Alignment of whole genomes using MUMmer

Presentation in Algorithms in Bioinformatics (TÖ111F autumn 2014)

Hannes Pétur Eggertsson

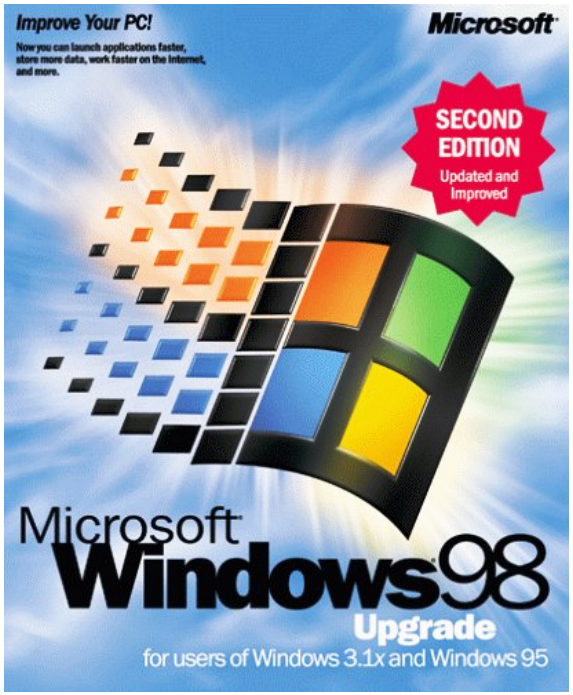
November 18, 2014

Motivation

First off. Let's travel back in time to 1999... (here are some things that will remind you of that wonderful time)

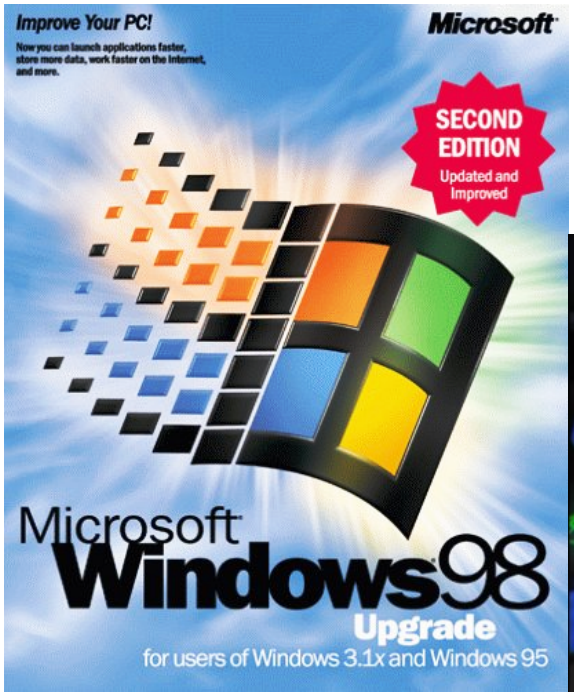
Motivation

First off. Let's travel back in time to 1999... (here are some things that will remind you of that wonderful time)



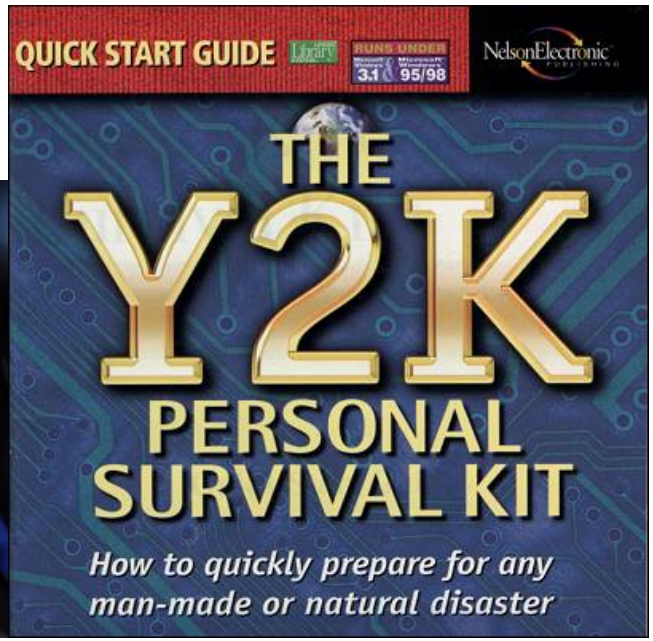
Motivation

First off. Let's travel back in time to 1999... (here are some things that will remind you of that wonderful time)



Motivation

First off. Let's travel back in time to 1999... (here are some things that will remind you of that wonderful time)



Motivation

But at that very same time in bioinformatics...

- The number of sequenced genomes was very limited but increasing very fast.
- Whenever a new genome is sequenced, one could ask himself:
  - How does this genome align to the other genomes we have sequenced?

roblem:

Motivation

But at that very same time in bioinformatics...

- The number of sequenced genomes was very limited but increasing very fast.
- Whenever a new genome is sequenced, one could ask himself:
  - How does this genome align to the other genomes we have sequenced?

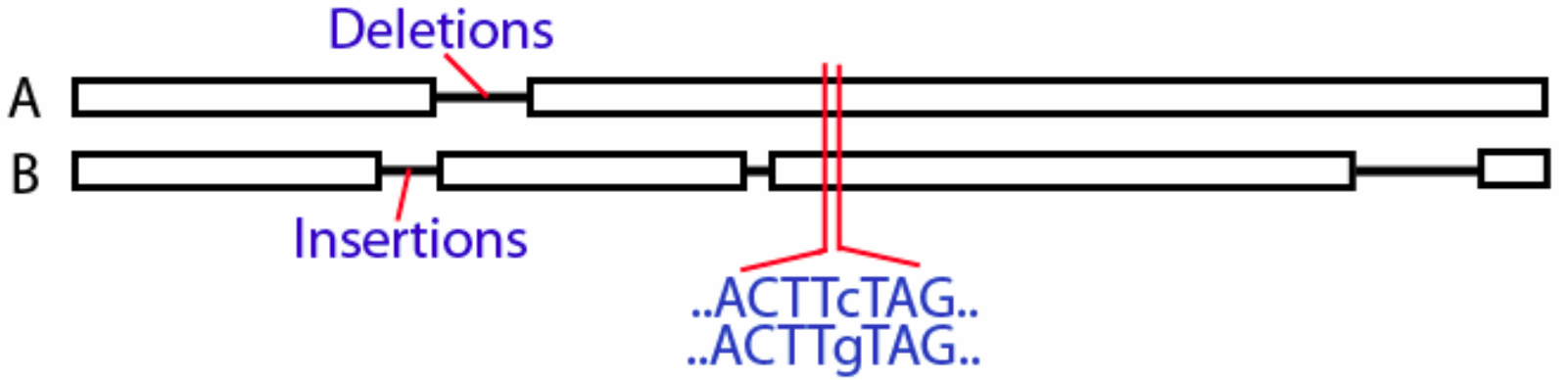
Problem:

- We had algorithms that were made for single gene sequences.
- But, they won't work well with whole genomes.
- It's simple: Size matters.
  - Take up way too much memory or
  - Have unacceptable computational time

Problem description

In Two genomes, A and B. Both could be very large (possibly over 1 Mbp)

Out Align the two genomes using insertions and deletions (or for short, indels) to maximize the matches.



