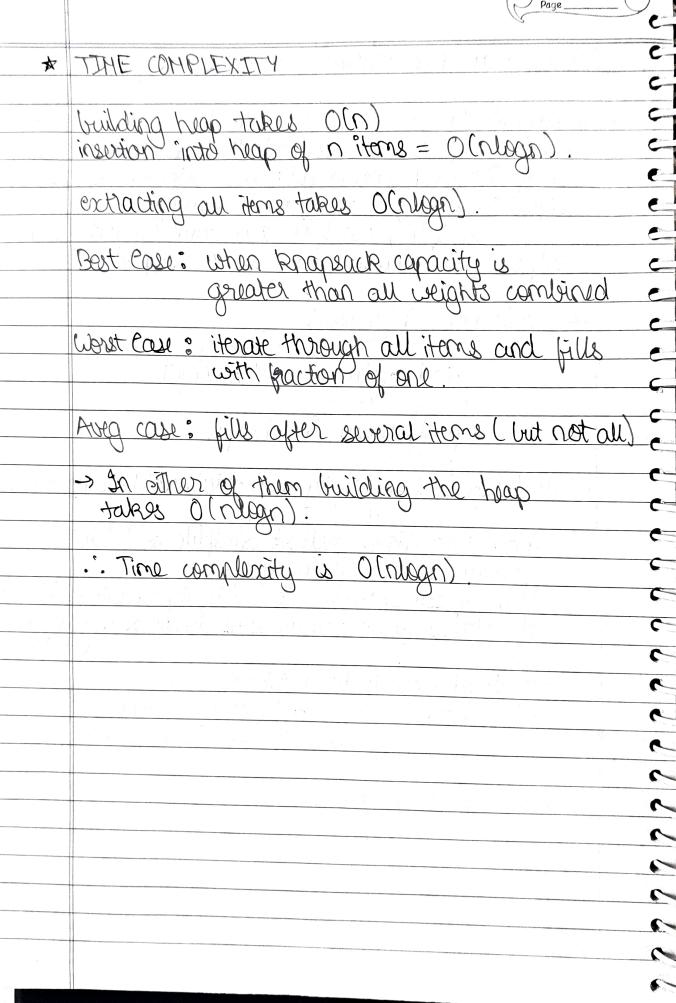


* ALGORITHM FRACTIONAL KNAPSACK 1/ compute the max value that can be shipped within capacity of 200 tonnes such that goods with use shelf life and higher cost are shipped first.

11 Input: List of goods (each having weight, shelf life, lost), man weight capacity (200) 11 Output: Marx value that can be shipped for each item in goods: find value = cost/weight push value in heap while heap is not empty and totalweight < capacity: app item with smallest shelf life and highest value density from heap b weight to ship = min (weight, capacity - weight of total_value+= value dons density x weight to ship total d total weight += weight_to ship e y total_weighth == capacity than coit with knapsack is full guturn total-value

6666666



•

classmate

* Time complexity (Brute Force) If we take the weight steps on to be m and there are n items then all possible combination amount to m. Therefore, time complexity will be O(m) This is computationally wery expensive and is not feasible to calculate for large values Test Case:

つかつつつかっちゃ

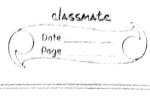
al Randomly generate 100 items with random value, random weight and random shelflife (All values in acceptable ranges).

