Journal of Theoretical and Applied Information Technology
28th February 2022. Vol.100. No 4
2022 Little Lion Scientific

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

DESIGN AND IMPLEMENTATION OF A 3-TASK CONSTRUCT FOR MANAGING REDUNDANT FILES

EMMANUEL CHUKWUDI UKEKWE[1], HYACINTH AGOZIE ENEH[2]
[1,2]DEPARTMENT OF COMPUTER SCIENCE, UNIVERSITY OF NIGERIA, NSUKKA
E-mail: [1]emmanuel.ukekwe@unn.edu.ng, [2]agozie.eneh@unn.edu.ng

ABSTRACT

The operating system ensures efficient memory and disk space management which contributes a lot to computing speed. However, not all users have the technical expertise to make use of operating system tools for disk space management. This paper therefore proposes and implements a 3-task user friendly construct for managing redundant files which inadvertently occupy so much space in the computer disk and slow down the system. Redundant files such as duplicate files, temporary files and dormant files which are often taken for granted are detected. The construct employs the concept of hashing by making use of python hashlib.SHA-1 to detect duplicate files while dormant files are identified using file attributes such as date of creation and access date. The deduplication construct earmarks redundant files for deletion or compression in order to free disk space. Algorithms for implementation of the constructs are presented. The construct was implemented using python programming language. In order to test the efficacy of the construct in detecting file duplicates, the New York Times Comments for March, 2018 dataset was extracted from www.kaggle.com and used for that purpose. The simulation recorded an efficiency of 99%. The Construct is designed to provide alternative user friendly disk management support tools for computer users other than that provided by Operating systems.

Keywords: *Deduplication, Duplicate, Redundant, Disk-Space, Compress, Hashing*

1. INTRODUCTION

Disk space is a much coveted computer resource which determines to an extent the speed as well as smooth operation of the computer system. A major bottleneck of effective disk and storage management is the accumulation of junk or redundant files within the system. Redundant files are files which are temporarily used by the computer system in the course of its operation. However, as soon as they serve their purpose, such files become junk which ought to be removed. Redundant/junk files include rarely used files, duplicate files, temporary internet files, temporary system files, cache files, thumbnails, cookies, log files etc. Redundant/junk files take quite a chunk of disk space and keeping them in the system is really unnecessary. For instance, rarely used files remain unused because the reason for tagging it such often do not change. As a result, such files may indeed remain unused. The concept "rarely used" in this context therefore refers to a very long period of time spanning across years. According to the author in [1], as people age, they age into new and much broader territory. As individuals evolve, their computational needs advance both quantitatively and qualitatively. Hence, a file created today will serve today's needs but will definitely need some upgrading to fit into the needs of tomorrow. When a file is updated, it becomes a new file, hence, it is therefore unnecessary to encourage accumulation of rarely used files in the system which continue to occupy space. Such files are better compressed or even deleted in order to free space.

Similarly, keeping duplicate copies of files is also unnecessary. A user may decide to update a copy of a file whilst keeping the original. In such situation, a new and different file is created and may not be regarded as a duplicate of the former because extra or different content has been added. For this reason, keeping duplicate files in the same computer system is irrelevant and not advisable. Besides that, the constant use of the computer for routine work attracts deposition of [1]temporary files, log files as well as cache files on the computer. It can be argued that such temporary files are needed by the computer especially to initiate an installation

[1] Copyleaks "Duplicate File finder", Copyleaks Technologies, LTD https://copyleaks.com/duplicate-file-finder

or to run a program, however, they occupy unnecessary space which could be utilized otherwise.

Apart from the nuisance associated with rarely used and duplicate files, traces of cache files, log files, cookies and temp files also constitute another major challenge in disk space management. Each time a web page is visited, the browser stores information about the page by caching it. Cookies on the other hand are created based on request from the web page. A cookie file is created and stored in the computer for ease of use by requesting web pages. Temporary files on the other hand could be a tool in speeding up previously visited web pages as agued by the author in [2]. However, these files discretely continue to accumulate within the disk as the user engage in downloads, internet activities and routine use of the computer. According to the authors in [3], temporary files are rampant with applications such as Microsoft word and they are used for tracking any changes made to the original and also for recovery if the program crashes. However, they also agree that dozens of temp files can be created while working on Microsoft word thus constituting a nuisance. The author in [4], on the concept of workspace management, states that temporary files ought not to be left in the system forgotten. All these attest to the fact that temporary files serve their purpose and ought to be disposed as soon as their purpose expires.

