\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Explanation problem 1 : Least Recently Used Cache

For this problem, I have used `OrderDict` from `collections` module. After instantiating class with max capacity, we

can use `set()` method to set cache and assign value at specific keys. All keys are `None` until assigned. It will try

to assign value to given key by removing the value if assigned else after checking capacity and key item is removed

from dict and value is assigned again. To get value of the key, we can use `get()` method.

Time complexity of `get()` is `O(1)` and of `set()` is `O(1)`. Space complexity of the LRU Cache is `O(n)`.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Explanation problem 2 : Finding Files

For this problem, I have used zipfile. zipfile.ZipFile is used to write and read the Zip files. It has some methods which are used to handle the Zip files.

By using namelist() we created a list which contains all the subdirectory paths from testdir.zip file.

By using endswith() we check for file and with extension by suffix passed.

Time complexity is `O(n)` and space complexity is `O(n)`.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Explanation problem 3 : Huffman Coding

For this problem, I am calculating occurrences of each characters in data string, sort and encode with minimum code

length.

Time complexity for this is `O(nlogn)` and space complexity is `O(logn)`.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Explanation problem 4 : Active Directory

For this problem, I two things. First I am checking if user is group's name or user is in group users, return true if

matches else using recursion, follow same process .

Time complexity of this process is `O(depth \* users)` and space complexity is `O(depth \* users)`

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Explanation problem 5 : Blockchain

There are two ways to solve this problem. Create blocks and append one on another or create linked list and append

blocks in list. Using linked list, I have initialized head and tail where head is first block or starting block of list

and tail is last block or last element of list.

Time complexity for appending a block is `O(1)` and space complexity is `O(n)` which is total space that is occupied

by blockchain.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Explanation problem 6 : Union and Intersection of Two Linked Lists

For this problem, first I created a set for making union & intersection of the two linked list. Then I have created a

new linked list (LinkedList object) from that set.

Time complexity for union is `O(n)` and space complexity is `O(n)`.

Time complexity for intersection is `O(n)` and space complexity is `O(n)`.