Finding love

Prize-winning poet Wendy Cope's beautiful love poem *Another Valentine*, commissioned by the *Daily Telegraph* in 2009, explores the frequent criticisms levelled at the most romantic day of the year. Yet as the narrator muses on the obligation behind Valentine's Day, romantic feelings are conjured.

> *Today we are obliged to be romantic*
> *And think of yet another valentine.*
> *We know the rules and we are both pedantic:*
> *Today's the day we have to be romantic.*
>
> *Our love is old and sure, not new and frantic.*
> *You know I'm yours and I know you are mine.*
> *And saying that has made me feel romantic,*
> *My dearest love, my darling valentine.*

The attraction of love is reinforced by a special feature of the poem. Pick any word in the first three lines, count its letters and move ahead the corresponding number of words. For example, if you start at the word *obliged* on the first line, you'd count 7 letters and move ahead 7 words, landing on the word *yet* on the second line. Count that word's letters and continue in this manner until you've reached the end of the poem. Whatever word you choose as a starting point in the first three lines, you'll always arrive at the word *love* on the last line of the poem. You can check this by hoovering the mouse over the words of the poem. This way you can also check that if you start at the word *day* on the fourth line, you finally arrive at the word *romantic* on the penultimate line of the poem.

This special property is less surprising than it seems — it's based on a principle called the *Kruskal count*. It was originally proffered by Rutgers University physicist Martin Kruskal as a card trick, whose magical effect has been known to perplex professional magicians because — as he liked to say — it was based not on sleight of hand but on a mathematical phenomenon. In both the card trick and the love poem various tributaries merge into a common stream that arrives at a predictable destination. We further test the counting trick on the first three verses of Genesis:

> *In the beginning God created the heaven and the earth.*
>
> *And the earth was without form, and void;*
> *and darkness was upon the face of the deep.*
> *And the Spirit of God moved upon the face of the waters.*
>
> *And God said, Let there be light: and there was light.*

Pick any word in the first verse, count its letters, and move ahead by the corresponding number of words. For example, if you start at *beginning*, you'd count 9 letters and move ahead 9 words, landing on *the* in the second verse. Count that word's letters and continue in this manner until you've entered the third verse. You'll always arrive at *God*. It also works for this popular lullaby:

> *Twinkle, twinkle, little star,*
> *How I wonder what you are.*

Up above the world so high,
Like a diamond in the sky.
Twinkle, twinkle, little star,
How I wonder what you are.

Choose any word in the first two lines, count its letters, and count forward that number of words. For example, if you choose *star*, which has four letters, you'd count ahead four words, beginning with *how*, to reach *what*. Count the number of letters in that word and count ahead as before. Continue until you can't go any further. You'll always land on *you* in the last line. The last example was found by Martin Gardner in the opening of *Alice's Adventures in Wonderland*:

> *Alice was beginning to get very tired of sitting by her sister on the bank, and of having nothing to do; once or twice she had peeped into the book her sister was reading.*

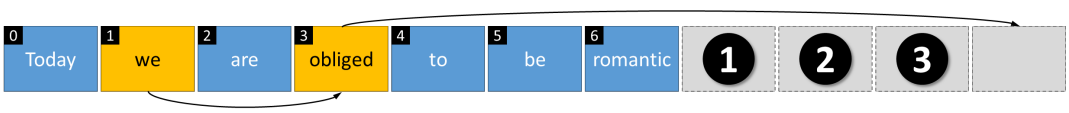
Choose any of the first 26 words and tap your way forward in the sentence from that point, tapping one word for each letter. So, for example, if you choose the word *Alice*, which has five letters, you'd tap *was*, *beginning*, *to*, *get*, and land on *very*. Then do the same thing with that word, advancing four letters to land on *by*. If you keep this up you'll always arrive at the word *sister* near the end of the fragment.

