Computational Problem Solving — CSCI-603
Mining Bitcoin — Lab 4

7/28/2015

1 Problem

Bitcoin is a digital payment system that is regarded as the first secure cryptocurrency. It stores all transactions in a global ledger referred to as the blockchain. In order to prevent attacks such as double spending (a single token is spent more than once), each transaction must be verified by a process called mining.

The miners in the network keep the blockchain consistent by generating a new group of transactions called a block. The next valid block to be accepted by the network uses a cryptographic hash of the previous block using the SHA-256 hashing algorithm. It is a process that repeats every 10 minutes and is leveraged by a difficulty setting which at present requires roughly 57,500,000,000,000 hashes (57.5 Terahashes) to find the next successful block.

You will be performing one pass of the hashing algorithm on a single 32 bit piece of transaction data. You will first watch a video that visually describes the process, and then you will implement portions of it.

1.1 Mining Bitcoins by Hand

Your instructor will first show the video,“Mining bitcoins by hand - The SHA-256 hash”, that can be found in the full writeup at:

`http://www.righto.com/2014/09/mining-bitcoin-with-pencil-and-paper.html`

In addition, you will be given a handout which is the final result of the video.

1.1.1 Binary, Decimal and Hexadecimal Numbers

Watching the video, you noticed that a lot of what is going on is converting from hexadecimal (base 16) to binary (base 2), and then comparing zero's and one's to come up with new binary values (that are eventually converted back to hex).

Python, by default, works with integers as decimals (base 10). However, it can directly convert to decimal from either binary (leading 0b) or hexadecimal (leading tt0x). For example:

```
0b0 -> 0
0b10 -> 2
0b101 -> 5
0x0 -> 0
0xa -> 10
0xff -> 255
```

1.1.2 Understanding Binary Strings

The easiest way to work with binary data in Python is using strings as a sequence of '0' (False) and '1' (True). As such, Python provides a built in function, `bin`, that takes an integer and produces a binary string from it. For example:

```
bin(0) -> '0b0'
bin(1) -> '0b1'
bin(12) -> '0b1100'
bin(45) -> '0b101101'
bin(0b11) -> '0b11'
bin(0xa) -> '0b1010'
```

In the same fashion, hexadecimal numbers are also represented by strings. The only difference is they use the `hex` function. It takes an integer and produces a string that starts with '0x' instead of '0b':

```
hex(0) -> '0x0'
hex(1) -> '0x1'
hex(10) -> '0xa'
hex(15) -> '0xf'
hex(16) -> '0x10'
hex(0b10011) -> '0x13'
hex(0x5fe) -> '0x5fe'
```

In addition, you can convert from either a binary or hexadecimal string to decimal using the `int` function and providing the string, followed by its base:

```
int('500', 10) -> 500       # default assumes base 10
int('0b1011', 2) -> 11
int('0x5bc', 16) -> 1468
```

1.2 Problem-solving Session (20%)

You will work in a team of three or four students as determined by the instructor. Each team will work together to complete the following activities.

1. What integers result from the following operations?
   (a) `15`
   (b) `0b100101`
   (c) `0x2a`
2. What strings result from the following operations?
   (a) `bin(83)`
   (b) `bin(0x2a)`
   (c) `hex(0b101101)`
   (d) `hex(299)`
3. We always want to be working with 32 bits of binary data. If an integer is less than 32 bits, we would like to pad it with leading 0's. Write a function, `padIntTo32Bits`, that takes an integer argument which is guaranteed to be 32 bits or less, and returns a 32 bit binary string out of it. For example:

```
padIntTo32Bits(11) -> '0b00000000000000000000000000001011'
padIntTo32Bits(0x6a09e667) -> '0b01101010000010011110011001100111'
```

4. The Majority box looks at the bits of three 32 bit padded binary strings, A, B and C. It compares each bit in A, B, C and constructs a new 32 bit binary string by computing whether the majority of bits in each position is '0' or '1', and taking each winner. Write a function `Ma` that takes the three 32 bit binary strings and returns the resulting 32 bit binary string. For example:

```
                  A = '0b01101010000010011110011001100111'
                  B = '0b10111011011001111010111010000101'
                  C = '0b00111100011011101111001101110010'
Ma(A, B, C) - >
                      '0b00111010011011111110011001100111'
```

1.3 Implementation (80%)

You will work with your partner on this assignment and deliver one implementation of the program `bitcoin.py`.

Download the starter code, `bitcoin.py` from MyCourses (Content...Labs...Lab 4 - Bitcoin). You are given a `main` function that should not be modified, and seven support functions that you will develop. Each function is already defined and gives detailed documentation about it. Do not change the function name or arguments.

Before you begin you should look over the `main` function and familiarize yourself with how it follows the steps performed in the video. It is suggested you implement the functions in the following order:

1. `padIntTo32Bits`
2. `Ma`
3. `rightShift`
4. `Sum0`
5. `Ch`
6. `Sum1`
7. `trimTo32Bits`

Keep in mind that all of these functions deal with binary strings. As such, the strings that are passed in, generated and returned, will all be prepended with `'0b'`, followed by the 32 bits of binary data represented by the strings `'0'` and `'1'`.

1.3.1 Sample Run

The main program outputs the results of each of the functions you are writing. Here is a sample run of a correct solution. It prompts for the input data halfway through execution. Here we are using the same input as in the video, `0x02000000`:

```
$ python3 bitcoin.py
Ma: bin=0b00111010011011111110011001100111 , hex=0x3a6fe667
Sum0: bin=0b11001110001000001011010001111110, hex=0xce20b47e
Ch: bin=0b00011111100001011100100110001100 , hex=0x1f85c98c
Sum1: bin=0b00110101100001110010011100101011 , hex=0x3587272b
Enter 32 bit input, e.g. 0x02000000:0x02000000
sum: bin=0b11110101011101111110110101101000 , hex=0xf577ed68
newA: 0xfe08884d
newB: 0x6a09e667
newC: 0xbb67ae85
newD: 0x3c6ef372
newE: 0x9ac7e2a2
newF: 0x510e527f
newG: 0x9b05688c
newH: 0x1f83d9ab
```

1.4 Grading

The majority of your grade for this assignment is based on the correctness of the seven functions your group implements.

- Functionality: 75%
  - `padIntTo32Bits`: 5%
  - `Ma`: 15%
  - `rightShift`: 5%
  - `Sum0`: 15%
  - `Ch`: 15%
  - `Sum1`: 15%
  - `trimTo32Bits`: 5%
- Code Style and Documentation: 5%

1.5 Submission

Transfer your program to the CS machines (using sftp). Submit your program before the deadline using `try`:

```
try grd-603 lab4-1 bitcoin.py
```