Explanatory Article

Team: The “Definitely CS Majors”

Project Software Engineers

Isaiah Mumaw - imumaw@nd.edu

Peter Ainsworth - painswor@nd.edu

Ed Stifter - estifter@nd.edu

Adam Mazurek - amazure2@nd.edu

Project Manager:

Julia Buckley - jbuckle2@nd.edu

Project Director:

Matt Morrison - Assistant Teaching Professor - University of Notre Dame - matt.morrison@nd.edu

Intro & Relevance

The purpose of our project was to create an understandable yet realistic tool to teach blockchain to individuals with limited programming experience. As this article will explain, we built a fully functional blockchain that is linked to an interactive webpage. At different points during this article you can see the concepts we talk about in action by visiting the webpage at: https://www.estifter.com/definitely-cs-project/.

Before we get into how blockchain technology works we first want to give a short explanation of why learning it is relevant. Due to the fact that blockchain is both new and rapidly growing, the potential uses have scaled up faster than the educational resources. Blockchain was only first conceptualized in 2008 and a majority of people had never heard of it until the rise of cryptocurrencies in the past few years, most notably the recent Bitcoin spike.

Students looking for jobs post-graduation may find opportunities involving blockchain. In 2021 alone there were $2.5 billion of venture capital investments into blockchain companies – a clear signal of industry growth and job opportunities. It is also not unlikely that this number grows exponentially as blockchain moves into fields like finance, healthcare or government.

Despite this, there are very few blockchain related courses offered in colleges, let alone highschools. Online resources are equally lackluster, and struggle to explain the abstract nature of blockchain and how the user interacts with the system behind it. This article, in conjunction with our website, will help to bridge this knowledge gap.

What is Blockchain (Top Level Explanation)

Most articles about blockchain describe it with words such as “decentralized ledger” or “immutable record for transactions,” but what does this really mean?

The goal of blockchain is to store a record of something in a more secure way than a conventional centralized database. For example, take a record of messages sent between people. The most straightforward way to do this would be to store every single message sent in a dataset, perhaps a linked list or hash table. While this is perfectly acceptable for many people, the drawback of this method is that there must be a secure central authority who we can trust.

Blockchain offers an alternative to this through the use of a hash function to encode the record of messages. A **hash function** takes an input and then converts it into a numerical value. What sets a hash function apart is that it is **once directional**, meaning that there is no way to convert it back to the input value. It is also very sensitive to the input, meaning that changing a single character in the input can generate a vastly different hash. While there are many different hash functions available, almost all of them create an output through manipulating the binary representation of the input data. For example, **SHA265** adds a 1 to the end of the binary representation and then adds 0's until the binary number can be divided by 512.

Example:

1) Lets try putting in two similar strings as input for the hash function

   input_1 = "message"
   input_2 = "message5"

2) Run each input through the hash function and see the output

   hash(input_1) = ab530a13e45914982b79f9b7e3fba
   hash(input_2) = 3de98fe6c314caf46f1c63b3596bc6

3) Notice how even though only one character is being added, the resulting hash is completely different

We can use this aspect of hashing to help the blockchain securely store information. The blockchain first makes an arbitrary initial hash. When a new piece of information is recorded, the blockchain program takes it and combines it with the previous hash. The new hash is then calculated from this, meaning that subsequent hashes are always dependent on the previous ones. In other words, any change further up in the blockchain results in changes throughout the chain.

Example:

1) **The blockchain starts with an arbitrary first hash:**

   root_hash = 0x02022

2) **A new piece of data is recorded by the blockchain:**

   data = "hello world"

3) **The blockchain takes the new data and it combines it with the previous hash. The string value of the hash is added to the string value of the message:**

   root_hash + data = "0x02022hello world"

4) **The combined value is rehashed to create a new hash:**

hash(“0x02022hello world”) = 91c90d3aa47114a58f538ebde60b308d3

5) **This process is repeated for any other new data recorded in the blockchain. Multiple new inputs can be added at the same time**

   Previous hash: 91c90d3aa47114a58f538ebde60b308d3
   New data: “multiple” , “messages” , “added”

   Combine: “91c90d3aa47114a58f538ebde60b308d3multiplemessagesadded”

   Rehash: hash(“91c90d3aa47114a58f538ebde60b308d3multiplemessagesadded”) = 031808f5c0e5ba66980680a525ecc7d80d10b

In the blockchain each new hash iteration is known as a block. The “chain” part comes from how each new hash relies on the previous hash in the chain.

