

Dropbox like Storage Report

Team Information

Names	BUID	SCC user name
Yucheng He	U90654754	hyc1011
Li Cao	U60827070	licao
Lixu He	U67329607	masonhe
Haipang Liu	U18757354	haipengl
Jianing Geng	U65291070	jgeng

Abstract

For this project, a local files system was created to satisfy the following requirements: Able to load files and list the name of the existing files; Able to retrieve back and regenerate files; Able to delete files. After discussion and further exploration of the data storage system, less storage capacity was chosen as the top priority of this project. To achieve this goal, three major steps were designed. First, after the program receives the input file, it will chunk the input file into several parts. Second, divided data chunks will be compressed using Huffman coding. Last, those compressed chunks will be stored on a virtual hard disk. The main programming language we use is c++ because it provides programmers a low-level manipulation of data and control over memory management.

Background Information

Huffman coding is a lossless data compression algorithm. The idea is to assign variable-length codes to input characters; lengths of the assigned codes are based on the frequencies of corresponding characters. The most frequent character gets the smallest code, and the least frequent character gets the largest code.

For example, there is a string of text: aaaabbbccd. If we store the string in the file alone, it will take at least 10 bytes. So in order to save our valuable disk space and transfer time, we can use Huffman coding. So now we can convert the original string of text into 00001010 10111111 110. By replacing the original characters with binary bits, the string is compressed by replacing the more frequent characters with a shorter code. The codes obtained by this method are prefix codes: the code of any character is not a prefix of the code of another character. This ensures the uniqueness of the decoding code.

The advantage of this: Huffman coding is the theoretically optimal compression coding. It is lossless so that we can unpack it correctly. It is more space-efficient than binary encoding. However, it also has some disadvantages: It needs to be decoded from the beginning of the text

to get the correct result. We cannot decode halfway, and even if the text is long, we also need to decode from the beginning. If the order is messed up somewhere, the whole text may be wrong.

Instruction for running code

The project files are as following:

- disk.c
- disk.h
- fileSystem.c
- fileSystem.h
- fileCompress.cpp
- fileCompress.h
- HuffmanTree.hpp
- compress.c
- compress.h
- UI.c
- Makefile

Test files

Sample1.txt

sample2.txt

To use the UI, make the project and use the following command:

makefile

```
```. /make -k -f makefile``
```

Creating a new disk with its name:

```
```. /UI -newdisk {DiskName}``
```

Listing the content and usage of a disk:

```
```. /UI -ls {DiskName}``
```

Saving a file to the disk:

```
```. /UI -save {DiskName} {OriginalFileName} {FileNameInVFS}``
```

Deleting a file in the disk:

```
```. /UI -rm {DiskName} {FileNameInVFS}``
```

Loading a file from the disk:

```
```. /UI -load {DiskName} {FileName} {FileNameInVFS}``
```

Loading a file from the disk, without decoding:

```
```.UI -load_NoDecode {DiskName} {FileName} {FileNameInVFS}```
```

Encoding a file:

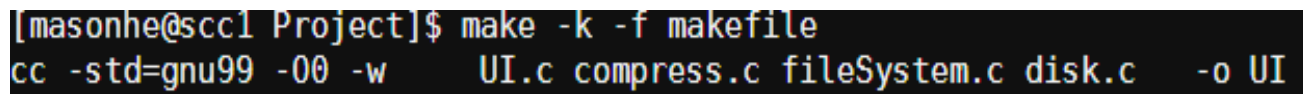
```
```.UI -encode {ASCIIFileName} {TrainBigramFileName}```
```

