Data Analytics - Assignment-IV

Anirban Biswas (Sr.No. - 14382)

October 15, 2018

Color Blindness Detection

Given four possible configurations of red-green gene, the task is to find out the most probable configuration leading to color-blindness. For this purpose, a reference sequence is provided along with the reads. We have to count the number of times the reads are matching with the range of locations provided for exons of red and green genes. Based on these counts, the job is to identify the most probable configuration.

Approach

The approach taken to solve the problem is same as mentioned in the class. Firstly, create four binary arrays for each of the characters A, G, C and T from the BWT column. The storage requirement for these binary arrays is $4n$ bits. Use these binary arrays to store the ranks in four different integer arrays, with Δ milestones. This requires $\frac{4n}{\Delta}$ bytes. The time complexity to get the rank of a character is $\mathcal{O}(\Delta)$. The Δ parameter balances the time-space trade-off in this problem.

Once the rank arrays are built, for a particular read, we can recursively search for the band (range of indices in BWT column) in which the suffix has exact match. The rank query is used to find this band. The mapping of the indices in the BWT column to the reference sequence is provided and we can make use of it to find the matching indices in the reference sequence.

Finally, we check the how many of these matched indices falls in the red or green exon range for a each and every read. If a read matches with only red or green exon (unambiguous match), we increment the respective count by 1. If it is an ambiguous match, the count is incremented by 0.5 for both red and green exons.

In the scenario when two mismatches are allowed, the read is segmented into three (almost) equal sized parts and each segment is checked for exact matches. It will give a set of indices and for each index we need to check in the original reference sequence the count of mismatches. For this problem up to two mismatches are allowed.

How to find most probable configuration

Let us consider the matched exon count for red and green genes are respectively $R_c = \{r_i | i \in \{1,2,3,4,5,6\}\}$ and $G_c = \{g_i | i \in \{1,2,3,4,5,6\}\}$. The probability of generating these counts given a configuration k is $C_k = \{p_{kj} | j \in \{2,3,4,5\}\}$ can written, assuming binomial distribution, as follows:

$$P(\{R_c, G_c\}|C_k) = \prod_{i=2}^{5} \binom{r_i + g_i}{r_i} {p_{ki}}^{r_i} (1 - p_{ki})^{c_i}$$

The value p_{kj} is the probability that the j^{th} exon is red for the given configuration k. Naturally, $(1 - p_{kj})$ is the probability that the j^{th} exon is green.

Result

The matching counts obtained for red exons are: R_c = **{97, 237, 107.5, 167.5, 303.5, 235}**. For green exons, the values are: G_c = **{97, 156.5, 99.5, 87.5, 217.5, 235}**. As we can see the count for first and last exons for red and green genes are same which is expected.

The absolute probability values along with the log probabilities for four different configurations as shown in the slides are mentioned in the following table. These probabilities are calculated by taking the floor value of the counts obtained previously.

Configuration	Probability	Log(Probability)
I	$5.6 \exp -98$	-223.92
II	0	$-\inf$
III	$3.9 \exp -77$	-175.93
IV	$4.3 \exp -112$	-256.41

Hence, it can be concluded that **Configuration-III** is the most probable one and the person with color-blindness has this red-green gene structure.

Select Query on Binary Array

The problem is to find i^{th} 1 from a binary array of size n. The extra space allowed is $\frac{cn}{\Delta}$ and time complexity in $\mathcal{O}(\Delta)$. The c value needs to be as small as possible.

Approach - I

We can keep count of Δ-rank milestones. This array will be of size $\frac{n}{\Delta}$ and it will contain the indices of the correspondingly ranked 1 in the original binary array.

Given a value of i, we can straight away jump to that index in the original array and start traversing from there until we reach the i^{th} 1.

In this approach, it is not guaranteed that for any given i, the search process will be completed in $\mathcal{O}(\Delta)$ time and extra space required is $\frac{n}{\Delta}$ bytes. In worst case it can take $\mathcal{O}(n)$ time, if array is is extremely sparse in 1 and all these 1's are far apart from each other and non-uniformly distributed. But, in amortized sense, it will take $\mathcal{O}(\Delta)$ time. Also, to improve the non-uniform situation, we can maintain one more array with Δ milestone to get to the next 1. This way the time taken for search can be further reduced, though not guaranteed to be done in $\mathcal{O}(\Delta)$ time and the extra space required in this case increases by $\frac{n}{\Delta}$ bytes.

Approach - I

Another approach to solve this problem can be using the array used for rank query. We can store the count of 1's in Δ milestone, hence space required is still $\frac{n}{\Delta}$ bytes. Given an i, we can do a binary search in this array to find the nearest value to it and can be done in $\mathcal{O}(\log \frac{n}{\Delta})$. This will give the index to start traversing and finally finding the i^{th} 1. The traversal will take Δ time in worst case. Hence the overall time complexity for this approach is $\mathcal{O}(\log \frac{n}{\Delta} + \Delta)$ and extra space required is $\frac{n}{\Delta}$