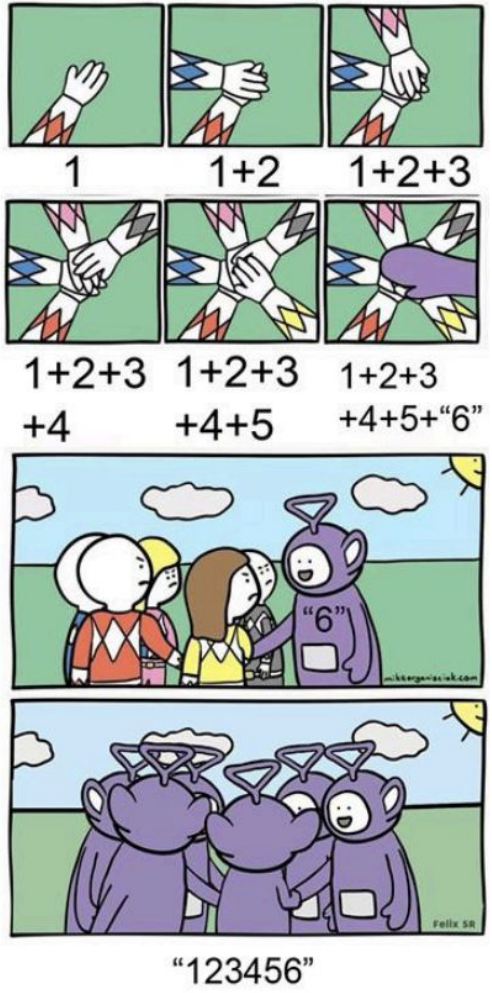

CS 1332R
WEEK 12

Galil Rule

Rabin-Karp

Pattern Matching Efficiencies

← I don't get this

ANNOUNCEMENTS

-

REFRESHER: Pattern Matching Algorithms

➔ **PROBLEM GOAL**: Finding a pattern (smaller string of characters) in a text (longer string of characters) - the same as your Command/Control F function.

Typically...

`n = text length` `m = pattern length`

`i = text index` `j = pattern index`

- **All Occurrences**: return a list of all indexes where the pattern occurs in the text
  - We must iterate through the entire text of length n.

- **Single Occurrence:** find the first index where the pattern occurs in the text
  - We stop at the first occurrence of the pattern.

REFRESHER: Boyer-Moore

INTUITION: If a character in the text is not present in the pattern, we can move our pattern completely past this character in the text.

Algorithm

1. Start `i = 0, j = m - 1`. **We compare from right to left within the pattern.** Decrement j.
2. If mismatch…
   a. `lot.get(text[i + j]) == -1` → shift pattern completely past the text,
   b. `lot.get(text[i + j]) != -1` → shift pattern *forward* to last occurrence of the text character in the pattern
      i. If `lot.get(text[i + j]) > j`, just shift the pattern over by 1 (increment `i`).

```
procedure BoyerMoore(text, pattern):
  lastTable is pattern's last occurrence table
  m is pattern's length, n is text's length
  i starts at 0
  while (i <= n - m):
    j starts at m - 1
    while (j >= 0 and text[i + j] matches pattern[j]):
      decrement j
    end while
    if (j == -1):
      // match found at i                    match
      move i forward
    else:
      shift is the lastTable index for text[i + j]
      if (shift < j):
        add j - shift to i                   mismatch
      else:
        move i forward
      end if
    end if
  end while
end procedure
```

Boyer-Moore with Galil Rule

INTUITION: After a match is found, shift by the period.

- Requires a last occurrence table AND a failure table
- **`period = m - failureTable[m - 1]`**
  - This is simply the shift we perform in KMP - `shift = j - f[j - 1]`
- Keep track of a **`lowBound`** for `j` that alternates between **`0`** (after a mismatch) and **`m - period`** (after a match)
  - Remember, the KMP shift allows us to not have to compare the characters in the pattern before the shift on the next iteration.

```
procedure BoyerMooreGalilRule(text, pattern):
  create lastTable and failureTable for pattern
  m is pattern's length, n is text's length
  periodicity is the last failureTable entry subtracted from m
  i and lowBound start at 0
  while (i <= n - m):
    j starts at m - 1
    while (j >= lowBound and text[i + j] matches pattern[j]):
      move j backward
    end while
    if (j < lowBound):
      // match found at i
      add periodicity to i
      set lowBound to periodicity subtracted from m
    else:
      shift is the lastTable index for text[i + j]
      if (shift < j):
        move i forward by j - shift
      else:
        move i forward
      end if
      set lowBound to 0
    end if
  end while
end procedure
```

