Project4 Assemble

Shengyuan Wang
Apr 2, 2020

In a practical, we saw the scs function (copied below along with overlap) for finding the shortest common superstring of a set of strings.

In [1]:
```
def overlap(a, b, min_length=3):
    """ Return length of longest suffix of 'a' matching
        a prefix of 'b' that is at least 'min_length'
        characters long.  If no such overlap exists,
        return 0. """
    start = 0  # start all the way at the left
    while True:
        start = a.find(b[:min_length], start)  # look for b's suffx in a
        if start == -1:  # no more occurrences to right
            return 0
        # found occurrence; check for full suffix/prefix match
        if b.startswith(a[start:]):
            return len(a)-start
        start += 1  # move just past previous match

import itertools

def scs(ss):
    """ Returns shortest common superstring of given
        strings, which must be the same length """
    shortest_sup = None
    for ssperm in itertools.permutations(ss):
        sup = ssperm[0]  # superstring starts as first string
        for i in range(len(ss)-1):
            # overlap adjacent strings A and B in the permutation
            olen = overlap(ssperm[i], ssperm[i+1], min_length=1)
            # add non-overlapping portion of B to superstring
            sup += ssperm[i+1][olen:]
        if shortest_sup is None or len(sup) < len(shortest_sup):
            shortest_sup = sup  # found shorter superstring
    return shortest_sup  # return shortest
```

It's possible for there to be multiple different shortest common superstrings for the same set of input strings. Consider the input strings ABC, BCA, CAB. One shortest common superstring is ABCAB but another is BCABC and another is CABCA.

Q1. What is the length of the shortest common superstring of the following strings?
CCT, CTT, TGC, TGG, GAT, ATT

Q2. How many different shortest common superstrings are there for the input strings given in the previous question?

In [4]:
```
def revised_scs(ss):
    """ Returns shortest common superstring of given
        strings, which must be the same length """
    shortest_sup = []
    for ssperm in itertools.permutations(ss):
        sup = ssperm[0]  # superstring starts as first string
        for i in range(len(ss)-1):
            # overlap adjacent strings A and B in the permutation
            olen = overlap(ssperm[i], ssperm[i+1], min_length=1)
            # add non-overlapping portion of B to superstring
            sup += ssperm[i+1][olen:]
        shortest_sup.append(sup)  # found shorter superstring
    shortest_len = len(ss) * len(ss[0])
    for sup in shortest_sup:
        if len(sup) <= shortest_len:
            shortest_len = len(sup)
    shortest_sup = [sup for sup in shortest_sup if len(sup) == shortest_len]
    return list(set(shortest_sup))  # return shortest
```

In [5]:
```
# Question 1, 2
shortest_sup_list = revised_scs(['CCT', 'CTT', 'TGC', 'TGG', 'GAT', 'ATT'])
print("Length of the shortest common superstring:", len(shortest_sup_list[0]))
print("Number of different shortest common superstrings:", len(shortest_sup_list))
```

```
Length of the shortest common superstring: 11
Number of different shortest common superstrings: 4
```

Assemble reads using one of the approaches discussed, such as greedy shortest common superstring. Since there are many reads, you might consider ways to make the algorithm faster, such as the one discussed in the programming assignment in the previous module.

Q3. How many As are there in the full, assembled genome?

Q4. How many Ts are there in the full, assembled genome from the previous question?

In [10]:
```
import operator


def readFastq(filename):
    sequences = []
    qualities = []
    with open(filename) as fh:
        while True:
            fh.readline()  # skip name line
            seq = fh.readline().rstrip()  # read base sequence
            fh.readline()  # skip placeholder line
            qual = fh.readline().rstrip() # base quality line
            if len(seq) == 0:
                break
            sequences.append(seq)
            qualities.append(qual)
    return sequences, qualities


def smart_overlap_map(reads, k):
    olaps = {}
    result = {}
    for read in reads:
        for i in range(len(read)-k+1):
            if read[i:i+k] not in olaps:
                olaps[read[i:i+k]] = [read]
            else:
                olaps[read[i:i+k]].append(read)

    count = 0
    for read in reads:
        read_suffix = read[-k:]
        for possible_read in olaps[read_suffix]:
            if possible_read != read:
                olen = overlap(read, possible_read, k)
                if olen > 0:
                    count += 1
                    result[(read, possible_read)] = olen

    return result, count


def pick_maximal_overlap(reads, k):
    """ Return a pair of reads from the list with a
        maximal suffix/prefix overlap >= k.  Returns
        overlap length 0 if there are no such overlaps."""
    reada, readb = None, None
    best_olen = 0
    for a, b in itertools.permutations(reads, 2):
        olen = overlap(a, b, min_length=k)
        if olen > best_olen:
            reada, readb = a, b
            best_olen = olen
    return reada, readb, best_olen


def smart_greedy_scs(reads, k):
    """ Greedy shortest-common-superstring merge.
        Repeat until no edges (overlaps of length >= k)
        remain. """
    pairs_olen, pairs_count = smart_overlap_map(reads, k)
    sorted_pairs_olen = sorted(pairs_olen.items(), key=operator.itemgetter(1), reverse=True)
    read_a, read_b, olen = sorted_pairs_olen[0][0][0], sorted_pairs_olen[0][0][1], sorted_pairs_olen[0][1]
    while olen > 0:
        reads.remove(read_a)
        reads.remove(read_b)
        reads.append(read_a + read_b[olen:])
        pairs_olen, pairs_count = smart_overlap_map(reads, k)
        if pairs_olen != {}:
            sorted_pairs_olen = sorted(pairs_olen.items(), key=operator.itemgetter(1), reverse=True)
            read_a, read_b, olen = sorted_pairs_olen[0][0][0], sorted_pairs_olen[0][0][1], sorted_pairs_olen[0][1]
        else:
            read_a, read_b, olen = pick_maximal_overlap(reads, k)
    return ''.join(reads)
```

In [12]:
```
# Question 3, 4
reads_filename = 'ads1_week4_reads.fastq'
fastq_reads, _ = readFastq(reads_filename)

genome = smart_greedy_scs(fastq_reads, 10)
print("Number of As in the assembled genome:", genome.count('A'))
print("Number of Ts in the assembled genome:", genome.count('T'))
```

```
Number of As in the assembled genome: 4633
Number of Ts in the assembled genome: 3723
```