Assignment

Your task is to examine whether the counting trick works for the text contained in a given text file. We agree that whenever we split a text into words, the words are made up of the longest possible contiguous sequence of letters (uppercase or lowercase). That means for example that the string `Today's` contains two words: `Today` and `s`. You are asked to:

- Write a function `split` that takes a string as its argument. The function must return a list that contains the sequence of words in the given string.

- Write a function `sequence` that takes a sequence of words (a list or a tuple). The function also has a second optional parameter `start` that takes an integer argument (default value: 0). The function must return a tuple with two elements. The first element is the list of words that are visited when Kruskal count is applied on the given sequence of words, starting at the word whose position was passed to the parameter `start`. The second element is an integer that indicates the maximal number of words that could be appended to the given sequence, without the possibility to jump to the next word when Kruskal count is applied to the word at the given starting position.



  The above figure illustrates how the function `sequence` processes the words on the first line in the love poem. If we start at position 1, the list of words marked in yellow must be returned as the first element of the tuple. From the word *obliged* we would move ahead another 7 words, which is four words beyond the edge of the sequence. As a result, we could append 3 more words to the sequence without making it possible to jump to another word, which is the value that needs to be returned as the second element of the tuple.

- Write a function `kruskal` that takes the location of a text file. The function also has a second

optional parameter start that takes an integer argument (default value: 0). The function also has a third optional parameter last that takes a Boolean argument (default value: False). The function must return the list of words visited when Kruskal count is applied to the text contained in the given file, starting at the word whose position was passed to the parameter start. Positions of words are indexed starting from 0. In case the value False was passed to the parameter last, Kruskal count must be applied until the end of the text is reached. In case the value True was passed to the parameter last, Kruskal count must be applied until the first word is reached on the last line of the text. In the latter case, if no word is reached on the last line of the text, the function must raise an AssertionError with the message no word visited on last line.

- Write a function trick that takes the location of a text file. The function also has an optional parameter start that has the same meaning as with the function kruskal. The function must return a Boolean value that indicates whether or not all words on the first line of the given text file finally end up at the same word if Kruskal count is applied as in the function kruskal. Note that this means that it is possible that the function must raise an AssertionError with the message no word visited on last line.

Example

In the following interactive session we assume that the text files cope.txt, genesis.txt, twinkle.txt and alice.txt are located in the current directory.

```
>>> split('Today we are obliged to be romantic')
['Today', 'we', 'are', 'obliged', 'to', 'be', 'romantic']
>>> split('And think of yet another valentine.')
['And', 'think', 'of', 'yet', 'another', 'valentine']
>>> split('We know the rules and we are both pedantic:')
['We', 'know', 'the', 'rules', 'and', 'we', 'are', 'both', 'pedantic']
>>> split("Today's the day we have to be romantic.")
['Today', 's', 'the', 'day', 'we', 'have', 'to', 'be', 'romantic']

>>> words = ['Today', 'we', 'are', 'obliged', 'to', 'be', 'romantic']
>>> sequence(words)
(['Today', 'be'], 0)
>>> sequence(words, 1)
(['we', 'obliged'], 3)
>>> sequence(words, start=2)
(['are', 'be'], 0)

>>> kruskal('cope.txt', 3)
['obliged', 'yet', 'We', 'the', 'we', 'both', 'the', 'have', 'Our', 'old', 'not', 'frantic', 'I', 'know', 'And', 'has', 'feel', 'love']
>>> kruskal('cope.txt', 21)
['pedantic', 'be', 'Our', 'old', 'not', 'frantic', 'I', 'know', 'And', 'has', 'feel', 'love']
>>> kruskal('cope.txt', 25)
['day', 'to', 'romantic', 'new', 'You', 'm', 'yours', 'are', 'saying', 'romantic']
>>> kruskal('cope.txt', 25, last=True)
Traceback (most recent call last):
AssertionError: no word visited on last line

>>> trick('cope.txt')
True
>>> trick('cope.txt', last=True)
True

>>> trick('genesis.txt')
```

```
True
>>> trick('genesis.txt', last=True)
True

>>> trick('twinkle.txt')
True
>>> trick('twinkle.txt', last=True)
True

>>> trick('alice.txt')
True
>>> trick('alice.txt', last=True)
False
```

Resources

- **Lagarias JC, Rains E, Vanderbei RJ (2009)**. The Kruskal Count. *The Mathematics of Preference, Choice and Order*. Springer Berlin Heidelberg, 371-391.

