Algorithms in bioinformatics

The suffix trie

As you progress within the tutorial, you should develop your intuition on the data structure, as well as gaining automy for good practices in algorithmics (proper definition of problems, running time analysis, high-level description of algorithms) and implementations (extensive code documentation with docstrings and comments, robust tests).

The aim of this tutorial is to implement a `SuffixTrie` class, that support basic pattern matching queries for the Membership, Count and LocateAll problems. We start by implementing a `Trie` class, and we then modify it iteratively to support each type of pattern matching request (and even more).

(1) ■ Tries

The trie T_S of a string collection $\mathcal{S}$ is a tree such that a string S belongs to $\mathcal{S}$ if and only if it spells a root-to-marked-node path within T_S. It is also such that the paths that correspond to string S and S' coincide in the tree while spelling their longest common prefix.

▶ **Question 1.** *(✎) Recall the algorithm that constructs the trie of a set of string given as input.*

▶ **Question 2.** *(✎) What are its asymptotical complexity in time and memory? Justify.*

▶ **Question 3.** *(✎) Use it to build the trie of $\mathcal{S} = \{climate, climb, climbing, close, cloth\}$ by hand.*

Now, open `suffixes_template.py`.

▶ **Question 4.** *(💻) What does the function `Trie.__init__` do?*

▶ **Question 5.** *(💻) Fill in the method `Trie.insert` so that it insert the word `word` into the trie `self`.*

▶ **Question 6.** *(💻) Check your implementation by building the trie of $\mathcal{S} = \{climate, climb, climbing, close, cloth\}$. You can display the resulting trie by using the `pretty_print()` method.*

(2) ■ Suffix tries

Suffix tries are particular tries suited for pattern matching, because a pattern P is a substring of some string S if it is the prefix of one of S's suffixes.

▶ **Question 7.** *(✎) Recall the definition of the suffix trie.*

▶ **Question 8.** *(✎) What is the purpose of introducing the fresh letter \$ in the definition?*

▶ **Question 9.** *(💻) Rename a copy of the `Trie` class by `SuffixTrie`, and change the `__init__` method so that it takes a single word as input and build its suffix trie.*

(3) ■ Pattern matching from the suffix tries

Pattern matching is the task of finding occurences of a given pattern within a string. Depending on the output, this define different flavours of pattern matching.

▶ **Question 10.** *(✎) Recall the definition of the Membership, Count and LocateAll problems, being as formal as possible.*

(3.1) ◆ The Membership problem

▶ **Question 11.** *(✎) Propose an algorithm that solves the Membership problem in linear time, with respect to the pattern size.*

▶ **Question 12.** *(✎) Give a trie $\mathcal{T}$ and two patterns*[*] (P, P') of same length such that the execution of your algorithm on the input $(\mathcal{T}, P)$ takes $\Theta(|P|)$ time, but only $\Theta(1)$ time on the input $(\mathcal{T}, P')$.*

▶ **Question 13.** *(💻) Implement a corresponding method `membership` within the `SuffixTrie` class.*

(3.2) ◆ The Count problem

▶ **Question 14.** *(💻) Implement a method `count_covered_leaves` for `TrieNode` that uses a depth-first traversal of the node's subtree to count the number of leaves it contains.*

▶ **Question 15.** *(💻) Implement a method `count` for `SuffixTrie`.*

▶ **Question 16.** *(💻) Implement a method `preprocess_leaf_covering` for `SuffixTrie` that stores, at the level of nodes, the number of leaves it covers. You can use the dictionnary `TrieNode.misc`, and add a value for the key `'nb_covered_leaves'`. This method should run in time $\mathcal{O}(|\mathcal{T}|)$.*

▶ **Question 17.** *(💻) Implement a method `count_fast` for `SuffixTrie`, that take advantage of the information `'nb_covered_leaves'`.*

▶ **Question 18.** *(✎) Compare the theoretical complexity of the algorithms under `count` and `count_fast`.*

▶ **Question 19.** *(💻) Can you observe this difference in practice? For (wall-clock) timing executions, you can use the `time` module:*

```
import time
start_time = time.time()
# PUT HERE THE EXECUTION TO MEASURE
print(f"--- {time.time() - start_time} seconds ---")
```

(3.3) ◆ The LocateAll problem

▶ **Question 20.** *(💻) Implement a method `locate_all` for `SuffixTrie`. This requires to change the `__init__` function of `TrieNode` to link leaves with their corresponding suffix starting position. You can modify the `pretty_print` function to help debugging.*

(4) ■ More insights from suffix tries

▶ **Question 21.** *(✎ **BONUS**) Propose a high-level algorithm to compute the longest common substring of two strings. What is its complexity?*

▶ **Question 22.** *(💻 **BONUS**) What is the longest common substring between* Hepatitis delta virus *and* Human immunodeficiency virus-1 *genomes?*

[*] To be rigourous one should give a family of such patterns, for any possible length. We let this consideration aside.