Introduction to MUMmer

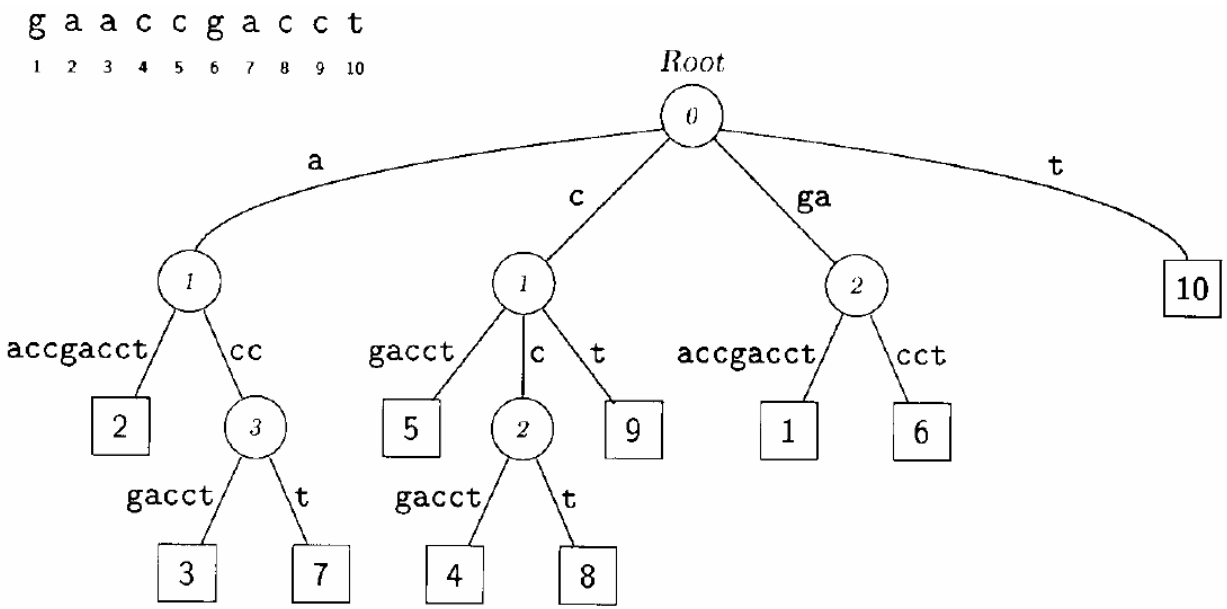
MUMmer:

- Was published in 1999.
- Is a system used to align whole genome sequences.
- Uses suffix trees as a data structure.
- Idea: Finding large chunks of exact matches on both genomes in linear time.
- Doesn't guarantee the optimal solution, just a good one.

Step 1: Creating a suffix tree

A suffix tree stores all possible suffixes in a tree.

- Square nodes are leaves.
  - Store information about the starting position of the suffix.
- Circular nodes are internal nodes.
  - Store information about the length of the shared prefix.



Creating a suffix tree takes $O(n)$ time and space.

Step 2: MUM decomposition

Let us first define what a MUM is

Definition

A subsequence is a MUM (Maximal Unique Matches) if and only if:

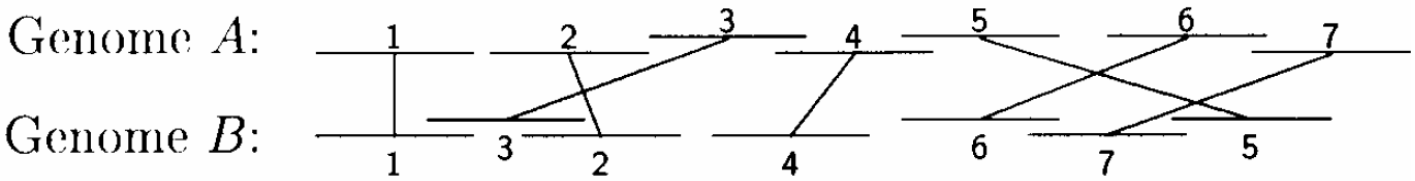
- The subsequence has an exact match on both genomes
- It is not a subsequence of another matched sequence
- It is unique

Example

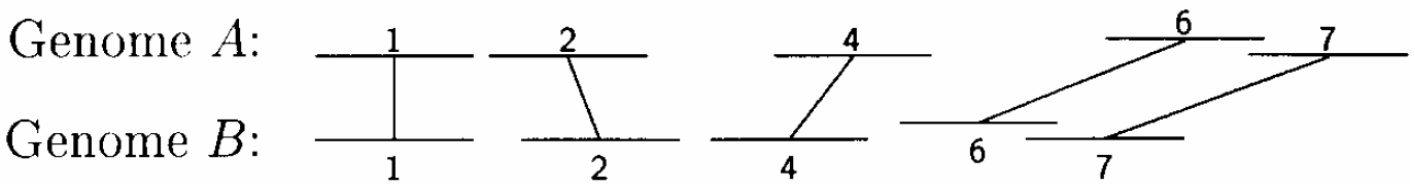
```
tcgatcGACGATCGCGGCCGTAGATCGAATAACGAGAGAGCATAAcgactta
gcattaGACGATCGCGGCCGTAGATCGAATAACGAGAGAGCATAAtccagag
```

Step 3: Sorting the matches found in the MUM alignment

Once we have found the MUMs, we enumerate them like this:

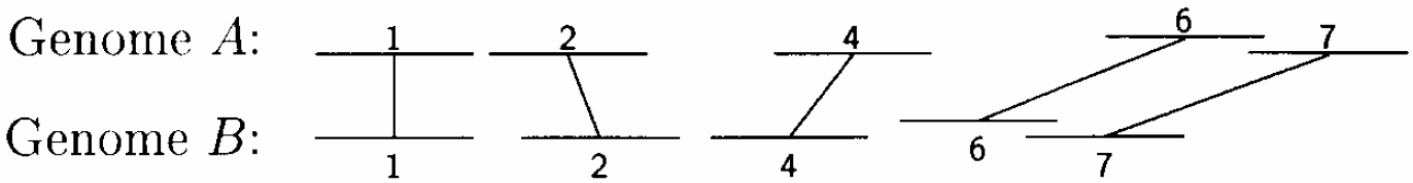


Problem: We cannot align all MUMs because they aren't in the same order in both genomes.
Solution: Align as many MUMs as we can:



Step 4: Closing the gaps

Everything in between the MUMs is called **gaps**.



