CS213/293 Data Structure and Algorithms 2025

Lecture 10: Pattern matching

Instructor: Ashutosh Gupta

IITB India

Compile date: October 3, 2025

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.

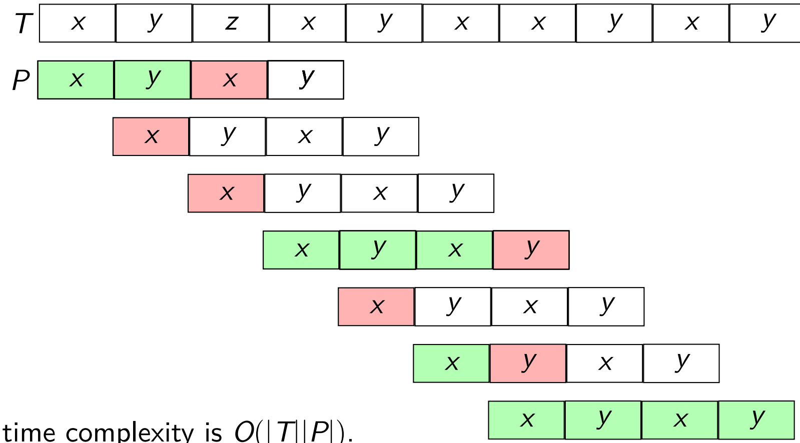
Usage:

- Text editor
- DNA sequencing

Example : Naive approach for pattern matching

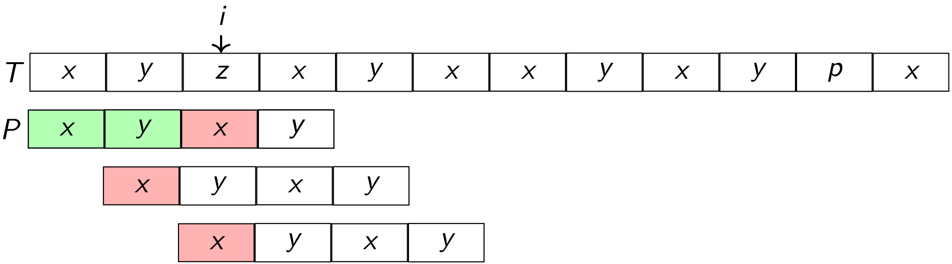
Example 10.1

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 the pattern at the second and third positions?

No.

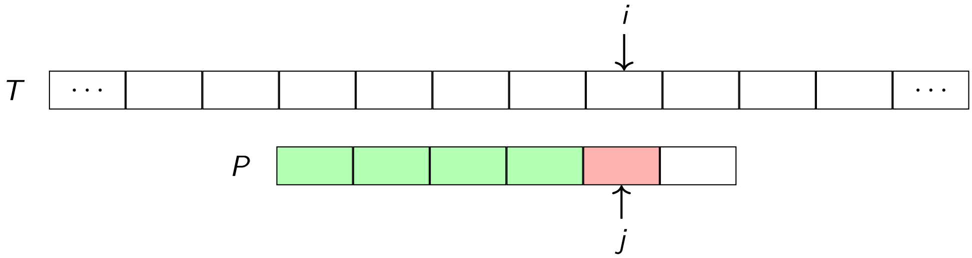
Commentary: In the drawing i is 2. However, we have named the position i to illustrate the argument using symbolic expressions.

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

Side note: 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.