Each browser handles cookies, temporary files and log files differently. These files could be deleted by the user through rigorous processes as enumerated in [5]. The process requires user's indebt knowledge and understanding of the browser operations. In contemporary computer use, more than one browser could exist in a computer, it becomes a bit tasking for the user to get acquainted with almost every browser in order to delete temporary files and cookies. In addition, there is no definite operating system command that explicitly identifies rarely used files. Identifying duplicate files through operating system command is dependent on the user's level of expertise as it is not usually a direct command that will produce desired result. It is therefore necessary to have an intelligent construct that identifies and locates temporary files, duplicate files as well as redundant files within the computer disk. These files could then be deleted or compressed for further use. When files are compressed, they tend to occupy lesser space.

There are several third party software applications that detect duplicate files, delete them and free disk space. The *Duplicate File Finder* developed by Copyleaks Technologies, LTD is noteable for having features such as finding and extracting duplicates in database, reducing database storage size, guaranteeing document security and lots more. However, it does not make provision for temporary files, cookies as well as log files within the system.

Similarly, the *DuplicateFileDetective* application is prominent third party software for extracting duplicates. Its other features include; Move, Delete, Archive Dupes, schedule duplicate search, link duplicate files, calculate file hashers and more. However, the features of the software are limited to handling duplicates and do not include managing redundant files. Other third party software for file finding duplicate include; Wise Duplicate finder, DupInOut, CCcleaner, Disk drill, Gemini, Hash MD5 check and others. Their functions regretfully are limited to duplicate files only.

In this work, we present a theoretical deduplication construct as well as implementation that assist the user in detecting and managing redundant files in the system with the sole aim of freeing and managing the limited disk space. The work is organized in different sections. Section 2 presents a review of related literature while Section 3 presents the methodology and concepts employed in the work. Section 4 presents the results of the implemented construct while section 5 summarizes the findings.

2. LITERATURE REVIEW

A major concept employed in the 3-task construct is known as deduplication. Data deduplication simply involves elimination of redundant data especially duplicate data. The process deletes extra copies of same data leaving only the original copy to be stored. Duplicate byte patterns are analysed and used to ensure that the remaining single instance of the data is indeed the original file. A deduplication approach known as *DARE* (Deduplication-Aware Resemblance Detection and Elimination scheme) was used to eliminate redundancy in [6,7]. DARE combines two techniques which are data deduplication and delta compression to achieve high data reduction efficiency. The delta compression refers to techniques that store data as the difference between successive samples or characters, rather than directly storing the samples themselves. The DARE technique makes provision for detection of duplicate data as well as achieving data, but it does not make provision for detection of dormant files which occupy disk space. Dormant files can be argued to occupy more space than duplicate files because they usually occur in numbers. In addition, unlike the proposed 3-task construct, DARE technique focuses on deduplication and compression of data but not on the actual files containing the data.

Another deduplication technique known as the *DupAdj* (duplicate adjacency) approach was used in [8]. This approach detects similar files by exploiting duplicate-adjacency data of a deduplication system. Again, the DupAdj technique sole objective is to fish out duplicates and unlike the proposed 3-task construct, the *DupAdj* focuses on the data only.

A more definite approach which focuses on file deduplication is seen in [6] which proposed a new approach to data reduction and storage management known as "*Redundancy Elimination within Large Collections of Files*". They argue that the new scheme is cost effective and most appropriate for large storage. However, functionality of the approach is limited to only duplicate files which does not proide a more holistic solution to disk space management.

Other approaches exist which also focus on the data content rather than the file content. The authors in [9] evaluated three methods used to discover identical portions of data with the aim of eliminating them and reducing disk space. The three methods evaluated include file content hashing, fixed size blocking, Rabin fingerprints strategy that delimits content-defined data chunks. The authors in [7] also developed a cloud based deduplication. They successfully used an AES (Advanced Encryption Standard) encryption algorithm to encrypt redundant files. Also, a deduplication solution for eliminating redundant files in resource constrained devices such as smartphones and the Internet of Things (IoT) devices is proposed in [10]. The developed solution "SmartDedup" combines in-line and out-of-line deduplication techniques by taking advantage of their complementary strengths, and optimizes their use for resource-constrained devices. Even so, their solution does not provide measures to address temporary files, cookies and log files.

