

Övnings Närstar PROV 1

Problem set = Creating a greedy algorithm, that will arrange words on lines, making sure to minimise the number of lines used - whilst adhering to maximum line lengths.

Given:

n: number of words

len: maximum length allowed per line.

w<sub>1</sub>, w<sub>2</sub> . . . , w<sub>n</sub>: length of the words in order.

Note

each line can fit words as long as the total length of the words and spaces doesn't exceed len.

Objective = Find maximum number of lines required to fit the entire text, - while preserving word order.

Manual - Run-through

• n=10 • len=25

words      total len

LINKE 1	w <sub>1</sub> + w <sub>2</sub>	17 + 1 + 4
LINKE 2	w <sub>3</sub> + w <sub>4</sub> + w <sub>5</sub>	7 + 1 + 3 + 1 + 3
LINKE 3	w <sub>6</sub> + w <sub>7</sub> + w <sub>8</sub> + w <sub>9</sub>	2 + 1 + 9 + 1 + 2 + 1 + 7
LINKE 4	w <sub>10</sub>	6

$$\left\{ \begin{array}{l} w_1 = 17 \\ w_2 = 1 \\ w_3 = 7 \\ w_4 = 3 \\ w_5 = 13 \\ w_6 = 2 \\ w_7 = 9 \\ w_8 = 2 \\ w_9 = 7 \\ w_{10} = 6 \end{array} \right.$$

Tidskomplexiteten  $O(1)$  betyder att komplexiteten är oberoende av  $w_1, w_2, w_3, w_4, w_5, w_6, w_7, w_8, w_9, w_{10}$  längd

Greedy Algorithm

func minimal-num-lines (n, len, word-lengths [ ])

    current-line-len = 0

    num-of-lines = 1

    for i = 0 to n

        if (current-line-len == 0)

            current-line-len = word-lengths[i]

        else: ( $i + \text{current-line-len} + \text{word-lengths}[i] \leq \text{len}$ )

            current-line-len += 1 + word-lengths[i]

    else

        num-of-lines += 1

        current-line-len = word-lengths[i]

    return num-of-lines;

## Explanation of pseudo code:

- fn funktion `minimal_num_lines` is created which takes 3 parameters.
- The parameters "n" is the number of words, "len" gives the maximum length/line and `word_lengths [ ]` is an array that contains the length of each word.
- `word_lengths = [ 17, 4, 7, 3, 13, 2, 9, 2, 7, 6 ]`
- Variable Int `current_line_len = 0` is initiated to keep track of the length of the current lines length
- `num_of_lines = 1` is initiated to  $1 \rightarrow$  Since we start filling the words on line 1
- for i=0 to n. Here we began looping through the words starting at word 1
- We first check If (current\_line\_len = 0) when this is true, line is empty. then we can put w1 on the row. we will only put the words length without the extra space ↴
- Else, If the line already has a word  $\Rightarrow$  `current_line_len + 1 + word_length[i]` we need to put a space ↴ between w1 and w2.
- Then we check if we can put a space + new word there by checking if its less than len.
- If the word doesn't fit on the current line, we began a new line and start again
- Lastly, once we have gone through all the words, we can return `num_of_lines`.

## MOTIVERING

Girig strategi =

Algoritmen använder en girig strategi - vid varje steg försöker placera så många ord som möjligt utan att överskrida maximala rad längden

Vid varje steg gör algoritmen det lokala bästa valet

Vartför den fungerar?

Giriga algoritmen returnerar en optimal lösning =

- 1) Om vi skulle starta en ny rad tidigare än nödvändig, skulle vi använda fler rader än rad som krävs - icke OPTIMALT
- 2) Å andra sidan, om vi packar in så många ord som möjligt på varje rad, utan att överskrida gränsen - inte slårer rad utrymen.

Beweis

Anta att den giriga algoritmen delar upp orden i ett antal rader, och att det finns en bättre lösning som använder färre rader. Det skulle innebära att den alternativa lösningen packar orden mer effektivt på någon rad än den giriga lösningen.

Men eftersom den giriga algoritmen alltid fyller varje rad så mycket som möjligt utan att överskrida längdgränsen, är det omöjligt för den alternativa lösningen att packa orden bättre.

Detta leder till en motsägelse, eftersom den alternativa lösningen inte kan använda färre rader utan att bryta mot gränsen.

Därför är den giriga lösningen optimal.

## TIDSKOMPLEXITET

- **Antal iterationer:** Algoritmen itererar över alla ord en gång. Det betyder att om vi har n ord, så kommer algoritmen att utföra n iterationer.
- **Arbetet per iteration:** För varje iteration (dvs. för varje ord) gör algoritmen några konstanta operationer: den kontrollerar om ordet får plats på den aktuella raden (en jämförelse och en addition), och eventuellt uppdaterar den nuvarande radens längd. Om ordet inte får plats på raden, startar algoritmen en ny rad, vilket också är en konstant operation.

Eftersom algoritmen gör en konstant mängd arbete per ord och vi itererar över alla n ord, är den totala tidskomplexiteten för algoritmen **O(n)**. Detta innebär att algoritmen är linjär i förhållande till antalet ord, vilket är optimalt för detta problem.