import copy

import logging

from Bio import SeqIO

from Bio.Seq import Seq

from Bio.Alphabet import IUPAC

from Bio.SeqRecord import SeqRecord

class Grange:

def \_\_init\_\_(self, genome\_name, chrom, start, end, strand, genome\_seq, genome\_len, circular=True):

print("Start: %s" % start)

print("End: %s" % end)

print("Length: %s" % genome\_len)

if start == 0 and end > start: # custom line added (from developers of Bacannot)

start += 1 # custom line added

end += 1 # custom line added

elif start == -1 and end > start:

start += 2 # LW--for all?

end += 2 # --for all?

if start <= 0 or end <= 0 or start > genome\_len or end > genome\_len:

raise ValueError("GRange: start or end position is out of the range.")

self.genome\_name = genome\_name

self.chr = chrom

self.start = start

self.end = end

self.strand = strand

self.genome\_seq = genome\_seq

self.genome\_len = genome\_len

self.circular = circular

self.width = self.\_\_len\_\_()

def set\_start(self, start):

self.start = start

self.width = self.\_\_len\_\_()

def set\_end(self, end):

self.end = end

self.width = self.\_\_len\_\_()

def get\_flank\_range(self, flank):

if flank + flank + self.width > self.genome\_len:

max\_flank = int((self.genome\_len - self.width) / 2)

else:

max\_flank = flank

flank\_start = self.start - max\_flank

if flank\_start <= 0:

flank\_start = flank\_start + self.genome\_len if self.circular else 1

flank\_end = self.end + max\_flank

if flank\_end > self.genome\_len:

flank\_end = flank\_end - self.genome\_len if self.circular else self.genome\_len

return Grange(self.genome\_name, self.chr, flank\_start, flank\_end, self.strand,

self.genome\_seq, self.genome\_len, self.circular)

def shift\_left(self, size):

self.start -= size

if self.start <= 0:

self.start = self.start + self.genome\_len if self.circular else 1

self.end -= size

if self.end <= 0:

self.end = self.end + self.genome\_len if self.circular else 1

self.width = self.\_\_len\_\_()

def remap\_offsets(self, left\_offset, right\_offset):

if self.strand == "+":

self.end = self.start + right\_offset - 1

self.start = self.start + left\_offset - 1

if self.start > self.genome\_len:

self.start = self.start - self.genome\_len if self.circular else self.genome\_len

if self.end > self.genome\_len:

self.end = self.end - self.genome\_len if self.circular else self.genome\_len

else:

self.start = self.end - right\_offset

self.end = self.end - left\_offset

if self.start <= 0:

self.start = self.start + self.genome\_len if self.circular else 1

if self.end <= 0:

self.end = self.end + self.genome\_len if self.circular else 1

self.width = self.\_\_len\_\_()

def has\_overlap(self, other, ignore\_strand=False, flank=0):

return self.get\_overlap\_length(other, ignore\_strand, flank) > 0

def get\_overlap\_length(self, other, ignore\_strand=False, flank=0):

other\_range = copy.copy(other)

new\_range = self.get\_flank\_range(flank)

size = min(new\_range.start, other\_range.start)-1

new\_range.shift\_left(size)

other\_range.shift\_left(size)

if new\_range.start <= other\_range.start:

overlap = min(new\_range.end-other\_range.start+1, other.width)

else:

overlap = min(other\_range.end-new\_range.start+1, new\_range.width)

if self.strand != other.strand and not ignore\_strand:

overlap = 0

# print(self.start, self.end, self.strand, other.start, other.end, other.strand, flank, overlap)

return overlap

def is\_inside(self, other, ignore\_strand=False):

return self.get\_overlap\_length(other, ignore\_strand, 0) == self.width

def get\_flank\_lengths(self, flank):

new\_range = self.get\_flank\_range(flank)

left\_flank = self.start - new\_range.start

if left\_flank <= 0:

left\_flank += self.genome\_len

right\_flank = new\_range.end - self.end

if right\_flank <= 0:

right\_flank += self.genome\_len

return left\_flank, right\_flank

def get\_sequence(self, flank=0, protein=False):

new\_range = self.get\_flank\_range(flank)

if new\_range.start <= new\_range.end:

seq = self.genome\_seq[new\_range.start-1:new\_range.end]

else: # element crossing the genome boundary

seq = self.genome\_seq[new\_range.start-1:self.genome\_len] + self.genome\_seq[0:new\_range.end]

if self.strand == '-':

seq = seq.reverse\_complement()

if protein:

seq = seq.translate(table=11)

return SeqRecord(seq, id=self.genome\_name, description='')

def \_\_str\_\_(self):

return "{} {} {} {} {}".format(self.genome\_name, self.chr, self.start, self.end, self.strand)

def \_\_len\_\_(self):

out\_len = self.end - self.start + 1

if out\_len <= 0: # element crossing the genome boundary

out\_len += self.genome\_len

return out\_len