Try it yourself:

1) Go to https://www.estifter.com/definitely-cs-project/

2) Type in a message into the box on the left. This will be the new data added to the next block. Hit send.

3) Look on the right hand side of the page and wait a few seconds. A new block will appear at the bottom of the list. Notice how it includes the previous hash, added message and then the new hash. The code running on the website took your new message, added it to the previous hash and then reshashed it all to get the new current block hash.

One of the biggest benefits of this method is that it is extremely difficult to tamper with the record of data that the block chain recorded. For example someone using the blockchain might want to change something they previously added to it. They would first need to recompute the hash in the block being changed, then recompute the hash of all following blocks as all blocks reference their previous block. This process requires immense computational power making modifications virtually impossible to perform. This aspect of the blockchain is what makes it appealing for recording financial transactions.

The last aspect about blockchain that makes it special is that it is decentralized. This means that instead of a single central chain every user has their own copy of the blockchain. Each copy of the blockchain is referred to as a node. Because the hash function is immutable everyone will have the same copy. When new information needs to be stored on the blockchain nodes can start mining (essentially solving the next hash from the previous). Whatever node solves it first sends its results to all the other nodes. They then double check the calculations. When a majority of the nodes agree that the new block is correct it is accepted into the chain.

At this point you should have a solid understanding of how a blockchain works. In the next sections we will go into how a blockchain is coded as well as diving into some of its more challenging aspects such as proof-of-work and mining.

Blockchain Structure and Classes

We built our blockchain code entirely in Python. Python is a highly popular, general-use, and high-level language, so most users with some programming background will readily be able to understand code. It also utilizes OOP (object-oriented programming), giving us access to thousands of code libraries which will reduce complexity and increase functionality. The following sections will break down the basic structure and operation of the code.

At the core of our blockchain were a set of three classes, which provide definitions for objects which can be created, interacted with, and destroyed as needed. Inside each class are members, which define the attributes of the object, and methods, which define the behavior of the object through the use of functions.

The most elementary structural unit of a blockchain is a transaction. Transactions can be any exchange of information or data between two endpoints, and we contain these within a class structure as described above. For our implementation, transactions took the form of simple messages, rather than monetary exchanges, as we wanted to teach about the fundamentals of blockchain without becoming distracted by the complexities and connotations of modern cryptocurrency markets.

The message structure is implemented as a user-defined class. Inside this class are 4 string-type members which define the message, reminiscent of an email message:

1. **Sender**: Who sent the message (the origin point)
2. **Receiver**: Who will get the message (the destination point)
3. **Time**: When the message was sent
4. **Content**: What the message says

Try it yourself:

1) Go to https://www.estifter.com/definitely-cs-project/

2) Pay attention to the sending a message feature on the left. Notice how the fields include: to, from and message. These are the sender, receiver and content members of the transaction class

3) **Where does the time come from?** The time is automatically recorded when the message is sent using Python's datetime module.

4) Try typing a simple message and sending it. You'll see it appear momentarily! Remember that messages cannot be deleted or modified – we'll get into this later.

Two additional functions are present inside the message class which are designed for output generation in either multiline string format or HTML code. String output is primarily used for command line tests and is mostly relegated to development tasks. HTML output is used for display on the webpage.

As in a typical blockchain structure, messages can then be grouped together and stored within blocks. Blocks are implemented as another user-defined class, with messages being stored in a Python list. Python lists are extremely versatile, capable of storing objects of any type in an indexed array, similar to a C++ vector, however unlike vectors the object type does not have to be constant across all indices. Lists, like vectors, can be extended or reduced as needed.

