String-searching Algorithms

Naive search

Sergio Peignier sergio.peignier@insa-lyon.fr

Associate Professor
INSA Lyon
Biosciences department

How would you search a word within a document?

An intuitive example: Searching a word in a book

How would you search a word within a document?

Given $S, T \in \mathcal{S}^2$ two strings, such that $|S| < |T|$

Problem 1: Is S a sub-string of T?
Problem 2: Where are the instances of sub-string S in T?

Exercise 1: Express both problems formally

Exercise 2: Write a function that tackles this problems

How would you search a word within a document?

Problem 2: Where are the instances of sub-string S in T?

$$f : \mathcal{S}^2 \to \{u \mid u \subseteq \{0, 1, ..., |T|\}\}$$

$$f(S, T) = \{i \quad \forall i \in \{1, ..., |T| - |S|\} \mid d(S, T[i, |S| + i]) = 0\}$$

Problem 1: Is S a sub-string of T?

$$f : \mathcal{S}^2 \to \{0, 1\}$$

$$f(S, T) = |\{i \quad \forall i \in \{1, ..., |T| - |S|\} \mid d(S, T[i, |S| + i]) = 0\}| > 0$$

Naive search a word within a document

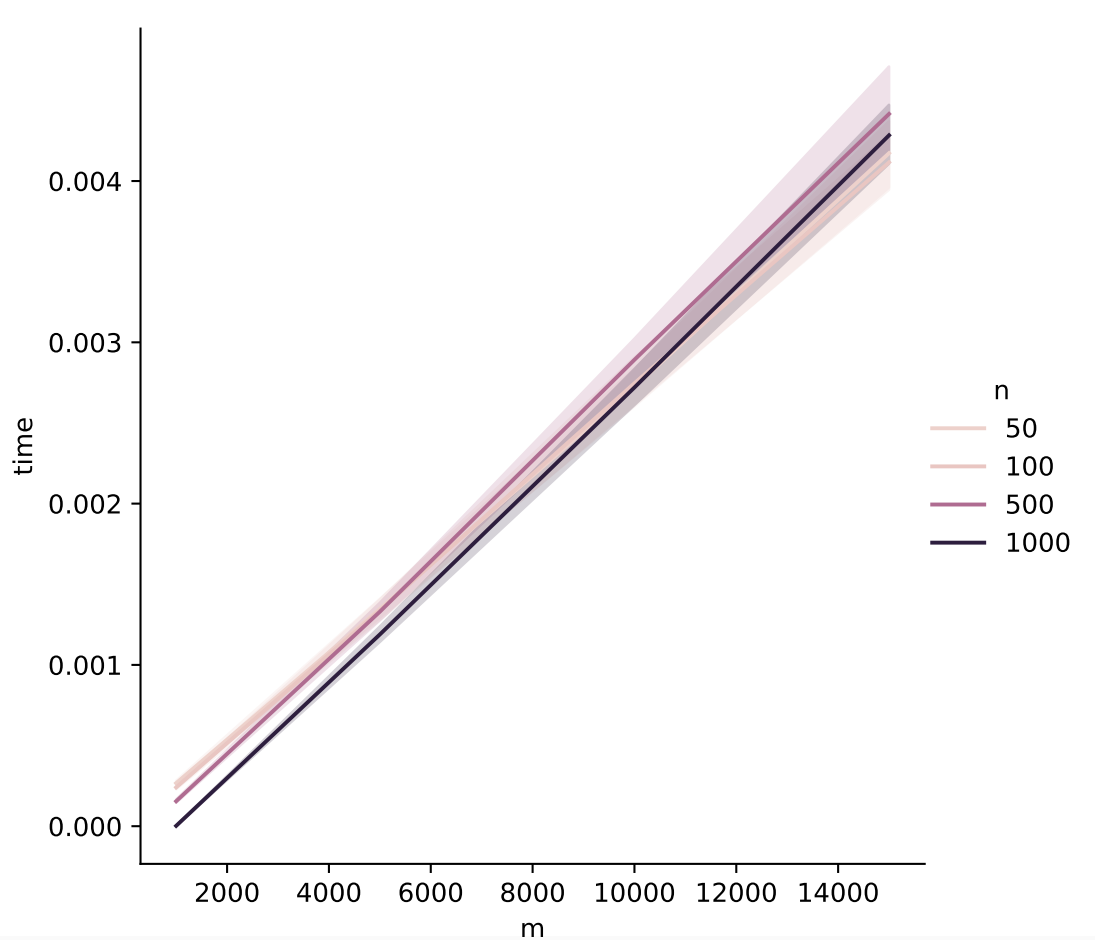
```python
def naive_string_search(T,S):
    for i in range(len(T) - len(S) + 1):
        j = 0
        while j < len(S) and T[i+j] == S[j]:
            j += 1
        if j == len(S):
            return(i)
```

