CS213/293 Data Structure and Algorithms 2023

Lecture 10: Pattern matching

Instructor: Ashutosh Gupta

IITB India

Compile date: 2023-09-05

Topic 10.1

Pattern matching problem

Pattern matching

Definition 10.1

In a pattern-matching problem, we need to find the position of all occurrences of a pattern string P in a string T.

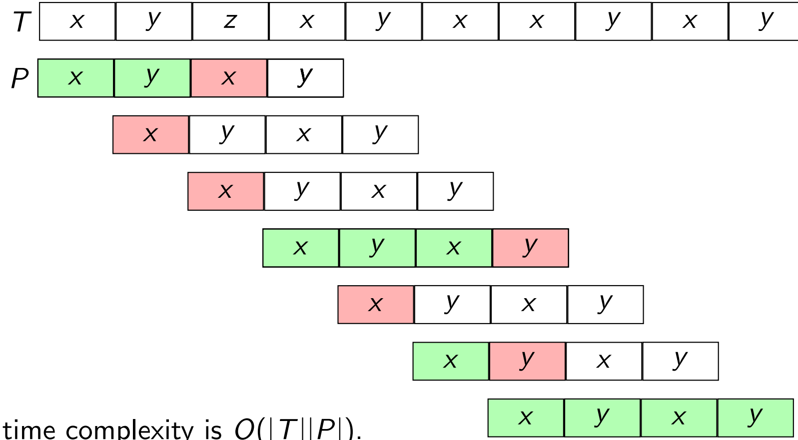
Usage:

- Text editor
- DNA sequencing

Example : Näive approach for pattern matching

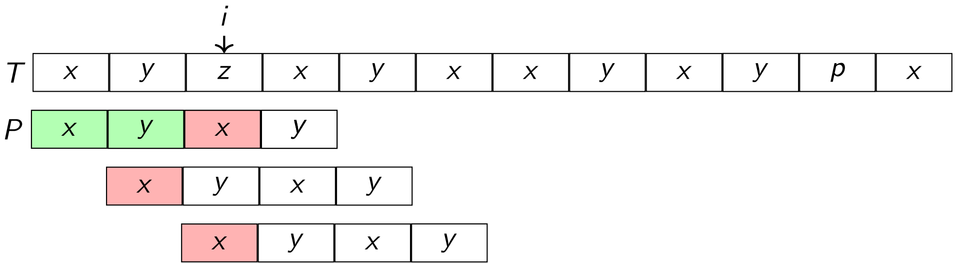
Example 10.1

Consider the following text T and pattern P. We try to match the pattern in every position.



Running time complexity is $O(|T||P|)$.

Wasteful attempts of matching.



Should we have tried to match at the second and third positions?

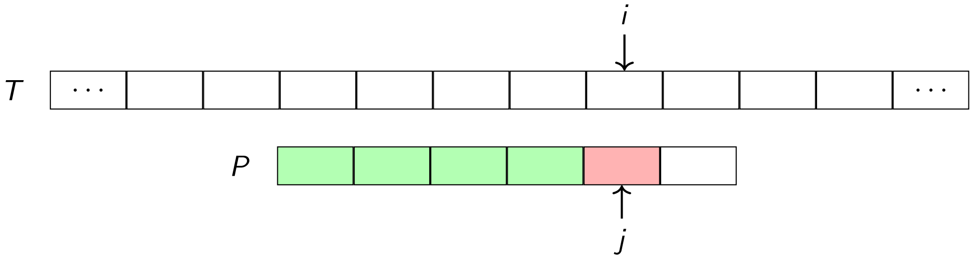
No.

Let us suppose we failed to match at position i of T and position 2 of P.

- We know that $T[i-1] = y$. Therefore, there is no match starting at $i-1$. (Why?)
- We know that $T[i] \neq x$. Therefore, there is no match starting at i. (Why?)

Shifting the pattern

Let us suppose at position i of T and j of P the matching fails.



Let us suppose we want to resume the search by only updating j.

If we assign j some value k, we are shifting the pattern forward by $j - k$.

Exercise 10.1

What is the meaning of $k = j - 1$, $k = 0$, or $k = -1$?