Hashing algorithms have also been employed in disk management of redundant files [11],[12]. The author in [13] used an N-layer hash table for matching and identifying file duplicates and redundant files. Their work revealed that the N-Layered algorithm could achieve a O(n) time complexity and indeed an improvement in the accuracy of identifying redundant file. However, their research focused on only duplicate redundant files with no emphasis to the ever replicating temporary files which bedevils computer space .

Hash function was also used for deduplication as seen in [14]. In their work, Huffman code and hash function were combined to remove duplicate files in mobile phones. Once again, the functionality is limited to finding duplicate files in phones. In [15], Huffman code was also employed for image compression and disk space management. Their findings show how efficient the code can be in compressing image files. However, the functionality was specifically limited to image files.

In summary, most deduplication and hashing technologies are employed to address specific issues such as finding duplicates, and compressing files.. However, the 3-task construct which makes use of hashing algorithm provides the functionality of deduplication, compression as well as dormant file reduction. Unlike several other findings, The 3-task construct focuses on computer file content rather than on data content. The user friendly construct also goes a step further in aiding disk space management.

3. METHODOLOGY

This research employs a simulation methodology to investigate as well as demonstrate disk space management within the computer. The investigation and implementation focuses on finding duplicates a well as temporary files within the system. Data on 2018 New York Times comments were collected from Kaggle.com and used for the simulation. Four relevant fields were extracted from the heavy CSV (Comma separated values) comment dataset. The extracted fields of interest include; *articleID*, *articleWordCount, commentType* and *newsCategory*. From the extracted fields, a count of duplicate rows and columns were computed and recorded as initial count of duplicates using the excel COUNTIF function. The count values were subsequently compared with the results obtained after simulating the construct. At the end, a total of 65,526 files were obtained each one containing text extracted from the different fields. In order to accommodate the report in this paper, a random sample of 20,000 files accruing to 30% of the 65,000 distinct files was chosen for the simulation. After recording the initial duplicate count, a user defined excel function was coded for the purpose of exporting and saving the fields as separate CSV files each with distinct file names. All the files were saved in one folder. Some of these files were noted to be duplicates of others while some were just unique. Before deploying the construct, the initial size occupied by the folder (containing the 20,000 files) was also recorded for comparison after the simulation.

3.1 Conceptual formulation

The major concepts and theory behind the construct are presented in this section.

3.1.1 Redundant files

In this work, we define redundant files to be files that occupy space in the disk which do not in any way contribute to user's overall expected output. Redundant files are categorized as:

a. Duplicate files

These are files that serve as a copy of an original file. Such files are saved in the computer disk with the intent of later use. Unfortunately they are usually not used rather the original copy is modified and improved on thereby leaving a deprecated copy behind to occupy unnecessary space.

b. Temporary files

Temporary files are associated to word documents and other text files. When a new document file is created, a temporary file associated with it is also created in order to keep track of changes and recovery in case of a crash. In as much as these files have a specific role to play, they are temporary as the name implies and ought to be done with as soon as their purpose is served. Temporary files can be known to slow down system operation especially when they accumulate for so long. They can also be a source of infection to the main files. Temporary files bear the file name extension ".tmp", and usually reside in a temporary folder within the disk ("\AppData\Local\Temp"). Clearing temporary files usually require some expertise because the "temp" folder is usually hidden. It basically requires the following steps:

i. Open the "Run" dialogue box by pressing the Windows button + R

ii. Type %temp% in the box and click ok in order to open the "Temporary" folder

iii. Press "Ctrl + A" to select every temporary file

iv. Press "Delete" key to delete temporary files

The process is tedious and does not guarantee complete deletion of temporary files. In this paper, the files classified as temporary files include .sqm (Software Quality Metrics), .tmp and .log files. Sqm files extensions are associated to some applications that run on the Windows operating system platform. A log file keeps track of user's activity as well as information about the system.

c. Dormant Files

Dormant files are regarded as files that have not been used for a very long time. Such files exist in the computer without the knowledge of the user. Sometimes the user might have forgotten the existence of such files and may not really need them again. They occupy space inadvertently. Such files should therefore be deleted or zipped for subsequent use. The concept of dormancy and its duration is a controversial issue. The authors in [16] define it as an inactive period of two or more seasons. From economic perspective, it could be seen as a period that hinders poverty alleviation and human economic growth [17]. On the other hand, financial institutions see it as a period of inactivity on withdrawals, deposits or transfers that could span up to two (2) years and above [18]. The Central Bank of Nigeria in a communiqué stipulates a period of six (6) years before an account could be tagged dormant by Nigerian banks [19]. However, computer usage is dynamic and user's needs keep evolving. Computer files are therefore seen as transient objects that need to be constantly updated. In this paper therefore, we define a dormant file as a file that has not been accessed (updated) after two (2) years.

