# FOCS Homework 22

Edit your answers into this file, or add a separate file in the same directory. If you add a separate file, please include the following at the top:

```

Student Name: Frankly Olin [change to your name]

Check one:

[ ] I completed this assignment without assistance or external resources.

[ ] I completed this assignment with assistance from \_\_\_

and/or using these external resources: \_\_\_

```

## 1. Break dancing

A \*\*line break algorithm\*\*, or \*\*paragraph wrap algorithm\*\*, inserts newlines between words in order to maintain a maximum line length. An [early implementation](http://massis.lcs.mit.edu/archives/technical/western-union-tech-review/10-1/p040.htm) of this algorithm ran on mechanical relays.

Consider the following algorithm:

```python

def wrap(text, max\_width):

"""Return the words of `text`, wrapped to a max line length of `line\_width`."""

column = 0

lines = [[]]

for word in text.split():

# int(column > 0) is 0 if this is the first word on the line;

# else 1 (to account for the interpolated space)

if column + int(column > 0) + len(word) > max\_width:

column = 0

lines.append([])

lines[-1].append(word) # lines[-1] is the last item in the lines[] array

column += int(column > 0) + len(word)

return '\n'.join(' '.join(line) for line in lines)

print(wrap("aaa bb cc ddddd", 6))

print()

print(wrap("a b c d e f g h i j k l m n o p qqqqqqqqq", 9))

```

Output:

```

aaa bb

cc

ddddd

a b c d e

f g h i j

k l m n o

p

qqqqqqqqq

```

Questions:

a. Is this algorithm any of: greedy, recursive, dynamic programming (top down), dynamic programming (bottom up)?

**It’s greedy.**

b. If the text contains a word that is longer than `max\_width`, what does this algorithm do?

**The algorithm puts it all on one line anyway.**

c. Does this algorithm produce an optimal result? (Do you have enough information to answer this question?)

**I suppose that depends on the definition of optimal. Highest average words per line (and thus least lines) may not always be the same as least ragged (least squares of variance from the average).**

## 2. That's a wrap

Here's a ~~more complicated~~ alternative paragraph algorithm. This uses an explicit representation of Infinity to avoid a special case to rule out lines that are too long.

a. Is this algorithm any of: greedy, recursive, dynamic programming (top down), dynamic programming (bottom up)? **There’s some sort of dynamic programming going on here, I think. We compute the cost of breaking the paragraph basically everywhere, then choose the best ones. It seems like we’re building on past computation to do that, too, so I guess it’s bottom-up?**

b. [no question] See [this post by Juraj Sukop](http://xxyxyz.org/line-breaking/) for a more extensive exploration of line breaking algorithms using Python.

```python

import math

def less\_ragged(text, max\_width):

"""Return the words of `text`, wrapped to a max line length of `line\_width`.

Use Knuth-Plass to minimize the sum of the lines' raggedness

(the cube of each lines' amount of wasted space)."""

words = text.split()

word\_count = len(words)

word\_lengths = [len(w) for w in words]

line\_cost = {} # line\_cost[i, j] = the raggedness of a line consisting of words[i:j]

# line\_cost[i, j] is equivalent to line\_cost[(i, j)]. This code uses a dict (hash) to

# simulate a 2D array, without as much work to set up. It's less efficient by a

# constant factor.

for i in range(word\_count):

line\_length = -1 # accumulated length of words[i:j] and their interpolated

# spaces. initialize to -1 to compensate when general case

# adds the first word, that should't include an interpolated

# space, to the empty string

for j in range(i + 1, word\_count + 1):

# the length of the line containing words[i:j] + the j-i interpolated spaces

line\_length = sum(word\_lengths[k] for k in range(i, j)) + (j - i - 1)

slack = max\_width - line\_length

line\_cost[i, j] = slack \*\* 3 if slack >= 0 else math.inf

# breaks[j] represents the best place i to break the paragraph words[0:j]

# into a paragraph words[0:i] followed by a line words[i:j].

# Its value is the tuple (cost, i) where cost is the cost of this paragraph.

breaks = [(0, 0)]

for j in range(1, word\_count+1):

# generate a set of candidate positions at which to start the final line,

# and the cost of the best paragraph at of these final line positions.

#

# each candidate is (cost, i), so that subsequent computation can use the cost

# (to select the next candidate), and so that the final result can use the line

# break position i.

candidates = ((breaks[i][0] + line\_cost[i, j], i) for i in range(j))

# the order is (cost, i) in order to use `min` to select the best candidate.

# this works because tuple comparison uses lexicographic order.

breaks.append(min(candidates))

# start at the end of the paragraph. iterate from the last line to each preceding

# line, accumulating tuples (i, j) that indicate words[i:j] on each line

word\_ranges = []

j = word\_count

while j > 0:

i = breaks[j][1]

word\_ranges.append((i, j))

j = i

# word\_ranges is ordered from last line to first. iterate over it backwards,

# to collect the actual words in the order first line -> last line.

lines = ([words[i:j] for i, j in reversed(word\_ranges)])

return '\n'.join(' '.join(line) for line in lines)

print(less\_ragged("aaa bb cc ddddd", 5))

print()

print(less\_ragged("a b c d e f g h i j k l m n o p qqqqqqqqq", 9))

```

Output:

```

aaa

bb cc

ddddd

a b c d

e f g h

i j k l

m n o p

qqqqqqqqq

```

## 3. Knuth break

The algorithm in #2 was created by [Donald Knuth](https://en.wikipedia.org/wiki/Donald\_Knuth), for use in TeX (which he created).

Let's take a break to admire Knuth. Aside from foundational work in computer science:

1. In high school, he invented the [quater-imaginary numeral system](https://en.wikipedia.org/wiki/Quater-imaginary\_base), which uses the digits 0, 1, 2, and 3, and the base 2i to represent all the complex numbers.

2. That same year, he published the [Potrzebie system of weights and measures](https://github.com/focs16fall/focs-assignments/blob/master/day22/images/potrzebie\_system.gif). For a while, you could enter “1 farshimmelt potrzebie”, “1 ngogn”, or “1 furshlugginer blintz” into Google, to convert the measurement into the corresponding English or metric measure of weight, volume, or mass. Google has dropped this feature, and you now need Mathematica, Wolfram Language, or Wolfram Alpha.

3. Knuth is the originator of “Beware of bugs in the above code; I have only proved it correct, not tried it.”

4. He has his own [t-shirt](https://github.com/focs16fall/focs-assignments/blob/master/day22/images/knuth-tshirt-show.jpg).