In addition to the list of messages, there are members which contain the hexadecimal (base 16) value of the block's hash, the hexadecimal hash of the previous block, and a nonce (number only used once) value. We will explain what the nonce value does in the Proof of Work section. While by default the block's hash value is set to 0x0 (hexadecimal for 0), the block contains the compute_hash function, which chains all other block members (messages, previous hash, and nonce) together in a single string and then computes the true hash using this string and the SHA256 algorithm.

Much like the message class, there are also additional functions which create output in both string and HTML form, creating visually appealing drawings that can be used for testing or website implementation, respectively.

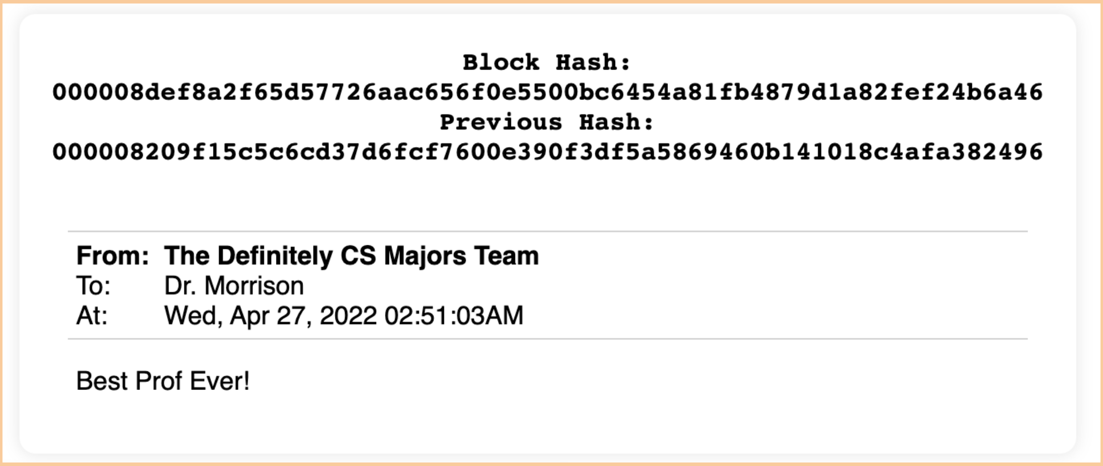
Try it yourself:

1) Go to https://www.estifter.com/definitely-cs-project/

2) Send a message by filling out the form, and then wait for the block to appear on the right.

    a) As you'll learn soon, adding a block takes quite a bit of computational power, so this may take anywhere from a few seconds to a few minutes. As a result, a single block can easily contain multiple messages.

3) The displayed block will show the resulting hash, as well as the hash of the previous block, with the contents of the message(s) at the bottom. Notice how the classes are starting to build on each other!



Block Hash:
000008def8a2f65d57726aac656f0e5500bc6454a81fb4879d1a82fef24b6a46
Previous Hash:
000008209f15c5c6cd37d6fcf7600e390f3df5a5869460b141018c4afa382496

From: The Definitely CS Majors Team
To: Dr. Morrison
At: Wed, Apr 27, 2022 02:51:03AM

Best Prof Ever!

Blocks are managed using the blockchain class, which stores them inside of a Python list in much the same way that blocks store messages. Newly added blocks are appended to the end of this list, such that the order of the blocks is always preserved.

While adding a block sounds relatively straightforward, the new hash of the block must obey certain rules, meaning that new blocks cannot be immediately added. The process of acquiring a new block is known as "mining," and will be discussed in the next section. Suffice it to say, mining requires significant computational power in order to complete, and there are quite a few interacting functions which are involved in the process.

In addition to mining, the blockchain must also store received messages until they can be added to a new block. This is done using another Python list, but our implementation is similar to a queue, meaning that the messages are accessed in "first in, first out" order, or FIFO. As soon as a new message is received, it is added to the queue, and as blocks are added, this queue is emptied into them. Thanks to multithreading, this operation does not interfere with the mining function, meaning that as soon as the queue is not empty, mining can begin in the background.