- To find alignment for the gaps we can use any alignment algorithm.
- MUMmer uses the Smith-Waterman algorithm.

Diving deeper

How do we go from use suffix trees to find MUMs?

Example

In Two genomes

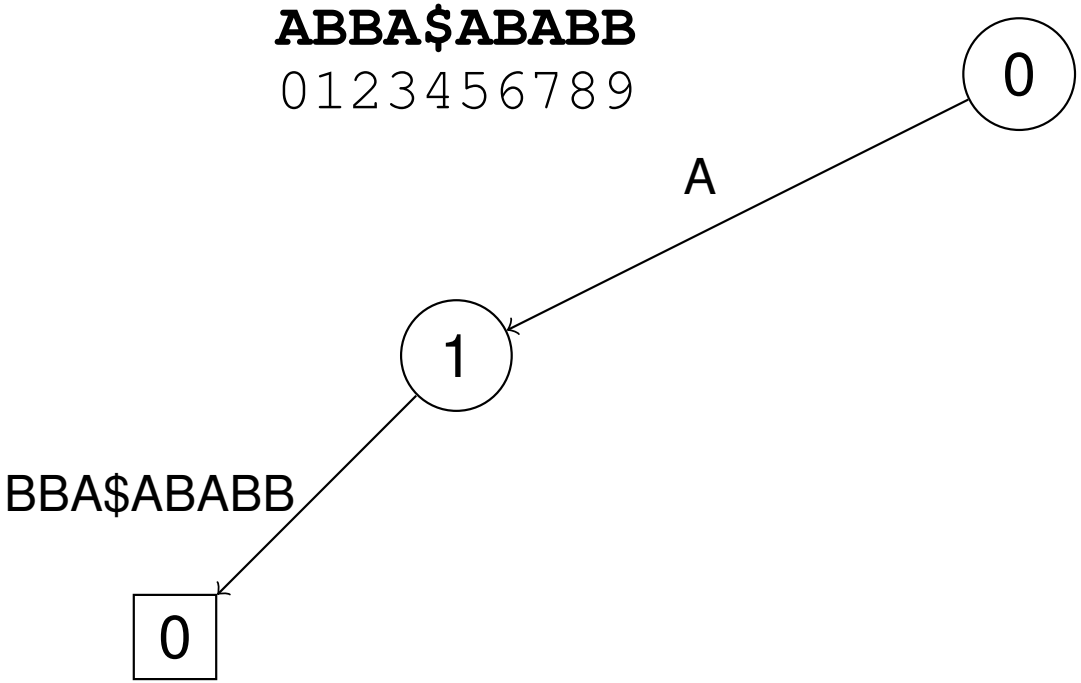
Genome A: `ABBA`

Genome B: `ABABB`

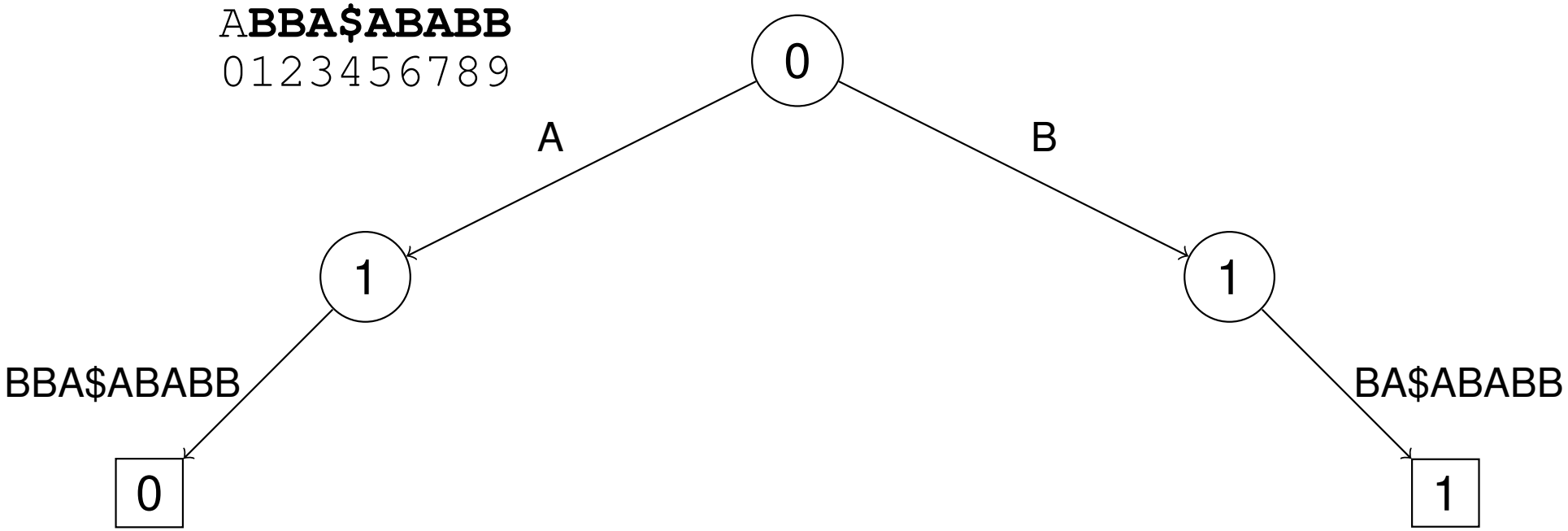
Out List of all MUMs.