match

Failure Table

Char:	b	u	r	b	u
Value:	0	0	0	1	2

Last Occurrence Table

Key	b	u	r	*
Value	3	4	2	-1

period = 3

Galil Rule: Practice

Answer:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
text	l	a	b	u	r	b	u	j	a	b	u	b	u	r	u	r	b	u	r	b	u
pattern	b	u	r	b	u																

EXERCISE: At each iteration, note why the shift was made.

Failure Table

Char:	b	u	r	b	u
Value:	0	0	0	1	2

Last Occurrence Table

Key	b	u	r	*
Value	3	4	2	-1

period = 3

Galil Rule: Practice

Answer:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
	l	a	b	u	r	b	u	j	a	b	u	b	u	r	u	r	b	u	r	b	u
1	b	u	r	b	u																
2			b	u	r	b	u														
3																					
4																					
5																					
6																					
7																					
8																					
9																					
10																					

Failure Table

Char:	b	u	r	b	u
Value:	0	0	0	1	2

Last Occurrence Table

Key	b	u	r	*
Value	3	4	2	-1

period = 3

Galil Rule: Practice

Answer: 2

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
	l	a	b	u	r	b	u	j	a	b	u	b	u	r	u	r	b	u	r	b	u
1	b	u	r	b	u																
2			b	u	r	b	u														
3						b	u	r	b	u											
4																					
5																					
6																					
7																					
8																					
9																					
10																					

Failure Table

Char:	b	u	r	b	u
Value:	0	0	0	1	2

Last Occurrence Table

Key	b	u	r	*
Value	3	4	2	-1

period = 3

Galil Rule: Practice

Answer: 2

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
	l	a	b	u	r	b	u	j	a	b	u	b	u	r	u	r	b	u	r	b	u
1	b	u	r	b	u																
2			b	u	r	b	u														
3						b	u	r	b	u											
4							b	u	r	b	u										
5																					
6																					
7																					
8																					
9																					
10																					

Failure Table

Char:	b	u	r	b	u
Value:	0	0	0	1	2

Last Occurrence Table

Key	b	u	r	*
Value	3	4	2	-1

period = 3

Galil Rule: Practice

Answer: 2

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
	l	a	b	u	r	b	u	j	a	b	u	b	u	r	u	r	b	u	r	b	u
1	b	u	r	b	u																
2			b	u	r	b	u														
3						b	u	r	b	u											
4							b	u	r	b	u										
5										b	u	r	b	u							
6																					
7																					
8																					
9																					
10																					

Failure Table

Char:	b	u	r	b	u
Value:	0	0	0	1	2

Last Occurrence Table

Key	b	u	r	*
Value	3	4	2	-1

period = 3

Galil Rule: Practice

Answer: 2

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
	l	a	b	u	r	b	u	j	a	b	u	b	u	r	u	r	b	u	r	b	u
1	b	u	r	b	u																
2			b	u	r	b	u														
3						b	u	r	b	u											
4							b	u	r	b	u										
5										b	u	r	b	u							
6												b	u	r	b	u					
7																					
8																					
9																					
10																					

Failure Table

Char:	b	u	r	b	u
Value:	0	0	0	1	2

Last Occurrence Table

Key	b	u	r	*
Value	3	4	2	-1

period = 3

Galil Rule: Practice

Answer: 2

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
	l	a	b	u	r	b	u	j	a	b	u	b	u	r	u	r	b	u	r	b	u
1	b	u	r	b	u																
2			b	u	r	b	u														
3						b	u	r	b	u											
4							b	u	r	b	u										
5										b	u	r	b	u							
6												b	u	r	b	u					
7														b	u	r	b	u			
8																					
9																					
10																					

