# CS213/293 Data Structure and Algorithms 2024

### Lecture 9: Pattern matching

Instructor: Ashutosh Gupta

IITB India

Compile date: 2024-09-13



## Topic 9.1

### Pattern matching problem

#### Pattern matching

Definition 9.1

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

Usage:



DNA sequencing

## Example : Näive approach for pattern matching

#### Example 9.1

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



## 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?) @**()**\$0

Instructor: Ashutosh Gupta

## 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 9.1

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 9.2 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.



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

Exercise 9.2

 $\Theta$ 

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 pre-compute array h such that h[j] = k. Example 9.2



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

Exercise 9.3 a. Show that  $j > h(j) \ge -1$  for each  $j \in [0..|P|)$ b. Show that  $|P| > h(|P|) \ge 0$  if |P| > 0. Is it true if |P| = 0? c. If we drop condition  $P[j] \ne P[k]$ , what may go wrong?

©**()**§0

9

## Knuth–Morris–Pratt algorithm

**Algorithm 9.1:** KMP(string T, string P)

1 assume(|P| > 0); **2** i := 0; i := 0; found  $:= \emptyset;$ 3  $h := \text{KMPTABLE}(\mathsf{P});$ 4 while i < |T| do if P[i] = T[i] then 5  $i := i + 1; \quad i := i + 1;$ 6 if i = |P| then 7 found.insert(i - i): 8 i := h[i];g else 10 i := h[i];11 12

13

 $\Theta(\mathbf{i} \otimes \mathbf{0})$ 

if j < 0 then

 $i := i + 1; \ i := i + 1;$ 

Running time complexity:

Since no. of increments of  $i \leq |T|$ , the line 6 and 13 will execute < |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  $j \ge 0$  (why?), no. of reductions of i < no. of increments of i.
- 3. Since *i* and *i* are always incremented together.

no. of reductions of  $i \leq no$ . of increments of i.

4. no. of reductions of  $i \leq |T|$ .

 $\triangleright$  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.

14 return found

## Example : KMP execution

Example 9.3

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



## Topic 9.2

### How to compute array 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  $k \in (j, i)$ ,  $\neg (P[0: k-1] = P[i-k: i-1] \land P[i] \neq P[k])$ .

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 conservatively.

Commentary	: Due to the assumption, no k larger $j$ is a valid candidate or $h[i]$ .			
0000				
()()()()()()()()()()()()()()()()()()()	CS213/293 Data Structure and Algorithms 2024	Instructor: Ashutosh Gupta	III B India	14

## 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  $j \le 0$ . Exercise 9.4

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

b. Prove that  $\forall k \in (h[j], j] : \neg (P[0:k-1] = P[i-k:i-1] \land P[i] \neq P[k])$ . (important poin ©DS® CS213/293 Data Structure and Algorithms 2024 Instructor: Ashutosh Gupta

## 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 9.2: KMPTABLE(string P)

```
i := 1; i := 0; h[0] := -1;
while i < |P| do
   if P[i] \neq P[i] then
      h[i] := j;
      while j \ge 0 and P[j] \ne P[i] do
        i := h[i];
   else
    | h[i] := h[j];
   i := i + 1; \quad j := j + 1;
h[|P|] := j;
return h
```

// Prepare for the next iteration

**Commentary:** Let  $prop(i, k) = (P[0: k - 1] = P[i - k: i - 1] \land P[i] \neq P[k])$ . 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  $\forall l < j \quad prop(h[l], l) \land \forall k \in (h[l], l), \neg prop(l, k)$ . We prove the correctness by proving the validity of the loop invariant.

#### Exercise 9.5

Give proof of correctness of the algorithm.

000

## Example: computing h

Example 9.4

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;.

©()(S)()

## Example: computing h (cotinued) (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  $i \ge |P|$ .



## Example: computing h (cotinued) (3)

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



## Topic 9.3

#### Tutorial problems



#### Exercise 9.6

Compute array h for pattern "babbaabba".



Exercise: version of  $\mathrm{KMPTABLE}$ 

Exercise 9.7 *Is the following version of* KMPTABLE *correct?* 

```
Algorithm 9.3: KMPTABLEV2(string P)
i := 1; j := 0; h[0] := -1;
while i < |P| do
   h[i] := i;
   while j \ge 0 and P[j] \ne P[i] do
   j := h[j];
                // Moving forward the pattern in minimum steps as in {
m KMP}
  i := i + 1; \quad j := j + 1;
h[|P|] := j;
return h
```

## Exercise: compute h(i)

#### Exercise 9.8

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?



# End of Lecture 9

