# Algorithms in Genome Analysis, Spring 2023

Veli Mäkinen

# Week 7

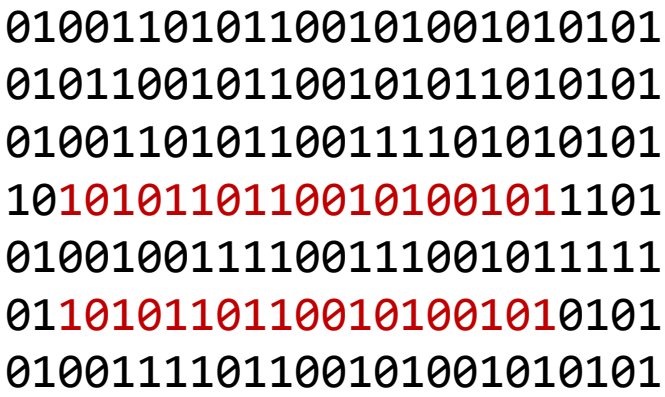
Haplotype matching and O(kn) time approximate pattern matching

# Haplotype matching

From suffix trees to positional BWT

# Haplotype matching

- Recall that from successful haplotype assembly, we have two binary strings representing a diploid genome:
  - Binary string A[1..n] has A[j]=1 iff haplotype A supports variant j
  - Binary string B[1..n] has B[j]=1 iff haplotype B supports variant j
- Assuming haplotype assembly has been done with the same set of variants on d individuals, we obtain a matrix P[1..m,1..n], where m=2d, where rows are haplotypes and columns are variants and P[i,j]=1 iff i-th haplotype supports variant j.
- Matrix P is called *haplotype panel*.
- *Haplotype matching* is the problem of finding (repeating) haplotype blocks P[i,j+k-1] such that P[i,j+k-1]= P[i',j+k-1] for some $i \neq i'$, and k is a given threshold.

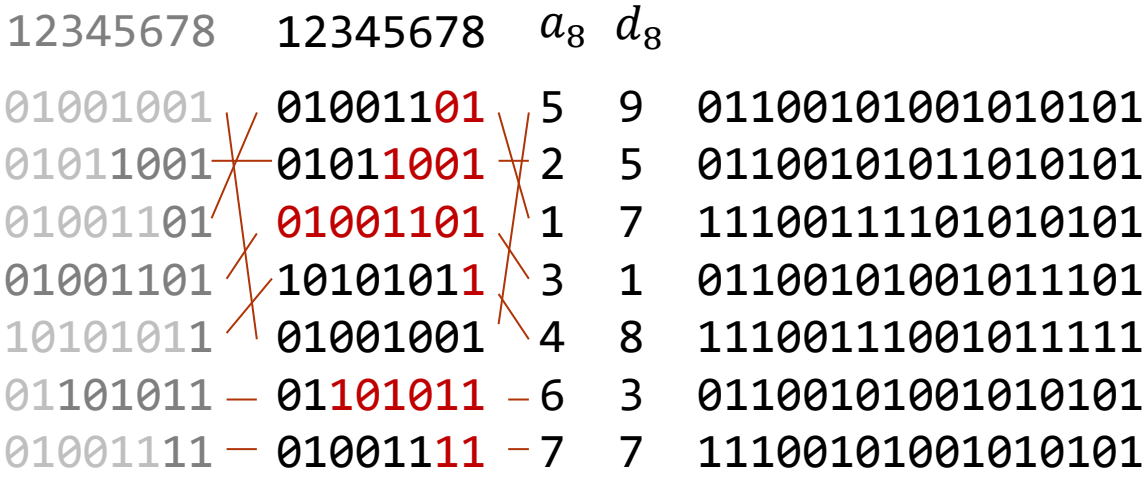

01001101011001010010101
01011001011001010110101
01001101011001111010101
10101011011001010010111101
01001001111001110010111111
01101011011001010010100101
01001111011001010010101

