Assignment 1

# Question 1

The total reward of first blocks is , according to the question, we can deduce that . So, the total amount of currency that can be issued is

when r is larger than one. While the total amount grows to infinity when r is not greater than one.

# Question 2

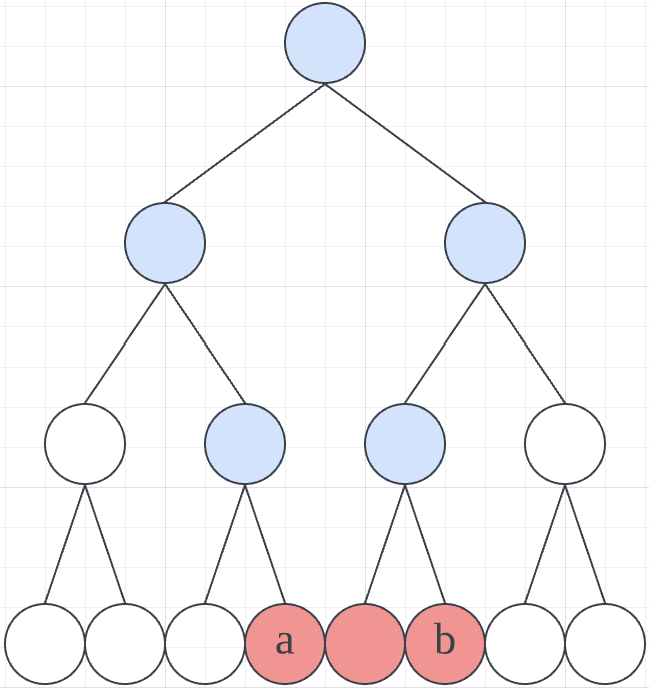
1. *Describe the data structure stored on the server, and how it is computed. Give an expression for the size of this data structure as a function of the length of L.*

**For clarity, is used to represent the length of and is used to represent the length of answer to the queries. Also assume that the size of hash code is bits.**

Merkel tree can be used in this question for generating proof information. But directly use Merkel tree is time consuming and the tree may take much more space comparing to the original list. The server may pass a subset of the Merkel tree to client as proof information. And the subset is simply union of single verification paths of each element in the answer to the query.

Because the number of nodes is two times of number of leaf nodes, as grows, the size of it would eventually becomes . As time complexity of validating is proportional to size, the time complexity is also . When is a small proportion of , the complexity is near to .

In addition to time complexity of verifying each node in the returned tree, there are also some “wasted nodes” in the tree: lots of nodes in the returned data are used for validation only.



The coloured nodes in the diagram are the data sent to client while red nodes are the nodes relevant to the answer to the query, and the blues are only used for validation.

**Binary search tree (BST) is an efficient data structure to optimize Merkel tree:**

As given in the question, the data is static, so efficiency of inserting or deleting data is not considered. Therefore, in order to store the number list, complete BST can be used to speed up the search operation. The server could use an array to represent the tree by defining the following additional rules:

Define the index of root as

Define the index of left child of element as

,

and that of right child as

.

Then the index of parent of element becomes:

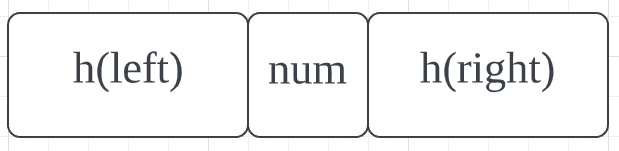
.

**Structure and computation:**

In stead of storing hash value only, each node in the BST Merkel tree stores both hash values of children and number value of itself, as illustrated below:



A node in standard Merkel tree



A node in BST Merkel tree

In the diagrams above, h(child) is obtained by calculating the hash value for concatenated h(left), num, and h(right) of that child. And the leaf nodes store the number only.

The construction of the tree is almost the same as a standard BST, in addition to it, the hash value for each node is computed.

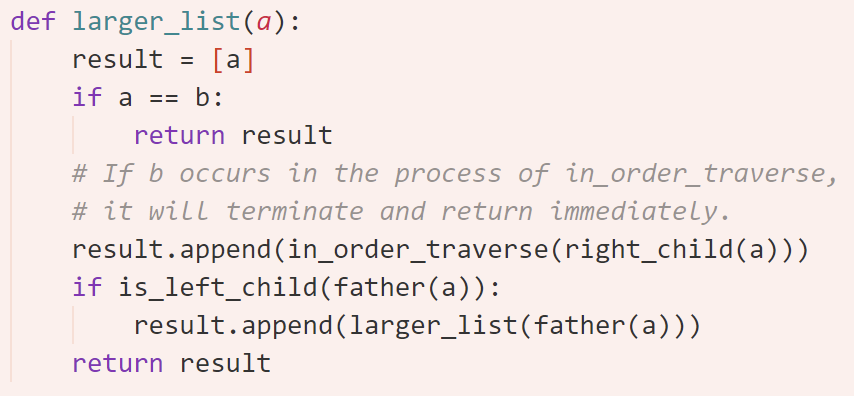
**Answering queries:**

The way of getting all the numbers between and is to first search and then get numbers that are greater than until .

The searching process is the same as that of a regular BST which time complexity is . The algorithm for getting the number list is recursively adding the right child tree of the father nodes until the current node is right child of the father node or is met. The time complexity of this process is .

And the way of adding right child tree is simply done by in-order traversing the right child.