Combined string: `ABBA$ABABB`

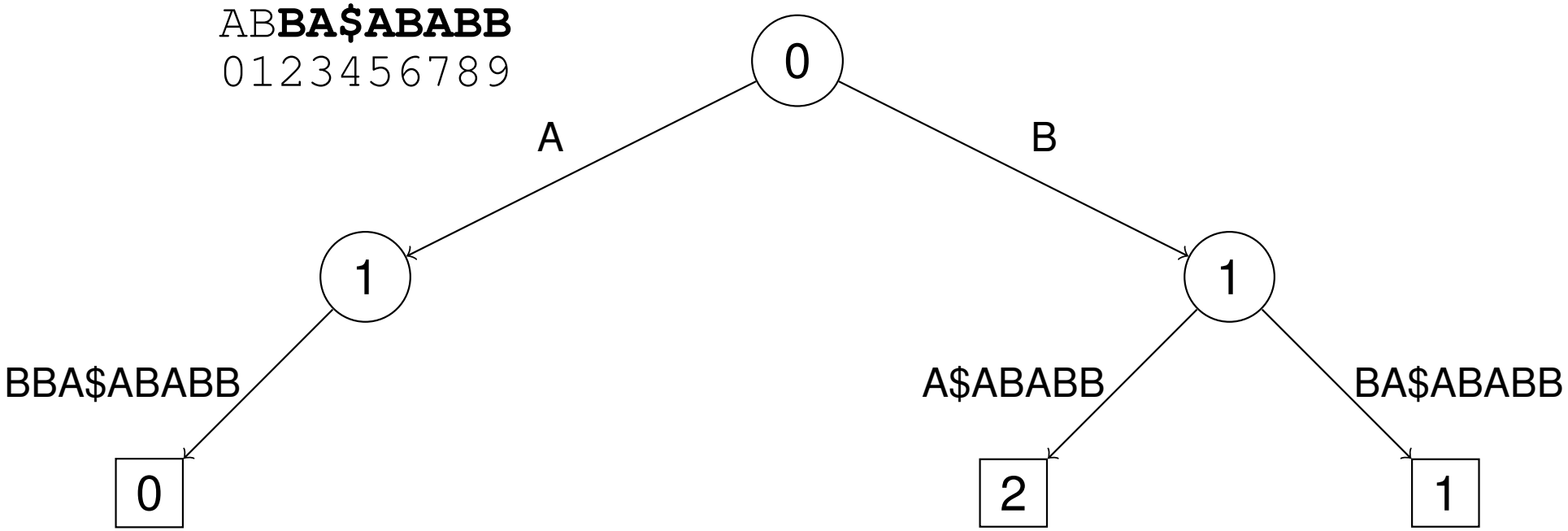
Creating a suffix tree



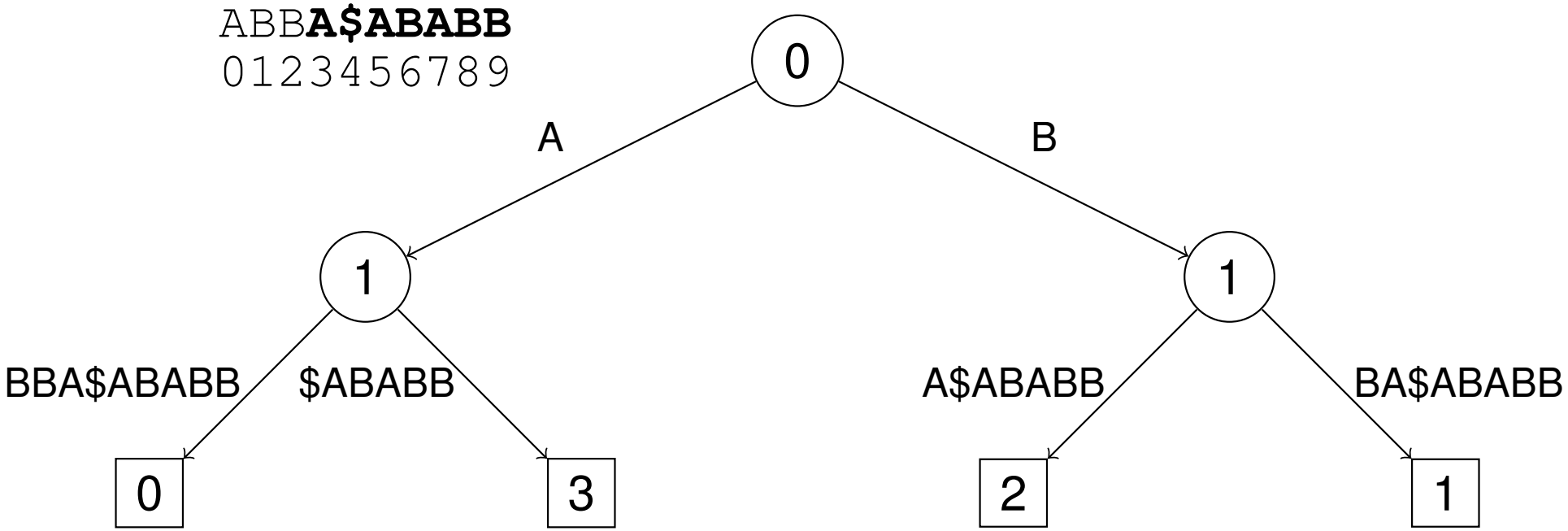
Creating a suffix tree



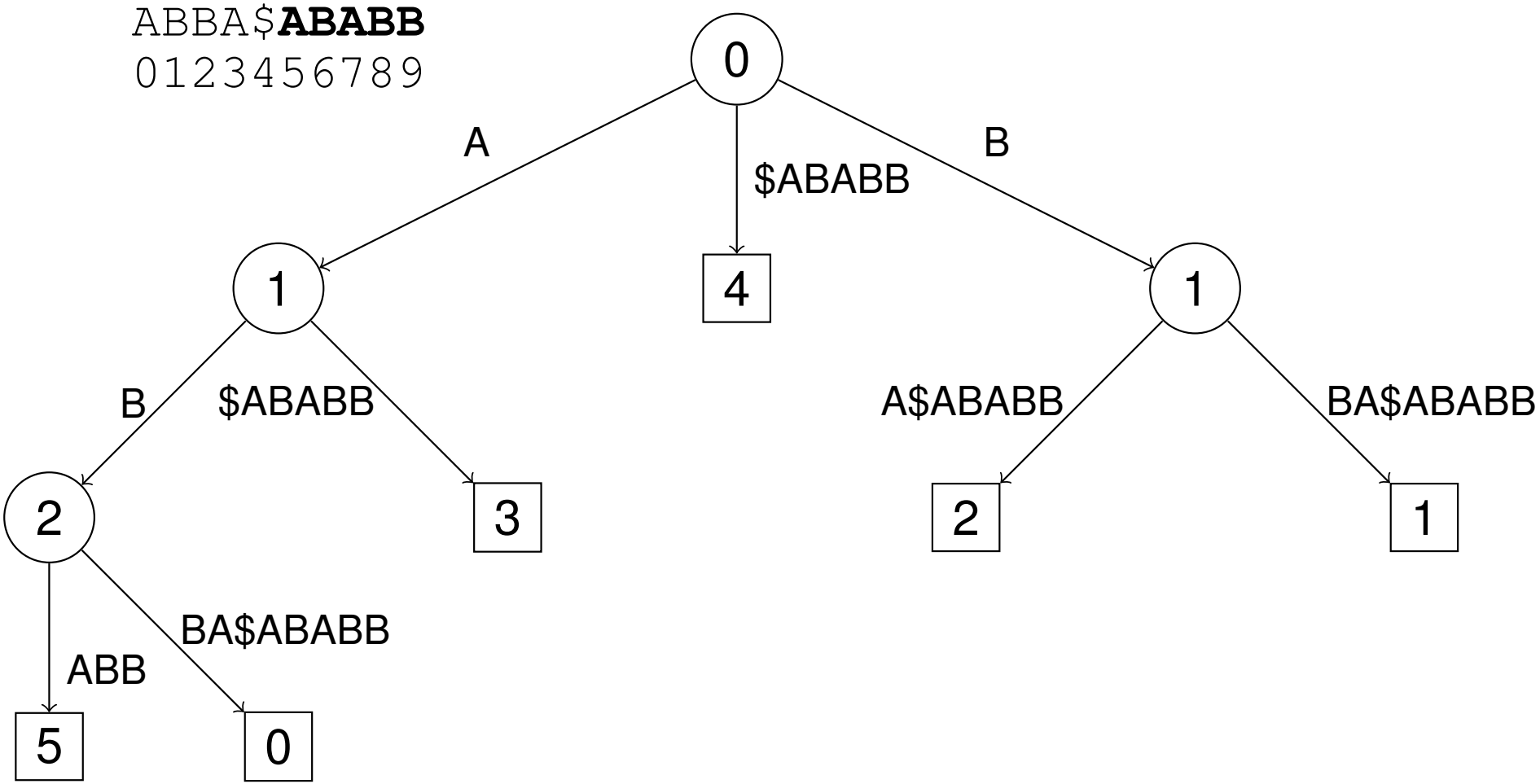
