Solution sheet

High Throughput Sequencing Methods
Bioinformatics Leipzig

WS 13/14

1 Bowtie

1. Bowtie searches subsequences in a given genome or sequence to know where exactly the subsequences are located in the genome.
2. Methods used: Suffix arrays to store all possible suffixes of the genome in an ordered way; BWT:to store the genome data in an efficient way (less storage space);FM-index: to find positions faster using just a bit more storage.

1.1 Suffix array

pos	source index	suffix
1	14	$
2	10	atik$
3	1	bioinformatik$
4	6	formatik$
5	12	ik$
6	4	informatik$
7	2	ioinformatik$
8	13	k$
9	9	matik$
10	5	nformatik$
11	3	oinformatik$
12	7	ormatik$
13	8	rmatik$
14	11	tik$

Since the suffixes are in alphabetical order, just the source index column can be used as the suffix array.

BWT table:

pos	source index	suffix	letter
1	14	$bioinformati	k
2	10	atik$bioinfor	m
3	1	bioinformatik	$
4	6	formatik$bioi	n
5	12	ik$bioinforma	t
6	4	informatik$bi	o
7	2	ioinformatik$	b
8	13	k$bioinformat	i
9	9	matik$bioinfo	r
10	5	nformatik$bio	i
11	3	oinformatik$b	i
12	7	ormatik$bioin	f
13	8	rmatik$bioinf	o
14	11	tik$bioinform	a

1.2 Burrows Wheeler Transformation

First, get the second column by sorting the given letters alphabetically. Then do the BWT backwars. The string is: ALGORITHMUS.

1.3 FM-index

sequence: **GCGAATATCTGAAATGCTTA**

pos	source index	suffix	letter
0	21	$	A
1	20	A$...	T
2	12	AAATG...	G
3	4	AATAT...	G
4	13	AATGC...	A
5	5	ATATC...	A
6	7	ATCTG...	T
7	14	ATGCT...	A
8	2	CGAAT...	G
9	9	CTGAA...	T
10	17	CTTA$...	G
11	11	GAAAT...	T
12	3	GAATA...	C
13	1	GCGAA...	$
14	16	GCTTA	T
15	19	TA$...	T
16	6	TATCT...	A
17	8	TCTGA...	A
18	10	TGAAA...	C
19	15	TGCTT...	A
20	18	TTA$...	C

O-matrix

ind	A	C	T	G
0	1	0	0	0
1	0	0	1	0
2	0	0	1	1
3	0	0	1	2
4	2	0	1	2
5	3	0	1	2
6	3	0	2	2
7	4	0	2	2
8	4	0	2	3
9	4	0	3	3
10	4	0	3	4
11	4	0	4	4
12	4	1	4	4
13	4	1	4	4
14	4	1	5	4
15	4	1	6	4
16	5	1	6	4
17	6	1	6	4
18	6	2	6	4
19	7	2	6	4
20	7	3	6	4

C-array: (0,7,10,14)
Starting either by 0 or by 1 when counting the positions is both possible.
Find subsequence aW:
$\underline{R} = C(a) + O(a, \underline{R}(W) - 1) + 1$
$\overline{R} = C(a) + O(a, \overline{R}(W))$
whereas: $\underline{R}(\emptyset) = 1$ and $\overline{R}(\emptyset) = n$.

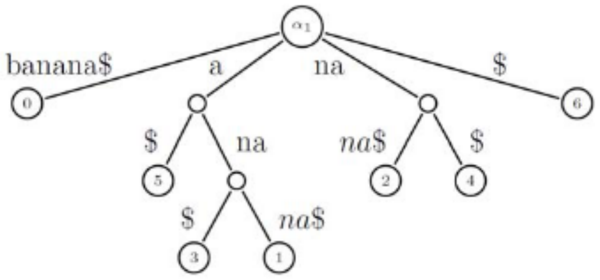
seq	1 letter	2 letters	3 letters
AAT	$T : a = T, W = \emptyset$ $\underline{R} = 14 + 0 + 1 = 15$ $\overline{R} = 14 + 6 = 20$	$AT : a = A, W = T$ $\underline{R} = 0 + 4 + 1 = 5$ $\overline{R} = 0 + 7 = 7$	$AAT : a = A, W = AT$ $\underline{R} = 0 + 2 + 1 = 3$ $\overline{R} = 0 + 4 = 4$
CAA:	$A : a = A, W = \emptyset$ $\underline{R} = 0 + 0 + 1 = 1$ $\overline{R} = 0 + 7 = 7$	$AA : a = A, W = A$ $\underline{R} = 0 + 1 + 1 = 2$ $\overline{R} = 0 + 4 = 4$	$CAA : a = C, W = AA$ $\underline{R} = 7 + 0 + 1 = 8$ $\overline{R} = 7 + 0 = 7$
TAT:	$T : a = T, W = \emptyset$ $\underline{R} = 14 + 0 + 1 = 15$ $\overline{R} = 14 + 6 = 20$	$AT : a = A, W = T$ $\underline{R} = 0 + 4 + 1 = 5$ $\overline{R} = 0 + 7 = 7$	$TAT : a = T, W = AT$ $\underline{R} = 14 + 1 + 1 = 16$ $\overline{R} = 14 + 2 = 16$

