The Minichiello-Durbin Algorithm

cs1820

Background

The Minichiello-Durbin algorithm infers a genealogy from population genotype data. Using genealogies generated by the MD algorithm, we can fine map disease loci and interpret association signals.

Input: A population of haplotypes, each labeled with a given trait (eg. association with a disease).

Output: An Ancestral Recombination Graph (ARG) which explains the inheritance patterns of the trait across the population.

Since the "true" ARG is generally unknown, the Minichiello-Durbin algorithm employs heuristics to infer many possible ARGs, and conducts statistical tests to determine the "most likely" patterns of inheritance which explain the occurrence of the trait in question. Each ARG is constructed "backward in time", with each step corresponding to one of the events defined below (coalescence, mutation or recombination).

Notation

The algorithm works backward in time from the contemporary population of chromosome sequences to a single ancestor sequence. Each step back in time, accomplished with a recombination, mutation, or coalescence, defines an ancestral population of sequences.

We denote the set of sequences at time T as S_T. The sequences are strings of length m from the alphabet $\{0,1,\cdot\}$, where m is the number of markers (SNPs), 0 is one of the SNP alleles, 1 is another allele, and "$\cdot$" denotes and undefined allele (an allele that was not inherited by any sequence in the contemporary population).

Notation

- The allelic state of SNP i on sequence C is denoted $C[i]$, where i is the marker position and $1 \leq i \leq m$
- We define $C_1[i] \sim C_2[i]$ if and only if $C_1[i] = C_2[i]$, or $C_1[i] = \cdot$, or $C_2[i] = \cdot$
- We define the complement operator, $\neg$, such that if $C[i] = 0$, then $\neg C[i] = 1$ and vice versa. $\cdot$ is its own complement

Notation

- There is a shared tract between sequences C_1 and C_2, over the contiguous set of markers $a, ..., b$ if

  1. $C_1[i] \sim C_2[i]$ for all $a \leq i \leq b$
  2. If there is at least one i for which $C_1[i] = C_2[i] \neq \cdot$
  3. If $a > 1$, then $C_1[a-1] \neq C_2[a-1]$ and neither is $\cdot$
  4. If $b < m$, then $C_1[b+1] \neq C_2[b+1]$ and neither is $\cdot$

Rules

The algorithm proceeds by finding which coalescences, mutations, and recombinations can be performed to create an ancestral sequence population S_{T+1} from the current population S_T. We begin with the set of contemporary sequences, S_1.

Coalescence:

- *Rule* : If there exists two sequences, C_1 and C_2, in S_T such taht for all i, $C_1[i] \sim C_2[i]$, then C_1 and C_2 can be coalesced into an ancestor.
- $Transition$: $S_{T+1} = (S_T \backslash \{C_1, C_2\}) \cup \{C'\}$ where $C'[i] = C_1[i]$ when $C_1[i] \neq \cdot$, and $C'[i] = C_2[i]$ otherwise.

Rules

Mutation:

- *Rule* : If there exists a sequence C_1 in S_T and a marker, i, where for all C_2 in $S_T \backslash \{C_1\}$ we have $C_2[i] = \neg C_1[i]$ or $\cdot$, then we can remove the derived allele ($C_1[i]$) from the population.
- $Transition$: $S_{T+1} = (S_T \backslash \{C_1\}) \cup \{C'\}$, where $C'[i] = \neg C_1[i]$ and $C'[j] = C_1[j]$ for all $j \neq i$

Rules

Recombination:

- *Rule* : When it is not possible to perform a mutation or coalescence transition, we must perform a recombination (or pair of recombinations) instead. We denote a recombination breakpoint as (α, β), meaning that it occurs between indices α and β. Picking a shared tract $\{C_1, C_2\}[a, b]$ from those available in S_T, we aim to put recombinations on the lineages of C_1 and C_2 such that one recombination parent of C_1 and one recombination parent of C_2 satisfy the rule for coalescence. To do this, we must put a breakpoint at $(a-1, a)$ if $a \neq 1$ and put a breakpoint at $(b, b+1)$ if $b \neq m$.

- *Transition* : From the tract $\{C_1, C_2\}[a, b]$, pick (1) a valid breakpoint (α, β) where either $(\alpha, \beta) = (a-1, a)$ or $(\alpha, \beta) = (b, b+1)$ and (2) a recombinant sequence C_R, where either $C_R = C_1$ or $C_R = C_2$. Then $S_{T+1} = (S_T \backslash \{C_R\})\{C_1', C_2'\}$, where $C_1'[i] = C_R[i]$ for all $i \leq \alpha$ and $C_1'[i] = \cdot$ otherwise, and $C_2'[i] = C_R[i]$ for all $i \geq \beta$ and $C_2'[i] = \cdot$ otherwise. If both $(a-1, a)$ and $(b, b+1)$ are valid breakpoints (ie, $a \neq 1$ and $b \neq m$), we must put the second recombination (taking us to state $S_{\{}T+2\}$) on an appropriate ancestor of C_1 or C_2.

Heuristics

These rules define the constraints on the algorithm that must be enforced if it is to produce legal ARGs. However, at any stage of the algorithm, there may be several different coalescences, mutations, or recombinations that satisfy the rules. We choose between these, using the heuristics below, and the stochastic elements mean that different ARGs aregenerated each time the algorithmis run.