Creating a suffix tree



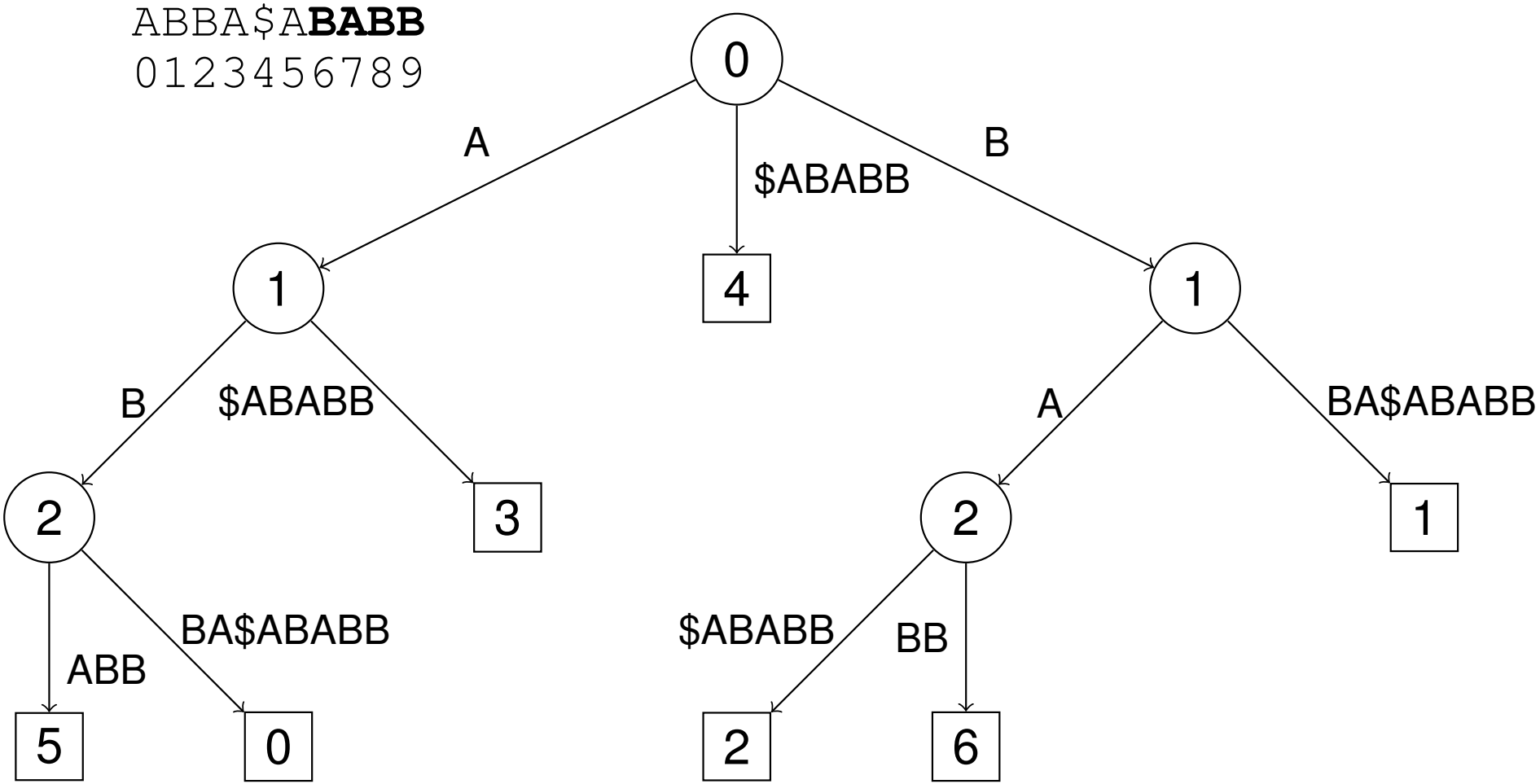
Creating a suffix tree



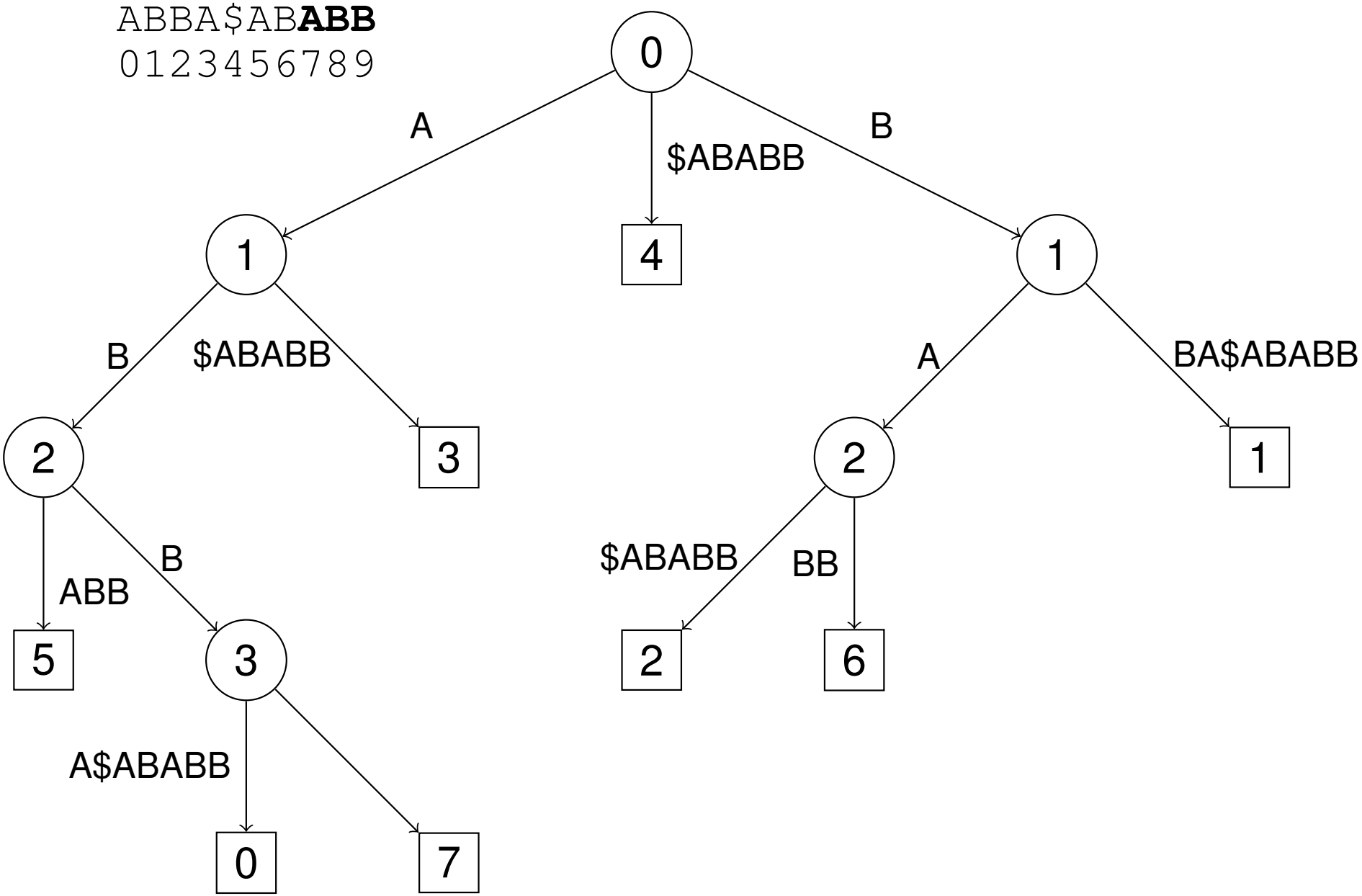
Creating a suffix tree