b) weights are regative or equal to zero

I values of items are zero

al shall life are regative or tos zero

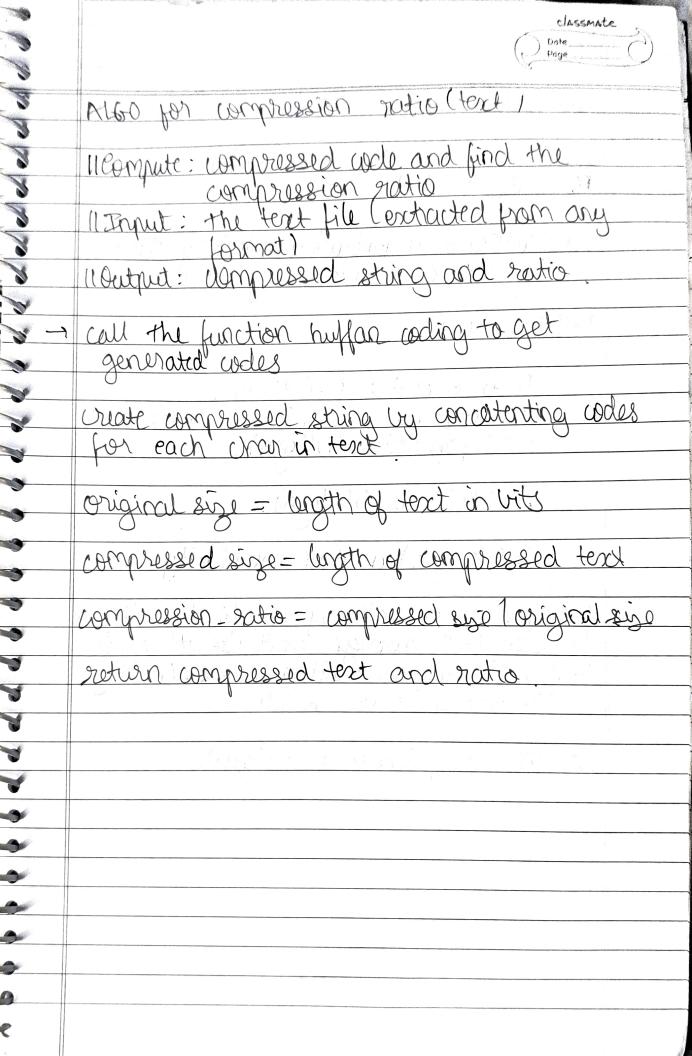
e vehicle capacity less than O.



* Muffman cooling: 11 Brout: Lext
11 Output: huffman tree. ritialise a freq dictionary to count occurrence of each char in text. reate a min heap priority queue with (weighty node) for each char. > while size of heap >1: pop 2 nodes with buest freq (x, y) new_freq = rc. freq + y. freq. (x, create resonant and assign re, y as children nodes of new node al push new node in heap seturn noot node.

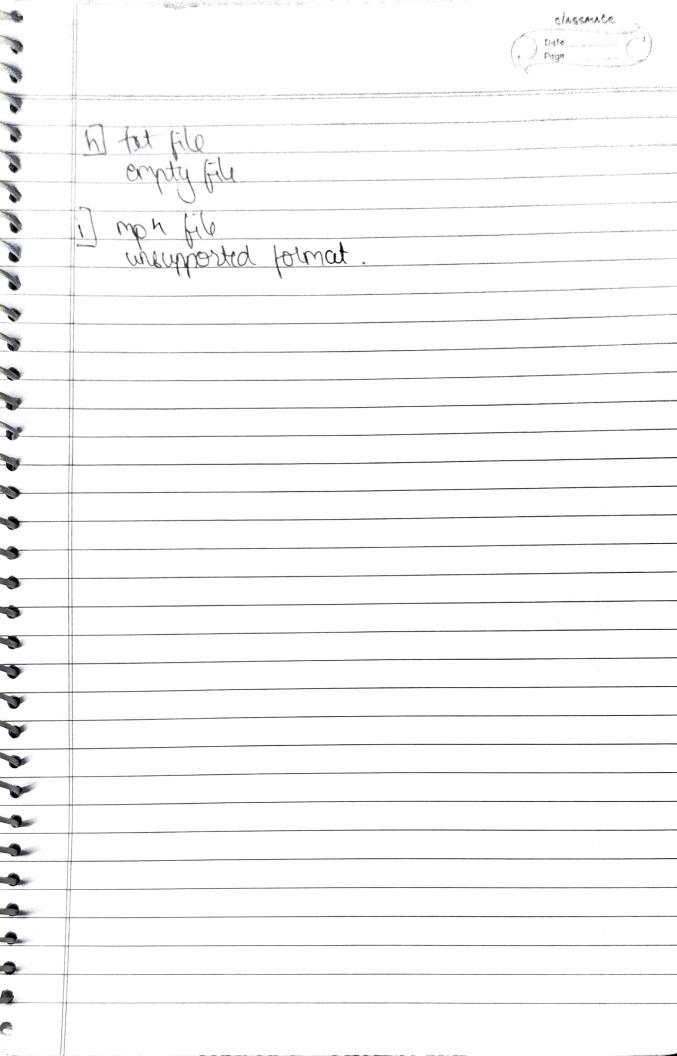
generate ades (node, vodes, currentiode) Hampule the huffman codes for each character from the tree. 11 Input: node of hulfan tree, empty dictionary, current ode. C C-11 Output: wodes dictionary containing <u>_</u> huffman codes at each character. - if node is not none: al il it has a symbol to sumbol and C corresponding wrrent code **C** to code dict to row b) recursively call generate colles for left and child both current code appended with "O" of recursively call generate codes current node appended with "1" geturn codes 3 C