3.2 File attribute recognition and extraction

Every file has peculiar features attributed to it. File attributes include;

i. Name: A unique name for identification.
ii. Location: The source location path for the file.
iii. Type: The file type and the associated file extension.
iv. Size. Content size of the file.
v. Access control. Denotes if a file can be read only or read and write mode.
vi. Time and date of creation: The date of creation of file
vii. Time and date of last modification: The last time/date the file was modified

The file attributes of creation and access will be used in this work to categorize redundant files..

3.3 The Python Hashlib.SHA1

Hashing is a data structure mechanism most suitable for searching large datasets. In Hashing, the time taken to perform a search is independent of the total number of elements that exist. Hashing also has an advantage for it completes a search with constant time complexity O(1). In this work, the python hashlib.SHA-1 will be used for developing the construct. The SHA-1 algorithm is a known cryptographic hash function which converts an input to a 160-bit hash value known as a message digest. The message digest is a hexadecimal number of length 40 digits.SHA-1 algorithms have been applied in data integrity [20, 21]. The python hashlib hashing function generally works by taking a variable length of bytes (input) and converting it into a fixed length sequence of strings. In this paper, the Hashlib.SHA-1 detects duplicate files by taking a portion of a file (2048 bytes) and searching through other files for similarities. If a file is identified as unique (i.e no similarities) then such file is stored in a set data structure. On the other hand, if a similarity is found between files based on the chunk taken, then such file is marked as duplicate of the former.

3.3.1 The Find-duplicate construct

The find-duplicate construct makes use of the hashlib.SHA-1 algorithm. It takes a path and a file and finds duplicates. It has only one action which is to delete identified duplicate file. The find-duplicate construct have the following action path as shown in Figure 1;

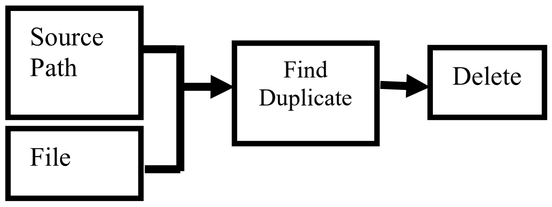


Figure 1: Action Path for identifying Duplicate Files

3.3.1.1 The implementation algorithm

The hash algorithm for implementing the find-duplicate construct is presented as follows:

```
1. Start
2. Declare Variables
    2.1 Hash function: hashlib.SHA1
    2.2 set: to store unique data
    2.3 file path to check for duplicates
3. Read 2048 bytes from file in tone iteration
4. Scan and Read from file until reach EOF and
   EOP (End of File/Path)
5. Generate Hash Value using Hash function
6.  if (Data is Unique)
6.1 store in set
      else
6.2 Data or File is duplicate
7. Display the Result for each iteration
8. END
```

The algorithm starts by declaring variables. The variables include the hashlib.SHA-1 function which is initialized before use, a set data structure which will be used to store the files/data identified as unique and the possible source path where duplicates are sought. A chunk of data as much as 2048bytes is read off a file and used as a comparing factor on all the other files within the path until an End of file (EOF) is reached. As each file is compared, a hash value is generated. For every non-matching comparison, the hash key is stored in the set data structure while for every matching comparison, the file is listed as a duplicate.

3.3.2 The Find-temporary construct

The find-temporary construct takes a path and finds temporary files. It has only one action which is to delete identified temporary file. The find-temporary construct have the following action path as shown in Figure 2;

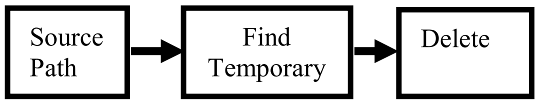


Figure 2: Action Path for identifying Temporary Files

3.3.2.1 The Implementation Algorithm

The implementation algorithm is presented as follows:

```
1. Start
2. Declare Variables
    2.1 Initialize the temporary function
    2.2 Set the file source path to point to
"Documents\AppData\Local\Temp"
2. Call Zip function with source path
3. List temporary files (.temp, .log, .sqm)
4. Delete file
5. Else compress file(s)
6. End
```

