String Matching

Knuth-Morris-Pratt

A.J. Craig
COMP261, 2017

Problem

- Inputs:
  - Some input text of length N
  - A string of length M
- Output:
  - First position in text where the string occurs (or -1 if it doesn't occur)
  - OR, all positions in the text where the string occurs
- Example:
  - text = "hamadan", string = "ada"
  - text = "cellardoor", string = "lard"

Naive Algorithm

- Look at every position in the text, try to match S

```
stringSearch(text, string):
    for t = 0 to N-1 loop:
        s = 0
        while s < M loop:
            if text[t + s] != string[s] then break
            else s++
        if s = m then return t
     return -1
```

Naive Algorithm

- Cost?
  - Outer loop runs N times, inner loop runs M times
  - We perform O(N * M) comparisons
  - No space needed, O(1) spac

Text: ababcabcababa
Prefix: ababa

First iteration
(t = 0)

a	b	a	b	c	a	b	c	a	b	a	b	a
a	b	a	b	a								

Second iteration
(t = 1)

a	b	a	b	c	a	b	c	a	b	a	b	a
	a	b	a	b	a							

Third iteration
(t = 2)

a	b	a	b	c	a	b	c	a	b	a	b	a
		a	b	a	b	a						

etc.

KMP Algorithm

- KMP = Knuth-Morris-Pratt
- Observation: if we get a mismatch, we don't always have to start matching from the beginning of the string again
  - If part of the string matched so far contains its own beginning, then we can start matching from that part
  - Otherwise, we can skip ahead by the number of characters matched
- Use a prefix table to tell you how far ahead to skip based on what character you mismatched

KMP Algorithm

- Input: text, string, table where entries tell you how far to jump

```
KMP(text, string, table):
    t = 0, s = 0 // indices into text and string
    while t + s < |text| loop
        if string[s] = text[t+s] then    // matched another character
            s++
            if s = |string| then return t
        else if table[s] = -1 then       // mismatched with no overlap
            t = t + s + 1, s = 0
        else                             // mismatched with overlap
            t = t + s - table[s]
            s = table[s]
    return -1
```

Computing the Table

- If you mismatch at position s, jump ahead by s - table[s]
- table[s] tells you how much of the beginning of the string is matched so far by the characters before position s
  - e.g., if table[s] = 2 then table[s-2] and table[s-1] match the beginning of the string
- table[0] is always -1
  - Different implementations of the algorithm have table[0] = 0

Computing the Table

a	b	a	b	a
-1				

First entry is always -1

Computing the Table

a	b	a	b	a
-1	0			

We have not yet matched any characters, and ‘b’ is not a prefix

Computing the Table

a	b	a	b	a
-1	0	0		

We have not yet matched any characters, ‘a’ is a prefix

Computing the Table

a	b	a	b	a
-1	0	0	1	

We have matched ‘a’, and ‘ab’ is a prefix

Computing the Table

a	b	a	b	a
-1	0	0	1	2

We have matched ‘ab’ (and ‘abc’ is a prefix)

KMP: Constructing the Table

```
makeTable(string):
    M = |string|
    table = new int[M]
    table[0] = -1 // first position is a special case
    matchedSoFar = 0
    for i = 1 to M - 1 do // start iterating from 1, not 0
        table[i] = matchedSoFar
        if string[i] = string[matchedSoFar]
            matchedSoFar++
        else
            matchedSoFar = 0
    return table
```

Text: ababcabcababa
String: ababa

Table:

a	b	a	b	a
-1	0	0	1	2

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

a	b	a	b	c	a	b	c	a	b	a	b	a
a	b	a	b	a								

Begin matching the string against the text

Text: ababcabcababa
String: ababa

Table:

a	b	a	b	a
-1	0	0	1	2

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

a	b	a	b	c	a	b	c	a	b	a	b	a
a	b	a	b	a								

We mismatch at s = 4
Increment t by s - table[s]. Now, t = 2
Set s to be table[s]. Now, s = 2

Text: ababcabcababa
String: ababa

Table:

a	b	a	b	a
-1	0	0	1	2

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

a	b	a	b	c	a	b	c	a	b	a	b	a
		a	b	a	b	a						

Begin matching the string against the text.