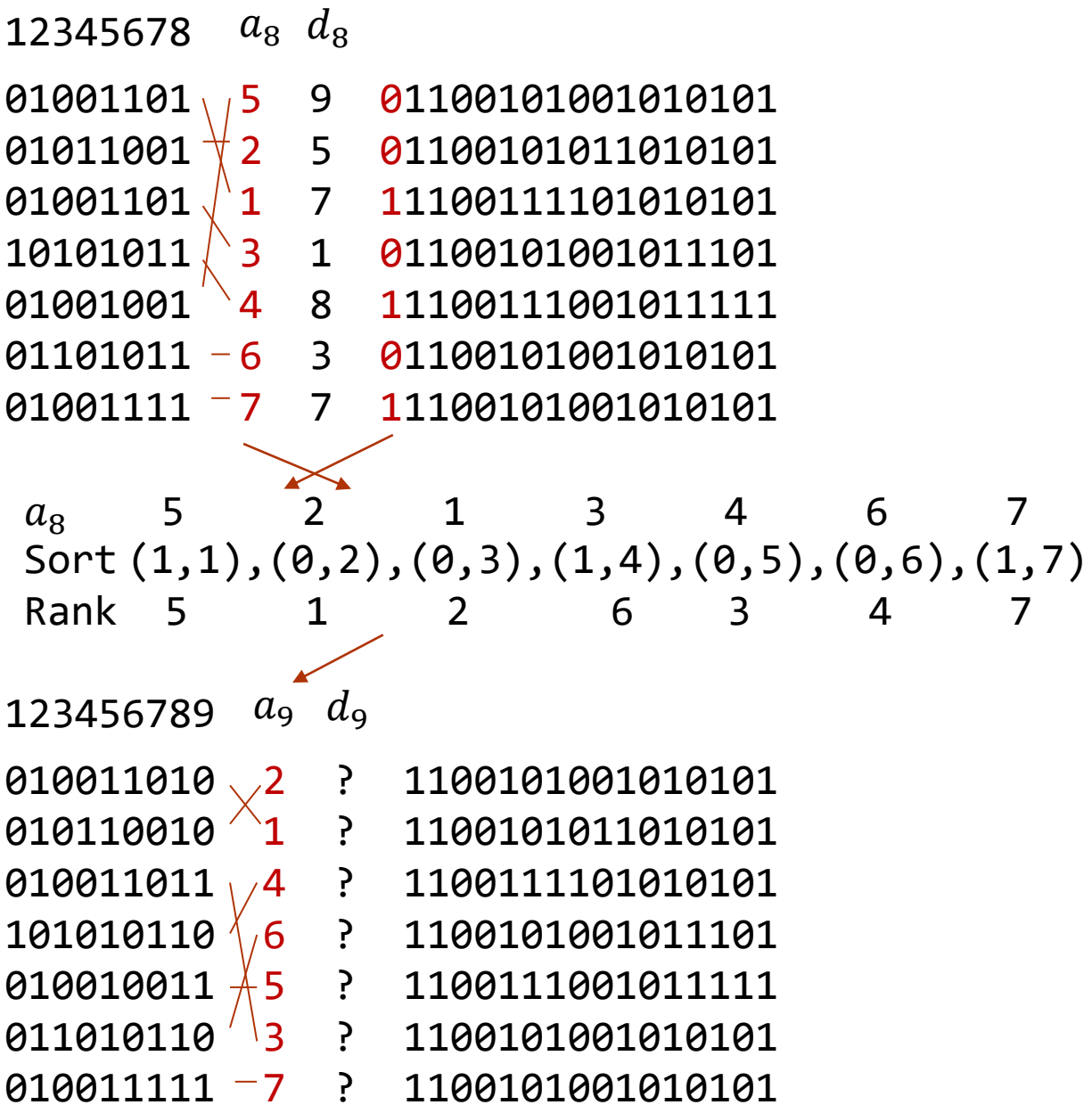
# Haplotype matching with suffix trees

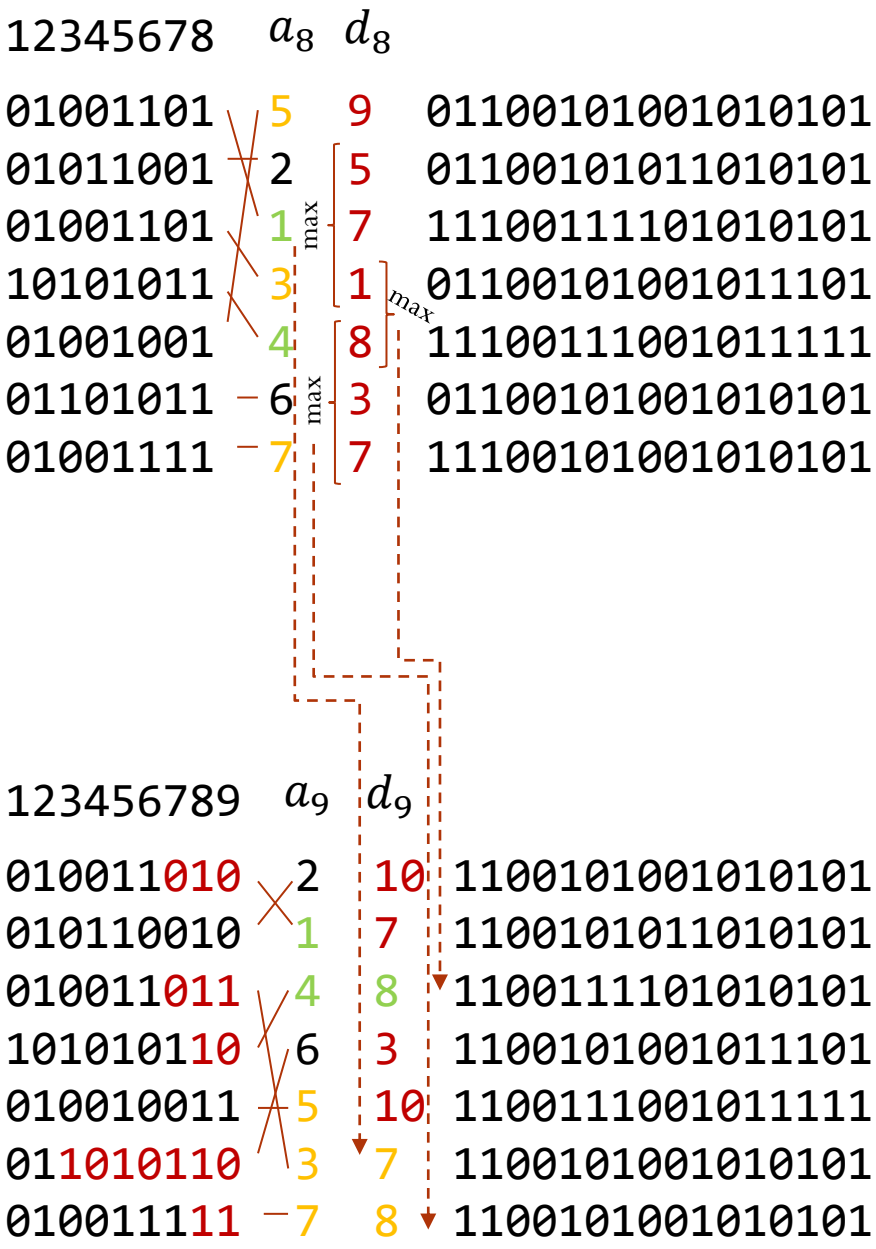
- Consider the positional sequence C:
  - C=(1,P[1,1]),(2,P[1,2]),…,(n,P[1,n]),
    (1,P[2,1]), (2,P[2,2]),…,(n,P[2,n]),
    …
    (1,P[m,1]),(2,P[m,2]),…,(n,P[m,n])
- Build suffix tree ST(C) of C (considering pairs as symbols)
- Consider (an implicit) node v in ST(C) with string depth k
- Root to v path spells a haplotype block P[i,j..j+k-1] for some i and j and the subtree of v contains information on all rows i' such P[i,j+k-1]=P[i',j+k-1]
- Considering all such v reveals all distinct haplotype blocks
- One could add maximality constraints to haplotype blocks and find them using the maximal repeat finding we covered earlier

# Isn't radix sorting enough

- Yes, we can sort prefixes of the matrix from left-to-right to produce two arrays $a_j[1..m]$ and $d_j[1..m]$ forming the *positional BWT (pBWT)* such that
  - $P[a_j[i], 1..j]$ is the co-lexicographically (compared right-to-left) i-th smallest string among $\{P[i', 1..j]\}$, where the row number is used for breaking ties
  - $P[a_j[i], d_j[i]..j] = P[a_j[i-1], d_j[i]..j]$ and $P[a_j[i], d_j[i]-1] \neq P[a_j[i-1], d_j[i]-1]$
  - $d_j$[i]=j+1 if the above does not hold for any value
- Haplotype matching can be solved, e.g., by checking all values $j - d_j[i] + 1 \geq k$, where $j - d_j[i-1] + 1 < k$, as this reveals all distinct haplotype blocks $P[a_j[i], j-k+1..j]$


