

Arab Academy for Science and Technology

Faculty of Engineering



الأكاديمية العربية للعلوم والتكنولوجيا والنقل البحري

Arab Academy for Science, Technology & Maritime Transport

Project Huffman File Compression (May 2023)

Submitted by: Philimon Elkess Dawoud
Department: Computer Engineering
Registration Number: 19102648
Submitted to: Dr. Ashraf Said
Submitted on: 24/05/2023

Welcome

fileParser.py X

fileParser.py > Parser > parseOriginal

```
1  class Parser:
2
3      def __init__(self, path: str) -> None:
4          self.path = path
5          self.text = ""
6          self.frequencies = {}
7          self.byteNum = 0
8
9          self.compressedPath = path[:-4] + "cmpr.txt"
10         self.encodedText = ""
11
12
13     def countFrequencies(self, char: str):
14         if char in self.frequencies:
15             self.frequencies[char] = self.frequencies[char] + 1
16         else:
17             self.frequencies[char] = 1
18
19
20     def parseOriginal(self):
21         with open(self.path, "r") as file:
22             self.text = file.read()
23         for char in self.text:
24             self.byteNum += 1
25             self.countFrequencies(char)
26
27     def parseCompressed(self):
28         with open(self.compressedPath, "r", encoding="utf-8") as file:
29             self.encodedText = file.read()
30
31
32     def write(self, text: str):
33         with open(self.compressedPath, "w", encoding="utf-8") as file:
34             file.write(text)
```

Welcome  priorityQ.py X

 priorityQ.py >  heap >  build

```
1  import heapq
2  from BinaryTree import Node
3
4  class heap:
5
6      def __init__(self) -> None:
7          self.queue = []
8
9      def build(self, frequencies: dict):
10         for key in frequencies:
11             frequency = frequencies[key]
12             node = Node(key,frequency)
13             heapq.heappush(self.queue , node)
```

Welcome

BinaryTree.py X

BinaryTree.py > BinaryTree

```
1  import heapq
2
3  class Node:
4      def __init__(self, data, freq) -> None:
5          self.data = data
6          self.freq = freq
7          self.left = None
8          self.right = None
9
10     def __lt__(self, other):
11         return self.freq < other.freq
12
13     def __eq__(self, other):
14         return self.freq == other.freq
15
16 class BinaryTree:
17
18     def __init__(self) -> None:
19         pass
20
21     def buildBinaryTree(self, queue: list):
22         while len(queue) > 1:
23
24             node1 = heapq.heappop(queue)
25             node2 = heapq.heappop(queue)
26
27             freqSum = node1.freq + node2.freq
28             newnode = Node(None, freqSum)
29
30             newnode.left = node1
31             newnode.right = node2
32
33             heapq.heappush(queue, newnode)
```

```
Welcome Huffman.py X
Huffman.py > HuffmanCode > decode
1  import heapq
2  from binAscii import binascii
3
4  class HuffmanCode:
5
6
7      def __init__(self) -> None:
8          self.codes = {}
9          self.huffmanTree = None
10
11
12      def __generateCodeHelper(self, root, code: str):
13          if root is None:
14              return
15
16          if root.data is not None:
17              self.codes[root.data] = code
18              return
19
20          self.__generateCodeHelper(root.left, code + '0')
21          self.__generateCodeHelper(root.right, code + '1')
22
23
24      def generateCode(self, queue: list):
25          self.huffmanTree = heapq.heappop(queue)
26          duplicatePointer = self.huffmanTree
27          self.__generateCodeHelper(duplicatePointer, '')
28
```

```
Welcome Huffman.py X
Huffman.py > HuffmanCode > encode
29
30     def encode(self, text):
31
32         encoded = ""
33         binary = ""
34
35         for char in text:
36             binary += self.codes[char]
37
38
39         if (len(binary) % 8) != 0:
40             remaining = 8 - (len(binary) % 8)
41             for i in range(remaining):
42                 binary += "0"
43
44         encoded = binascii.toString(binary)
45
46         return encoded
47
```

```
Welcome Huffman.py X
Huffman.py > HuffmanCode > encode
47
48
49     def decode(self, text):
50
51         decoded = ""
52         binary = binascii.toBinary(text)
53         root = self.huffmanTree
54
55         for bit in binary:
56
57             if root.data is not None:
58                 decoded += root.data
59                 root = self.huffmanTree
60
61                 if bit == "0":
62                     root = root.left
63
64                 elif bit == "1":
65                     root = root.right
66
67             elif bit == "0":
68                 root = root.left
69
70             elif bit == "1":
71                 root = root.right
72
73         return decoded
74
```

```
Welcome binAscii.py X
binAscii.py > ...
1 class binascii:
2
3     def __init__(self) -> None:
4         pass
5
6     def toBinary(string):
7         return "".join([format(ord(char), '#010b')[2:] for char in string])
8
9     def toString(binaryString):
10        return "".join([chr(int(binaryString[i : i + 8], 2)) for i in range(0, len(binaryString), 8)])
11
```