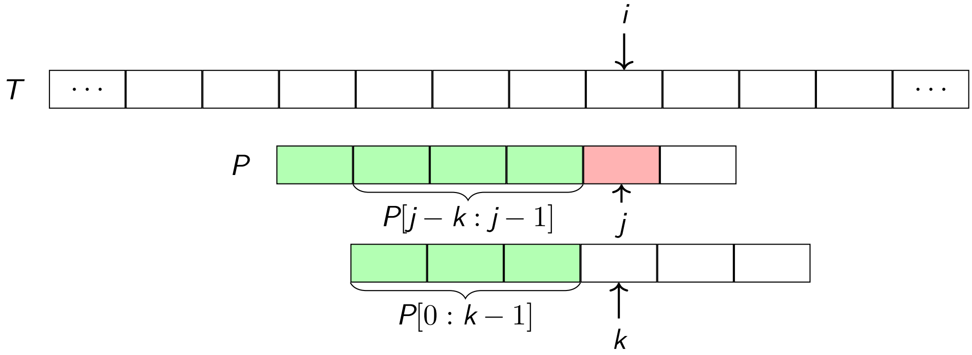
Definition 10.2

Let $P[i:j]$ indicates the array containing elements $P[i], ...P[j]$.

Commentary: In a formal definition, we may overlook or simplify some implementation issues. This allows us to write clean mathematical definitions. However, the implementations need to be careful about the issues.

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

Commentary: $P[0 : k-1]$ is the longest proper prefix that is also suffix (LPS) of $P[0 : j-1]$ and $P[j] \neq P[k]$. Array h keeps the records of LPSs.

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 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.

Exercise 10.3

a. Show that $k \neq j$.

b. Show that $j > h(j) \geq -1$ for each $j \in [0..|P|)$

c. Show that $|P| > h(|P|) \geq 0$ if $|P| > 0$. Is it true if $|P| = 0$?

d. If we drop condition $P[j] \neq P[k]$, what may go wrong?

Commentary: Answer of c: Since $P[|P|] = null$, we are guaranteed that $P[|P|] \neq P[0]$. Since we have $P[0 : -1] = P[j : j-1]$. $k = 0$ will satisfy the condition for $P[|P|]$. Since we are looking the largest k, $k \geq 0$.

Knuth–Morris–Pratt algorithm

Algorithm 10.1: KMP(string T,string P)

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

Running time complexity:

- Since no. of increments of $i \leq |T|$, the line 6 and 13 will execute $\leq |T|$ times in total.
- How do we bound the number of iterations when the **else** branch does not increment i?
  1. The **else** branch reduces j because $h[j] < j$.
  2. Since every time at the loop head $j \geq 0$ (Why?), no. of reductions of $j \leq$ no. of increments of j.
  3. Since i and j are always incremented together, no. of reductions of $j \leq$ no. of increments of i.
  4. no. of reductions of $j \leq |T|$.
- $O(|T|)$ algorithm

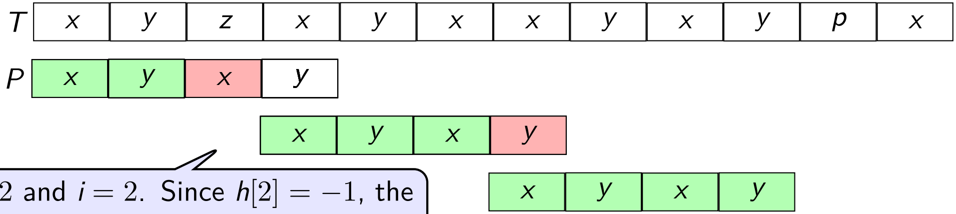
Commentary: The step two is bounding the number of reductions over all iterations of the loop (needs some thinking). It is called amortized complexity. Note that the argument does not guarantee a constant bound over the number of consecutive reduction steps.

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, $h[i]$ is the largest k such that

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

We use KMP like algorithm again to compute h.

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

Self-matching: use KMP again for computing h

We run two indexes i and j on P such that $j < i$.

We assume that for each $k \in (j, i), \neg(P[0 : k-1] = P[i-k : i-1] \wedge P[i] \neq P[k])$.

We will be computing $h[i]$. Let j be the current running match,i.e, $P[i-j : i-1] = P[0 : j-1]$.

When we consider position i, we have two cases.

