Q5. Boyer-Moore String Search Algorithm

Group:
1. Choe Choon Ho - 1132702963
2. Wong Tiong Kiat - 1132702943

How do we compare two strings?

Text	A	B	C	A	B	C	1	2	3
Pattern	A	B	C						

Did the pattern match the text?

How do we compare strings naturally?

We first look at each character sequentially, then if any of the character pair didn’t match, we stop and say it’s NOT a match, and proceed to the next comparison.

Naive Method (Brute Force)

```
for i <- 0 to text.length-pattern.length do
    j <- 0
    for j to pattern.length-1 do
        if text[i+j] != pattern[j] then
            break
    if j = pattern.length-1 then
            match
```

Visualization of Naive Method

A	B	C	D	E
A	0			

Did the character match? Yes, check next one.

A	B	C	D	E
A	0			

Did the character match? No, move the pattern.

Visualization of Naive Method

A	B	C	D	E
	A	0		

Did the character match? No, move pattern and repeat.

Naive Method Analysis

```
for i <- 0 to n-m do ..... n
    j <- 0 ..... n
    for j to m-1 do ..... n * m
        if text[i+j] != pattern[j] then ..... n * m
            break ..... n
    if j = m-1 then ..... n
            match ..... n
```

O(n * m)

Note:
n is the length of text
m is the length of pattern

Boyer-Moore Method

Before comparing, construct a mismatch table based on the pattern.

The mismatch table will tell how much should we move the pattern instead of a constant ONE box ahead in the Naive Method.

Then perform comparing but Boyer Moore compares starting from the end of the pattern. It looks at the last character of the text where that character is equal to the position of the last character in the pattern, obtain the skip value from the mismatch table to move.

Visualization of Boyer-Moore Method

A	B	C	D	E
A	0			

0 didn’t match with B, instead of moving right one box, we move equal size to the pattern because we will definitely NOT find a match even moving one by one.

A	B	C	D	E
		A	0	

Visualization of Boyer-Moore Method

A	A	B	0	1	2	3	4	5	A	A	A	0	0	1
A	B	0												

A	A	B	0	1	2	3	4	5	A	A	A	0	0	1
	A	B	0											

This will result in a match. What Boyer Moore is trying to do is, at the current text character, B, does my pattern contain a B? If it does, align the most right B before 0 from the pattern to the text's B. *If it doesn't, it would move the length of the pattern, 3.*

Mismatch Table

What’s wrong with the previous idea?

If we need to iterate and find our most right character from the current character, it makes NO differences compared to the Naive Method.

Therefore, Boyer Moore uses a mismatch table. The table will store the each character skip value so that the skip value could be obtained in O(1) time.

Boyer-Moore Method Algorithm

Part I, Construct Mismatch Table

```
for i <- 0 to m-1 do
    table[pattern[i]] <- m-i-1
table[others] <- m
```

Boyer-Moore Method Algorithm

Part II, Comparing

```
i <- m-1
while i < n do
        t <- m-1
        for j <- i to i-m+1 do
                if text[j] != pattern[t] then
                        break
                t <- t - 1
        if t = -1 then
                match
        i <- i + table[text[i]]
```

Visualization of Boyer-Moore Method

Key	1	A	*
Value	1	3	3

Mismatch Table

A	1	2	1	1	A	2	3	4	1
1	1	A							

So, we find 2 in the table, not found, take the default, move pattern ahead by 3.

A	1	2	1	1	A	2	3	4	1
			1	1	A				

*A is a match, we check again, 1 is a match, and again 1 is a match. Therefore, **MATCH**.*