Class 7: Merkle Trees

Schedule

- If you didn't get full credit for Project 1 because of failure to post something interesting, you can (and should!) redeem yourself and earn full credit by **posting an interesting comment by Thursday**. It can be on (1) Discussion questions from Project 1 (2) notes from classes, or (3) general forum.
- **Quiz**: We'll have a short, closed resources quiz in class on **Wednesday, 11 February**. The point of the quiz is to see how well people are understanding the core ideas we've covered so far (including today).
- **Read** (by Monday): *Chapter 6: The Bitcoin Network*, *Chapter 7: The Blockchain* from Andreas Antonopoulos' book.
- I am away the rest of this week, so will not have office hours on Thursday. (I am, of course, still available by email and course web site.)

Exploring Blocks

Label	Bytes	Description
`version`	4	Block version information
`prev_block`	32	Hash of the previous block
`merkle_root`	32	Hash of Merkle tree of all transactions
`timestamp`	4	When block was created (overflows in 2106)
`bits`	4	Difficulty target used for this block
`nonce`	4	Nonce found to generate this block

Block 341537

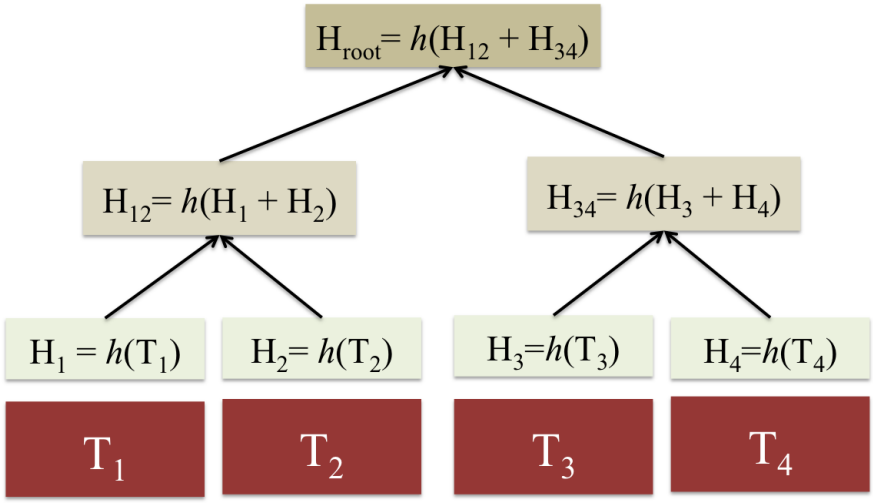

```
{
  "hash":"00000000000000002b32e242989056214fef31c5aac08ae517840db3e3e7fd2",
  "ver":2,
  "prev_block":"00000000000000014a97984448f2b3e5b8582ece719be1a1ea7db1d1fce5561",
  "mrkl_root":"d634ceec0d9a8b065ad3203555b74877d0476e0b302972be41671a6b92a0a066",
  "time":1422830051,
  "bits":404399040,
  "nonce":527809407,
  "n_tx":1511,
  "size":760657,
```

```
  "tx":[
    {
      "hash":"57db55dadb51ceeee6417af30946f234b2f77613e40586a2c03ce5e3a2be8bbb",
      "ver":1,
      ...
      "size":3533,
      "in":[
        {
          "prev_out":{
            "hash":"0000000000000000000000000000000000000000000000000000000000000000",
            "n":4294967295
          }, ...
      ],
      "out":[
        {
          "value":"1.00076629", ...
   },
   {
      "hash":"675e40df163d5ae4556774b325cdd7b0885d552bf7989d5e24f7039fce315a5b",
      ...
      "in":[
        {
          "prev_out":{
            "hash":"04e9c75d42093094a486a1c898527f4b50e1788fe4fda3ecb2574662b75b6f90",
        ...
        }
      ],
      "out":[
        {
          "value":"0.00100000",
          "scriptPubKey":"OP_DUP OP_HASH160 9e21abc1748a1df63b4016ac313c0f88e557d5fd ..."
        },
        {
          "value":"0.00710182",
          "scriptPubKey":"OP_DUP OP_HASH160 e37cd341540dd1e912568ae5b004d62422bd6b38 ..."
        }
      ]
    },
    ...
  ],
  "mrkl_tree":[
    "57db55dadb51ceeee6417af30946f234b2f77613e40586a2c03ce5e3a2be8bbb",
    "66a3eea4610dfb7d7987ad4fc22392c7964340d21006c1eea4c88174fd660c58",
    ...
    "675e40df163d5ae4556774b325cdd7b0885d552bf7989d5e24f7039fce315a5b",
    ...
  ]
}
```

Merkle Trees

https://github.com/btcsuite/btcd/blob/master/blockchain/merkle.go (some comments removed)

```
// HashMerkleBranches takes two hashes, treated as the left and right tree
// nodes, and returns the hash of their concatenation.
func HashMerkleBranches(left *btcwire.ShaHash, right *btcwire.ShaHash) *btcwire.ShaHash {
   var sha [btcwire.HashSize * 2]byte
   copy(sha[:btcwire.HashSize], left.Bytes())
   copy(sha[btcwire.HashSize:], right.Bytes())
   newSha, _ := btcwire.NewShaHash(btcwire.DoubleSha256(sha[:]))
   return newSha
}

func BuildMerkleTreeStore(transactions []*btcutil.Tx) []*btcwire.ShaHash {
    nextPoT := nextPowerOfTwo(len(transactions))
    arraySize := nextPoT*2 - 1
    merkles := make([]*btcwire.ShaHash, arraySize)

    // Create the base transaction shas and populate the array with them.
    for i, tx := range transactions { merkles[i] = tx.Sha() }

    // Start the array offset after the last transaction and adjusted to the
    // next power of two.
    offset := nextPoT
    for i := 0; i < arraySize-1; i += 2 {
        switch {
           case merkles[i] == nil:
              merkles[offset] = nil

           case merkles[i+1] == nil:
              newSha := HashMerkleBranches(merkles[i], merkles[i])
              merkles[offset] = newSha

           default:
              newSha := HashMerkleBranches(merkles[i], merkles[i+1])
              merkles[offset] = newSha
       }
       offset++
    }
    return merkles
}
```



What is needed to verify T_2 in H_{root}?

What must be recomputed if T_3 is replaced?

What must be computed if a new node, T_5, is added?

How many SHA-256 hashes must be computed to verify Block 341537?