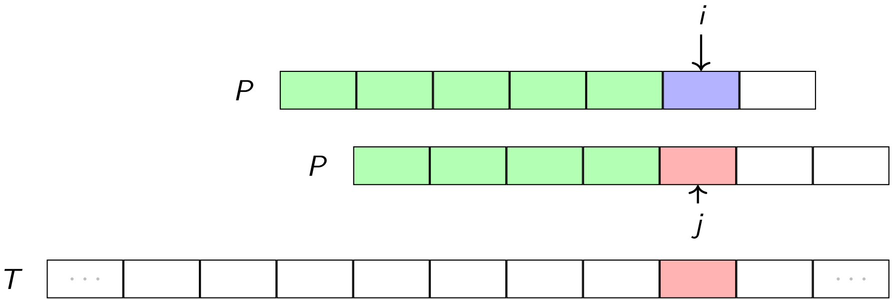
1. $P[i] \neq P[j]$
2. $P[i] = P[j]$

In both the cases, we need to update $h[i]$ and may update j.

We ensure that j is largest by updating j as little as possible.

Commentary: Due to the assumption, no k larger than j is a valid candidate of $h[i]$.

Case 1: $P[i] \neq P[j]$



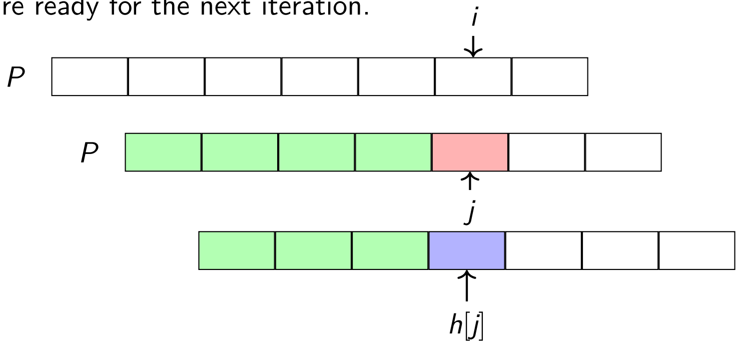
We assign $h[i] := j$, since j meets the requirements.

We have found the shift position for i.Now, we need to prepare for the next index $i + 1$.

Now we need to move the pattern forward as little as possible.

Case 1 (continued): $P[i] \neq P[j]$

After the mismatch, we move the pattern forward as little as possible such that we have a match at position i and are ready for the next iteration.



We must have computed h for earlier indexes. We set $j := h[j]$.
We need to keep reducing j until $P[j] = P[i]$ or $j \leq 0$.

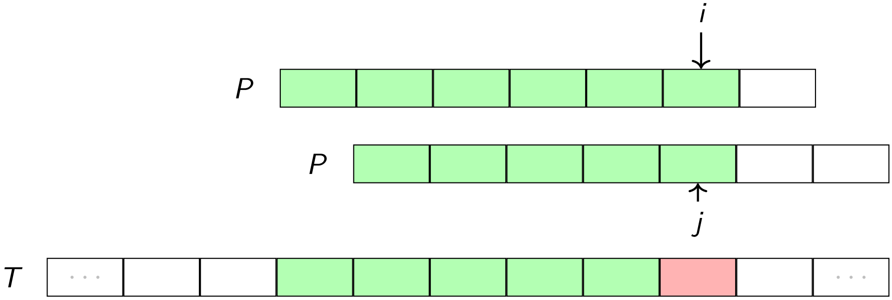
Exercise 10.4

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

b. Prove that $\forall k \in (h[j], j] : \neg(P[0:k-1] = P[i-k:i-1] \wedge P[i] \neq P[k])$. (important point!)

Case 2: $P[i] = P[j]$

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



We may use $h[i] := j$, but it does not meet the requirement $P[i] \neq P[j]$. (Why?)