In 2009 schreef de gelauwerde dichteres Wendy Cope in opdracht van de *Daily Telegraph* het prachtige liefdesgedicht *Another Valentine*. Hierin gaat ze op onderzoek naar de grond van de kritiek die vaak te beurt valt aan de meest romantische dag van het jaar. Terwijl de verteller uit het gedicht mijmert over de verplichtingen van Valentijn, wordt ze zelf overvallen door romantische gevoelens.

> *Today we are obliged to be romantic*
> *And think of yet another valentine.*
> *We know the rules and we are both pedantic:*
> *Today's the day we have to be romantic.*
>
> *Our love is old and sure, not new and frantic.*
> *You know I'm yours and I know you are mine.*
> *And saying that has made me feel romantic,*
> *My dearest love, my darling valentine.*

De aantrekkingskracht van de liefde wordt nog versterkt door een wel heel bijzondere eigenschap van het gedicht. Kies een willekeurig woord uit de eerste drie regels, tel de letters van dit woord, en spring het overeenkomstig aantal woorden vooruit. Stel bijvoorbeeld dat je begint bij het woord *obliged* op de eerste regel: dit woord bestaat uit 7 letters, dus spring je 7 woorden vooruit naar het woord *yet* op de tweede regel. Tel het aantal letters van dat woord, en blijf op die manier vooruitspringen tot je niet meer verder kan. Welk woord op de eerste drie regels je ook kiest als vertrekpunt, je komt altijd uit bij het woord *love* op de laatste regel. Je kan dit nagaan door met de muis over de woorden van het gedicht te bewegen. Op die manier kan je ook zien dat als je vertrekt vanaf het woord *day* op de vierde regel, je finaal uitkomt bij het woord *romantic* op de voorlaatste regel.

Deze bijzondere eigenschap is echter minder toevallig dan je op het eerste gezicht misschien zou vermoeden — het is gebaseerd op een principe dat *Kruskal's telprocedure* genoemd wordt. Deze telprocedure werd door de fysicus Martin Kruskal van Rutgers University voor het eerst gebruikt in een kaarttruc, waarmee hij goochelaars van over de hele wereld introduceerde in de wiskundige theorie van de Markovketens. Zowel bij de kaarttruc als bij het liefdesgedicht komt het erop neer dat verschillende vertakkingen samenkomen in een gemeenschappelijke stroom

die uitmondt in een voorspelbare bestemming. We testen deze telprocedure verder uit op de eerste drie verzen van het bijbelboek Genesis:

> *In the beginning God created the heaven and the earth.*
>
> *And the earth was without form, and void;*
> *and darkness was upon the face of the deep.*
> *And the Spirit of God moved upon the face of the waters.*
>
> *And God said, Let there be light: and there was light.*

Kies ook hier een willekeurig woord uit het eerste vers, tel de letters van dit woord, en spring het overeenkomstig aantal woorden vooruit. Stel bijvoorbeeld dat je start bij het woord *beginning*: dat woord bestaat uit 9 letters, dus spring je 9 woorden vooruit naar het eerste voorkomen van het woord *the* in het tweede vers. Tel het aantal letters van dat woord, en blijf op die manier vooruitspringen tot je een woord van het derde vers bereikt. Welk woord uit het eerste vers je ook kiest als vertrekpunt, alle wegen leiden uiteindelijk naar het woord *God* in het derde vers. Het werkt ook op dit bekende Engelse slaapliedje:

> *Twinkle, twinkle, little star,*
> *How I wonder what you are.*
> *Up above the world so high,*
> *Like a diamond in the sky.*
> *Twinkle, twinkle, little star,*
> *How I wonder what you are.*

Kies een willekeurig woord op de eerste twee regels, tel het aantal letters, en spring dat aantal woorden vooruit. Als je bijvoorbeeld het woord *star* kiest, dat bestaat uit vier letters, dan spring je vier woorden vooruit naar het woord *what* op de tweede regel. Als je dit blijft volhouden, dan is het laatste woord waar je naartoe kan springen altijd het woord *you*. Het laatste voorbeeld werd gevonden door Martin Gardner in de openingszinnen van *Alice's Adventures in Wonderland*:

> *Alice was beginning to get very tired of sitting by her sister on the bank, and of having nothing to do; once or twice she had peeped into the book her sister was reading.*

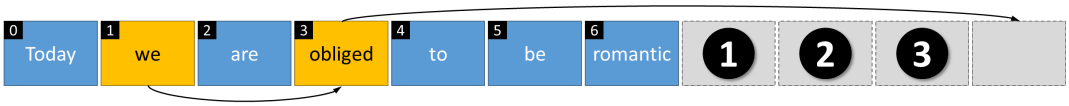
De eerste 26 woorden leiden allemaal naar het woord *sister* op het einde van dit tekstfragment.

Opgave

Je opdracht bestaat erin om voor de tekst in een gegeven tekstbestand na te gaan of de truc werkt. Bij het opsplitsen van een tekst in woorden leggen we vast dat de woorden bestaan uit de langst mogelijke opeenvolging van letters (hoofdletters of kleine letters). Dat betekent dus bijvoorbeeld dat de string `Today's` twee woorden bevat: `Today` en `s`. Gevraagd wordt:

- Schrijf een functie `splits` waaraan een string moet doorgegeven worden. De functie moet een lijst teruggeven met alle opeenvolgende woorden uit de gegeven string.
- Schrijf een functie `reeks` waaraan een reeks woorden (een lijst of een tuple) moet doorgegeven worden. De functie heeft ook nog een tweede optionele parameter `start` waaraan een natuurlijk getal kan doorgegeven worden (standaardwaarde: 0). De functie

moet een tuple met twee elementen teruggeven. Het eerste element is de lijst van woorden die men bezoekt als men Kruskal's telprocedure toepast op de gegeven reeks woorden, te beginnen met het woord op de positie die werd doorgegeven aan de parameter start. Het tweede element is een natuurlijk getal dat aangeeft hoeveel woorden men maximaal nog aan de gegeven reeks zou kunnen toevoegen, zonder dat nog naar een volgend woord kan gesprongen worden als men Kruskal's telprocedure toepast op het woord op de gegeven startpositie.



In bovenstaande figuur illustreren we de werking van de functie reeks op de woorden uit de eerste regel van het liefdesgedicht. Als we starten op positie 1, dan wordt de lijst met woorden die in het geel aangeduid zijn teruggegeven als eerste element van het tuple. Vanaf het woord *obliged* zouden we 7 woorden moeten vooruitspringen naar een volgende woord, wat vier woorden voorbij het laatste woord van de gegeven reeks woorden ligt. Er zouden dus nog 3 woorden aan de reeks kunnen toegevoegd worden zonder dat naar een volgende woord kan gesprongen worden, wat de waarde is die als tweede element van het tuple moet teruggegeven worden.