Failure Table

Char:	b	u	r	b	u
Value:	0	0	0	1	2

Last Occurrence Table

Key	b	u	r	*
Value	3	4	2	-1

period = 3

Galil Rule: Practice

Answer: 2

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
	l	a	b	u	r	b	u	j	a	b	u	b	u	r	u	r	b	u	r	b	u
1	b	u	r	b	u																
2			b	u	r	b	u														
3						b	u	r	b	u											
4							b	u	r	b	u										
5										b	u	r	b	u							
6												b	u	r	b	u					
7														b	u	r	b	u			
8															b	u	r	b	u		
9																					
10																					

Failure Table

Char:	b	u	r	b	u
Value:	0	0	0	1	2

Last Occurrence Table

Key	b	u	r	*
Value	3	4	2	-1

period = 3

Galil Rule: Practice

Answer: 2

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
	l	a	b	u	r	b	u	j	a	b	u	b	u	r	u	r	b	u	r	b	u
1	b	u	r	b	u																
2			b	u	r	b	u														
3						b	u	r	b	u											
4							b	u	r	b	u										
5										b	u	r	b	u							
6												b	u	r	b	u					
7														b	u	r	b	u			
8															b	u	r	b	u		
9																	b	u	r	b	u
10																					

Failure Table

Char:	b	u	r	b	u
Value:	0	0	0	1	2

Last Occurrence Table

Key	b	u	r	*
Value	3	4	2	-1

period = 3

Galil Rule: Practice

Answer: 2, 16

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
	l	a	b	u	r	b	u	j	a	b	u	b	u	r	u	r	b	u	r	b	u
1	b	u	r	b	u																
2			b	u	r	b	u														
3						b	u	r	b	u											
4							b	u	r	b	u										
5										b	u	r	b	u							
6												b	u	r	b	u					
7														b	u	r	b	u			
8															b	u	r	b	u		
9																	b	u	r	b	u
10																					

Galil Rule: Efficiencies

Can Boyer Moore with the Galil Rule ever degenerate into regular Boyer-Moore?

Yes, if there are no matches.

The Galil Rule only optimizes Boyer Moore's behavior after a match.

What if k == 1?
Isn't that just shifting by one after a match, which is what we do in regular Boyer-Moore?

If k == 1, our lowBound = m - 1. This means on the next iteration, we would only ever perform one comparison.

Galil Rule: Efficiencies

Best	Average	Worst
Single: O(m) *All: O(n/m + m)*	*very text/pattern dependent*	*O(mn)*

Single: O(m) - occurrence at i = 0

text = baaacdegfbaaa

pattern = baaa

All: O(n/m + m) - text character compared with last character of pattern is never in the pattern

text = aacaacaacaac

pattern = aab

Worst case with no matches: O(mn)

text = aaaaaaaaaa

pattern = caa

Worst case with matches: O(m + n)

text = caacaacaa

pattern = caa

We run in to the true worst case less frequently with the Galil Rule.

INTUITION: When there are a lot of matches, the Galil Rule linearizes Boyer Moore's worst case.

REFRESHER: Brute Force

INTUITION:

No optimizations, most basic search.

1. Line up index 0 of the pattern with index 0 of the text.
2. Compare each character of the pattern with each character in the text.
3. MISMATCH → shift pattern right by 1. Repeat step 1.

PSEUDOCODE:

```
procedure BruteForce(text, pattern):
  n is text's length, m is pattern's length
  for (i from 0 to n - m):
    j starts at 0
    while (j < m and pattern[j] matches text[i + j]):
      move j forward
    end while
    if (j == m):
      // match found at i
    end if
  end for
end procedure
```

Rabin-Karp

INTUITION: Brute force but with hashing.

- Pre-processing: the initial hashes of the pattern and the first m characters of the text
- Before comparing the individual characters of the pattern and text, check that the pattern hash equals the text hash
- Always shift i by 1.
- "Roll" the text hash on every iteration with the following formula:

```
textHash =
(textHash - text[i] * BASE^m-1) * BASE + text[i + m]
```

```
procedure RabinKarp(text, pattern):
  m is pattern's length, n is text's length
  patternHash is a hash of the pattern
  textHash is a hash of the first m chars in the text
  i starts at 0
  while (i <= n - m):
    if (patternHash equals textHash):
      j starts at 0
      while (j < m and text[i + j] matches pattern[j]):
        move j forward
      end while
      if (j is m):
        // match found at i
      end if
    end if
    if (i < n - m):
      roll textHash forward one
    end if
    move i forward
  end while
end procedure
```

Rabin-Karp: Computing the Hash

`patternHash = pattern[0] + pattern[1] * BASE^1 + ... + pattern[m-1] * BASE^m-1`

We must calculate the hash from "back to front" to avoid using Math.pow().