Let us jump to $h[j]$, which will meet the requirements. (Why?) We 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];                                  // Prepare for the next iteration
    else
        h[i] := h[j];
    i := i + 1;   j := j + 1;
h[|P|] := j;
return h
```

Commentary: Let $prop(i,k) = (P[0:k-1] = P[i-k:i-1] \wedge P[i] \neq P[k])$. The loop invariant at the head of the outer loop is $P[i-j:i-1] = P[0:j-1]$, $\forall k \in (j,i), \neg prop(i,k)$, and $\forall l < j\ prop(h[l], l) \wedge \forall k \in (h[l], l), \neg prop(l,k)$. We prove the correctness by proving the validity of the loop invariant.

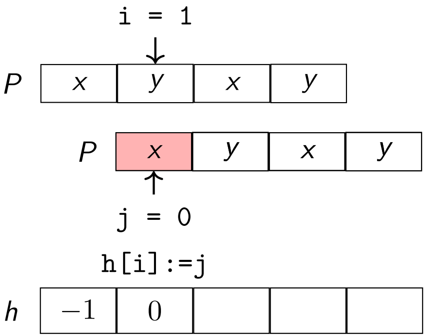
Exercise 10.5

Give proof of correctness of the algorithm.

Example: computing h

Example 10.4

Consider the following pattern P and the first iteration of the outer loop, which is case 1.

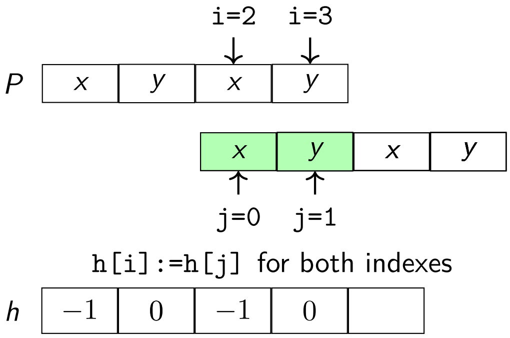


We need to update `j := h[j]`. Therefore, `j = -1`.

Afterwards, we increment both j and i. Therefore, `i = 2;j = 0;`.

Example: computing h (continued) (2)

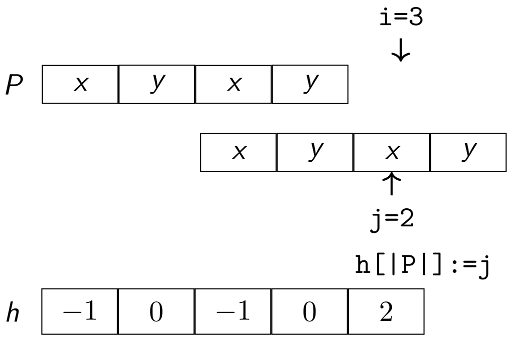
Let us consider the second and third iteration of the outer loop, which are case 2.



After the third iteration, the loop exits since $i \geq |P|$.

Example: computing h (continued) (3)

After the third iteration, the loop terminates and we update $h[|P|]$.



Topic 10.3

Tutorial problems

Exercise: compute h

Exercise 10.6

Compute array h for pattern "babbaabba".

Exercise: version of KMPTable

Exercise 10.7

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 10.8

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?

Topic 10.4

Problems

True or False

Exercise 10.9

Mark the following statements True / False and also provide justification.

1. KMP is $O(n+m)$ for text of size n and pattern of size m.
2. The h array in KMP cannot have 10 as an entry.
3. The h array in KMP cannot have -10 as an entry.

longestPrefixSuffix

Exercise 10.10

Given a string s, our goal is to find the length of the longest proper prefix which is also a suffix(LPS). A proper prefix is a prefix that doesn't include whole string. For example, prefixes of "abc" are "", "a", "ab" and "abc" but proper prefixes are "", "a" and "ab" only. Here is a code with missing parts that computes length. Give code that completes the code.

```
#include <iostream>
#include <string>
using namespace std;
int longestPrefixSuffix(string s) {
    int res = 0;
    for (int len = ___(1)___; len < s.length(); len++) {
      int j = s.length() - ___(2)___;
      bool flag = ___(3)___;
      for (int k = 0; k < len; k++) {
        if ( s[k] != s[___(4)___] ) {
          flag = ___(5)___;
```

End of Lecture 10