12345678 12345678 $a_8$ $d_8$
01001001 01001101 5 9 0110010100101010 1
01011001 01011001 2 5 0110010101101010 1
01001101 01001101 1 7 1110011110101010 1
01001101 10101011 3 1 0110010100101110 1
10101011 01001001 4 8 1110011100101111 1
01101011 01101011 6 3 0110010100101010 1
01001111 01001111 7 7 1110010100101010 1


12345678 $a_8$ $d_8$
01001101 5 9 0110010100101010 1
01011001 2 5 0110010101101010 1
01001101 1 7 1110011110101010 1
10101011 3 1 0110010100101110 1
01001001 4 8 1110011100101111 1
01101011 6 3 0110010100101010 1
01001111 7 7 1110010100101010 1
$a_8$ 5 2 1 3 4 6 7
Sort (1,1),(0,2),(0,3),(1,4),(0,5),(0,6),(1,7)
Rank 5 1 2 6 3 4 7
123456789 $a_9$ $d_9$
010011010 2 ? 110010100101010 1
010110010 1 ? 110010101101010 1
010011011 4 ? 110011110101010 1
101010110 6 ? 110010100101110 1
010010011 5 ? 110011100101111 1
011010110 3 ? 110010100101010 1
010011111 7 ? 110010100101010 1


12345678 $a_8$ $d_8$
01001101 5 9 0110010100101010 1
01011001 2 5 0110010101101010 1
01001101 1 7 1110011110101010 1
10101011 3 1 0110010100101110 1
01001001 4 8 1110011100101111 1
01101011 6 3 0110010100101010 1
01001111 7 7 1110010100101010 1
max
max
max
123456789 $a_9$ $d_9$
010011010 2 10 1100101001010101
010110010 1 7 1100101011010101
010011011 4 8 1100111101010101
101010110 6 3 1100101001011101
010010011 5 10 1100111001011111
011010110 3 7 1100101001010101
010011111 7 8 1100101001010101

# O(kn) time approximate pattern matching

Suffix tree with LCA-queries to speed-up alignment computation

# RMQ / LCA

- Computing the divergence array $d_j[1..m]$ for haplotype matching required fast computation of maximum value in a range (range max).
- For this, one can use a data structure for range minimum queries (RMQs):
  - Theorem: One can preprocess an array A[1..n] in O(n) time such that for any given query range [i..j] one can retrieve $RMQ(i,j) = \text{argmin}_{k\in[i..j]} A[k]$ in constant time.
    - Proof: See course book.
- We will need the same data structure for speeding-up approximate pattern matching through LCA queries:
  - Theorem: A rooted tree with n nodes can be preprocessed in O(n) time so that the lowest common ancestor LCA(u,v) of any two nodes v and w can be computed in constant time.
  - Proof: See course book. Idea: Print node depths into an array along a full walk of the tree. LCA(u,v) can be mapped into RMQ(i,j).

# RECAP

- Consider unit cost edit distance d() and the related approximate string matching problem of finding out if pattern P[1..m] occurs with k errors in a text T[1..n], i.e., if there is substring T[j'..j] such that $d(P, T[j'..j]) \leq k$.
- Let d[0..m,0..n] be the DP matrix s.t. $d[i,j] = \min_{j'} d(P[1..i], T[j'..j])$.
- Recall that we can initialize zeroth row d[0,0..n] to zeros, and use the recurrence
  - $d[i,j] = \min(d[i-1,j-1] + (P[i] = T[j]\,?\,0:1),$
    $d[i-1,j] + 1,$
    $d[i,j-1] + 1)$
- Then we can check the last row d[m,0..n] for values not greater than k to identify ending positions of approximate matches.