The algorithm accepts a source path and searches for temporary files. \it simply lists them afterwards with an option to delete them.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

3.3.3 The Find-dormant construct

The find-dormant construct takes a path and finds dormant files. It has two actions which are to delete or to compress a dormant file for future use. The find-dormant construct have the following action path as shown in Figure 3

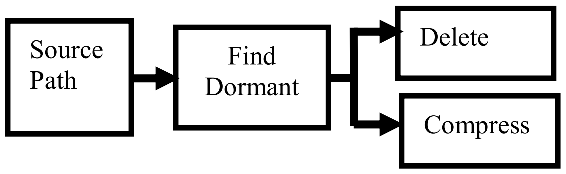


Figure3: Action Path for identifying Dormant Files

3.3.3.1 The Implementation Algorithm

The implementation algorithm for the find-dormant construct is also presented as follows:

```
1. Start
2. Declare Variables
    2.1 Initialize the dormant function
    2.2 Set the file source path
2. Select file(s) in a loop
    2.1 Extract creation_date
   2.2 Extract access_date
3. Call dormant function with creation_date, access_date
4. Loop again till EOF
4. Select option delete
5. Else select option compress
6. End
```

The algorithm also accepts a source path and loops through files. The file dormancy is determined by the difference between the date of creation and the last access date. At the end, an option to either delete or compress is provided.

3.3.4 The Compress-file construct

The compress-file construct takes a file and compresses it. It has only one action which is to compress selected file. The selected file can from temporary files or from dormant files. The compress-file construct have the following action path as shown in Figure 4;

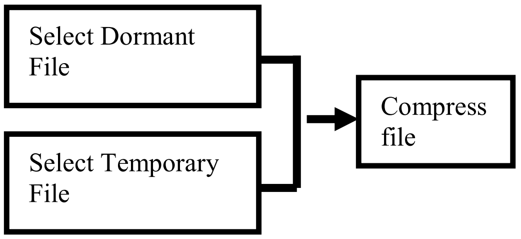


Figure 4: Action Path for identifying Dormant Files

3.3.4.1 The Implementation Algorithm

Similarly, the algorithm for compressing a file is also presented. As follows:

```
1. Start
2. Declare Variables
    2.1 Initialize the Zip function
    2.2 Set the file source path
2. Select file(s)
3. Call Zip function with source path
4. Save Zip file in source path
5. Delete file
6. End
```

The compress file algorithm accepts file(s) and compresses it as a zipped file and saves it in the same folder it was picked.

3.4 Conceptual Framework for the construct

The conceptual framework for the construct is shown in Figure 5.

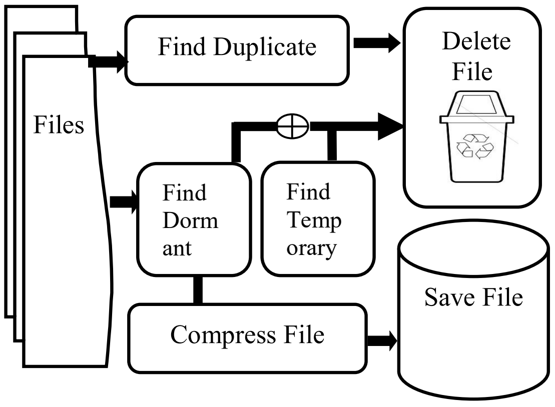


Figure5: Conceptual framework of the construct

The framework shows a selection of file(s) and four (4) subsequent actions to be performed on it. A file can be compared for duplicates, tagged as temporary or dormant file. A temporary or dormant file can either be compressed or deleted while a duplicate file can be deleted outrightly.

3.5 Implementation and Testing

The construct was implemented using python programming language. Specifically, the find-duplicate and the find-temporary constructs were implemented as a standalone application by making use of the Hashlib.SHA1 library. In order to test the efficacy of the construct, a dataset on the New York Times comments for March, 2018 from Kaggle.com was used. The developed software was deployed on a sample of 20,000 created files and simulated for results.

Initially, the target folder contains 20,000 files extracted from the dataset. Some of the files' contents are duplicates which also exist in other files. Such files were identified and deleted. Initially, before the software was deployed, the target folder was seen to occupy a disk space of 78.1MB.

4. RESULTS

After the simulation, the target folder was found to contain 141 files without duplicates and the size on disk was drastically reduced from 78.1MB to 564KB of disk space which amounts to a significant 99.3% storage space recovery. The number of duplicates found for each category was also recorded (After) against the initial (Before) value before the simulation. The summary of the output is presented in Table 1.