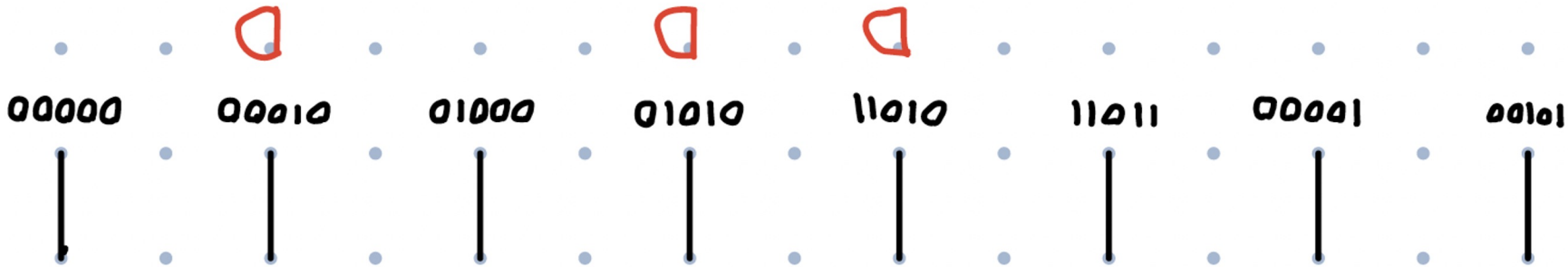
Heuristics

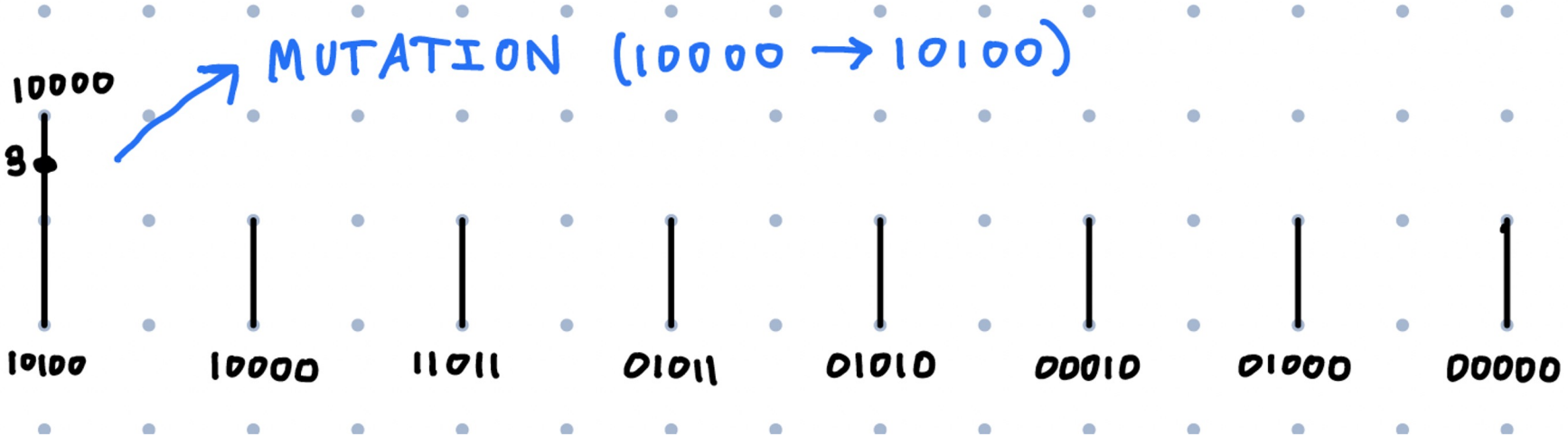
- Perform Recombination if and only if no mutations or coalescences are possible.
- If it is possible to add multiple mutations and/or multiple coalescences at the same time, the order in which these are done is chosen arbitrarily.
- Coalesce sequences only if the have an overlapping region of defined material–that is, the two sequences must match for at least one position that is not ".".
- Recombinations are added at the ends of longer shared tracts first. During the recombination step, we choose a shared tract $\{C_1, C_2\}[a, b]$ such that the base-pair distance between markers a and b is maximized, reflecting that longer shared tracts tend to arise from more-recent recombination events.
- The first coalescence after a recombination is based on the shared segment that was used to decide the location of that recombination.

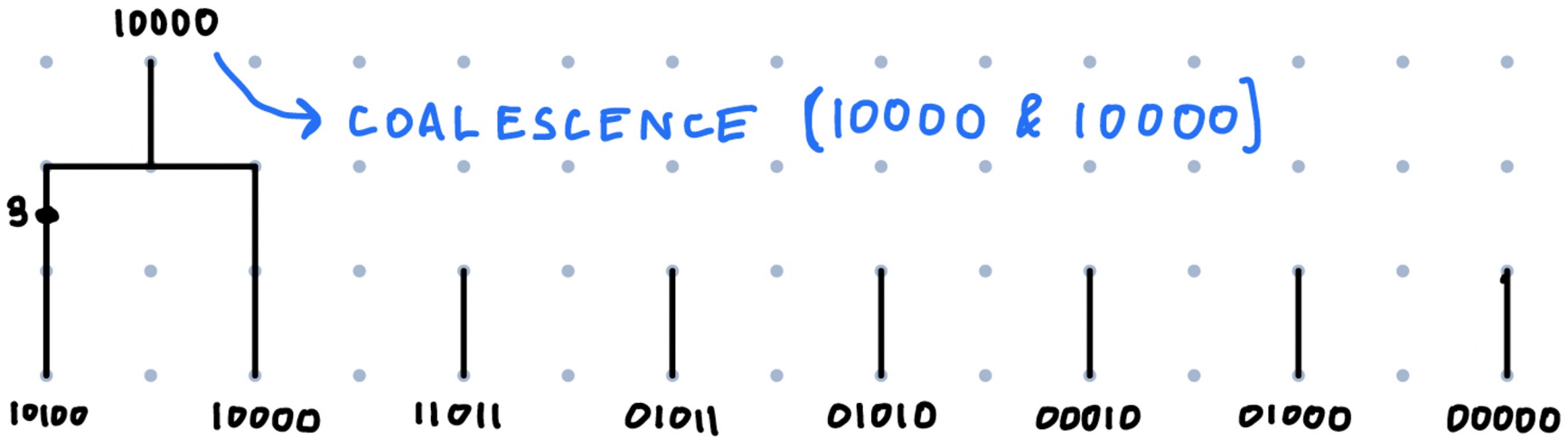
Example

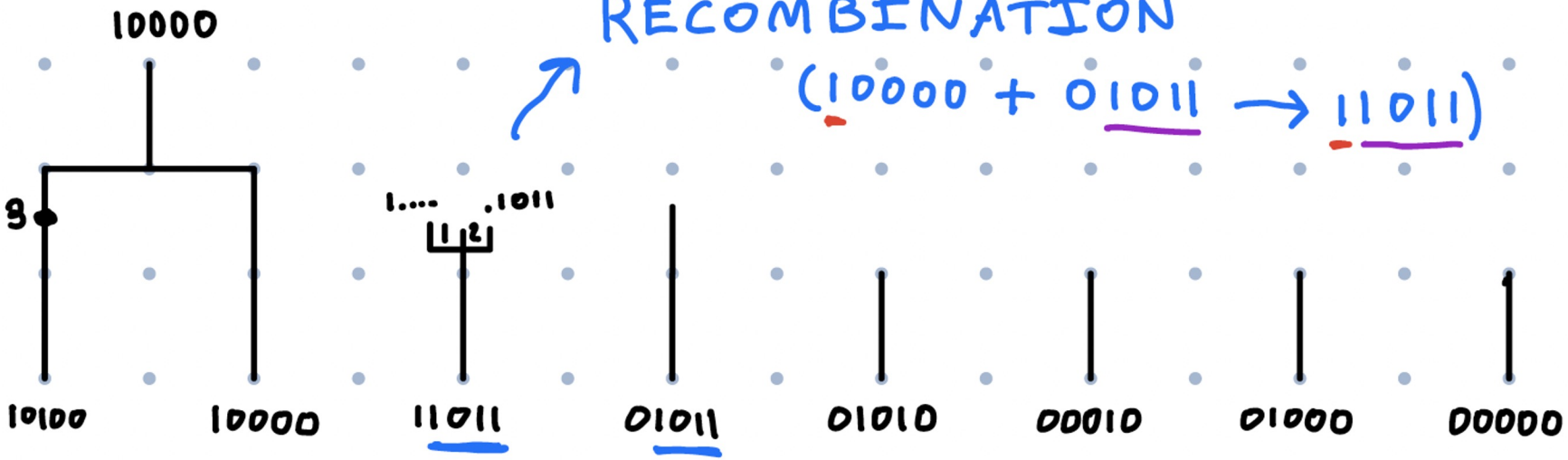
We now consider an example of applying the Minichiello-Durbin algorithm to the following set of haplotype sequences. The presence of a "D" next to a haplotype indicates that this haplotype is present in individuals showing symptoms of some disease.

