String Matching

Deskripsi Persoalan

Persoalan: Diberikan

a. teks (*text*), yaitu (*long*) *string* yang panjangnya *n* karakter

b. *pattern*, yaitu *string* dengan panjang *m* karakter ($m < n$) yang akan dicari di dalam teks.

Carilah lokasi pertama di dalam teks yang bersesuaian dengan *pattern*.

Algoritma *brute force*:

1. Mula-mula *pattern* dicocokkan pada awal teks.

2. Dengan bergerak dari kiri ke kanan, bandingkan setiap karakter di dalam *pattern* dengan karakter yang bersesuaian di dalam teks sampai:

   – semua karakter yang dibandingkan cocok atau sama (pencarian berhasil), atau
   – dijumpai sebuah ketidakcocokan karakter (pencarian belum berhasil)

3. Bila *pattern* belum ditemukan kecocokannya dan teks belum habis, geser *pattern* satu karakter ke kanan dan ulangi langkah 2.

Contoh 1:

Pattern: NOT

Teks: NOBODY NOTICED HIM

```
  NOBODY NOTICED HIM
1 NOT
2  NOT
3   NOT
4    NOT
5     NOT
6      NOT
7       NOT
8        NOT
```

ALGORITHM *BruteForceStringMatch*$(T[0..n-1],\ P[0..m-1])$

```
//Implements brute-force string matching
//Input: An array T[0..n − 1] of n characters representing a text and
//       an array P[0..m − 1] of m characters representing a pattern
//Output: The index of the first character in the text that starts a
//        matching substring or −1 if the search is unsuccessful
for i ← 0 to n − m do
    j ← 0
    while j < m and P[j] = T[i + j] do
        j ← j + 1
    if j = m return i
return −1
```

Horspool's Algorithm

- Determines the sizes of such a shift by looking at the character c of the text that was aligned against the last character of the pattern

- Case 1

  if there are no **c**'s in pattern

  Shift the pattern by entire length

s_0 .... D .... s_{n-1}

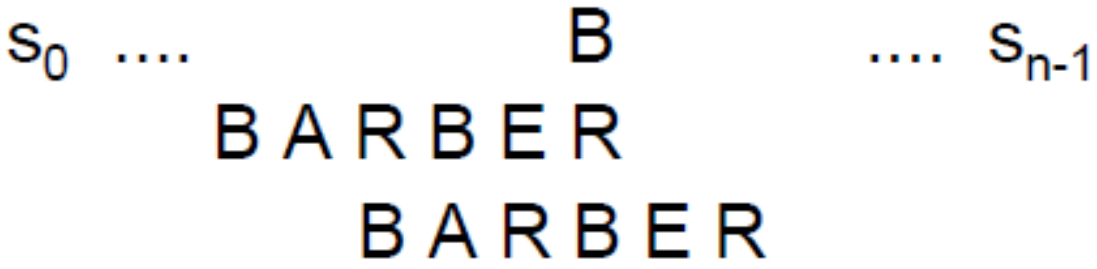
B A R B E R

B A R B E R

- Case 2

  if there are occurences of character **c** in pattern but is not the last one

  Shift should align the rightmost occurence of c in the pattern with the **c** in the text



- Case 3

  if **c** happens to be the last character in the pattern but there are no **c**'s among its other **m**-1 character

  Shift : similar with case 1

s_0 .... M E R .... s_{n-1}

L E A D E R

L E A D E R

- Case 4

  if **c** happens to be the last character in the pattern and there are other **c**'s among its first **m**-1 character

  Shift : similar with case 2

  s_0 .... O R .... s_{n-1}

  R E O R D E R

  R E O R D E R

- **Algorithm**

ShiftTable(P[0..m-1])

// input:Pattern P[0..m-1] and an alphabet of possible character

// output:Table[0..size-1] indexed by the alphabet's characters and filled with shift sizes

Initialize all the elements of table with m

```
For j ← 0 to m-2 do
      Table[P[j]] ← m-1-j
Return Table
```

Matching Algorithm

The steps of Horspool's algorithm

1. Construct the shift table
2. Align the pattern against the beginning of the text

3. Repeat until either matching substring is found or the pattern reach beyond the last character of the text :
   - ❑ starting the last character in the pattern, compare corresponding character in the pattern and text until either all **m** character are matched or mismatching pair encountered
   - ❑ In the latter case, retrieve the entri **t(c)** from the **c**'s columb on the shift table where **c** is the text's character currently aligned against the last character of the pattern
   - ❑ Shift the pattern by **t(c)** characters to the right

```
HorspoolMatching(P[0..m-1],,T[0..n-1])
//input:pattern P[0..m-1] and text T[0..n-1]
//output:index of the left end of the first matching
   substring or -1 if there are no matches
ShiftTable(P[0..m-1]) // generate shift table
i←m-1                    //position of the pattern's right end
While i ≤ n do
  k ← 0
  while k ≤ m-1 and P[m-1-k]=T[i-k] do
       k ← k+1
  if k = m
      return i-m+1
  else  i ← i+table[T[i]]
Return -1
```

Example

Pattern : BARBER

Shift table:

Character c	A	B	C	D	E	F	...	R	...	Z	-
Shift t(c)	4	2	6	6	1	6	6	3	6	6	6

Matching Process

J	I	M		S	A	W		M	E		I	N		A		B	A	R	B	E	R	S	H	O	P
B	A	R	B	E	R																				
				B	A	R	B	E	R																
					B	A	R	B	E	R															
											B	A	R	B	E	R									
													B	A	R	B	E	R							
																B	A	R	B	E	R				