Out-of-bounds access of P

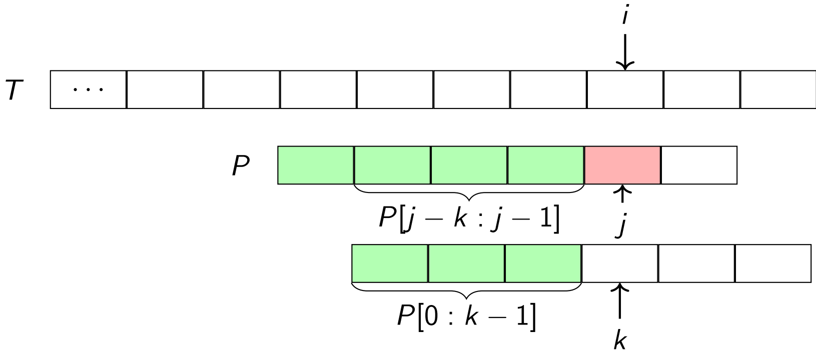
If k takes value -1 or $|P|$, $P[k]$ is accessing the array out of bounds.

For consistency of the definitions, we will say $P[-1] = P[|P|] = Null$.

However, the algorithms will be carefully written and there will be no out-of-bound access in them.

What is a good value of k?

We know $T[i-j:i-1] = P[0:j-1]$ and $T[i] \neq P[j]$.



We must have $P[0:k-1] = P[j-k:j-1]$ and $P[j] \neq P[k]$(Why?).

Exercise 10.2

Should we choose the largest k or smallest k?

The largest k implies the minimum shift

We choose the largest k such that

$$P[0 : k-1] = P[j-k : j-1] \text{ and } P[j] \neq P[k].$$

k only depends on P and j.

Since P is typically small, we may pre-compute array h such that $h[j] = k$.

Example 10.2

P	x	y	x	y	
h	-1	0	-1	0	2

P	x	y	x	z	
h	-1	0	-1	1	0

We can compute h in $O(|P|)$ time. We will discuss this later.

Knuth–Morris–Pratt algorithm

Algorithm 10.1: KMP(string T,string P)

```
1  i := 0; j := 0;  found := ∅;
2  h := KMPTable(P);
3  while i < |T| do
4      if P[j] = T[i] then
5          i := i + 1;  j := j + 1;
6          if j = |P| then
7              found.insert(i − j);
8              j = h[j];
9      else
10         j = h[j];
11         if j < 0 then
12             i := i + 1;  j := j + 1;
13 return found
```

Running time complexity:

- In total, line 5 and 12 will execute $\leq |T|$ times.
- How do we bound the number of iterations when the **else** branch does not increment i?
  1. The **else** branch reduces j.
  2. Since $j \geq 0$ at loop head, the no. of reductions of $j \leq$ no. of the increments of j.
  3. i and j are incremented together.
  4. no. of reductions of $j \leq$ no. of increments of i.
  5. no. of reductions of $j \leq |T|$.
- $O(|T|)$ algorithm

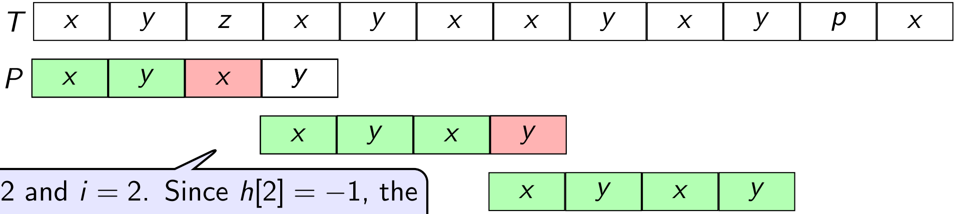
Example : KMP execution

Example 10.3

Consider the following text T and pattern P. Let us suppose, we have h.

P	x	y	x	y

h	-1	0	-1	0	2



Shifting at $j = 2$ and $i = 2$. Since $h[2] = -1$, the pattern is shifted to 3, $j = 0$ and $i = 3$.

Topic 10.2

How to compute array h?