Text: ababcabcababa
String: ababa

Table:

a	b	a	b	a
-1	0	0	1	2

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

a	b	a	b	c	a	b	c	a	b	a	b	a
		a	b	a	b	a						

We mismatch at s = 2
Increment t by s - table[s]. Now, t = 4
Set s to be table[s]. Now, s = 0

Text: ababcabcababa
String: ababa

Table:

a	b	a	b	a
-1	0	0	1	2

3rd iteration
(t = 4, s = 0)

a	b	a	b	c	a	b	c	a	b	a	b	a
				a	b	a	b	a				

Begin matching the string against the text

Text: ababcabcababa
String: ababa

Table:

a	b	a	b	a
-1	0	0	1	2

3rd iteration
(t = 4, s = 0)

a	b	a	b	c	a	b	c	a	b	a	b	a
				a	b	a	b	a				

We mismatch at s = 0, and table[0] = -1 (no overlap)
Increment t by s + 1, now t = 5
Set s = 0

Text: ababcabcababa
String: ababa

Table:

a	b	a	b	a
-1	0	0	1	2

4th iteration
(t = 5, s = 0)

a	b	a	b	c	a	b	c	a	b	a	b	a
					a	b	a	b	a			

Begin matching the string against the text

Text: ababcabcababa
String: ababa

Table:

a	b	a	b	a
-1	0	0	1	2

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

a	b	a	b	c	a	b	c	a	b	a	b	a
					a	b	a	b	a			

Mismatch at s = 2
Increment t by s - table[s]. Now, t = 4
Set s to be table[s]. Now, s = 0

Text: ababcabcababa
String: ababa

Table:

a	b	a	b	a
-1	0	0	1	2

5th iteration
(t = 7, s = 0)

a	b	a	b	c	a	b	c	a	b	a	b	a
							a	b	a	b	a	

Begin matching the string against the text

Text: ababcabcababa
String: ababa

Table:

a	b	a	b	a
-1	0	0	1	2

5th iteration
(t = 7, s = 0)

a	b	a	b	c	a	b	c	a	b	a	b	a
							a	b	a	b	a	

We mismatch at s = 0, and table[0] = -1 (no overlap)
Increment t by s + 1, now t = 8
Set s = 0

Text: ababcabcababa
String: ababa

Table:

a	b	a	b	a
-1	0	0	1	2

6th iteration
(t = 8, s = 0)

a	b	a	b	c	a	b	c	a	b	a	b	a
								a	b	a	b	a

Begin matching the string against the text

Text: ababcabcababa
String: ababa

Table:

a	b	a	b	a
-1	0	0	1	2

6th iteration
(t = 8, s = 4)

a	b	a	b	c	a	b	c	a	b	a	b	a
								a	b	a	b	a

Successfully match the string against the text

More Tables...

m	i	m	i	c

What are the entries?

More Tables...

m	i	m	i	c
-1	0	0	1	2

More Tables...

a	a	r	o	n

What are the entries?

More Tables...

a	a	r	o	n
-1	0	1	0	0

Analysing KMP

- Finding the string in the text is O(N) in the length of the text
  - Constant num. operations per iteration. How many iterations?
  - Note that the algorithm terminates when $t + s < N$
  - Want to analyse how each of the three branches in the loop affects t and $t + s$
  - Let s_i, t_i be the values of s and t on iteration i
  - Let delta $\Delta_k(t) = t_{i+1} - t_i$ be the change in t on branch k
  - Let delta $\Delta_k(t + s) = (t_{i+1} + s_{i+1}) - (t_i + s_i)$ be the change in $t + s$ on branch k

Analysing KMP

- First branch:
  - $t_{i+1} = t_i,\ s_{i+1} = s_i + 1$
  - $\Delta_1(t) = 0,\ \Delta_1(t + s) = 1$
- Second branch:
  - $t_{i+1} = t_i + s_i + 1,\ s_{i+1} = 0$
  - $\Delta_2(t) = s_i + 1 > 1,\ \Delta_2(t + s) = 1$
- Third branch:
  - $t_{i+1} = t_i + s_i - \text{table}[s_i],\ s_{i+1} = \text{table}[s_i]$
  - Note $s_i > \text{table}[s_i]$, so $\Delta_3(t) > 0$
  - $\Delta_3(t + s) = 0$

Analysing KMP

- Note that for any branch k, either $\Delta_k(t) > 0$ or $\Delta_k(t+s) > 0$
- If you enter branch k at least m + n times, then depending on which Δ_k is greater than zero, at least one of the following is true:
  - $t \geq \Delta_k(t) * (M + N) \geq M + N$
  - $t + s \geq \Delta_k(t + s) * (M + N) \geq M + N$
- In either case $t + s \geq N$, so you will not execute the loop again
- Therefore, if you enter a branch more than M + N times the loop will terminate
- On each iteration, the loop goes through one of the three branches
- So the loop iterates at most 3*(M + N) = O(M + N) times

KMP Complexity

- Finding the string in the text is O(M + N) in the length of the text
- Computing the table is O(M) space and O(M) time
- Total cost
  - O(M + M + N) = O(M + N) time
  - O(M) space
- When is it not that great?
  - Strings we're working with are not often prefixes of one another
  - Is it common for an English word to be a prefix of another?
  - Is it common for DNA strings (ACGT) to be a prefix of another?