Table 1: Output summary of the constuct

S/n	Category	Duplicate per unit	Before	After
1	Business	20,.,, 485	1 * 11 = 11	11
2	Washington	14,..,320	1 * 11 = 11	11
3	Editorial	255,.,448	1 * 5 = 5	5
4	Foreign	34, , 342	1 * 7 = 7	7
5	Games	81, ., 50	1 * 6 = 6	6
6	Culture	11, , … 1	1 * 8 = 8	9
7	Learning	19, .,,202	1* 7 = 7	7
8	Magazine	2, ….,7	1 * 2 = 2	2
9	Special Sect	22, …,7	1* 2 = 2	2
10	Weekend	207. .., 1	1 * 3 = 3	3
11	National	52, , 40	1 * 10 = 10	10
12	Travel	10	1 * 1 = 1	1
13	OpEd	28, ., , 43	1 * 29 = 29	29
14	Metro	9, …., 12	1 * 6 = 6	6
15	Arts & Leisure	119, 8	1 * 2 = 2	2
16	Styles	7, …,6	1 * 2 = 2	2
17	Dining	13,., 23	1 * 7 = 7	7
18	Well	163,.,94	1 * 3 = 3	3
19	Express	19	1 * 1 = 1	1
20	Climate	56, 146	1 * 2 = 2	2
21	Real Estate	2, …4	1 * 5 = 5	5
22	Upshot	188, 9	1 * 2 =2	2
23	Metropolitan	51	1 * 1 = 1	1
24	Sports	47	1 * 1 = 1	1
25	Insider	16	1 * 1 = 1	1
26	Sunday Bus.	10,.., 6	1 * 3 = 3	3
27	Photo	4	1 * 1 = 1	1
28	Science	121	1 * 1 = 1	1
		20,000 files	**140 files**	**141 files**

Table 1 shows the output from the implemented construct. The different News categories were extracted from the 20,000 files created from the dataset. The corresponding group numbers for each repeated category were also noted. The respective total number of resulting files for each category before applying the construct was also noted. Subsequently after applying the construct, the total number of resulting files for each category was compared with the initial files in order to observe any difference. It was observed that out of the 28 categories considered in the experiment, only the culture category showed a difference in expected output with a total of 8 distinct files before implementation and a total of 9 distinct files after implementation. The distinct category is highlighted a shown in Table 1.

4.1 Discussion of findings

The results obtained after deploying the software shows that the construct was able to detect and remove duplicate files with 99% efficiency. The 1% difference seen in row 6 of Table 1 could be attributed to editing issues resulting to foreign character in one of the files. The construct could therefore be said to be fool proof in detecting duplicate files. In addition, the 99.3% percentage disk space recovery from 78.1MB to 564KB demonstrates that the implemented construct is highly efficient in detecting and removing duplicate files within the system.

4.2 Screen shots from the developed application

The following screen shots were obtained from the application.

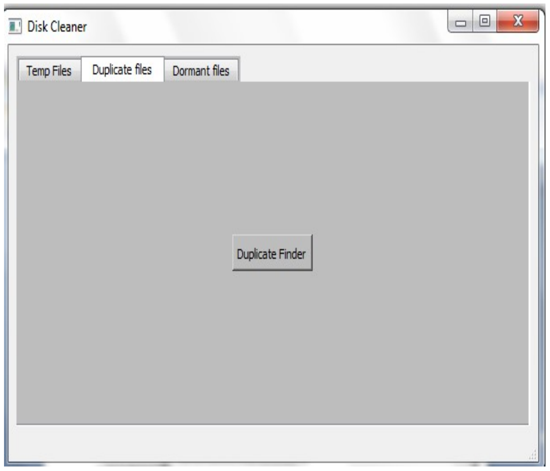
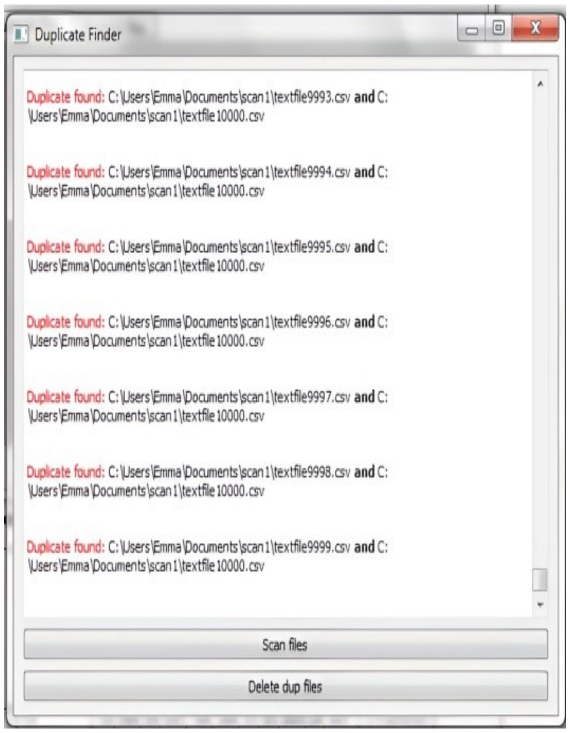


