EXCEPT ![n] = newWorkMap]
     The new block will be received by all other nodes in the network
    ∧ network′ = [m ∈ Nodes ↦ IF m ≠ n THEN network[m] ∪ {newBlock} ELSE network[m]]
     Update the global nextBlockId
    ∧ nextBlockId′ = nextBlockId + 1
     Local node immediately confirms the previous tip
    ∧ confirmedByNode′ = [confirmedByNode EXCEPT ![n] = confirmedByNode[n] ∪ {prevTip}]

 Action: Receive a block from the network
ReceiveBlockWithPreviousKnown(n, block) ≜
    ∧ block.prevHash ∈ chainsByNode[n].blocks
    ∧
      LET
          prevBlock ≜ block.prevHash
          updatedChain ≜ [
                          blocks ↦ chainsByNode[n].blocks ∪ {block.id},
                          edges ↦ chainsByNode[n].edges ∪ {⟨prevBlock, block.id⟩}
                      ]
          newWork ≜ workByNode[n][tipsByNode[n]] + WorkFor(block)
          newTip ≜ IF newWork > workByNode[n][tipsByNode[n]]
                    THEN block.id
                    ELSE tipsByNode[n]
      IN
       ∧ blocksByNode′ = [blocksByNode EXCEPT ![n] = @ ∪ {block}]
        Take the next block in the receieve queue
       ∧ network′ = [network EXCEPT ![n] = @ \ {block}]
       ∧ chainsByNode′ = [chainsByNode EXCEPT ![n] = updatedChain]
       ∧ LET
          newWorkMap ≜ [b ∈ DOMAIN workByNode[n] ∪ {block.id} ↦
```

```
                    IF b = block.id THEN newWork ELSE  workByNode[n][b]]
                 IN
                    workByNode′ = [workByNode EXCEPT ![n] = newWorkMap]
             Confirm the block if it is a new tip
           ∧ IF newWork > workByNode[n][tipsByNode[n]] THEN
                 ∧ confirmedByNode′ = [confirmedByNode EXCEPT ![n] = confirmedByNode[n] ∪ {tipsByNode[
               ELSE
                 ∧ confirmedByNode′ = confirmedByNode
             Update the tip for the local node
           ∧ tipsByNode′ = [tipsByNode EXCEPT ![n] = block.id]
           ∧ UNCHANGED ⟨nextBlockId⟩

 Next state relation
Next ≜
    ∨ ∃ n ∈ Nodes : CreateBlock(n)
    ∨ ∃ n ∈ Nodes : ∃ block ∈ network[n] : ReceiveBlockWithPreviousKnown(n, block)

 vars ≜ ⟨blocksByNode, chainsByNode, tipsByNode, workByNode, confirmedByNode, network, nextBlockId⟩

 Invariants
TypeInvariant ≜
    ∧ ∀ n ∈ Nodes :
        ∧ nextBlockId ∈ Nat
        ∧ tipsByNode[n] ∈ Nat
        ∧ chainsByNode[n].blocks ⊆ Nat
        ∧ chainsByNode[n].edges ⊆ (Nat × Nat)
        ∧ ∀ blockId ∈ DOMAIN workByNode[n] :
            ∧ blockId ∈ Nat
            ∧ workByNode[n][blockId] ∈ Nat
        ∧ confirmedByNode[n] ⊆ Nat
        ∧ network[n] ⊆ Block
        ∧ ∀ b ∈ blocksByNode[n] :
            ∧ b.id ∈ Nat
            ∧ b.prevHash ∈ Nat
            ∧ b.nonce ∈ Nat
            ∧ b.timestamp ∈ Nat

 Properties
TipHasMostWork ≜
    ∀ n ∈ Nodes :
      ∀ b ∈ chainsByNode[n].blocks :
        workByNode[n][tipsByNode[n]] ≥ workByNode[n][b]

 Complete specification
Spec ≜ Init ∧ □[Next]_vars
```