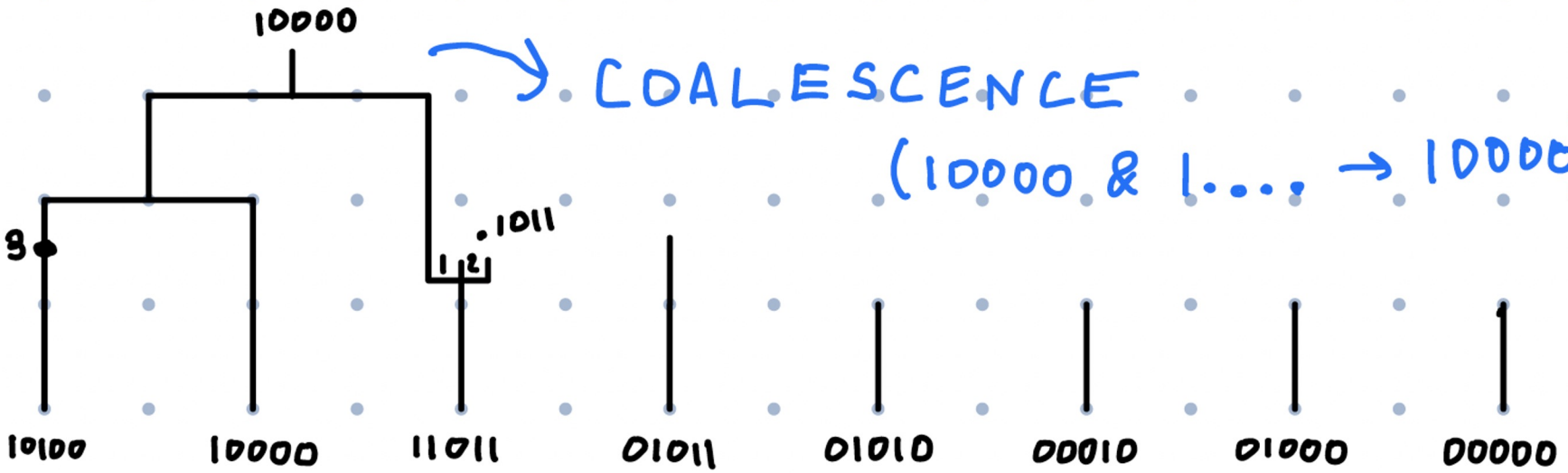
- 10100
- 10000
- 11011
- 01011 D
- 01010 D
- 00010
- 01000 D
- 00000

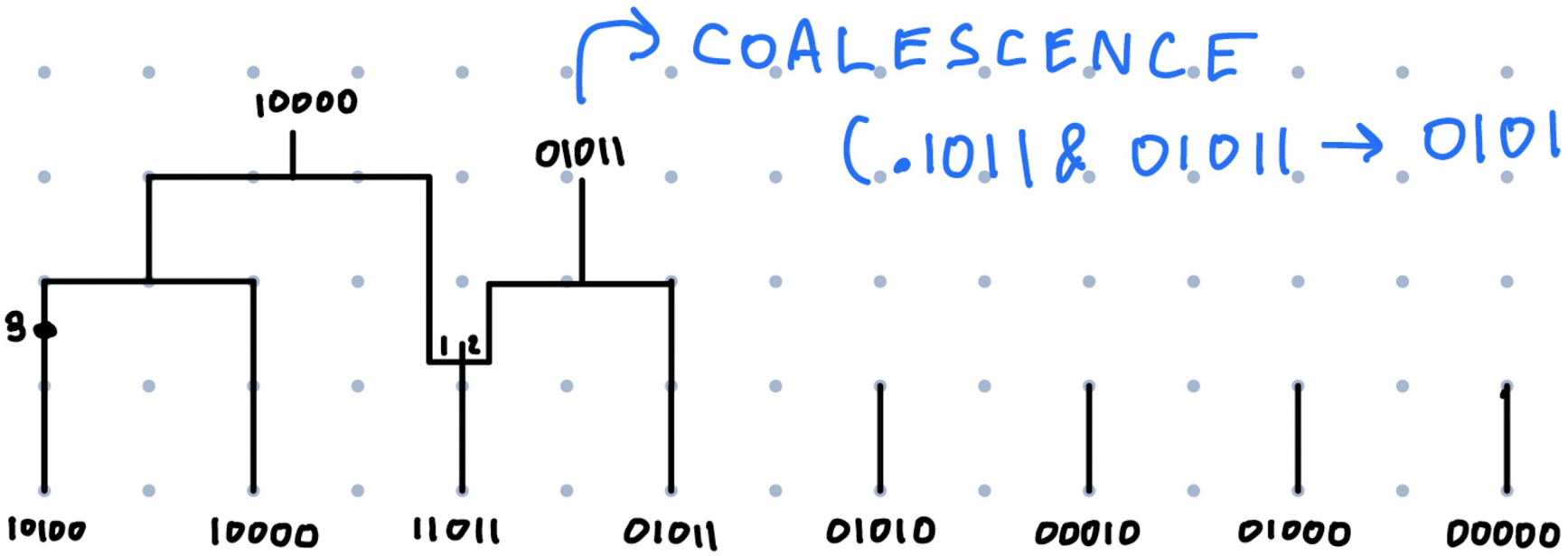

00000
00010
01000
01010
11010
11011
00001
00101

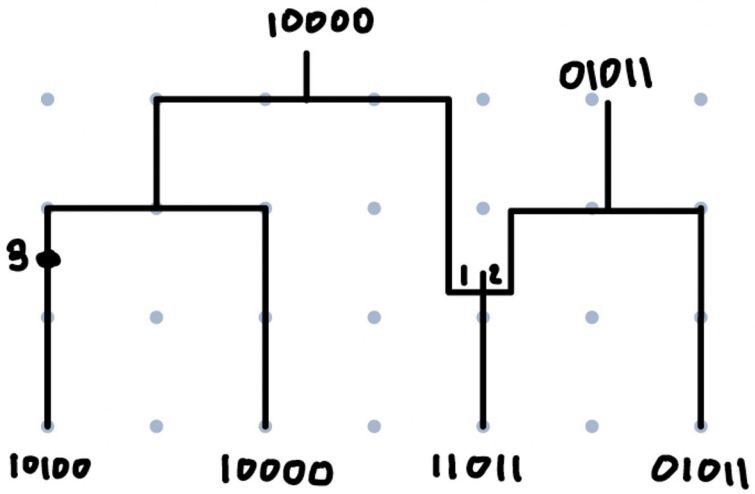

MUTATION (10000 → 10100)
10000
9
10100
10000
11011
01011
01010
00010
01000
00000


10000
COALESCENCE (10000 & 10000)
9
10100
10000
11011
01011
01010
00010
01000
00000


10000
9
10100
10000
1.... .1011
11011
01011
01010
00010
01000
00000
RECOMBINATION
(10000 + 01011 → 11011)