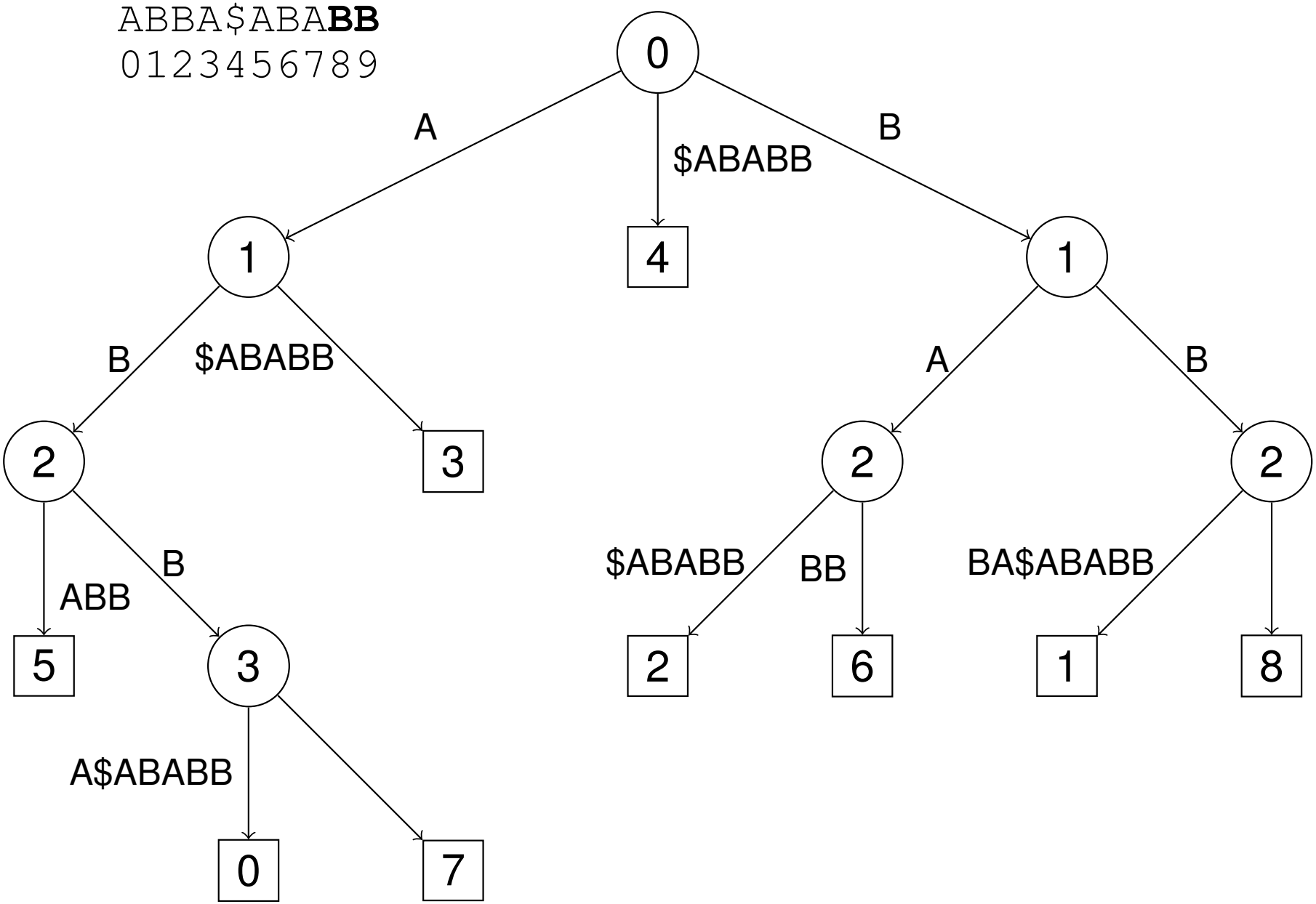
Creating a suffix tree



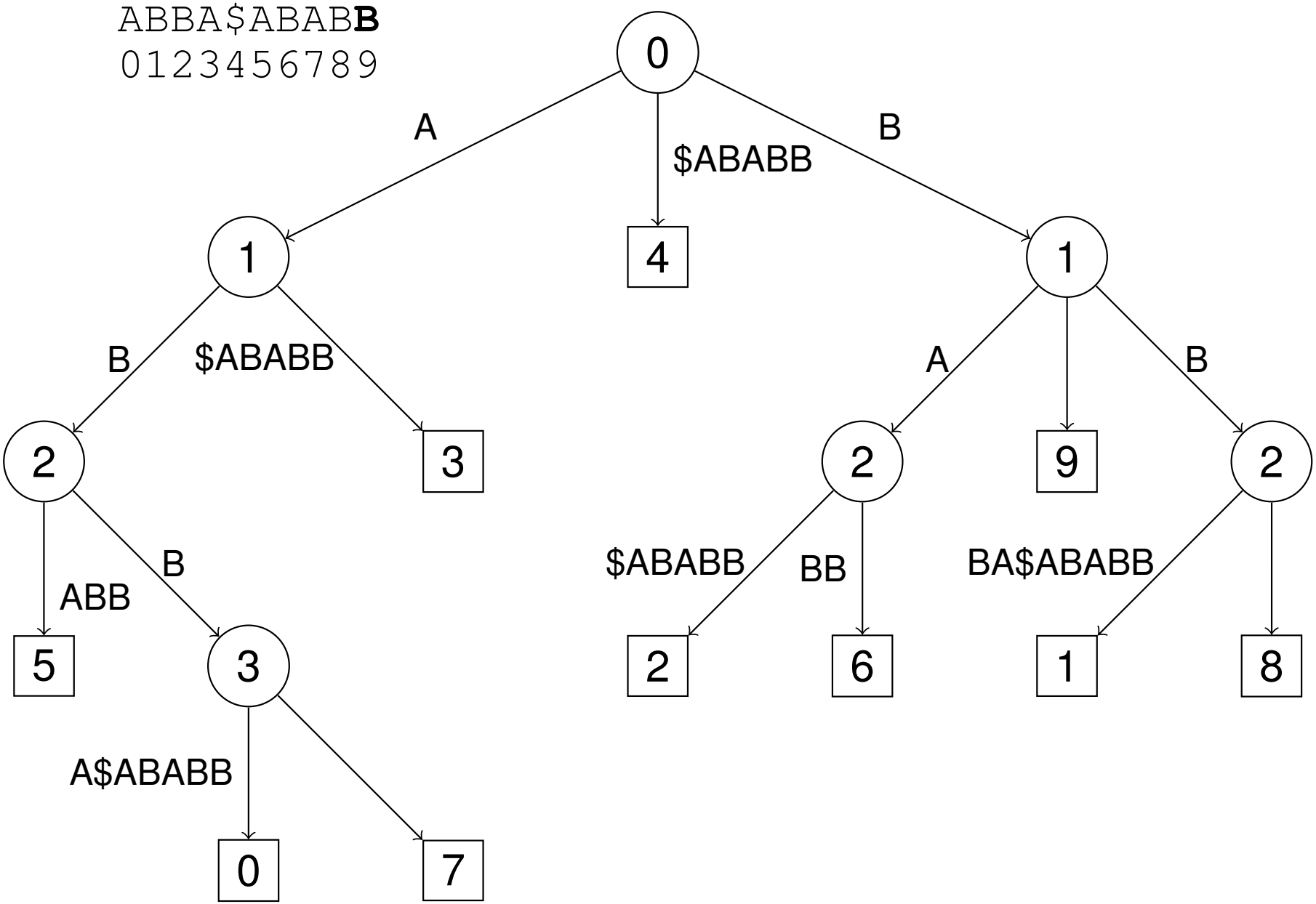
Creating a suffix tree



Creating a suffix tree



Creating a suffix tree