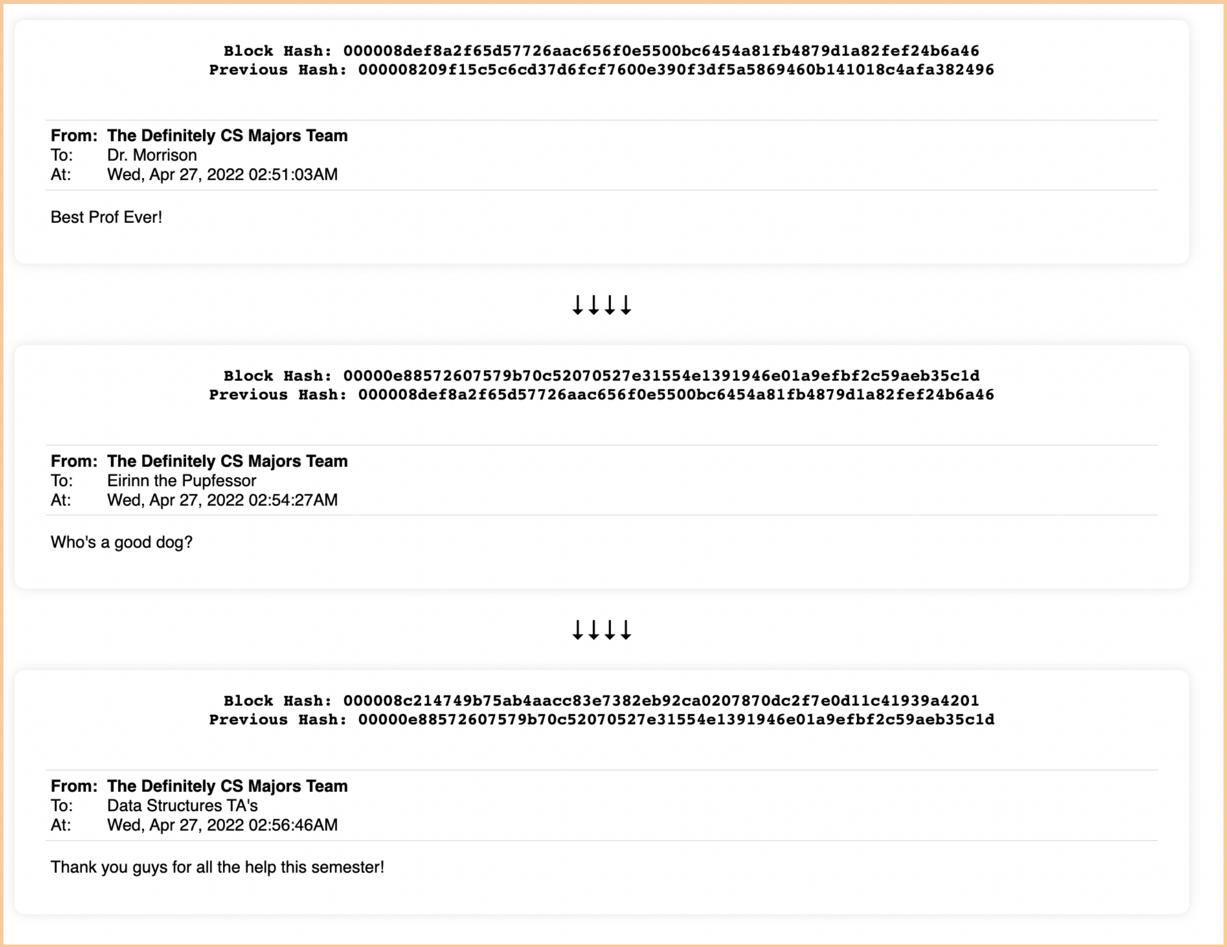
Finally, as before we have also included some basic functions that produce string and HTML output.