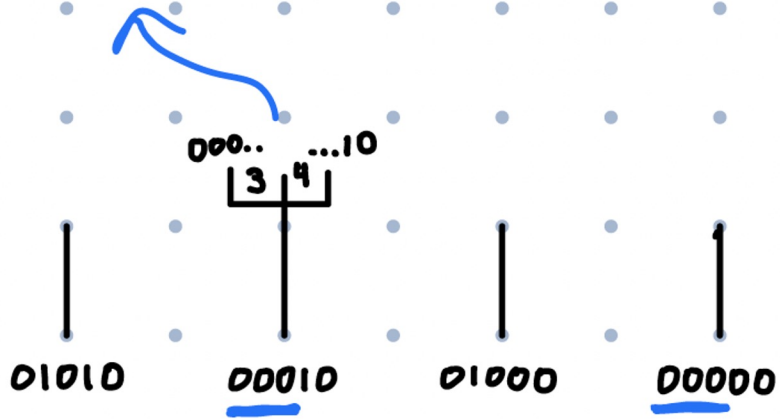

10000
9
1011
10100
10000
11011
01011
01010
00010
01000
00000
COALESCENCE
(10000 & 1... → 10000)

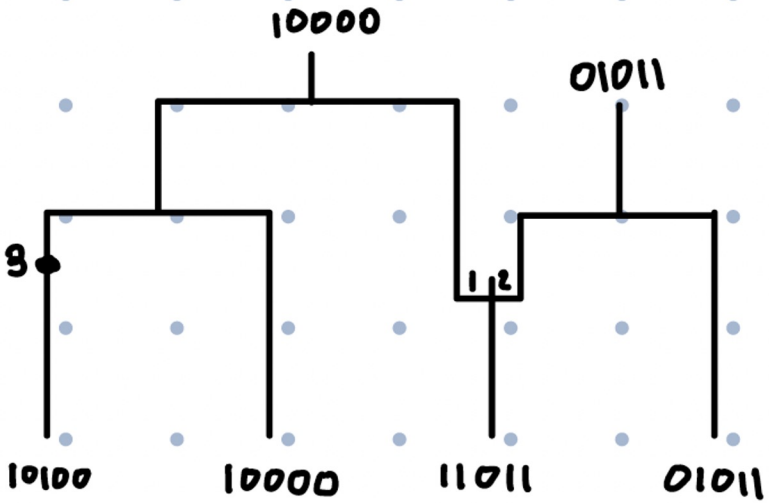

COALESCENCE
(.1011 & 01011 → 01011)
10000
01011
9
12
10100
10000
11011
01011
01010
00010
01000
00000



(01010 + 00000 → 00010)

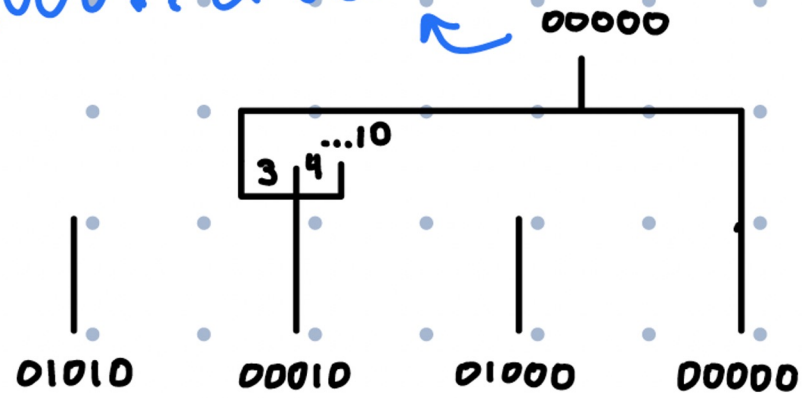
RECOMBINATION

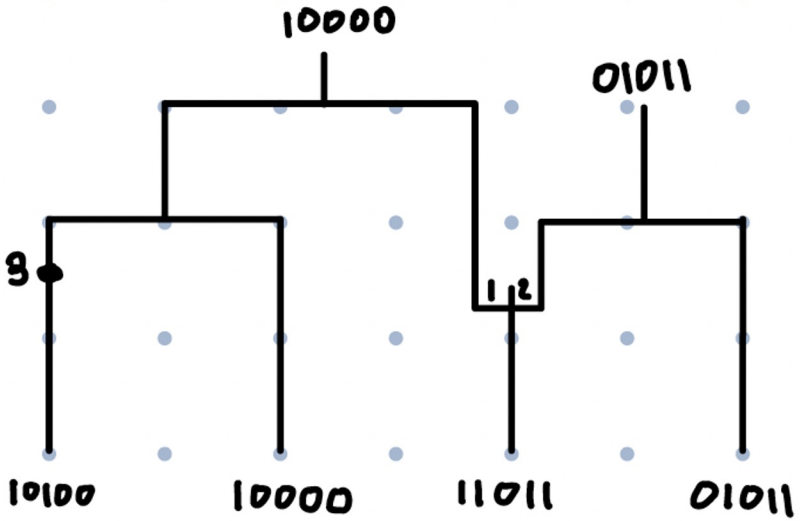




COALESCENCE

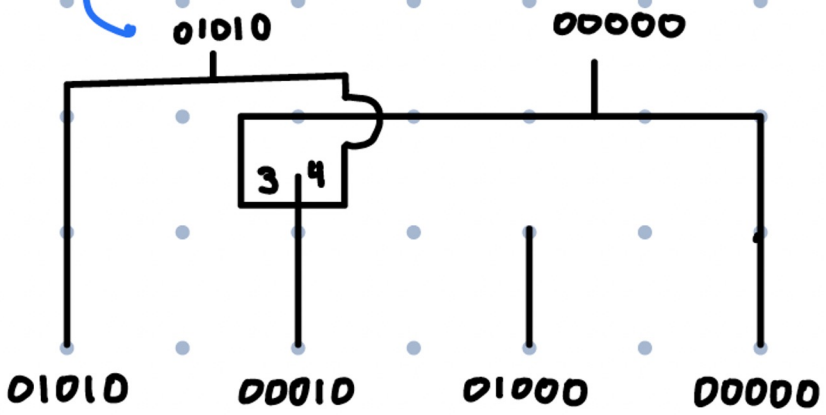
(000... & 00000 → 00000)

