CSE 373 / L-14 / 31.03.2024 /

⊛ Fractional knapsack Problem

- Problem describe on page-17 (ch-16)

⊛ Fractional knapsack Algorithm:

```
GREEDY-FRACTIONAL-KNAPSACK (w, v, W)

for i = 1 to n
        x[i] = 0
weight = 0
while weight < W
i = best remaining item
if weight + w[i] ≤ W
        x[i] = 1
        weight = weight + w[i]
else
        x[i] = (W - weight) / w[i]
        weight = W
return x
```

Time analysis:

sort in decreasing order $O(n \lg n)$

then, while loop will take $O(n)$

$$\text{Total} = O(n \lg n) + O(n)$$
$$= O(n \lg n).$$

⇒ BUILD-MAX-HEAP $= O(n)$

while loop $= O(\lg n)$

$$\text{Total} = O(\lg n)$$

→ its faster, if only a small number of items are need to fill the knapsack.

Huffman Codes

⇒ very effecting technique for compressing data

- Savings of 20% to 90% typical.
- use a table of the frequencies of occurrence of each character.
- then represent each character as a binary string.

Example: A 100,000 character data file that is to be compressed only characters a, b, c, d, e, f appear.

⇒

	a	b	c	d	e	f
frequency (in thousands)	45k	13k	12k	16k	9k	5k
fixed length codeword	000	001	010	011	100	101
variable length codeword	0	101	100	111	1101	1100
variable length codeword	0	10	110	1110	11110	11111

⇒ each character represented by a unique binary string.

Fixed-length code

- needs 3 bits to represent 6 characters

$2^2 = 4$ $\quad$ $4 \le 6 \le 8$

$2^3 = 8$

- requires $100{,}000 \times 3 = 300{,}000$ bits to code the entire file.

$\Rightarrow$ original size $= 100{,}000 \times 8 = 800{,}000$ bits

Variable-length Code:

- more frequency = less bits
- less frequency = more bits

for 2nd example:

$$\text{size} = (45 \times 1 + 13 \times 3 + 12 \times 3 + 16 \times 3 + 9 \times 4 + 5 \times 4)\,k$$

$$= \underbrace{224{,}000 \text{ bits}}$$

less than fixed-length code.

Prefix Code:

- simplify encoding and decoding
- no codeword will repeat in ~~other~~ prefix of other codeword.

$\Rightarrow$ Encoding $\Rightarrow$ concatenate the codewords representing each character

abc $\Rightarrow$ ~~010100~~ 0101100

$\Rightarrow$ Decoding $\Rightarrow$

(i) Read ~~one~~ bit

(ii) is it match with any codeword?

Yes: go to step (iv)

No: go to step (iii)

(iii) read another bit and go to step (ii)

(iv) translate the codeword and go to step (i)

001011101 $\Rightarrow$ 0.0.101.1101 $\Rightarrow$ aabe

* Convenient representation for the prefix code

- a binary tree whose leaves are the given characters.

0 $\Rightarrow$ left child

1 $\Rightarrow$ right child

* Connection:

Slide Page-7: fig $\Rightarrow$ fixed length code

Slide Page-8: fig $\Rightarrow$ variable-length code

* Huffman Code invented by Huffman

Algorithm:

set of leaves

HUFFMAN (C)

$n = |C|$

Queue least freq. $Q = C$

for i = 1 to n-1

allocate a new node z

z.left = x = EXTRACT-MIN (Q)

z.right = y = EXTRACT-MIN (Q)

z.freq = x.freq + y.freq

INSERT (Q, z)

return EXTRACT-MIN (Q) // return the root of the tree

Illustration ⇒ Slide - 12 - 14

L-15/ 02.04.2024

Quize-2