

CSE 3318 Notes 18: KMP String Search

(Last updated 8/20/20 1:25 PM)

CLRS, 32.4

SIMPLE RESCANNING

Pattern - m symbols

Text - n symbols

Example: Pattern: A B A C

Text: A B A B A B A C A B A B A C A B A

0 1 2 3

0 1 2 3

0

0 1 2 3

0

0 1 2 3

0

0 1

0

0 1 2 3

0

0 1 2 3

0

0 1

Worst-Case: $\Theta(mn)$

[illegible]

KNUTH-MORRIS-PRATT FAILURE LINK CONSTRUCTION

Fail link - seeks to reuse largest possible *suffix* before present position that matches a *prefix* of pattern.

Style 1: Choose maximum value of k with $0 \leq k < j$ such that

$$\text{for } 0 < i \leq k: \text{pattern}[k - i] == \text{pattern}[j - i]$$

Now set $\text{fail}[j] = k$

Style 2: Choose maximum value of k with $0 \leq k < j$ such that

$$\text{pattern}[k] != \text{pattern}[j], \text{ and}$$

$$\text{for } 0 < i \leq k: \text{pattern}[k - i] == \text{pattern}[j - i]$$

Now set $\text{fail}[j] = k$

Direct application of these definitions could take $\Theta(m^3)$ time!

Either style fail link table may be constructed in $\Theta(m)$ time.

For style 1:

Suppose fail links 0 through j have already been set and $\text{fail}[j] = k$.

If, in addition, $\text{pattern}[j] == \text{pattern}[k]$ then

Set $\text{fail}[j+1] = k+1$

But, what if $\text{pattern}[j] != \text{pattern}[k]$?

Move k back to $\text{fail}[k]$ and recheck for pattern match

0	b	-1	11	b	6
1	a	0	12	a	4
2	b	0	13	b	5
3	b	1	14	a	6
4	a	1	15	b	7
5	b	2	16	b	8
6	a	3	17	a	9
7	b	2	18	b	10
8	b	3	19	a	11
9	a	4	20	b	?
10	b	5			

For style 2:

Suppose fail links 0 through j have already been set and k is the *required maximum value* that was used when setting $\text{fail}[j]$.

If, in addition, $\text{pattern}[j] == \text{pattern}[k]$ then

If $\text{pattern}[j+1] != \text{pattern}[k+1]$

Set $\text{fail}[j+1] = k+1$

Else

Set $\text{fail}[j+1] = \text{fail}[k+1]$

But, what if $\text{pattern}[j] != \text{pattern}[k]$?

Move k back to $\text{fail}[k]$ and recheck for pattern match

0 b -1	11 b 6
1 a 0	12 a 0
2 b -1	13 b -1
3 b 1	14 a 3
4 a 0	15 b -1
5 b -1	16 b 1
6 a 3	17 a 0
7 b -1	18 b -1
8 b 1	19 a 11
9 a 0	20 b ?
10 b -1	

	Fail 1	Pattern	Fail 2
0		A	
1		A	
2		B	
3		A	
4		A	
5		B	
6		A	
7		A	
8		A	
9		B	

	Fail 1	Pattern	Fail 2
0		A	
1		B	
2		A	
3		A	
4		B	
5		A	
6		B	
7		A	
8		A	
9		B	
10		A	
11		A	
12		B	

	Fail 1	Pattern	Fail 2
0	-1	A	-1
1	0	A	-1
2	1	B	1
3	0	A	-1
4	1	A	-1
5	2	B	1
6	3	A	-1
7	4	A	-1
8	5	A	5
9	2	B	1

	Fail 1	Pattern	Fail 2
0	-1	A	-1
1	0	B	0
2	0	A	-1
3	1	A	1
4	1	B	0
5	2	A	-1
6	3	B	3
7	2	A	-1
8	3	A	1
9	4	B	0
10	5	A	-1
11	6	A	6
12	4	B	0

KMP.c:

```

/* Determine all possible occurrences of pattern string in text
string using KMP technique*/
#include <stdio.h>
#include <string.h>

#define MAXPATLEN 80
#define EOS 0

void preprocpatl(pat,next)
/* Produces slower failure links, but capable of continuing after
match */
char *pat;
int next[];
{
    int i,j;
    i=0;
    j=next[0]= -1;
    do
    {
        if (j==(-1)||pat[i]==pat[j])
        {
            i++;
            j++;
            next[i]=j;
        }
        else
            j=next[j];
    } while (pat[i]!=EOS);
    printf("Fail link table 1\n");
    for (i=0;pat[i]!=EOS;i++)
        printf("%d %c %d\n",i,pat[i],next[i]);
}

```

```

void preprocpat2(pat,next)
/* Produces faster failure links, but INcapable of continuing after
   match */
char *pat;
int next[];
{
    int i,j;
    i=0;
    j=next[0]=-1;
    do
    {
        if (j==(-1) || pat[i]==pat[j])
        {
            i++;
            j++;
            next[i]=(pat[j]==pat[i]) ? next[j] : j;
        }
        else
            j=next[j];
    } while (pat[i]!=EOS);
    printf("Fail link table 2\n");
    for (i=0;pat[i]!=EOS;i++)
        printf("%d %c %d\n",i,pat[i],next[i]);
}

void match(text1,pat)
char *text1,*pat;
{
    int next1[MAXPATLEN],next2[MAXPATLEN],i,j;
    char *text;
    char printSymbol=' ';

    if (*pat==EOS) return;
    preprocpat1(pat,next1);
    preprocpat2(pat,next2);

    printf("%s\n",text1);
    text=text1;
    for (j=0; *text!=EOS;)
        if (j==(-1) || pat[j]== *text)
        {
            if (pat[j+1]==EOS)
                printSymbol='^';
            if (pat[j+1]==EOS && *(text+1)!=EOS)
                j=next1[j]; /*restart*/
            else
            {
                printf("%c",printSymbol);
                printSymbol=' ';
                text++;
                j++;
            }
        }
        else
            j=next2[j];
    printf("\n");
}

main()
{
    char pat[80],text[80];

    printf("Enter text & pattern\n");
    /*variable text is string 1, variable pat is string 2*/
    while (scanf("%s %s",text,pat)!=EOF)
    {
        match(text,pat);
        printf("Enter text & pattern\n");
    }
}

```

KMP COMPLEXITY

m pattern symbols, n text symbols

Matcher - every comparison is preceded by a movement of one or both pointers

Text pointer always moves forward $\Rightarrow \Theta(n)$

Pattern pointer

Moves forward once for each text symbol $\Rightarrow \Theta(n)$

Total number of backward movements $\leq$ total number of forward movements $\Rightarrow O(n)$

Failure table construction

Lead pointer always moves forward $\Rightarrow \Theta(m)$

Prefix pointer

Moves forward once for each pattern symbol $\Rightarrow \Theta(m)$

Total number of backward movements $\leq$ total number of forward movements $\Rightarrow O(m)$

Overall: $\Theta(m + n)$

Aside: Worst-case number of comparisons (fail 2 links) when processing a text symbol is bounded by $1 + 1.44 \lg m$.

Fibonacci strings may be used as difficult cases:

$F_1 = \text{"a"}$ $F_2 = \text{"b"}$ $F_n = F_{n-1}F_{n-2}$

$F_3 = \text{"ba"}$

$F_4 = \text{"bab"}$

$F_5 = \text{"babba"}$

$F_6 = \text{"babbabab"}$

$F_7 = \text{"babbababbabba"}$