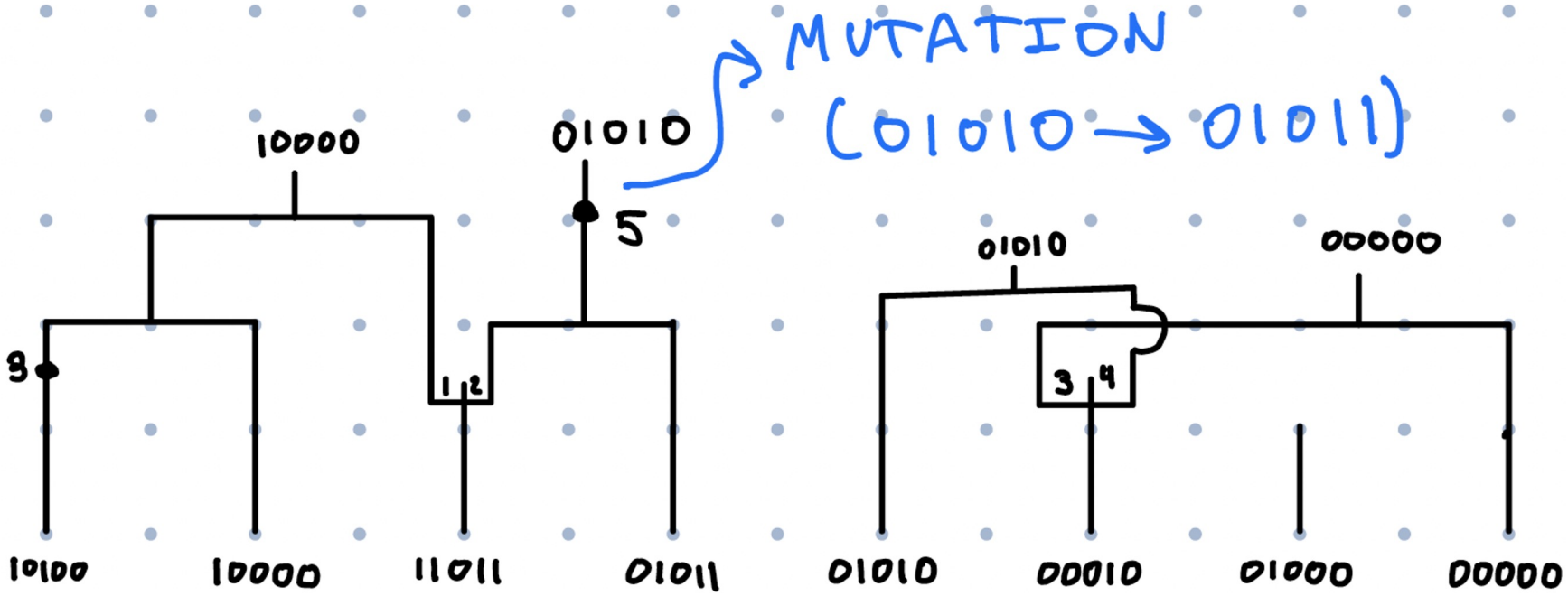


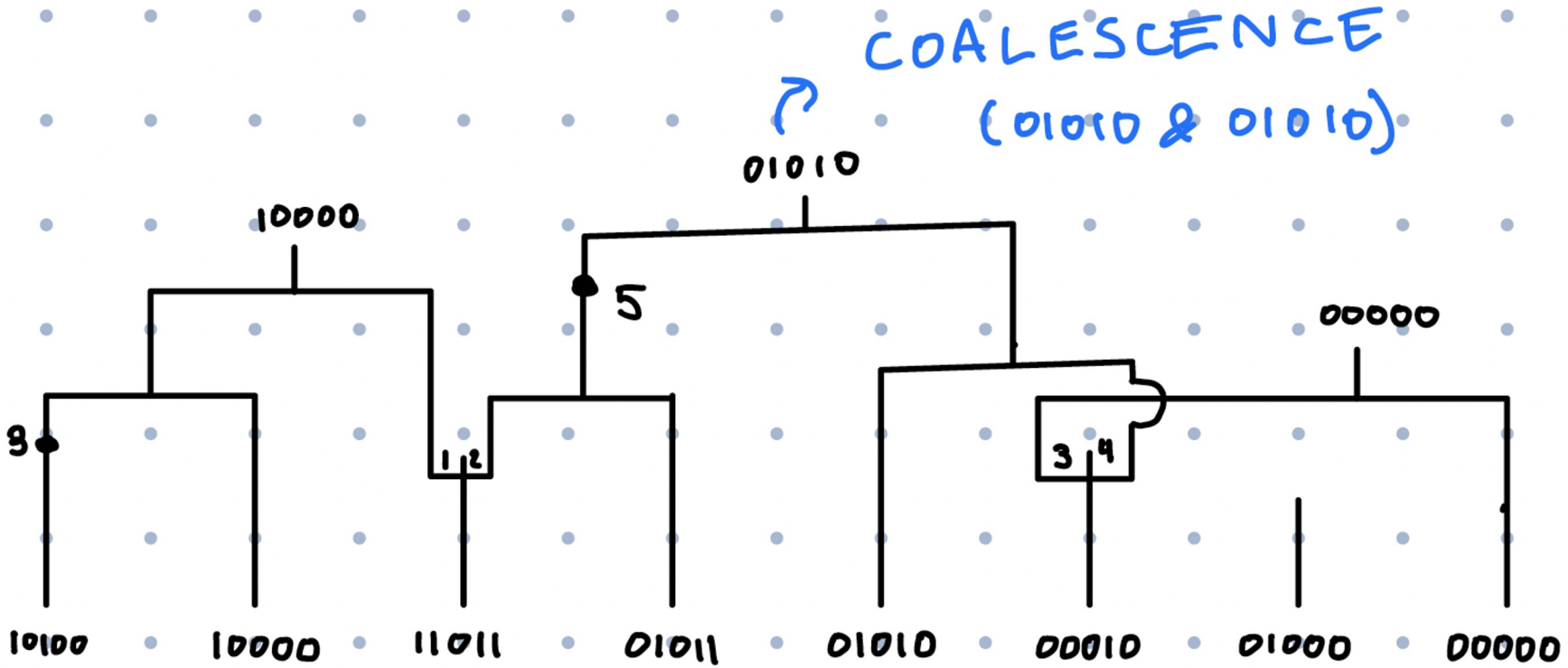