Decoding a file:

```
```.UI -decode {ASCIIFileName} {TrainBigramFileName}```
```

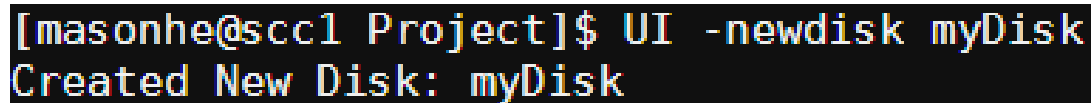
**\*\*Would output ``Segmentation Fault`` if the user try to read from a non-existing file.**

## Results



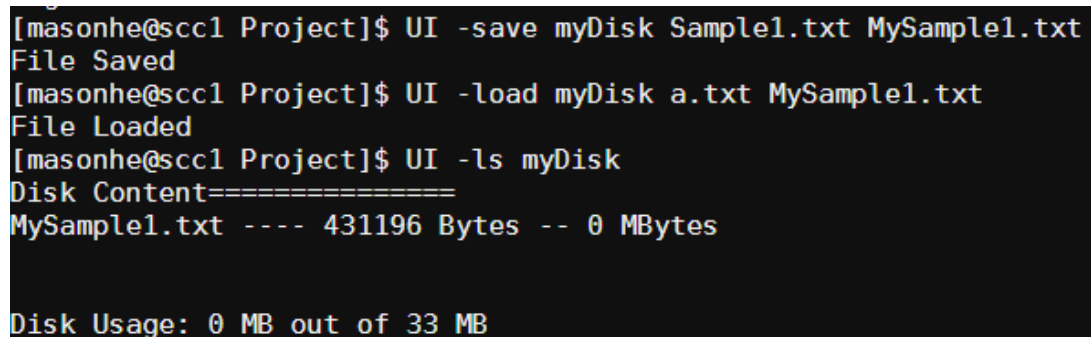
```
[masonhe@scc1 Project]$ make -k -f makefile
cc -std=gnu99 -O0 -w UI.c compress.c fileSystem.c disk.c -o UI
```

fig.1



```
[masonhe@scc1 Project]$ UI -newdisk myDisk
Created New Disk: myDisk
```

fig.2



```
[masonhe@scc1 Project]$ UI -save myDisk Sample1.txt MySample1.txt
File Saved
[masonhe@scc1 Project]$ UI -load myDisk a.txt MySample1.txt
File Loaded
[masonhe@scc1 Project]$ UI -ls myDisk
Disk Content=====
MySample1.txt ---- 431196 Bytes -- 0 MBytes