Figure 6: The 3-Task Construct Main Screen

Figure 6 screen shot shows the main screen of the implemented construct which allows the user to choose any of the 3-implemented tasks.



```
Duplicate found: C:\Users\Emma\Documents\scan1\textfile9993.csv and C:
\Users\Emma\Documents\scan1\textfile10000.csv

Duplicate found: C:\Users\Emma\Documents\scan1\textfile9994.csv and C:
\Users\Emma\Documents\scan1\textfile10000.csv

Duplicate found: C:\Users\Emma\Documents\scan1\textfile9995.csv and C:
\Users\Emma\Documents\scan1\textfile10000.csv

Duplicate found: C:\Users\Emma\Documents\scan1\textfile9996.csv and C:
\Users\Emma\Documents\scan1\textfile10000.csv

Duplicate found: C:\Users\Emma\Documents\scan1\textfile9997.csv and C:
\Users\Emma\Documents\scan1\textfile10000.csv

Duplicate found: C:\Users\Emma\Documents\scan1\textfile9998.csv and C:
\Users\Emma\Documents\scan1\textfile10000.csv

Duplicate found: C:\Users\Emma\Documents\scan1\textfile9999.csv and C:
\Users\Emma\Documents\scan1\textfile10000.csv
```

Figure7: The Find Duplicate Screen

Figure 7 shows the implementation window of the fin duplicate construct. The user can elect the option of deleting after the duplicate files are identified.

5. CONTRIBUTION TO KNOWLEDGE

The 3-task comprehensive construct (3 in 1) is specifically designed to assist the computer user in managing redundant files such as duplicate files, dormant files and temporary files. Unlike other contributions which seem to focus on duplicate data, the 3-task construct focuses on files. The construct employs the concept of hashing algorithm which is implemented as a standalone application. The user friendly environment provides a good alternative to the use of operating system tools for disk management which require rigourous steps.

6. CONCLUSION

The need for proper disk management cannot be overemphasized. In this work, redundant files such as temporary files, duplicate and dormant files have been identified as redundant files. Existing methods for managing such files were reviewed and a theoretical construct that makes use of Hash library for finding duplicate files was introduced. Dormant files were also determined by using file attributes while temporary files were also identified. The implementation algorithms for each action were also presented. Finally, the construct was automated using python programming language. The result from the automated construct was also tested and was found to be highly efficient in detecting duplicate files, as demonstrated by the significant gain in disk space observed from the test results of the simulation experiment.

Therefore, the 3-task construct for managing redundant files presents a significant opportunity for efficient and effective management of storage space.

REFRENCES:

[1] WHO , World report on Ageing and Health, WHO Press, World Health Organization, 20 Geneva, Switzerland, 2015. Retrieved from: https://apps.who.int/iris/bitstream/handle/10665/186463/9789240694811_eng.pdf;jsessionid=9DCF1CDAE67681687FD3877DC4EC7B73?sequence=1

[2] J. Sammons, Internet and e-mail in The Basics of Digital Forensics: The Primer for Getting Started in Digital Forensics, 2015. Elsevier Inc, ISBN:978-0-12-801635-0. DOI:https://doi.org/10.1016/B978-0-12-801635-0.00008-5.