***HOMEWORK TIP:** You must calculate the initial pattern hash, initial text hash, and BASE^m-1 all in one for loop.*

0	1	2	3
a	b	c	d

Red: character about to be chopped off
Green: character about to be added
Yellow: untouched – not leading or trailing character

Instruction	textHash
Old hash	$a * BASE^2 + b * BASE^1 + c * BASE^0$
Shave off oldest (index = 0) character	$(a * BASE^2 + b * BASE^1 + c * BASE^0) - a * BASE^2$
Increase each term's power by BASE	$(b * BASE^1 + c * BASE^0) * BASE$
Tack on newest (index = 3) character	$(b * BASE^2 + c * BASE^1) + d * BASE^0$
Updated hash!	$b * BASE^2 + c * BASE^1 + d * BASE^0$

```
textHash = (textHash - text[i] * BASE^m-1) * BASE + text[i + m]
```

Why must we pre-compute BASE^m-1?

to ensure that rolling the hash is O(1)

Rabin-Karp: Example

Green: match
Red: mismatch
Yellow: hashes not equal, no comparison

0	1	2	3	4	5	6	7	8	9	10	
a	b	d	a	a	b	d	c	a	c	c	Text Hash
a	c	c									abd → 4
	a	c	c								bda → 4
		a	c	c							daa → 3
			a	c	c						aab → 1
				a	c	c					abd → 4
					a	c	c				bdc → 6
						a	c	c			dca → 5
							a	c	c		cac → 4
								a	c	c	acc → 4

Rabin Karp: Efficiencies

Best	Average	Worst
Single: O(m) *All: O(m + n)*	*O(m + n)*	*O(mn)*

Single: O(m) - occurrence at i = 0

text = baaacdegfbaaa

pattern = baaa

All: O(m + n) - same reason as for average case

O(m): calculate the initial hashes

O(n): iterating through the text

If we have a good hash, it is likely that hashes won't be equal if the pattern and text are not equal. Therefore, a lot of comparisons are skipped.

If we have a bad hash, Rabin-Karp degenerates into Brute-Force.

Ex: h(any character) = 1.

text = aaaaaaaaaa

pattern = caa

Rolling Hash	*O(1)*

Pattern Matching: Efficiencies

Brute Force

Scenario	Best	Best Ex	Worst	Worst Ex
No Occurrences	*O(n)*	P: baa T: aaaaaaa	*O(mn)*	P: aab T: aaaaaaa
Single Occurrences	*O(m)*	P:aaa T: aaaaaaa	*O(mn)*	P: aab T: aaaaaaab
All Occurrences	*O(n)*	P: baa T: baaaaabaa	*O(mn)*	P: aaab T: aaaaaaab

Rabin Karp

Scenario	Best	Average	Worst
Rolling Hash	*O(1)*	*O(1)*	*O(1)*
Single Occurrence	*O(m)*	*O(m + n)*	*O(mn)*
All Occurrences	*O(m + n)*	*O(m + n)*	*O(mn)*

Boyer Moore

Scenario	Best	Worst
LOT (preprocess)	*O(m)*	*O(m)*
No Occurrences	*O(m + n/m)*	*O(mn)*
Single Occurrence	*O(m)*	*O(mn)*
All Occurrences	*O(m + n/m)*	*O(mn)*

KMP

Scenario	Best	Worst
FailureTable (preprocess)	*O(m)*	*O(m)*
No Occurrences	*O(m + n)*	*O(m + n)*
Single Occurrence	*O(m)*	*O(m + n)*
All Occurrences	*O(m + n)*	*O(m + n)*

LEETCODE PROBLEMS

28. Find Index of First Occurrence in a String

Any questions?

Name
Office Hours
Contact

Name
Office Hours
Contact

Let us know if there is anything specific you want out of recitation!