classmate



	classmate	
	Date Page	e
农	Time Complexity.	
	R= ungth of text	ر د
		ins deligion in page to the mass and a second in the secon
	contracting and intialising min heap takes	0
	building tree:	0
	10 each is and insert I can be audio	<u> </u>
	in each we pop 2, insert 1, each push and pop takes Ollhogn).	-6 -c
	contrie à Omloyn).	-e
	O (nlogn) is time complexity.	C
	Very thore is nothing specifically different between best average and worst since	Ç
	building tree (generating wells, finding satio books fame in all	¢
	i. Time complexity is Olaloga).	C
	- The confusion of the second	C
		6
		6
		c
		_0

		classmate	
		Date Page	
	Test Cases:		
	al post file		
	on sino: 1138 char	C	
College Andrews State Control of the College And	og size: 1138 chart compressed size: 5368 bits ratio: 1.70	C	
	natio : 1.70	C	
	bl docx file og size: 704 char corpressed size: 3390 lits ratio: 1.66	e	
	b) docx fill		
	00 size: 704 char	<u> </u>	
	Corpressed by a . 3390 UTS		
	Nat (0. 1066	•	
	Cl tot Gill		
	no sino . 718 char	e	
	commissed size: 1218 bits		
	og size: 718 char compressed size: 1218 bits ratio: 4.72	e	
		e	
	d hard file		
-	20 01 · · 2221 chan	C	
	og size: 2221 charl compressed size: 9150 bits ratio: 1.94.		
	natio : 1.94.	C	
	e jpg file unsupported format	C	
	insupported format	C	
		<u></u>	
<i>a</i> .	2 docse file incorrect path		
**	Maria Maria		
	a pat file	C	
	g) pat file empty file		
		•	
		•	



The code follows PEP 8 guidelines.