Finding MUMs from suffix tree

Let's recall what condition MUMs had to have:

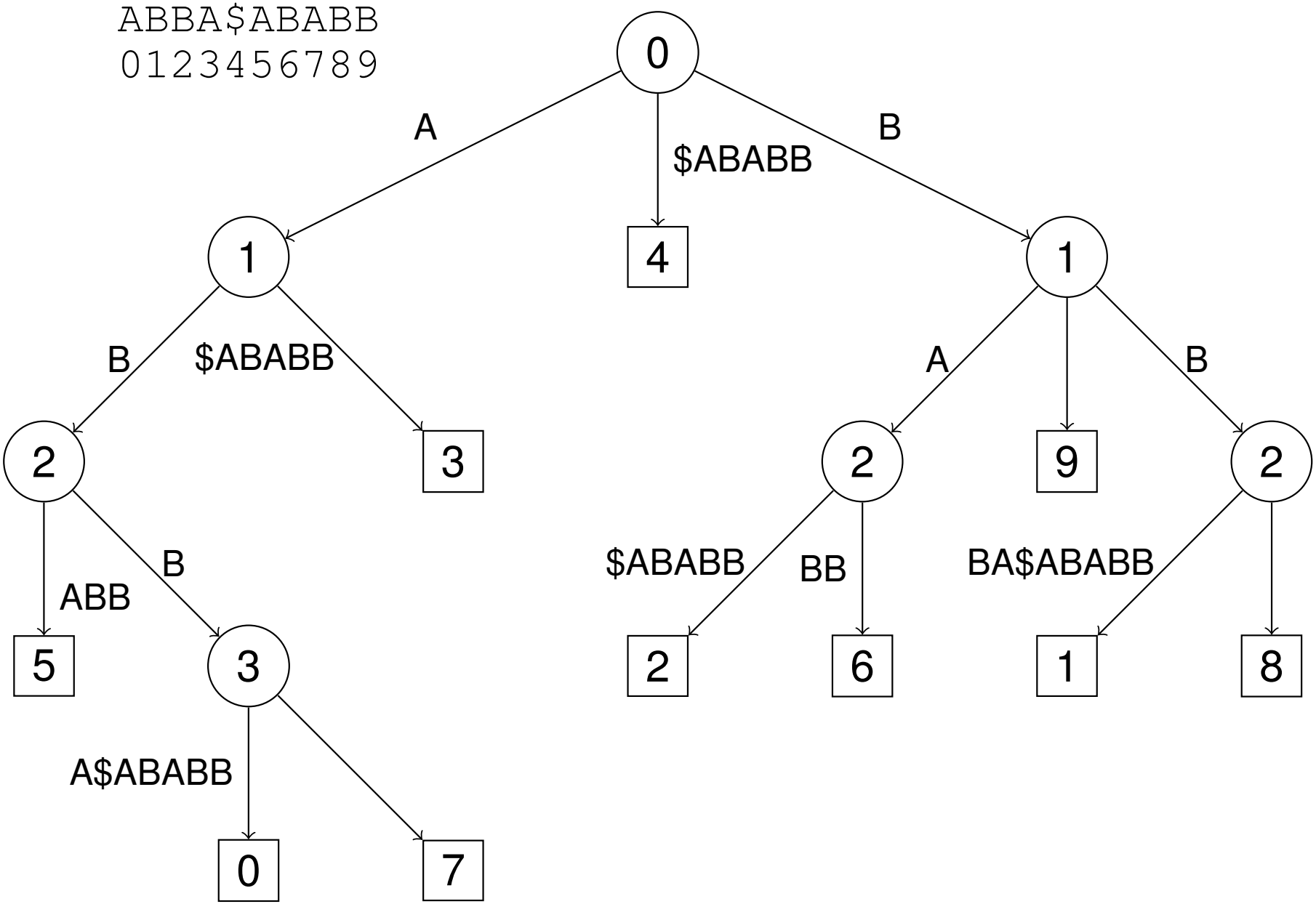
1. Exact matches on both genomes.
2. Surrounded by mismatches.
3. Unique.

