MIHIR KATAKDHOND
231070025

Date
Page

lab 5

1] fractional knapsack (greedy)

```
// input : set S of items, such that each item i∈S has
//         a positive benefit bi and a positive weight
//         Wi, positive total weight W
// output: Amount xi of each item i∈S that maximises
//         the total benefit while not exceeding total
//         weight W
for each item i∈S do
  xi ⇐0
  vi <- bi/wi
  w <- 0
while w < W
  remove from S an item i with highest value index
  a <- min{wi, W-w}
  xi <- a
  w <- a+w
```

2] Time complexity

→ best case : if only the first time is picked

$T(n) = O(1) = C$

sorting to arrange items in descending order of v_i which takes $T(n) = O(n \log n)$

$T(n) = O(n \log n) + O(1) = O(n \log n)$ //

→ worst case : if all the items are picked

$T(n) = O(n)$ (to traverse till end)

sorting to arrange items in descending order of v_i which takes $T(n) = O(n \log n)$

$T(n) = O(n \log n) + O(n) = O(n \log n)$

⇒ average case : $T(n) = O(n \log n)$ due to sorting

3] positive test cases

1] test case 1

items = [(110, 150, 6), (70, 60, 4), (200, 100, 8), (60, 75, 5), (150, 40, 3)]

items picked = [(150, 40, 3), (200, 100, 8), (70, 60, 4)]
Max value = 420

2] Test case 2

items = [(150, 140, 60), (130, 70, 5), (90, 80, 4), (200, 60, 7), (60, 45, 2)]

items picked = [(150, 140, 60), (200, 60, 7), (130, 70, 5), (80, 30, 2)]

Max value = 520

3] Test case 3

Items = [(160, 15, 6), (70, 20, 2), (90, 30, 4), (80, 80, 3), (71.4, 55, 5)]

Items picked = [(160, 15, 6), (70, 20, 2), (90, 30, 4), (80, 80, 3), (71.4, 55, 5)]

Max value = 339.28

4] Test case 4

Items = [(60, 10, 5), (20, 30, 6), (100, 20, 3), (40, 10, 7), (20, 5, 2)]

Items picked = [(60, 10, 5), (20, 30, 6), (100, 20, 3), (40, 10, 7), (20, 5, 2)]

Max value = 340

5] Test case 5

Items = [(150, 50, 8), (90, 95, 4), (100, 90, 2), (60, 100, 3), (80, 80, 5)]

Items picked = [(150, 50, 8), (100, 90, 2), (80, 80, 5)]

Max value = 310

4] Negative Test cases

1] Test case 1

```
item_1 = [
        Item (5, -10, 3),
        Item (10, -20, 4),
        Item (15, -30, 5),
        Item (20, -40, 6),
        ]
```

Output = Test case 1 : Negative values
Item 1 : invalid value : -10 , value cant be negative
Item 2 : invalid value : -20 , value cant be negative
Item 3 : invalid value : -30 , value cant be negative
Item 4 : invalid value : -40 , value cant be negative

2] Test case 2

```
Item_2 = [
        Item (-5, 10, 3),
        Item (-10, 20, 4),
        Item (-15, 30, 5),
        Item (-20, 40, 6),
        ]
```

Output = Test case 2 : Negative weight
Item 1 : Invalid weight : -5 , weight cant be negative
Item 2 : Invalid weight : -10 , weight cant be negative
Item 3 : Invalid weight : -15 , weight cant be negative
Item 4 : Invalid weight : -20 , weight cant be negative

3] Test case 3

```
Item - 3 = [
        Item (2, 20, 4),
        Item (3, 30, 6),
        Item (4, 44, 2),
        Item (10, 30, 3),
        ]
```

Knapsack capacity = 0

output = Test case 3: empty Knapsack capacity
error: capacity of Knapsack must be greater than 0

4] Test case 4

```
Item - 4 = [
        Item ("five", 10, 3),
        Item ("ten", 20, 4),
        Item ("fifteen", 30, 5),
        Item ("twenty", 40, 6),
        ]
```

output = Test case 4: String in weight
Item 1 : invalid weight : five, weight must be numeric
Item 2 : invalid weight : ten, weight must be numeric
Item 3 : invalid weight : fifteen, weight must be numeric
Item 4: invalid weight : twenty, weight must be numeric

5] Test case 5

```
Item_5 = [
        Item ( 0, 10, 3),
        Item ( 0, 20, 4),
        Item ( 0, 30, 5),
        Item ( 0, 40, 6),
        ]
```

output = Test case 5: all items with 0 weight
Item 1 : invalid weight: 0, weight must be greater than 0
Item 2 : invalid weight: 0, weight must be greater than 0
Item 3 : invalid weight: 0, weight must be greater than 0
Item 4: invalid weight: 0, weight must be greater than 0

Huffman

1) class huffmanNode
   char freq, left, right

2) function build_huffman_tree (txt)
   freq <- count_freq (txt)
   heap <- [huffmanNode (char, freq[char] for char in freq)]

   heapify (heap)
   while len (heap) > 1
      node 1 = heap_pop (heap)
      node 2 = heap_pop (heap)
      merged = huffman_node (node, node1.freq + node2.freq, node1, node 2)

      heap_push (heap, merged)

function __init__ (self, others)
   return self.freq < others.freq

function generate_code (node, code, codes)
   if node <- none
      return
   if node.char <- not none
      codes [node.char] = code

   generate_code (node.left, code + "0", codes)
   generate_code (node.right, code + "1", codes)

```
function compress(text)
    root = build_huffman_tree(text)
    generate_code(root, "", codes)
    compressed = "".join([codes[char] for char in text])
    return compressed, codes

function calculate_compression_ratio(original_text,
                                     compressed_text)
    size = len(original_text) * 8
    size_compressed = len(compressed_text) * 8
    ratio = size / size_compressed
    return size, size_compressed, ratio
```

→ Time complexity {n = size of input, m = no of chars}

1] counting char freq → O(n)
2] build heap with len(freq) → O(m log m)
3] generate huffman code → O(m)
4] compressing text → O(n) {iterating}
5] calculating ratio → O(1)

Total time complexity = $O(m \log m + n)$

- $m \log m$ → for constructing huffman tree
- n → for creating freq table

Date
Page

=) Test cases

a] positive test cases

1] intro.docx : input
original text size in bits = 1080
compressed text size in bits = 476
compression ratio : 0.44

2] input : intro.txt
original text size in bits = 7544
compressed text size in bits = 3356
compression ration = 0.44

3] input : intro.html
original text size in bits = 8200
compressed text size in bits = 3706
compression ratio = 0.45

4] input : intro.pdf
original text size in bits = 23456
compressed text size in bits = 10569
compression ratio = 0.45

b] negative test cases

1] input : intro.png
output : unsupported file type inputted

2] input : intro.jpeg
output : unsupported file type inputted

3] input: intro.ipynb
output: unsupported file type inputted

4] input: empty docx file
output: No content extracted from docx file, since it does not include anything and it is blank hence huffman compression cant be done and found here to proceed please enter some text

5] input: selected pdf optn but uploaded docx file
output: The file intro.docx is not a PDF file since you have selected the option of PDF file but uploaded a docx file which is a mismatch between formats

Date
Page

Conclusion:

We have implemented both fractional Knapsack and huffman encoding in this lab. By using frac. Knapsack we have optimised the transportation of XYZ courrier company by prioritising items based on their shell life and value. In huffman we took various formats of files such as text, docx, html and pdf and compressed them to find the compression ratio and smaller the value better is the ratio i.e should be less than 1