Welcome

compress.py X

compress.py > Compressor

```
1  from fileParser import Parser
2  from priorityQ import heap
3  from BinaryTree import BinaryTree
4  from Huffman import HuffmanCode
5
6  class Compressor:
7
8      def __init__(self, path) -> None:
9          self.path = path
10         self.file = Parser(self.path)
11         self.heap = heap()
12         self.tree = BinaryTree()
13         self.huffman = HuffmanCode()
14
15     def compress(self):
16
17         self.file.parseOriginal()
18
19         self.heap.build(self.file.frequencies)
20
21         self.tree.buildBinaryTree(self.heap.queue)
22
23         self.huffman.generateCode(self.heap.queue)
24
25         encodedtext = self.huffman.encode(self.file.text)
26
27         self.file.write(encodedtext)
28
29         print("File Compressed Successfully!")
30
31
32     def decompress(self):
33         self.file.parseCompressed()
34         decodedtext = self.huffman.decode(self.file.encodedText)
35         print(decodedtext)
```



```
Welcome  main.py X
main.py > ...
1  from compress import Compressor
2
3  path = "Sample.txt"
4
5  compressor = Compressor(path)
6
7  compressor.compress()
8
9  compressor.decompress()
```

```
ijfoglknjsdfgljksddfglk
File Decompressed Successfully!
PS F:\Edu. related\Term 8\Algorithms\Huffman Compression - Decompression Algorithm>
```

Welcome

test.py



test.py

```
1  # from fileParser import Parser
2  # from priorityQ import heap
3  # from BinaryTree import BinaryTree
4  # from Huffman import HuffmanCode
5
6
7  # file1 = Parser(path)
8  # file1.parseOriginal()
9
10 # print(file1.byteNum)
11 # print(file1.frequencies)
12
13 # heap1 = heap()
14 # heap1.build(file1.frequencies)
15
16 #print(heap1.queue)
17
18 # tree1 = BinaryTree()
19 # tree1.buildBinaryTree(heap1.queue)
20
21 #print(heap1.queue)
22
23 # huffman1 = HuffmanCode()
24 # huffman1.generateCode(heap1.queue)
25
26 # print(huffman1.codes)
27
28 # encodedtext = huffman1.encode(file1.text)
29
30 # print(text)
31
32 # file1.write(encodedtext, huffman1.codes)
33
34 # file1.parseCompressed()
35
36 #print(file1.encodedText)
37
```

```

Welcome  test.py  X
test.py
25
26 # print(huffman1.codes)
27
28 # encodedtext = huffman1.encode(file1.text)
29
30 # print(text)
31
32 # file1.write(encodedtext, huffman1.codes)
33
34 # file1.parseCompressed()
35
36 #print(file1.encodedText)
37
38 # decodedtext = huffman1.decode(file1.encodedText)
39
40 # print(decodedtext)

```

```

Welcome  Sample.txt  X
Sample.txt
1  ijfogljknsdfgljksddfglk

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```

PS F:\Edu. related\Term 8\Algorithms\Huffman Compression - Decompression Algorithm> & "C:/Program Files/Python310/python.exe" "F:/Edu. related/Term
8/Algorithms/Huffman Compression - Decompression Algorithm/main.py"
File Compressed Successfully!

```

```

Welcome  Samplecmpr.txt  X
Samplecmpr.txt
1  ACKEXÔñUÍý☐°

```

Sample Properties



General Security Details Previous Versions



Sample

Type of file: Text Document (.txt)

Opens with: Notepad

Change...

Location: F:\Edu. related\Term 8\Algorithms\Huffman Compre

Size: 24 bytes (24 bytes)

Size on disk: 0 bytes

Created: Thursday, May 18, 2023, 10:41:59 PM

Modified: Wednesday, May 24, 2023, 3:15:40 AM

Accessed: Wednesday, May 24, 2023, 3:15:40 AM

Attributes: ☐ Read-only ☐ Hidden

Advanced...

OK

Cancel

Apply