Try it yourself:

1) Go to https://www.estifter.com/definitely-cs-project/

2) Type in a few messages by filling out the form. You'll start to see blocks appear with your messages. The blockchain

3) Notice how the blocks on the right side of the page are linked together. This is how the individual blocks are included as members of the blockchain class.



Block Hash: 000008def8a2f65d57726aac656f0e5500bc6454a81fb4879d1a82fef24b6a46
Previous Hash: 000008209f15c5c6cd37d6fcf7600e390f3df5a5869460b141018c4afa382496

From: The Definitely CS Majors Team
To: Dr. Morrison
At: Wed, Apr 27, 2022 02:51:03AM

Best Prof Ever!

↓↓↓↓

Block Hash: 00000e88572607579b70c52070527e31554e1391946e01a9efbf2c59aeb35c1d
Previous Hash: 000008def8a2f65d57726aac656f0e5500bc6454a81fb4879d1a82fef24b6a46

From: The Definitely CS Majors Team
To: Eirinn the Pupfessor
At: Wed, Apr 27, 2022 02:54:27AM

Who's a good dog?

↓↓↓↓

Block Hash: 000008c214749b75ab4aacc83e7382eb92ca0207870dc2f7e0d11c41939a4201
Previous Hash: 00000e88572607579b70c52070527e31554e1391946e01a9efbf2c59aeb35c1d

From: The Definitely CS Majors Team
To: Data Structures TA's
At: Wed, Apr 27, 2022 02:56:46AM

Thank you guys for all the help this semester!

At this point you should be able to deeply understand the blockchain elements and how they work together. The next few sections will explain the hashing process and how it intrinsically links all blocks together.

Proof of Work and Mining

To begin the mining process, we need to implement the proof-of-work feature. With what is already established, someone with substantial computing power could theoretically alter tailing blocks of a blockchain, then recompute every following hash. Implementing a proof-of-work algorithm makes this even harder to do. Simply put, blockchain's **proof-of-work** is its process of verifying each transaction within a block, then verifying the hash of the previous block before adding a block to the chain. This ensures all nodes in a blockchain network are in agreement to hash of a block, preventing one node from tampering with what has already been established.

To illustrate how this process works, let's first introduce blockchain mining. **Mining** is the process of computing the hash value for a block to be added to the block chain by solving a complex mathematical problem. This mathematical problem is, in most cases, a restriction on what the next hash can be. In our case, we have set that the computed hash must begin with a certain number of 0's. The number of 0's is known as the **difficulty** level, and the more 0's there are, the harder it is to find that hash. Take a look at our code below to see how this works.

```
  def proof_of_work( self, block ):

       #set number only used once to 0
       block.nonce = 0

       #first hash calculation
       computed_hash = block.compute_hash()

       #loop until hash meets difficulty requirements (for high difficulty, this may take a
while)
       while not(computed_hash.startswith('0' * self.__difficulty)):

           #increment nonce and get new hash
           block.nonce += 1
           computed_hash = block.compute_hash()

       return computed_hash
```

Our code simulates solving this mining process by adding a number only used once value (or **nonce** value) to the block. Because of the hashing process, changing this value changes the hash of the block. Thus, our code increments this number by 1 until a hash is found with the appropriate number of leading 0's. This brute force process requires substantial computing power and can thus take several minutes to several hours just to find the hash of one block. Setting a difficulty of 5 took a few seconds per hash on a 6-core machine, but setting a difficulty of 6 took up to two and a half minutes.

To summarize, adding one block takes substantial computational power depending on the difficulty. Thus, given the reverse-linked nature of blockchain hashing, changing the contents of one block causes an immense ripple effect that cannot be solved. This makes block-chain immutable.

Conclusion

At this point, you should have a solid understanding of how a blockchain works. There really is not much more to it! If you are interested in seeing what a real-life blockchain looks like, check out the bitcoin whitepaper. You will find that even bitcoin, the canonical example of a blockchain, is described in only a handful of pages. If you are still hungry for more, feel free to check out the code for our complete project for a more thorough look at how a blockchain is constructed.