[3] L. Shinder, M. Cross, Acquiring Data, Duplicating Data, and Recovering Deleted Files, 2008, in Scene of the Cybercrime 2nd Edition, Elsevier Inc. ISBN:978-1-59749-276-8. DOI: https://doi.org/10.1016/B978-1-59749-276-8.X0001-5

[4] J. E. Olson, The Archive Data Extraction Component, 2009, in Database Archiving,. Elsevier Inc, ISBN: 978-0-12-374720-4. DOI: https://doi.org/10.1016/B978-0-12-374720-4.X0001-5

[5] T.Fair, M. Nordfelt, S. Ring, E. Cole, Counterspy: Are You Being Watched? 2005, in Cyber Spying. Elsevier Inc, ISBN:978-1-931836-41-8. DOI: https://doi.org/10.1016/B978-1-931836-41-8.X5000-X.

[6] P. Kulkarn, F. Douglis, J.LaVoie, J. M. Tracey, Redundancy Elimination Within Large Collections of Files, 2004, in *2004 USENIX Annual Technical Conferenc*, Boston

[7] R..S. Mahajan, R. Rajpurohit, An Efficient Deduplication Approach to Maximally Detectand Eliminate Data Redundancy in Cloud Enviroment, 2017, *International Journal of Innovative Research in Computer,* vol. V, no. 2, p. 7.

[8] G. Priyanka, G.V. Reddy, K.B. Maruthiram , Reliable De-Duplication Approach to Detect and Eliminate Data Redundancy and Finding the Sweet Spot through Delta Compression, 2017, International Journal of Innovative Research in Science, Engineering and Technology, Vol. 6, Issue 8

[9] C. Policroniades, I. Pratt, Alternatives for Detecting Redundancy in Storage Systems Data, 2004, ATEC '04: Proceedings of the annual conference on USENIX Annual Technical Conference. Available at: https://dl.acm.org/doi/10.5555/1247415.1247421

[10] Q.Yang, R. Jin, M. Zhao, Smartdedup: optimizing deduplication for resource-constrained devices 2019. In *{USENIX} Annual Technical Conference ({USENIX}{ATC} 19)* (pp. 633-646).

[11] D.V.V.B, Jacobson, Storage management system for concurrent generation and fair allocation of disk space among competing requests, 1995, USA Patent US5463776A.

[12] Y. Kano, System and method for managing disk space in a thin-provisioned storage subsystem, 2006, USA Patent US7130960B1

[13] M. Siladitya, G. P. Jose, C. Soumick & B. Priyanka, Duplicate File Analyzer using N-layer Hash and Hash Table, 2017. International Research Journal of Computer Science (IRJCS). 4. 24-30

[14] A. Asaad and A. A. Y. Alamri , A New Scheme for Removing Duplicate Files from Smart Mobile Devices: Images as a Case Study, Department of Computer, Education College for Pure Sciences, University of Basrah, 61004 Basrah, Iraq, CUESJ 2019, 3 (2): 5-13 . DOI: 10.24086/cuesj.v3n2y2019.pp5-13

[15] A. Nagarajan. and V. Devendran. An Enhanced Approach in Huffman Coding Scheme, International Journal of Advanced Research in Computer Science and Software Engineering, Volume 4, Issue 11, November 2014

[16] J.Walck, J.Baskin, C.Baskin, S.Hidayati, Defining transient and persistent seed banks in species with pronounced seasonal dormancy and germination patterns, 2005. Seed Science Research. 15. 189 - 196. 10.1079/SSR2005209.

[17] Grameen Foundation , Addressing Dormancy in Savings Accounts: Insights from the Cashpor BC Project, 2013, A Grameen Foundation India Series. Available at: https://grameenfoundation.org/documents/rr7hglaed054z3cnrhon.pdf

[18] D. M. Schydlowsky, E.F. Matuk, Converting Dormant Bank Accounts into a Dynamic Force in Latin America, 2018, Discussion Paper Nº IDB-DP- 5 96, Institutions for Development Sector Connectivity, Markets, and Finance Division, Inter-American Development bank. Available at: https://publications.iadb.org/publications/english/document/Converting-Dormant-Bank-Accounts-into-a-Dynamic-Force-in-Latin-America.pdf

[19] CBN, Exposure Draft guidelines for the management of Dormant Accounts by Banks in Nigeria, 2015, Central Bank of Nigeria. Available at : https://www.cbn.gov.ng/out/2015/fprd/dormant%20account%20guidelines.pdf

[20] R.Siddhartha, Advanced SHA-1 Algorithm Ensuring Stronger Data Integrity, 2015. International Journal of Computer Applications. 130. 25-27. 10.5120/ijca2015907056. DOI:10.5120/ijca2015907056

[21] J. Mohammed, Data Integrity Mechanism Using Hashing Verification, 2014. International Journal of Network Security.