Assignment 3 - Thor Malmby Jørgin

Prep work

As always I started out by type hinting the entire template.

Although I have experience with Python, I don't really have experience with NumPy.
Therefore, you will notice, that I have not really used any NumPy features myself.

To make the pseudocode easier to follow, I renamed the following variables in the functions:

- `f` → `qf`
- `transitions` → `a_transitions`
- `emissions` → `b_emissions`

This made the pseudocode much easier to follow and implement.

Forward algorithm

For the forward algorithm I implemented the pseudocode as it is written,
and initially I had a helper method for computing the sums.
However after I had implemented it at first, I compared my outputs,
with those of another student, and could see that they differed.
I suspected my summation to be wrong,
so I chose to use list comprehensions instead to generate the elements to be summed over:

```
forward[s][t] = sum(
    [
        forward[s_prime][t - 1] * a_transitions[s_prime][s] * b_emissions[s]
[observations[t]]
        for s_prime in inclusive_range(1, big_n)
    ]
)
```

In which the line `for s_prime in inclusive_range(1, big_n)`
can be mentally mapped to the Sigma symbol in the pseudocode.

List comprehensions are also used for the termination step.

Viterbi algorithm

For viterbi the story is similar. I followed the pseudocode and implemented it.

Here I tried to use the argMax provided, but at first I could not wrap my head around it.
Therefore, I decided to rewrite it to take in tuples that provide both the arg, and the value.
This way it is actually also a bit more robust to work with in case the provided arguments came from a "jumping" source.
i.e.

```
[
    (2, f(2)),
    (4, f(4)),
    (6, f(6))
]
```

This is also the reason that the list comprehensions that are generated for the arg-max calls are slightly different.
Since they have to provide a tuple at each generated element in the list, as opposed to a "raw" value.

I have however since realised that the provided arg-max would provide the correct arg as long as the input is of this format **exactly**:

```
[
    f(1),
    f(2),
    f(3),
    ...
]
```

(To be clear, the numbers are the argument and the result of evaluating f(x) is what would be written into the list.)

With that said, I think that the list comprehensions made the Python code resemble the pseudocode quite a lot.
They helped make it easier to check that the combinations of arguments was correct.