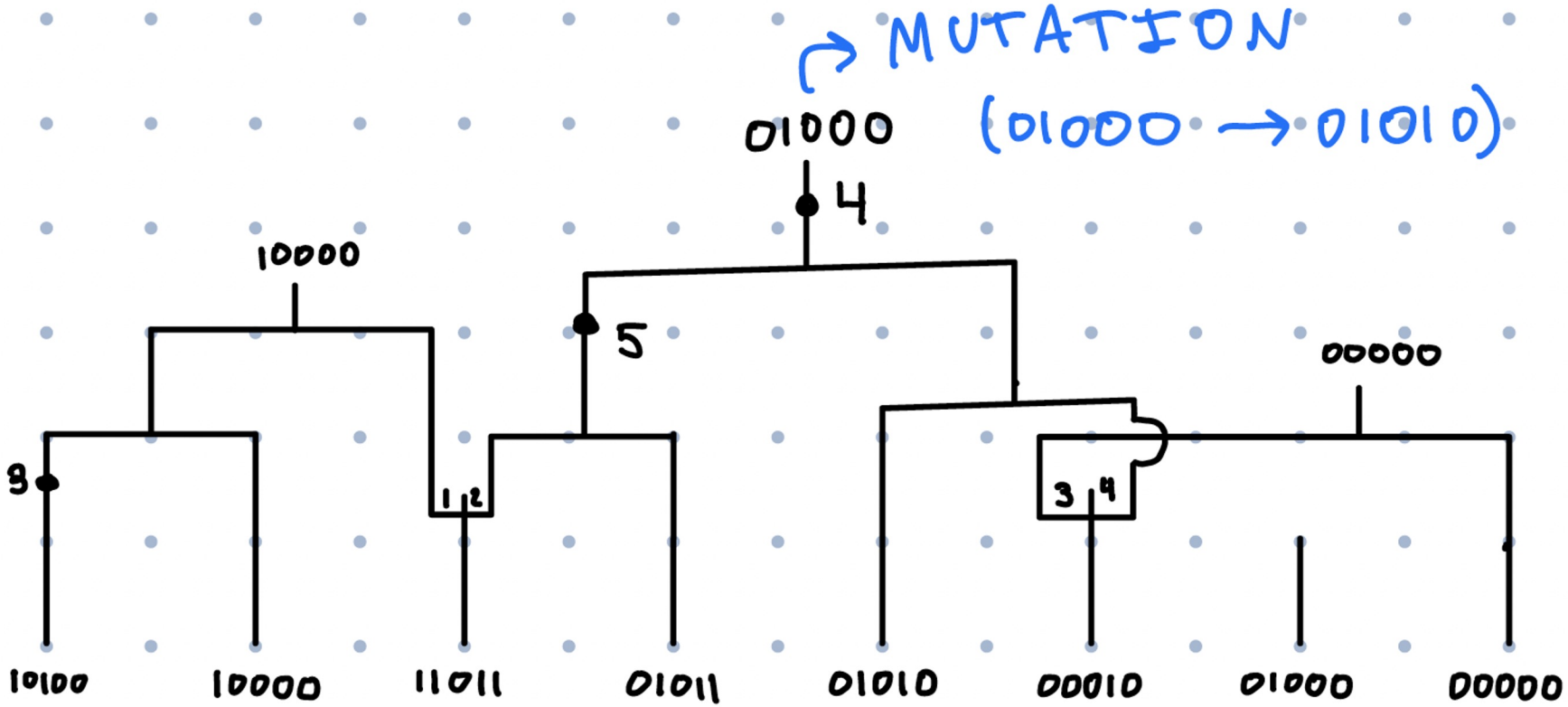
COALESCENCE

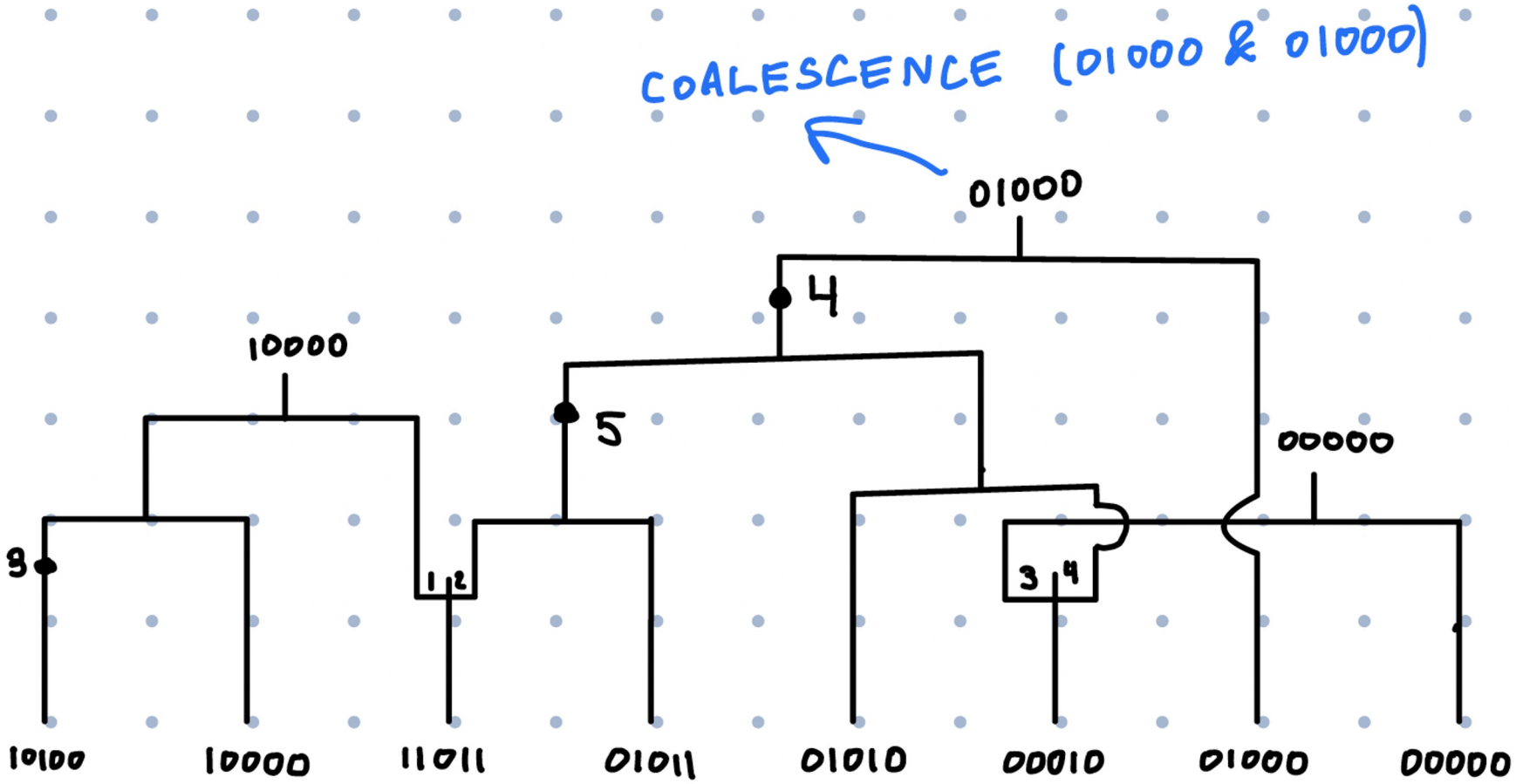
(01010 & ...10 → 01010)