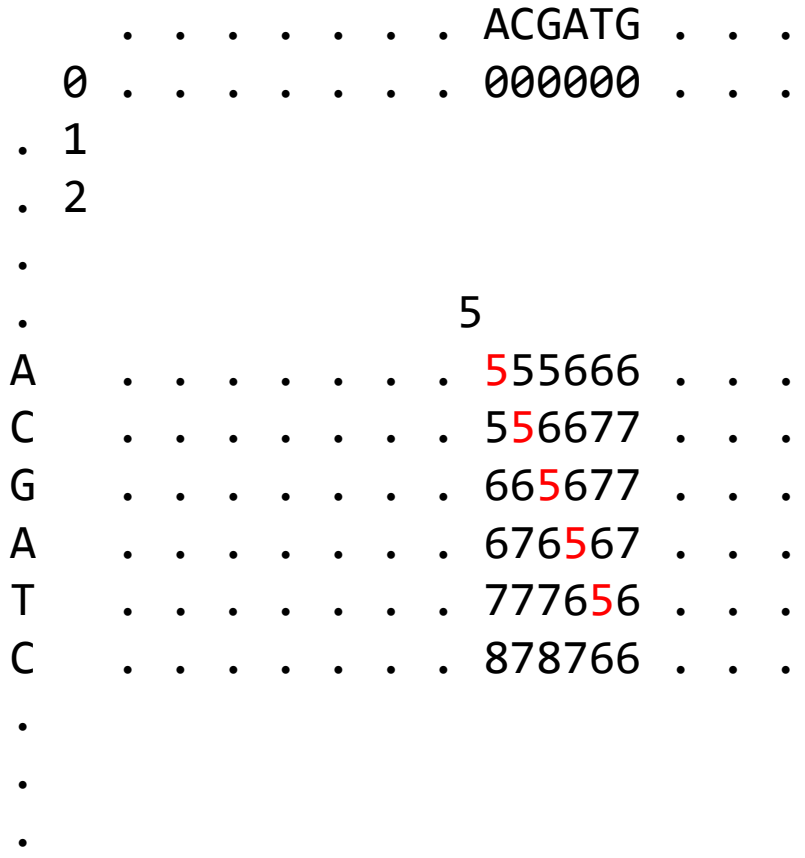
# LCE

- Let the *longest common extension* LCE(i,j)=k iff P[i..i+k-1]=T[j..j+k-1] and P[i+k]≠T[j+k] (or i+k>m or j+k>n)
- Consider suffix tree of the concatenation P$T#. Let v and w be the leafs corresponding to suffixes P[i..] and T[j..] in the concatenation, respectively
- LCE(i,j) is the length of the longest common prefix of P[i..] and T[j..], hence, it equals the length of the string spelled from the root to LCA(u,v), and can thus be computed in constant time

# Some key properties

- *Lemma 1:* d[i,j]-d[i-1,j-1] is either 0 or 1
- *Lemma 2:* d[i,j]-d[i,j-1] is either -1, 0 or 1
- *Lemma 3:* d[i,j]-d[i-1,j] is either -1, 0 or 1
- *Lemma 4:* d[i+k',j+k']=d[i-1,j-1], where $0 \leq k' < LCE(i,j)$
- *Observation*: d[i,j]=d[i-1,j-1] can hold even when $P[i] \neq T[j]$, e.g., when d[i,j]=d[i,j-1]+1 and d[i,j-1]=d[i-1,j-1]-1

# Diagonal propagation 1/4



# Diagonal propagation 2/4

- Diagonal g of matrix d[1..m,1..n] is formed by cells d[i,j] such that j-i=g
- Values along a diagonal can be partitioned into k+2 intervals, those with values 0, 1, …, k, and those with values greater than k
- Assume we know intervals corresponding to values 0,1,..e-1 for all diagonals, maybe we can compute the interval corresponding to values e for all diagonals

# Diagonal propagation 3/4

- Let $L_{e,g}$ denote the last row i at diagonal g with value e=d[i,g+i]
- One can observe that $L_{e,g}$ depends on interval boundaries $L_{e-1,g-1}$, $L_{e-1,g}$, and $L_{e-1,g+1}$, and LCEs that propagate these interval boundaries along the diagonal
- *Lemma*: $L_{e,g} = r + LCE(r, g+r) - 1, where\ r = \max(i, i', i''), \mathrm{i} = L_{e-1,g-1}, \mathrm{i}' = L_{e-1,g} + 1, \text{and } \mathrm{i}'' = L_{e-1,g+1} + 1$ (see next slide)
- Each cell of $L_{0..k,-k..n}$ can be computed in constant time, hence the running time is O(kn)

# Diagonal propagation 4/4