Q1. For our first problem, the goal will be to design a data structure known as a **Least Recently Used (LRU) cache**. An LRU cache is a type of cache in which we remove the least recently used entry when the cache memory reaches its limit. For the current problem, consider both get and set operations as an use operation.
Your job is to use an appropriate data structure(s) to implement the cache.

- In case of a cache hit, your get() operation should return the appropriate value.
- In case of a cache miss, your get() should return -1.
- While putting an element in the cache, your put() / set() operation must insert the element. If the cache is full, you must write code that removes the least recently used entry first and then insert the element.
  All operations must take O(1) time.
  For the current problem, you can consider the size of cache = 5.

Q2. For this problem, the goal is to write code for finding all files under a directory (and all directories beneath it) that end with ".c"

Q3. A Huffman code is a type of optimal prefix code that is used for compressing data. The Huffman encoding and decoding schema is also lossless, meaning that when compressing the data to make it smaller, there is no loss of information.
The Huffman algorithm works by assigning codes that correspond to the relative frequency of each character for each character. The Huffman code can be of any length and does not require a prefix; therefore, this binary code can be visualized on a binary tree with each encoded character being stored on leafs.
There are many types of pseudocode for this algorithm. At the basic core, it is comprised of building a Huffman tree, encoding the data, and, lastly, decoding the data.
Here is one type of pseudocode for this coding schema:

- Take a string and determine the relevant frequencies of the characters.
- Build and sort a list of tuples from lowest to highest frequencies.
- Build the Huffman Tree by assigning a binary code to each letter, using shorter codes for the more frequent letters. (This is the heart of the Huffman algorithm.)
- Trim the Huffman Tree (remove the frequencies from the previously built tree).
- Encode the text into its compressed form.
- Decode the text from its compressed form.

You then will need to create encoding, decoding, and sizing schemas.

Q4. In Windows Active Directory, a group can consist of user(s) and group(s) themselves. We can construct this hierarchy as such. Where User is represented by str representing their ids.

Write a function that provides an efficient look up of whether the user is in a group.

Q5. A **Blockchain** is a sequential chain of records, similar to a linked list. Each block contains some information and how it is connected related to the other blocks in the chain. Each block contains a cryptographic hash of the previous block, a timestamp, and transaction data. For our blockchain we will be using a **SHA-256** hash, the **Greenwich Mean Time** when the block was created, and text strings as the data.
Use your knowledge of linked lists and hashing to create a blockchain implementation.

Q6. Your task for this problem is to fill out the union and intersection functions. The union of two sets A and B is the set of elements which are in A, in B, or in both A and B. The intersection of two sets A and B, denoted by $A \cap B$, is the set of all objects that are members of both the sets A and B.

You will take in two linked lists and return a linked list that is composed of either the union or intersection, respectively. Once you have completed the problem you will create your own test cases and perform your own run time analysis on the code.