DESCRIPTION:

implement a Python program that can be invoked with the following options and arguments:

**identic [-f | -d ] [-c] [-n] [-s] [<dir1> < dir2> ..]**

**-f -> to find the identical files (default is -f)**

**-d -> to find the identical directories**

**-c -> identical by content (default is -c)**

**-n -> identical by name**

**-s -> gives the sizes of the outputs (This option is ignored when -n given)**

**[ <dir> ..] -> is the directories you want to search for (default is current directory)**

The identic program will traverse the given directories and look for files or directories that are duplicates of each other (i.e. identical). The full pathnames of duplicates will be printed as output (There will be a new line between the sets of duplicates).

PROJECT DETAILS :

First of all, I decided to structure of the project and learnt the function which I will use during implementation (sha256 and argparse). At the beginning, I thought about implementing it with OOP to build a directory tree, however I found out that os.walk(path) is already doing the same thing. So, I moved on with that.

THE STUCTURE OF THE PROJECT:

(Important note : takes inputs as argument and print output to the console)

VARIABLES:

1. “type” (integer) : keeps the meaning of identical. If by content (-c option), it is set to 1, for -n option 2, for -cn option 3. Default is -c option. So, it is set to 1 at the beginning.
2. “typefd” (integer) : keeps the data (file-1 or directory-2) type we are searching for. Default is -f option. So, it is set to 1 at the beginning.
3. “typesize” (integer) : stores if size flag is set(1), otherwise 0. Default is 0.
4. “dict” (dictionary) : keeps ‘HashValue’ : Size(of the directory or file) It stores all hash values we found.
5. “dict2” (dictionary) : keeps ‘HashValue’ : List of Paths ( of the files or directories which have the same hash value) It stores all hash values we traversed.
6. “dirDict” (dictionary) : keeps ‘Path of a directory’ : [Its size, Its hash value]

(These two are used in traversing directories block)

1. “currentDic” (dictionary) : dictionary for current root we are searching. Keeps ‘Hash value’ : Size (of the each directory or file inside the root)
2. “currentDup” (dictionary) : dictionary for current root we are searching. Keeps ‘Hash value’ : List of paths ( of the files or directories which have the same hash value inside the root)

(These two used to print paths)

1. “finalDic” (dictionary): We use this to sort paths alphabetically and by their sizes. It keeps ‘HashValue’ : [Size, List of duplicates]
2. “groups” (List): Stores the duplicate groups.

ARGUMENT PARSE:

I created an Argument Parser (“parser”), I used it to detect -c, -n, and -s options and to store given directories(in dirs). Also, created a mutually exclusive group (“group”) to detect either -f or -d options and I set the type, typefd and typesize variables according to arguments.

HELPER FUNCTIONS:

hashFile( type, path) : type is the same variable we mentioned before(-c,-n,-cn), path is the path of the file.

If only -c flag is set, it reads the content of the file in byte (it handles png, txt, mp3 etc.) and hashes the encoded content by hashlib sha256. Then returns this value as hexdigest.

If only -n flag is set, it takes the name of the file and hashes the encoded name and return.

If both of them are set, it creates a tuple like ( hash value for -c, hash value for -n) and returns it.

**hashDir( type, sum, path) :**

type is the same variable we mentioned before(-c,-n,-cn), sum is the concatenated (and sorted alphabetically) hash values of the files and directories inside the directory we want to hash, path is the path of the directory we want to hash.

If only -c flag is set, the content of the directory equal to the sum(all files and directories inside it). So, it just hashes the sum and returns it.

If only -n flag is set, it takes the name of the directory, hashes it, then concatenates with the sum and again hashes it, finally returns the value.

If both -c and -n flags are set, it creates a tuple like ( hash value for -c, hash value for -n) returns it.

HOW TO TRAVERSE AND FIND THE DUPLICATES:

My code traverses directory tree from bottom up by os.walk(path, topdown=False), and gets root directory, its children directories and files. At first, it gets the absolute path of the root, because we want all path as full path. Then creates currentDict and currentDup as empty dictionary.

For every file in the current root, it gets the full path of the file and its size. Then gets its hash value by hashFile function. If “typefd” is set to 1 (-f is invoked), it adds the hash value(key) and the size(value) to the “dict” and adds the hash value and full path to the “dict2”. If the hash value is already added to the “dict”, It just appends the path to the “dict2[hash value]”.

Regardless of the value of “typefd” it adds hash value and size to the “currentDict” and hash value and path to the “currentDup” (in the same manner with “dict” and “dict2”)

If we are searching for directories ( -d is invoked), for every directory in the root, it gets the full path and its size and hash value from the “dirDict” (we update it while traversing), then updates the “currentDict” and “currentDup”.

If -d is invoked, it sorts the “currentDup” alphabetically by keys which are hash values then, concatenates all hash values in “newHash” variable, also gets total size of the directory. Then, it gets the hash value of the root via “hashDir” function. If this hash value is already in “dict”, just appends the path to the “dict2[hash value]” otherwise, update both “dict” and “dict2”. Finally, adds the root to the “dirDict” with its size and hash value. It means we are done with this directory. So, even if we are not traversing the directories, it just adds path to the dictionary.

We run this block for every directory inside the args.dirs but before running, it checks if the directory is traversed before by looking inside the “dirDict” dictionary. If it is, it just continues to the next directory.

PRINTING :

At first, we sort the every path list in dict2[\*hash value here\*] (where we store duplicate groups) alphabetically, we also get the sizes of these duplicate groups and add the sorted lists and sizes to the “dicFinal”.

Then, sort the “dicFinal” by the first element inside the path lists. (first path in duplicate groups)

If -s flag is set, sort the “dicFinal” again by the sizes of the duplicate groups.

At the end, for every hash value in dicFinal, we get the duplicate groups and their sizes.

If -s flag is set, we print the groups with full path (tab) and size values. Otherwise, print only full path.

References:

CMPE 230 resources

<https://docs.python.org/3/library/argparse.html>

<https://thispointer.com/python-how-to-sort-a-dictionary-by-key-or-value/>