Recall: the definition of h

For a pattern P, $h[j]$ is the largest k such that

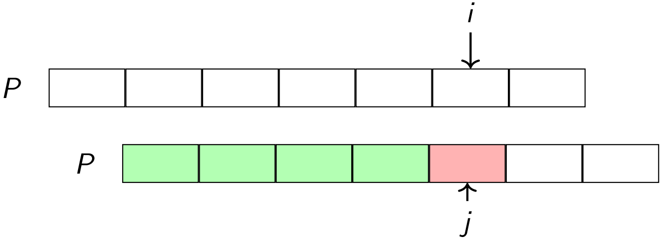
$$P[0:k-1] = P[j-k:j-1] \text{ and } P[j] \neq P[k].$$

We use KMP like algorithm again to compute h.

When we compute $h[j]$, we assume we have computed $h[j']$ for each $j' \in [0, j)$.

Self-matching: We use KMP again for computing h

For largest j such that $P[i-j:i-1] = P[0:j-1]$ and $P[i] \neq P[j]$.

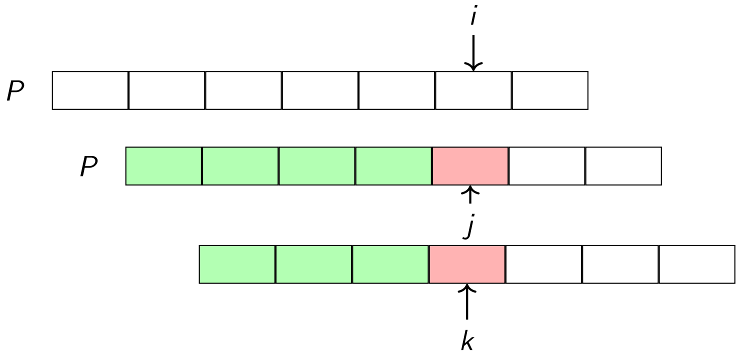


We assign $h[i] := j$.

Now we need to move the pattern forward.

Self-matching: Moving the pattern forward

After the mismatch, we need to move the pattern forward as little as possible.



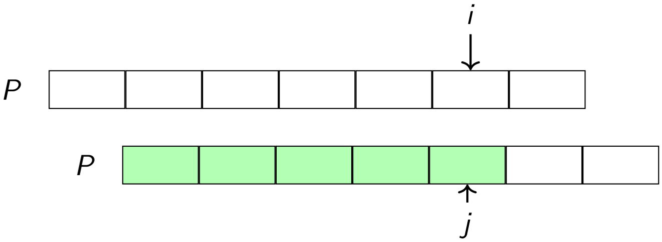
We must have computed h for earlier indexes. Therefore, $j := h[j]$.

Exercise 10.3

Why the value of $h[j]$ be available?

Self-matching: if no disagreement

Let us consider the case when matching continues. How should we assign $h[i]$?



$h[i] := j$ may not be efficient.

If the suffix of part of T does not match with $P[0:i]$ then it will also not match with $P[0:j]$.

We will be jumping again to $h[j]$. We should directly assign $h[i] := h[j]$.

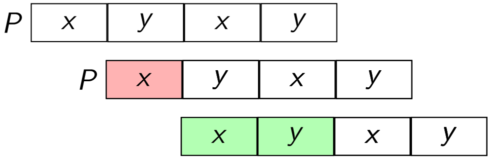
Computing h array

Algorithm 10.2: KMPTABLE(string P)

```
i := 1; j := 0;  h[0] := −1;
while i < |P| do
    if P[j] ≠ P[i] then
        h[i] := j;
        while j ≥ 0 and P[j] ≠ P[i] do
            j := h[j];                // Shifting
    else
        h[i] := h[j];
    i := i + 1;   j := j + 1;
h[|P|] := j;
return h
```

Example 10.4

Consider the following pattern P



h					
	-1	0	-1	0	2

Topic 10.3

Problem

Exercise: compute h

Exercise 10.4

Compute array h for pattern "babbaabba".

Exercise: version of KMPTable

Exercise 10.5

Is the following version of KMPTable *correct?*

Algorithm 10.3: KMPTableV2(string P)

```
i := 1; j := 0;  h[0] := −1;
while i < |P| do
    h[i] := j;
    while j ≥ 0 and P[j] ≠ P[i] do
        j := h[j];                    // Moving forward the pattern in minimum steps as in KMP
    i := i + 1;   j := j + 1;
h[|P|] := j;
return h
```

Exercise: compute $h(i)$

Exercise 10.6

Suppose that there is a letter z in P of length n such that it occurs in only one place, say k, which is given in advance. Can we optimize the computation of h?

End of Lecture 10