We can achieve conditions 1 and 3 by searching the tree for a internal node with:

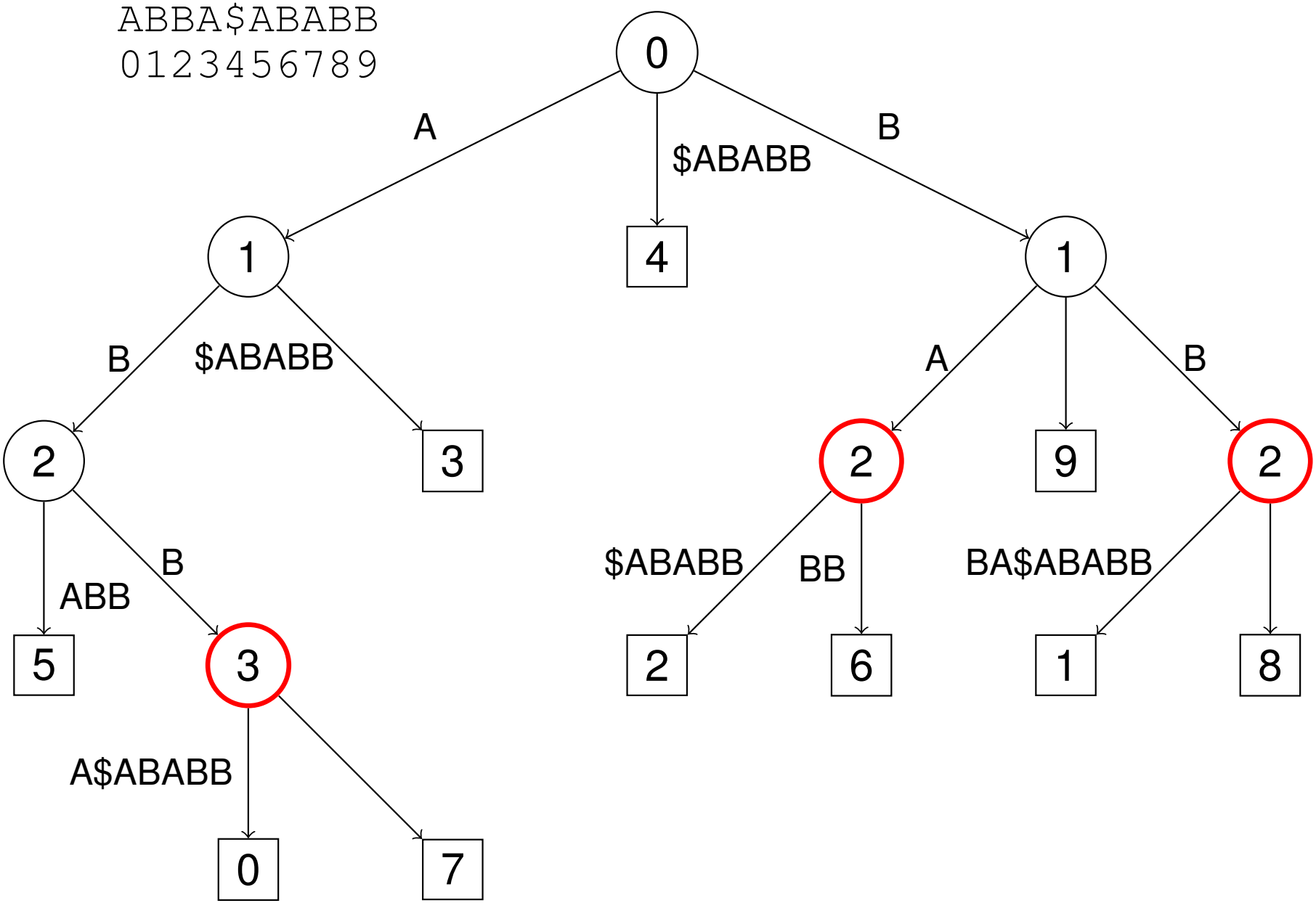
- Exactly two leafs.
- Both leaves start on each side of the dollar sign.

We'll need to check for condition 2 afterwards.

Finding potential MUMs



Finding potential MUMs



Potential MUMs to MUMs

We have the following potential MUMs:

- `ABB` at starting positions 0 and 7.
- `BA` at starting positions 2 and 6.
- `BB` at starting positions 1 and 8.

Our string: `ABBA$ABABB`. Let's check if condition 2 is satisfied

- `ABB` achieves condition 2 (it's surrounded by mismatches).
- `BA` achieves condition 2 as well.
- `BB` does NOT achieve condition 2. In both cases it is preceded by a `A`.

Resulting MUMs: **`ABB`** and **`BA`**.

Results and conclusion

MUMmer was put to the test on various genomes:

- Two strains of tuberculosis (bacteria) that are >99% identical
  - 5 seconds to create the suffix tree.
  - 45 seconds to sort the MUMs.
  - 5 seconds to generate alignments of the gaps.

Results and conclusion

MUMmer was put to the test on various genomes:

- Two strains of tuberculosis (bacteria) that are >99% identical
  - 5 seconds to create the suffix tree.
  - 45 seconds to sort the MUMs.
  - 5 seconds to generate alignments of the gaps.
- Two 'cousin' genomes. Genome of *M.genitalium* (580,074 nucleotides) and *M.pneumoniae* (816,394 nucleotides)
  - 6.5 seconds to create the suffix tree.
  - 0.02 seconds to sort the MUMs.
  - 116 seconds to generate alignments of the gaps.

Results and conclusion

MUMmer was put to the test on various genomes:

- Two strains of tuberculosis (bacteria) that are >99% identical
  - 5 seconds to create the suffix tree.
  - 45 seconds to sort the MUMs.
  - 5 seconds to generate alignments of the gaps.
- Two 'cousin' genomes. Genome of *M.genitalium* (580,074 nucleotides) and *M.pneumoniae* (816,394 nucleotides)
  - 6.5 seconds to create the suffix tree.
  - 0.02 seconds to sort the MUMs.
  - 116 seconds to generate alignments of the gaps.

So in conclusion:

- If the two genomes are very similar, MUMs sequences will...:
  - ...be long and cover most of the genome.
  - ...rarely be a random match. Therefore few or no errors.
  - ...make the algorithm run fast (almost linear time)
- If the genomes are very different, MUMs sequences will...:
  - ...be short and cover a small part of the genome.
  - ...often be a random match. Therefore many errors.
  - ...make the algorithm run slow.

MUMmer 3

- MUMmer 3 is the latest version of MUMmer
- It was released in 2004 and is open-source.
- Requires less than half the memory and more than twice as fast than the initial MUMmer.
- Most notable features added since the initial MUMmer:
  - You can allow tolarence for mismatches when finding MUMs.
  - Can handle non-unique MUMs.
  - All sorts of visualization tools.

Thank you!

Feel free to ask any questions.