Zuhal Didem Aytaç
2018400045

CMPE 230 SYSTEMS PROGRAMMING PROJECT DOCUMENTATION

This documentation expresses my approach to implement a Python program will traverse the directories and look for files or directories that are duplicates of each other (i.e. identical)

General Structure / Approach

The program reads the arguments with the use of ***argparse*** library. I selected different traversing approaches for -f and -d options, they will be specified below. The program compares files and directories based on their hash values, which are calculated accordingly to given options. When all hash values are calculated for directories given as argument, the program filters files or directories with the same hash value and outputs properly.
everything = defaultdict(list) : dictionary that holds hash values and corresponding file or directory paths
knownSubDirectoryHashes = dict() : dictionary that holds pre-calculated directory paths and hashes of those directories (used with -d option only)

Argument Passing

The program makes use of the ***argparse*** library to parse command line arguments. -f and -d options form a mutually exclusive group. When all arguments are read, the program makes sure that if nothing is specified, default arguments are taken into consideration.

Detecting Identical Files

The program starts with the directory given as argument (***args.dirList***) and looks the contents. If it encounters a directory, the directory is appended to the dirList for future traversal. If the program encounters a file, the hash value of the file is calculated as specified by the arguments. The {file_hash:full_file_path} pair is appended to the dictionary named *everything*.

Detecting Identical Directories

The program makes use of the ***os*** library to detect identical directories. The walk function is used with bottom-up option. For every directory, a list named ***hashlist*** is generated and holds the hash values of subdirectories and files in that directory. For files, the hash value of the file is calculated as specified by the arguments and appended to the *hashlist*. For directories, the hash value of the directory is got from the knownSubDirectoryHashes and appended to the *hashlist*. Names of the subdirectories and current directory are hashed and appended to the *hashlist* if necessary. After all necessary hashes are calculated, the *hashlist* is sorted and the hash of the current directory is calculated by concatenating those hashes in *hashlist* and taking the hash again.
The *{direcroty_hash:full_directory_path}* pair is appended to the dictionary named *everything*.
The *{ full_directory_path:directory_hash }* pair is appended to the dictionary named *knownSubDirectoryHashes*.

Filtering Duplicates

duplicatesSize = list() : dictionary that holds list of detected duplicate groups and their size as [(size,duplicate), …],[(size,duplicate), …]
duplicates = list() : dictionary that holds detected duplicates
sizeList = list() : list for each duplicate group holds (size, duplicate) pair
def directorySize(dir): function that calculates size of a directory recursively, summing the sizes of all files in its subdirectories
After the traversal of all argument directories and calculations are concluded, the program has hashes of all files/directories in the dictionary named *everything*. If a key has a value of length>1, this means a duplicate.
If the sizes are to be printed, *(size,duplicate)* pair for each group appended to the list named *sizeList*. If not, duplicate groups are appended as lists to the list named duplicates.

Printing Output

The full paths of duplicate groups are printed in lexical order, placing an empty line between different groups of duplicates. If the s option is specified, the duplicates are printed in descending order of size, every group in lexical order, and the sizes are also printed at every line.

Conclusion

My project reads and executes the arguments properly, and detects the duplicates properly, giving the correct output. No errors are detected, the program handles the necessary edge cases. In conclusion, I have properly implemented a program will traverse the directories and look for files or directories that are duplicates of each other, using Python3.