Output:

```
C:\Users\Krish\AppData\Local\Programs\Python\Python312\python.exe C:\Users\Krish\Downloads\Huffmancoding.py
Reading book from: C:\Users\Krish\Downloads\Pdf_File.pdf
Original size: 1138 characters
Compressed size: 5368 bits
Compression Ratio: 1.70
Reading book from: C:\Users\Krish\Downloads\Docx_File.docx
Original size: 704 characters
Compressed size: 3390 bits
Compression Ratio: 1.66
Reading book from: C:\Users\Krish\Downloads\Text_File.txt
Original size: 718 characters
Compressed size: 1218 bits
Compression Ratio: 4.72
Reading book from: C:\Users\Krish\Downloads\Html_file.html
Original size: 2221 characters
Compressed size: 9150 bits
Compression Ratio: 1.94
Reading book from: C:\Users\Krish\Downloads\Html_file2.html
Original size: 10521 characters
Compressed size: 39889 bits
Compression Ratio: 2.11
C:\Users\Krish\AppData\Local\Programs\Python\Python312\python.exe C:\Users\Krish\Downloads\Huffmancoding.py
Reading book from: C:\Users\Krish\Downloads\Image.jpg
Unsupported file format: C:\Users\Krish\Downloads\Image.jpg
Reading book from: C:\Users\Krish\Downloads\Doc.docx
Error reading C:\Users\Krish\Downloads\Doc.docx: Package not found at 'C:\Users\Krish\Downloads\Doc.docx'
Reading book from: C:\Users\Krish\Downloads\Zero_letter_File.pdf
Error reading C:\Users\Krish\Downloads\Zero_letter_File.pdf: Cannot read an empty file
Reading book from: C:\Users\Krish\Downloads\Single_Letter_File.txt
Compression failed or resulted in an empty string.
Reading book from: C:\Users\Krish\Downloads\Video_File.mp4
Unsupported file format: C:\Users\Krish\Downloads\Video_File.mp4
Process finished with exit code 0
```

C:\Users\Krish\AppData\Local\Programs\Python\Python312\python.exe C:\Users\Krish\Downloads\Fractional_Knapsack.py
Maximum profit achievable: 1171.49 units

Total weight used: 200.00 tons Remaining capacity: 0.00 tons

Items selected:

Item	Fraction (%)	Weight (Selected/Total)	Value (Selected/Total)	Shelf Life (days)
Item_12	100.0	7.00/7	91.00/91	12
Item_80	100.0	5.00/5	85.00/85	19
Item_76	100.0	10.00/10	90.00/90	12
Item_49	100.0	7.00/7	89.00/89	18
Item_81	100.0	11.00/11	83.00/83	12
Item_41	100.0	8.00/8	49.00/49	10
Item_99	100.0	9.00/9	64.00/64	14
Item_82	100.0	10.00/10	87.00/87	19
Item_62	100.0	12.00/12	63.00/63	12
Item_55	100.0	15.00/15	92.00/92	15
Item_53	100.0	19.00/19	82.00/82	14
Item_93	100.0	19.00/19	91.00/91	17
Item_19	100.0	13.00/13	44.00/44	13
Item_86	100.0	8.00/8	30.00/30	15
Item_4	100.0	15.00/15	66.00/66	18
Item_44	74.4	32.00/43	65.49/88	10

C:\Users\Krish\AppData\Local\Programs\Python\Python312\python.exe C:\Users\Krish\Downloads\Fractional_Knapsack.py
Error: Total weight of items is less than or equal to zero.

C:\Users\Krish\AppData\Local\Programs\Python\Python312\python.exe C:\Users\Krish\Downloads\Fractional_Knapsack.py
Error: Shelf life of items must be greater than 0.

C:\Users\Krish\AppData\Local\Programs\Python\Python312\python.exe C:\Users\Krish\Downloads\Fractional_Knapsack.py
Error: No value could be obtained from the items.

C:\Users\Krish\AppData\Local\Programs\Python\Python312\python.exe C:\Users\Krish\Downloads\Fractional_Knapsack.py
Error: Vehicle capacity must be greater than 0.

Conclusion:

The **fractional knapsack problem** involves selecting items with given weights and values to maximize the total value within a fixed capacity, allowing fractional parts of items to be taken. **Huffman coding** is a data compression technique that assigns variable-length binary codes to characters based on their frequency, minimizing the total encoded size while ensuring prefix-free encoding.

Fractional knapsack using brute force has time complexity of O(m^n) and for greedy approach is O(nlogn) which is due the use of min heap priority queue.

Huffman coding gives the time complexity of O(nlogn) for greedy approaches.

Greedy algorithms offer optimal and practical solutions to both problems by reducing time complexity compared to brute force.