- Schrijf een functie kruskal waaraan de locatie van een tekstbestand moet doorgegeven worden. De functie heeft ook nog een tweede optionele parameter start waaraan een natuurlijk getal kan doorgegeven worden (standaardwaarde: 0). De functie heeft ook nog een derde optionele parameter laatste waaraan een Booleaanse waarde kan doorgegeven worden (standaardwaarde: False). De functie moet de lijst met woorden teruggeven die men bezoekt als men Kruskal's telprocedure toepast op de tekst in het gegeven bestand, te beginnen met het woord op de positie die werd doorgegeven aan de parameter start. Hierbij worden de woorden genummerd vanaf 0. Indien de waarde False werd doorgegeven aan de parameter laatste, dan moet Kruskal's telprocedure worden toegepast tot men niet meer naar een volgend woord kan springen. Indien de waarde True werd doorgegeven aan de parameter laatste, dan moet Kruskal's telprocedure worden toegepast tot het eerste woord bereikt wordt op de laatste regel. Wanneer in dit laatste geval geen woord bereikt wordt op de laatste regel, dan moet de functie een AssertionError opwerpen met de boodschap geen woord bereikt op laatste regel.
- Schrijf een functie truc waaraan de locatie van een tekstbestand moet doorgegeven worden. De functie heeft ook nog een optionele parameter laatste (standaardwaarde: False) met dezelfde betekenis als bij de functie kruskal. De functie moet een Booleaanse waarde teruggeven, die aangeeft of alle woorden op de eerste regel in het gegeven tekstbestand finaal eindigen op eenzelfde woord als men Kruskal's telprocedure toepast zoals bij de functie kruskal. Merk dus op dat het ook mogelijk is dat deze functie een AssertionError moet opwerpen met de boodschap geen woord bereikt op laatste regel.

Voorbeeld

Bij onderstaande voorbeeldsessie gaan we ervan uit dat de tekstbestanden cope.txt, genesis.txt, twinkle.txt en alice.txt zich in de huidige directory bevinden.

```
>>> splits('Today we are obliged to be romantic')
['Today', 'we', 'are', 'obliged', 'to', 'be', 'romantic']
```

```
>>> splits('And think of yet another valentine.')
['And', 'think', 'of', 'yet', 'another', 'valentine']
>>> splits('We know the rules and we are both pedantic:')
['We', 'know', 'the', 'rules', 'and', 'we', 'are', 'both', 'pedantic']
>>> splits("Today's the day we have to be romantic.")
['Today', 's', 'the', 'day', 'we', 'have', 'to', 'be', 'romantic']

>>> woorden = ['Today', 'we', 'are', 'obliged', 'to', 'be', 'romantic']
>>> reeks(woorden)
(['Today', 'be'], 0)
>>> reeks(woorden, 1)
(['we', 'obliged'], 3)
>>> reeks(woorden, start=2)
(['are', 'be'], 0)

>>> kruskal('cope.txt', 3)
['obliged', 'yet', 'We', 'the', 'we', 'both', 'the', 'have', 'Our', 'old', 'not', 'frantic', 'I', 'know', 'And', 'has', 'feel', 'love']
>>> kruskal('cope.txt', 21)
['pedantic', 'be', 'Our', 'old', 'not', 'frantic', 'I', 'know', 'And', 'has', 'feel', 'love']
>>> kruskal('cope.txt', 25)
['day', 'to', 'romantic', 'new', 'You', 'm', 'yours', 'are', 'saying', 'romantic']
>>> kruskal('cope.txt', 25, laatste=True)
Traceback (most recent call last):
AssertionError: geen woord bereikt op laatste regel

>>> truc('cope.txt')
True
>>> truc('cope.txt', laatste=True)
True

>>> truc('genesis.txt')
True
>>> truc('genesis.txt', laatste=True)
True

>>> truc('twinkle.txt')
True
>>> truc('twinkle.txt', laatste=True)
True

>>> truc('alice.txt')
True
>>> truc('alice.txt', laatste=True)
False
```

Bronnen

- **Lagarias JC, Rains E, Vanderbei RJ (2009)**. The Kruskal Count. *The Mathematics of Preference, Choice and Order*. Springer Berlin Heidelberg, 371-391.