The pseudo code is:



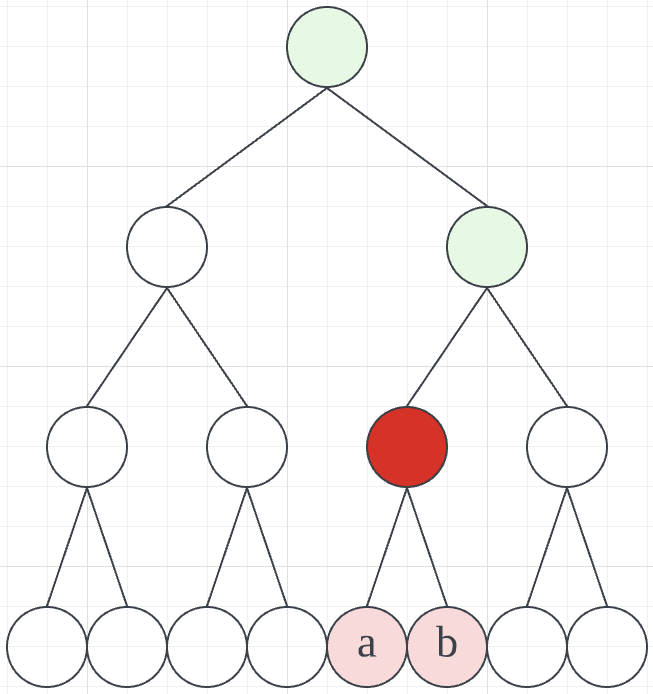
Because complete tree is used, the array is compact so that nearly no space is wasted. Therefore, the memory used to store the BST Merkel tree would be bits.

1. *Describe the data that is stored at the client, and state its size.*

The data stored at the client consists of two parts: answer and verification information. The answer is temporarily stored as a list from to in ascending order whose size is bits. Verification information is the root node of the BST Merkel tree, the size of it is bits.

1. *Describe the “proof information” and state its size as function of R*

The proof information is a subset of the BST Merkel tree which consists of all nodes in the answer list and all nodes on the shortest path from root to the answer list. For example:



The coloured nodes correspond is the data returned to the client with “top of the answer list R” coloured deep red. The path from root to deep red is the desired proof information. Client may directly use the deep red node as starting point to verify the correctness. The calculation is the same as computing the tree. The final value of root should match the stored version if the data is not corrupted.

The size of the green part is just the depth of the top node of , it is approximately .

If the client doesn’t trust the value stored in deep red and really desire to perform the validation thoroughly, they can choose to start from the deepest nodes. In this case, the proof information takes up .

1. *Describe the computation that the client performs to verify the proof state its size as function of R. Explain how this computation provides a guarantee.*

The client has two option to verify, one is to verify the existence of the answer list, another is to also verify the correctness and integrity of the list.

To verify existence of the list, the client just starts from top of R (deep red node) to perform the following computation recursively:

1. Concatenate , and to yield .
2. Pass to parent node, if this node is left child, pass to , vice versa.
3. Compare and hash value of parent, if not equal, then warn use that the data is corrupted.
4. Until root node.

This process is similar to proving existence of data in a Merkel tree. The running time is . To verify the correctness of the list, the client should follow the procedure below:

1. Add all leaf nodes to an array , then dispose these nodes.
2. Calculate and pass hash value as above.
3. Until top of the answer list.

The process guarantees that each node in the answer list exists in the correct place in the BST Merkel tree, thus the correctness and integrity of the answer is proved. The running time is .

The existence option is compulsory, when doing this, the client assumes that the answer list is complete and correct. This guarantees that the returned answer list is indeed a subset of the BST Merkel tree. They can also choose to verify the correctness and completeness if they want more certainty.

The combined complexity is . Which means that, for a given list (then becomes constant), the combined running time is linear to ().

By using collision-free hash functions, the data structure is guaranteed unchangeable because each hash value corresponds a unique subtree.

1. *Is your solution the most efficient possible solution for this special case?*

In this case, the client just needs the hash value chain in Merkel tree for the last number as proof information. And the information stored is the hash value of the last number in previous query. The verification process is to calculate hash sequence from to starting from stored by client and then check the existence of through Merkel tree. The time complexity and complexity are .

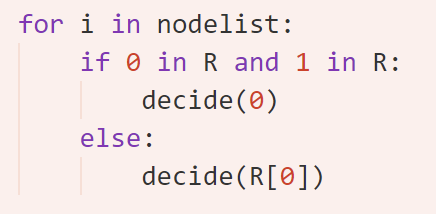
Although their complexities are the same in big O notation, my solution is less efficient in constant level. My solution uses bits more and concatenating two hash values and a number, it also costs more time to verify the subtree than directly computing the hash chain.

# Question 3

1. *Show that each node is able to compute a decision at time 1*

Because the network is fully connected and there is no faulty node, each node will know the *vote* information of all other nodes at time 1. Thus, they can make choice based on the same full information. If both 0 and 1 are available, each node just decides 0 by default.

Pseudo code:



Because the decision is made based on R, and R is the same for each node. Due to fully connectivity, R[i] is union of all {} sets so can make decision that satisfy Validity. And because they are using the same decision rule and same information, the Agreement specification also holds.

1. S

# Question 5

1. *Recombining the p + 1 shares allows k to be reconstructed.*

Firstly, add up the shares and then take modulus of n to yield :

Because and are constant, can be reconstructed by solving the congruence function where and .

To ensure that the function is solvable, we need to keep , that is, .

Also, unique solution is needed to guarantee that the solution is exactly , another constraint becomes . In other words, and need to be coprime. This also implies that .

*2） An attacker who learns any p of the p + 1 shares still gets no information about k.*

Now assume that the attacker can work out with only shares.