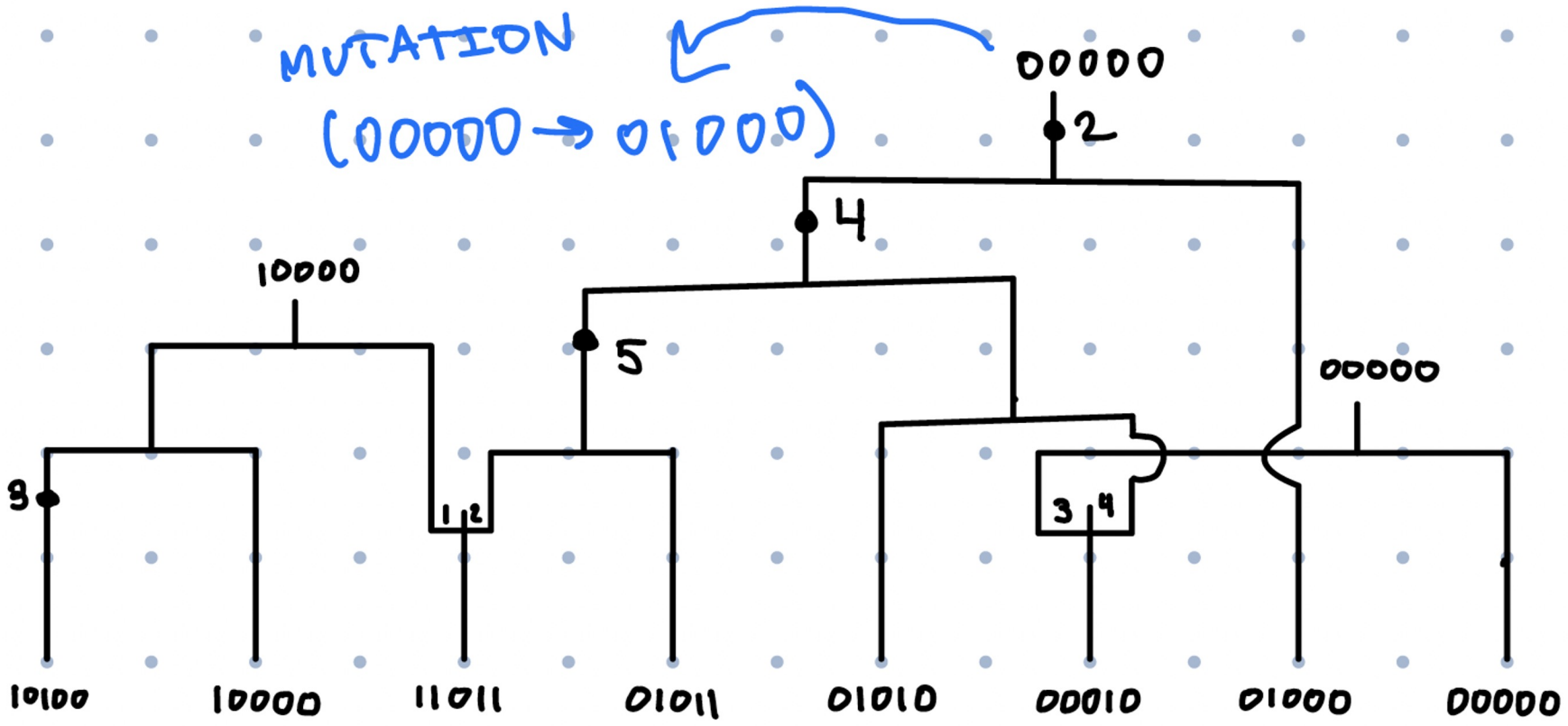


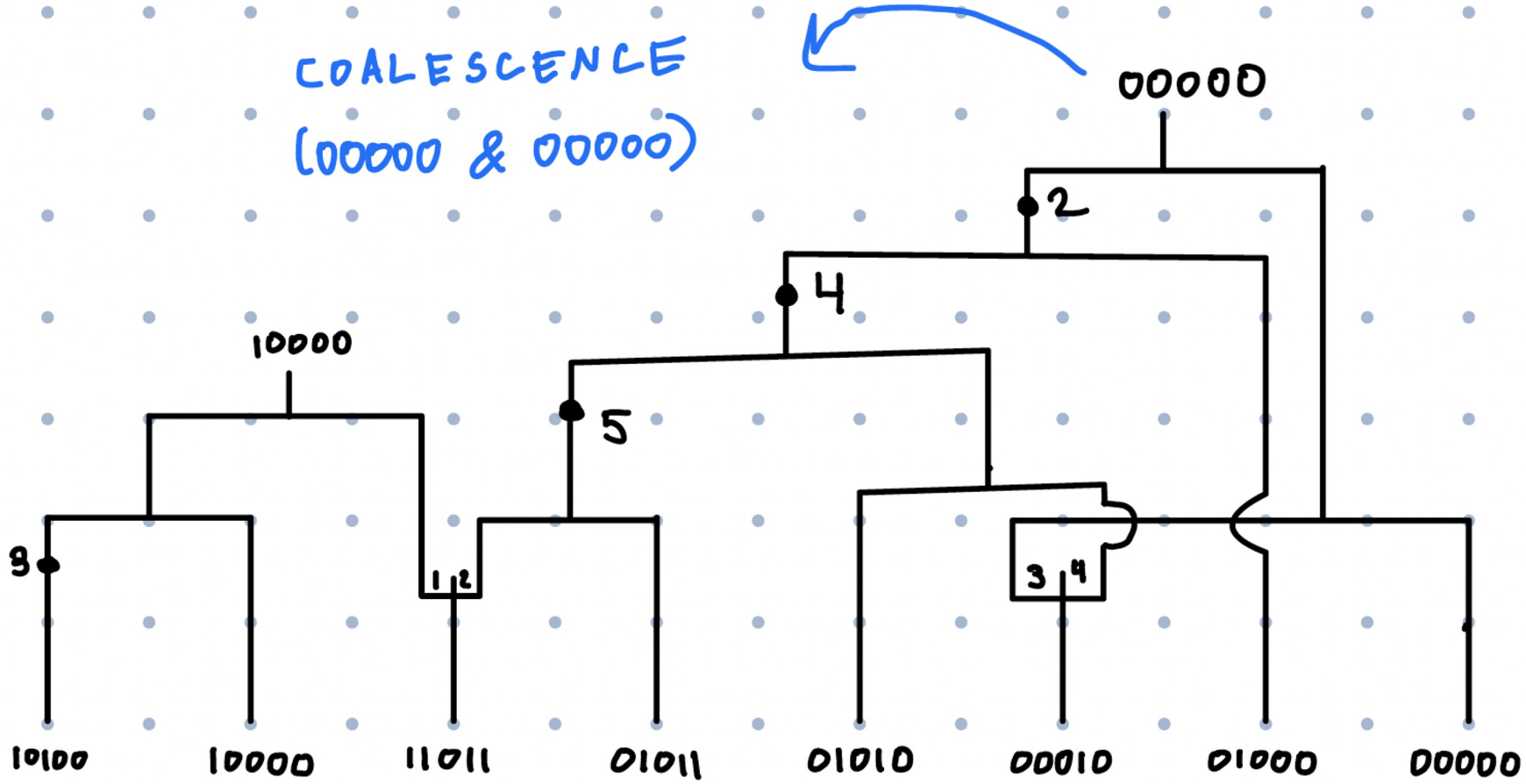

MUTATION
(01010 → 01011)
10000
01010
5
9
12
10100
10000
11011
01011
01010
00000
3
4
01010
00010
01000
00000

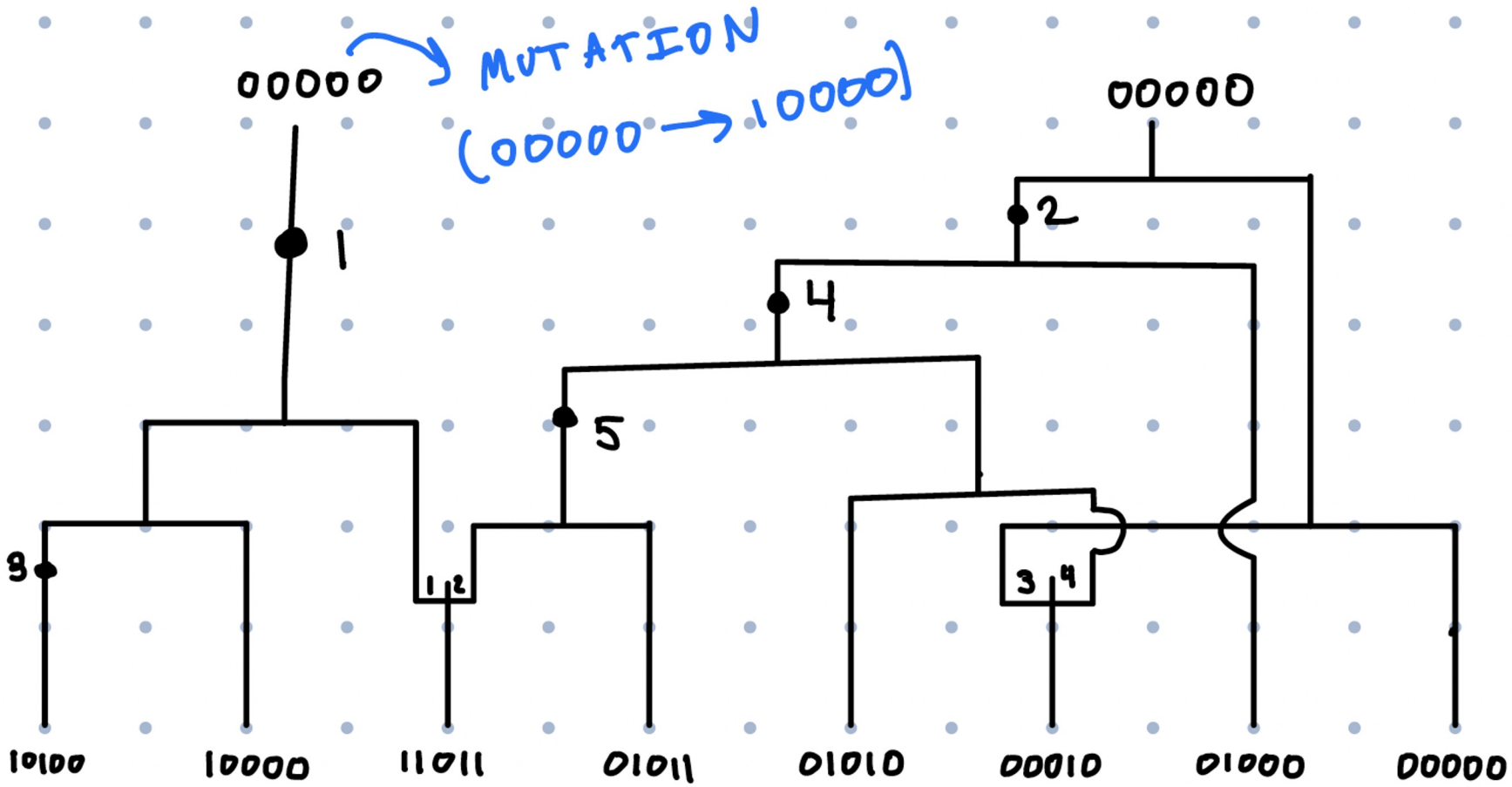

COALESCENCE
(01010 & 01010)
01010
10000
5
00000
9
1 2
3 4
10100
10000
11011
01011
01010
00010
01000
00000

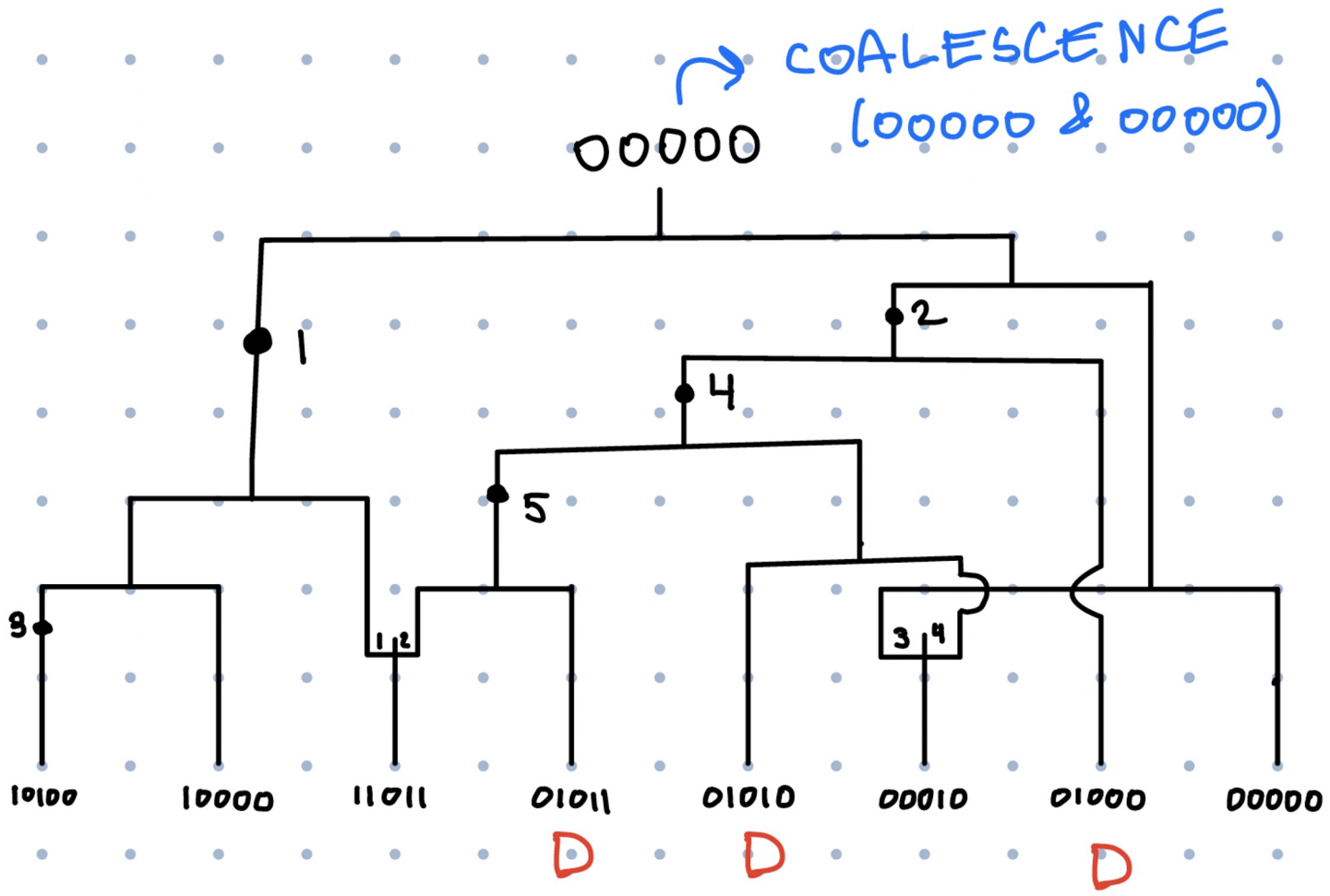

MUTATION
(01000 → 01010)
01000
4
10000
5
00000
9
12
3
4
10100
10000
11011
01011
01010
00010
01000
00000


COALESCENCE (01000 & 01000)
01000
4
10000
5
00000
9
1 2
3 4
10100
10000
11011
01011
01010
00010
01000
00000


MUTATION
(00000 → 01000)
00000
2
4
10000
5
00000
9
12
3
4
10100
10000
11011
01011
01010
00010
01000
00000


COALESCENCE
(00000 & 00000)
00000
2
4
10000
5
9
1 2
3 4
10100
10000
11011
01011
01010
00010
01000
00000


MUTATION
(00000 → 10000)
00000
00000
1
2
4
5
9
1 2
3 4
10100
10000
11011
01011
01010
00010
01000
00000


COALESCENCE
(00000 & 00000)
00000
1
2
4
5
9
12
3
4
10100
10000
11011
01011
01010
00010
01000
00000
D
D
D