Figure 1: Naive sub-string search

Question 1: Compute the space and time complexities.

Question 2: Modify the algorithm to increase the runtime.

Naive search a word within a document



Naive search runtime (seconds) for random sequences $|T| = m, |S| = n$ (for 100 repetitions)

How could we compare large strings simply?

- Let us image two large genomes $S, T \in \mathcal{S}^2$, s.t. $|S| < |T|$
- Does searching S in T make sense?

Exercise: Propose a simple method based on the naive sub-string search algorithm to compare S and T

Can we really compare (very) large strings naively?

Simple procedure to compare S and T:

- Split S in N small fragments of size n
- Search each one of the N fragments of S in T

Exercise: Human and dog genomes contain resp. 3×10^9 bps and 2.5×10^9 bps, let $n = 50$, and let us consider that the character comparison runs in 30 ns.

How many operations does this procedure need to run?

how long would it take to compare both genomes?

Can we really compare (very) large strings naively?

$n_{human} = 3 \times 10^9,\ n_{dog} = 2.5 \times 10^9,\ n = 50,\ r = 30 \times 10^{-9}$

- Number of fragments $N = \frac{n_{dog}}{n}$
- Number of operations per fragment
  $x = (n_{human} - n) \times n$
- Number of operations
  $X = x \times N = (n_{human} - n) \times n \times \frac{n_{dog}}{n} = (n_{human} - n) \times n_{dog}$
- Runtime: $R = X \times r = (n_{human} - n) \times n_{dog} \times r$
- Complexity: $\mathcal{O}(|S||T|)$

Numerical application:

- Operations: $X = 7.5 \times 10^{18}$
- Runtime: $R = 7134.7$ years

The dictionnary analogy

```python
def dichotomic_search(ordered_list, word, start, stop):
    search_position = (start + stop) // 2
    if start <= stop:
        if ordered_list[search_position] == word:
            return search_position
        elif ordered_list[search_position] > word:
            return dichotomic_search(ordered_list, word, start, search_position - 1)
        else:
            return dichotomic_search(ordered_list, word, search_position + 1, stop)
    return -1
```

Dichotomy search : efficient algorithm for ordered lists

Question: Compute the complexity of searching a word of size m in an ordered list of size n

(hint: Dichotomy search for list of n numbers $\sim \mathcal{O}(log\ n)$)

The dictionary analogy

```
def dichotomic_search(ordered_list, word, start, stop):
    search_position = (start + stop) // 2
    if start <= stop:
        if ordered_list[search_position] == word:
            return search_position
        elif ordered_list[search_position] > word:
            return dichotomic_search(ordered_list, word, start, search_position - 1)
        else:
            return dichotomic_search(ordered_list, word, search_position + 1, stop)
    return -1
```

Dichotomy search : efficient algorithm for ordered lists

Complexity: $\mathcal{O}(m \log n)$

Can we adapt this technique to compare genomes?

Can we adapt this technique to compare genomes?

Given two large strings $S, T \in \mathcal{S}^2$

- Extract all the suffixes of S
- Sort them
- Return a list of suffixes (with their respective indexes)

This data structure is called **suffix array**

Question: Compute the new complexity, and estimate the runtime for the dog/human comparison

Can we adapt this technique to compare genomes?

- Naive suffix table construction complexity
  $\sim \mathcal{O}(|S|^2 \; log \; |S|)$
- Can be reduced to $\sim \mathcal{O}(|S|)$
- Comparison complexity: $\sim \mathcal{O}(|T| \; log \; |S|)$
- Comparison runtime (rough) estimation: $\sim 30min$