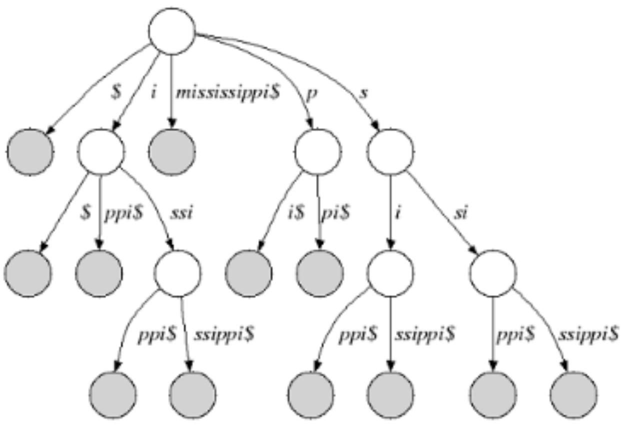
The subsequence **AAT** can be found at positions 3 and 4 in the suffix array. The subsequence **CAA** does not exist in the array since $\underline{R} > \overline{R}$. **TAT** exists once at position 16.

2 Segemehl

1. Segemehl works in a similar way as Bowtie, it also searches subsequences in a given sequence or genome. It also outputs the positions of the subsequences in the given data.
2. Segemehl uses suffix trees. To easily correct mistakes it uses the longest common prefix array and the so called suffix links, which are created by looking at the longest common prefixes of two strings.

2.1 Suffix tree





2.2 Longest common prefix

Mississippi$.

1. (ississsippi, issippi, ippi, i), (ssissippi, ssippi, sissippi, sippi), (ppi, pi)
2. (ississsippi, issippi), (ssissippi, ssippi), (sissippi, sippi)
3. (ississsippi, issippi), (ssissippi, ssippi)
4. (ississsippi, issippi)

banana$

1. (anana, ana, a) , (nana, na)
2. (anana, ana), (nana, na)
3. (anana, ana)

3 Sequence Assembly

3.1 De Bruijn Graphs

$$F = \{ACGAACG, TGCTGAC, ACTGCT, AACGG, CTGACGA\}$$

1. Set of 4-mers for each of the fragments:

   - ACGA, CGAA, GAAC, AACG
   - TGCT, GCTG, CTGA, TGAC
   - ACTG, CTGC, TGCT
   - AACG, ACGG
   - CTGA, TGAC, GACG, ACGA

   k-mers which occur more than once aren't used usually, but in this way they help when creating the De-Bruijn-Graph.

2. The De-Bruijn-Graph is created from $(k-1)$-mers as nodes and edges which indicate that the nodes' sequences overlap based on the information from the k-mers. The graph should look similar to the following graph. There are 11 nodes and 11 edges.

$$\begin{tikzcd}
\boxed{\text{CGG}} & \boxed{\text{ACG}} \arrow[l] \arrow[r] & \boxed{\text{CGA}} \arrow[r] & \boxed{\text{GAA}} \arrow[r] & \boxed{\text{AAC}} \arrow[lll, bend right] \\
\boxed{\text{GAC}} \arrow[ur, bend right] & \boxed{\text{TGA}} \arrow[l] & \boxed{\text{CTG}} \arrow[l] \arrow[rr, bend left] & \boxed{\text{GCT}} \arrow[l] & \boxed{\text{TGC}} \arrow[l] \\
\boxed{\text{ACT}} \arrow[urr, bend right] & & & &
\end{tikzcd}$$

3. There is one node in the graph (ACT) which only has an outgoing edge. This is the start node of the Euler path. The node with just an incoming edge (CGG) is the end node of the Euler path. Every edge in the path has to be used exactly once, the nodes can be used more than once. The numbers in the graph give the way of the Euler path. The resulting sequence is:

$$ACTGCTGACGAACGG$$

$$\begin{tikzcd}
\boxed{\textbf{13}\text{: CGG}} & \boxed{\textbf{8 and 12}\text{: ACG}} \arrow[l] \arrow[r] & \boxed{\textbf{9}\text{: CGA}} \arrow[r] & \boxed{\textbf{10}\text{: GAA}} \arrow[r] & \boxed{\textbf{11}\text{: AAC}} \arrow[lll, bend right] \\
\boxed{\textbf{7}\text{: GAC}} \arrow[ur, bend right] & \boxed{\textbf{6}\text{: TGA}} \arrow[l] & \boxed{\textbf{2 and 5}\text{: CTG}} \arrow[l] \arrow[rr, bend left] & \boxed{\textbf{4}\text{: GCT}} \arrow[l] & \boxed{\textbf{3}\text{: TGC}} \arrow[l] \\
\boxed{\textbf{1}\text{: ACT}} \arrow[urr, bend right] & & & &
\end{tikzcd}$$