# String Matching Boyer-Moore

A.J. Craig COMP261, 2017

#### **Boyer-Moore**

- Match backwards, starting from the last character in the string
  - If you mismatch on a character c in the text, then reposition so that c is aligned with the last occurrence of c in the string
  - If there is no c in the string, move the whole string along
- Other rules: the "good suffix rule" and "Galil rule"
  - Won't cover it, look it up
  - We will look at a simplified version of Boyer-Moore
- How to figure out how far ahead to jump?
  - Store the last index at which every character occurs in the string
  - Then jump ahead by max(1, s table.get(text[t+s]))

#### **Boyer-Moore: Computing the Table**

- In an alphabet of L characters, need to know where in the string each occurs
- If a character doesn't occur, store -1
- Solution 1: array
  - Store an array of L characters; O(L) space, O(1) lookup
  - Table stored in contiguous memory; low overhead
- Solution 2: map
  - Store a map from characters to index
  - Have a default value for characters not in the string
  - O(M) space, O(1) lookup
    - Assuming constant-time hashing function for characters
  - Table may not be stored in contiguous memory; bigger overhead

## **Boyer-Moore Table**

String: ababca

а	b	С	*

## **Boyer-Moore Table**

String: ababca

а	b	С	*
5	3	4	-1

### **Boyer-Moore: Constructing the Table**

```
makeTable(String string):
   ASCII SIZE = 256 // number of ASCII characters
   charTable = new int[ASCII SIZE]
   for i = 0 to ASCII SIZE - 1 do
      charTable[i] = -1
   for j = 0 to |string| - 1 do
      charTable[string[j]] = j
   return charTable
```

### **Boyer-Moore: Constructing the Table**

- If storing in a map, use a "default map"
- Can be done in Java by overriding an implementation of Map, e.g. by using an anonymous inner class

```
new HashMap<Character, Integer>(){
    @Override
    public Integer get(Object o) {
        Integer result = super.get(o);
        if (result == null) return -1; // key not in map else return result;
    }
};
```

## **Boyer-Moore**

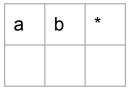
```
BoyerMoore(text, string):
    s = |string| - 1 // begin matching from end of string
    t = 0
    table = makeTable(string)
    while t < |text| do
        if string[s] = text[t + s] then
            if s = 0 then return t
            else s--
        else
            amt2skip = max(1, s - table.get(text[t+s]))
            t += amt2skip
            S = |string| - 1
    return -1;
```

## Example

Text: ababcabcababa

String: ababa

Table:



## Example

Text: ababcabcababa

String: ababa

Table:

а	b	*
4	3	-1

String: ababa

Table:

а	b	*		
4	3	-1		

1st iteration (t = 0, s = 4)

а	b	а	b	С	а	b	С	а	b	а	b	а
а	b	а	b	а								

Begin matching the string against the text

Table:

а	b	*
4	3	-1

1st iteration (t = 0, s = 4)

String: ababa

а	b	а	b	С	а	b	С	а	b	а	b	а
а	b	а	b	а								

Mismatch on c

Skip ahead in the text by max(1, s - table.get(text[t+s]))

$$= \max(1, 4 - (-1)) = 5$$

Now 
$$t = 0 + 5 = 5$$
,  $s = 4$ 

String: ababa

Table:

а	b	*
4	3	-1

2nd iteration (t = 5, s = 4)

а	b	а	b	С	а	b	С	а	b	а	b	а
					а	b	а	b	а			

Begin matching characters

String: ababa

Table:

а	b	*
4	3	-1

2nd iteration (t = 5, s = 4)

а	b	а	b	С	а	b	С	а	b	а	b	а
					а	b	а	b	а			

Mismatch on b

Skip ahead in the text by max(1, s - table.get(text[t+s]))

$$= \max(1, 4 - 3) = 1$$

Now 
$$t = 5 + 1 = 6$$
,  $s = 4$ 

String: ababa

Table:

а	b	*
4	3	-1

3rd iteration (t = 6, s = 4)

а	b	а	b	С	а	b	С	а	b	а	b	а
						а	b	а	b	а		

String: ababa

Table:

а	b	*
4	3	-1

3rd iteration (t = 6, s = 1)

а	b	а	b	С	а	b	С	а	b	а	b	а
						а	b	а	b	а		

Mismatch on c

Skip ahead in the text by max(1, s - table.get(text[t+s]))

$$= \max(1, 1 - (-1)) = 2$$

Now 
$$t = 6 + 2$$
,  $s = 4$ 

String: ababa

Table:

а	b	*		
4	3	-1		

4th iteration (t = 8, s = 4)

а	b	а	b	С	а	b	С	а	b	а	b	а
								а	b	а	b	а

Begin matching characters

String: ababa

Table:

а	b	*
4	3	-1

4th iteration (t = 8, s = 0)

а	b	а	b	С	а	b	С	а	b	а	b	а
								а	b	а	b	а

Successfully match the string, return t=8

### **Boyer-Moore Complexity**

- Space
  - $\circ$  O(M) if table as map, O(L) if table as array
- Time
  - Constant number of operations per iteration of the loop
  - Worst case: you match every character except the first one and increment t by 1
    - Example: match 'baaa' in 'aaaaaa'
  - $\circ$  Doing M comparisons at O(N) positions = O(N\*M)
  - But in practice you will skip over a lot of positions
    - Especially if there are characters not in your string
  - Improvements: good suffix rule and Galil rules