Disk Usage: 0 MB out of 33 MB
```

fig.3

## Discussion

The two files disk.c, disk. h were used to create a virtual hard disk. The hard disk could successfully open, close, and read data into its blocks. fileSystem.c and fileSystem.h files create a file system that could control the way data is stored and retrieved. The system is capable of interfering with the virtual hard disk, stores data in an organized way, and achieve most of the functions of this project (files load, data storage, file delete, file retrieve). For fileCompress.cpp,

fileCompress.h, HuffmanTree.hpp, compress.c and compress compress.h, those files are responsible for the chunking, compression, regeneration, and uncompression of input files. We acknowledge that using a Huffman tree to compress data may not be secure enough. If some errors happen that may lead to data loss, but because of its space-efficient and lossless nature, it is still our first choice.

The UI.c file allows all commands to be done in the terminal in an easy and fast way. The save a file to disk command will upload the file into the virtual hard disk after compression. The load a file from disk command will retrieve back the file and uncompress it.

Fig.1 shows that the makefile could be run successfully. Fig.2 shows the command used to create a virtual hard disk. We use two plain text files to test our project. The size of sample1.txt is 574,928 bytes and sample2.txt is 772,561 bytes. Fig.3 shows how the sample files could be saved and loaded. It also presents the function which lists the name of all the existing files in the storage system. Our test result shows that our final product satisfies the requirement of the goals of this project.

## Future Research

To reduce duplicate chunks and reduce the storage, we intend to use hashing. The basic principle is to calculate the hash value of a part of data by SHA-256 and use this hash value as a unique identifier for this part of data. When the hash values of two parts of the data are the same, we assume that the two parts of the data are the same.

As shown in the figure below, every 4 data are chunked into 6 chunks and the hash value of each chunk is calculated. The first data chunk and the 6th data chunk have the same value, so the data are also the same. Then the 6th data chunk only needs to record the index and does not need to be stored again, so as to achieve the effect of reducing storage space.



Fig.4

Ideally, different keys would be converted to different index values, but in some cases we need to deal with multiple keys being hashed to the same index value. Here we introduce two ways to handle hash collision conflicts.

1. Open Addressing: When a key conflicts with the hash address obtained by the hash function, we can use the calculated hash address as the basis to generate another address, and if it still conflicts, continue to take it in the same way until a non-conflicting hash address is found.

2. Chained Hash table: Store all records whose keywords are synonyms in the same linear chain table.

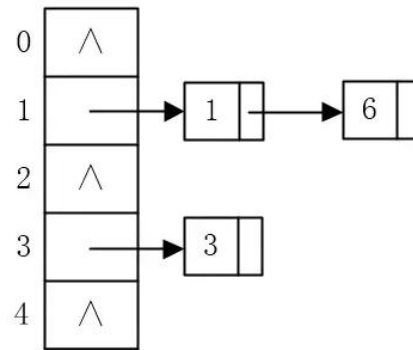


Fig.5 Chained Hash table

## Reference

“C++ Development: Advantages, Disadvantages, and Applications.” *VOLO*, 15 Apr. 2022, <https://volo.global/blog/news/c-plus-plus--development#:~:text=Low%2DLevel%20Manipulation%3A%20At%20a,terms%20of%20performance%20and%20memory>.

“Huffman Coding.” *Wikipedia*, Wikimedia Foundation, 7 Dec. 2022, [https://en.wikipedia.org/wiki/Huffman\\_coding](https://en.wikipedia.org/wiki/Huffman_coding).

Q 先生. (2020, June 10). 重复数据删除技术详解 (一)talkwithtrend.com. Retrieved December 5, 2022, from <https://www.talkwithtrend.com/Article/251221>

飒飒 程序员. (2021, June 24). 【数据结构与算法】哈希算法的原理和应用详解baijiahao.baidu.com. Retrieved December 5, 2022, from <https://baijiahao.baidu.com/s?id=1703426705379279178&wfr=spider&for=pc>

weixin\_30488313. (2018, November 1). 记录一下哈希表底层原理. CSDN. Retrieved December 3, 2022, from [https://blog.csdn.net/weixin\\_30488313/article/details/97284647](https://blog.csdn.net/weixin_30488313/article/details/97284647)

“哈夫曼实现文件压缩解压缩 (C语言) 哈夫曼实现文件压缩解压缩 (C语言) \_兔老大RabbitMQ 博客-CSDN博客\_c语言数据压缩, [https://fantianzuo.blog.csdn.net/article/details/86613332?spm=1001.2101.3001.6650.3&utm\\_medium=distribute.pc\\_relevant.none-task-blog-2~default~CTRLIST~Rate-3-86613332-blog-101710602.pc\\_relevant\\_recovery\\_v2&depth\\_1-utm\\_source=distribute.pc\\_relevant.none-task-blog-2~default~CTRLIST~Rate-3-86613332-blog-101710602.pc\\_relevant\\_recovery\\_v2&utm\\_relevant\\_index=6](https://fantianzuo.blog.csdn.net/article/details/86613332?spm=1001.2101.3001.6650.3&utm_medium=distribute.pc_relevant.none-task-blog-2~default~CTRLIST~Rate-3-86613332-blog-101710602.pc_relevant_recovery_v2&depth_1-utm_source=distribute.pc_relevant.none-task-blog-2~default~CTRLIST~Rate-3-86613332-blog-101710602.pc_relevant_recovery_v2&utm_relevant_index=6).

*Splitting Files and Regrouping Splitted - C++ Forum*, <https://cplusplus.com/forum/general/32742/>.

“Huffman Coding Compression Algorithm.” *Techie Delight*, 24 Sept. 2022,  
<https://www.techiedelight.com/huffman-coding/>.