Part1

September 15, 2020

1 Assignment 1.1

1.0.1 Question 1

```python
[1]: def suffix_array(DNA):
         """Takes in a sequence with/without $ and returns the suffix array.
         Can also return sorted suffixes"""
         n = len(DNA)
         indexes = list(range(n)) #create list of indexes for all suffixes
         suffixes = []

         #build suffixes in reverse order
         for i in range(n):
             s = DNA[i:n]
             suffixes.append(s)

         SA = []
         sorted_suffixes = []
         #sort indexes according to suffixes
         for s, i in sorted(zip(suffixes, indexes)):
             SA.append(i)
             sorted_suffixes.append(s)

         #print(sorted_suffixes)
         #print(SA)
         return SA
```

1.0.2 Question 2

```python
[2]: def binary_search(query, genome, SA):
         """Takes in a query string, a genome string with one or more $ and
         a suffix array (list of indexes of the genome suffixes)
         """
         #variables
         global depth
         n = len(query)
         #Base case
```

```python
    if len(SA) >= 1: #suffix array contains 1 or more objects so search
        #mid = int(np.floor(len(SA)/2))
        #mid = int(np.ceil(len(SA)/2 - 1))
        mid = len(SA)//2 #get midpoint of SA
        #match
        if query == genome[SA[mid]:SA[mid]+n]:
            matches = []
            #matches.append(SA[mid])
            start = mid # find all matches after the starting point
            while query == genome[SA[mid]:SA[mid]+n]:
                matches.append(SA[mid])
                mid += 1
            mid = start - 1 #find all matches before the starting point
            while query == genome[SA[mid]:SA[mid]+n]:
                matches.append(SA[mid])
                mid -= 1
            return matches
        elif query < genome[SA[mid]:SA[mid]+n]: #search smaller half
            depth +=1
            return binary_search(query, genome, SA[0:mid])
        else:
            depth +=1
            return binary_search(query, genome, SA[mid+1:len(SA)]) # search␣
 ↪bigger half
    else: #no match
        return []
```

```python
[3]: def read_fasta(file):
        """
        Takes in a text file amnd returns a string where all chromosomes are
        seperated by $
        """
        f = open(file, "r")
        lines = f.readlines()
        f.close()
        genome = ""
        for line in lines:
            line = line.rstrip() # get rid of \n
            #print(line)
            if line[0] == ">" and len(genome) == 0: # dont add $ if genome is empty
                pass
            elif line[0] == ">": # add $ for all new chromosomes
                genome = genome + "$"
            else:
                genome = genome + line
        genome = genome + "$" #add $ at the end
        return genome
```

1.0.3 Question 3

```
[4]: depth = 0
     genome = read_fasta("genome.fasta")
     sa = suffix_array(genome)
     hits = binary_search('ACCGT',genome,sa)
     hits
```

```
[4]: []
```

1.0.4 Question 4

```
[5]: def print_suffixInterval(genome,sa,i,j,length=10):
       ''' Print small part of a suffix array '''
       while i < j and i<len(genome):
         print('{:5}{:5} '.format(i,sa[i]),genome[sa[i]:sa[i]+length])
         i += 1

     # Here your suffix_array function is called
     sa = suffix_array(genome)
     # Here a small arbitrary part of it is printed
     print_suffixInterval(genome,sa,10,20)

     # Here your binary_search program is tested with the query sequence 'AACC'
     hits = binary_search('AACC',genome,sa)
     # The result is printed
     print("\nHits found:\n  Pos   Seq")
     for i in hits:
       print('{:5} '.format(i),genome[i:i+4])
```

```
   10   228   AAATCGGGGG
   11   631   AAATGCGCTC
   12   332   AAATGGCCGT
   13   597   AAATTCCCGG
   14   274   AACAAACTGG
   15   391   AACACCGCGC
   16   315   AACCGACTAC
   17   415   AACCTG$GTA
   18   359   AACGTGCTGC
   19   245   AACTCTG$GT

Hits found:
  Pos   Seq
  315   AACC
  415   AACC
```

1.0.5 Question 5

A genome of length L must be halved $\log_2(L)$ times.

A genome of length $2 \times 6 \times 10^9$ must be halved 33.4 times.

A genome of length 688 (this assignment) must be halved 9.4 times.

```python
[6]: print("Length of Genome:", len(genome))
     print("Search Depth:", depth)
```

```
Length of Genome: 688
Search Depth: 16
```