COMP10002 Workshop Week 8

Assignment 1 should be done

	ASS1: **Progress?** Marking rubric – some specific topics? Q&A?
	? • string searching, KMP & BMH • do 7.16 • Q&A on ASS1 &other exercises ? • ASS1 • Q&A
LAB	Finish ass1 during this workshop or today? Submissions close at **6pm this Friday**. Done ass1? Implement 7.16, then exercises from lec06
LMS	**Workshops**: • ***Your most important goal this week is to complete Assignment 1.*** • Submission is **via the LMS Assignment 1** page, and **not** by grok

Your assignment1:

A- All done

B- only stages 1+2 fully done

C- only stage 1 fully done

D- none of the above

String Searching

Input:

A (normally long) text T[0..n-1]. Example: T= “abab yxy aababcb”, with n=20

A (normally short) text pattern P[0..m-1]. Example: P= “ababc”, m=4.

Output:

index i such that T[i..i+m-1] == P[0..m-1], or NOTFOUND

How, in general:

- shift the pattern along the text until find a match

Output for the above example:

- position 10 (in T)

Measuring Efficiency:

- **how many (character) comparisons?**
- how many (pattern) shifts/alignments?

String matching: KMP & BMH

Both algorithms:

- start with aligning P with the start of T
- repeatedly shift P to the right as far as possible by comparing P with T character-by-character

a	b	a	b	y	a	y	b		a	a	b	a	b	c	a
a	b	a	b	c											

- *The number of positions to shift totally depends on P and hence is pre-computed at the start (with complexity O(m))*

But

- KMP compare pattern (with text) from *left to right*
- BMH compare pattern (with text) from *right to left*

How to run KMP *manually*

Compare T with P from left to right

current char in T

a	b	a	b	y	a	x	b		a	a	b	a	b	c	a

a	b	a	b	c

5 comp until mismatch;

When mismatch happens at position **i** in P, examine **only** the *matched part* of P (ie. P[0..i-1]) to decide the shift.

mismatched at position **i==4** in P, find **s**= length of the largest common proper *prefix* and *suffix*

a	b	a	b	

prefixes "a", "ab", "aba", suffixes "b", "ab", "bab"

longest common prefix and suffix: "ab", suffix "ab", **s** = 2,

case A: **s>0**

➔ shift P to the right **i – s** == 2 positions, then continue comparing from **y** (the current char in T)

that is, *shift the pattern so that its prefix aligns with the matched suffix of the text.*

a	b	a	b	y	a	x	b		a	a	b	a	b	c	a
		a	b	a	b	c									

Special cases:

- case B: **s==0** (no common prefix-suffix) ➔ align (the first character of) P to the current char in T
- case C: if empty matched part (ie. mismatch at the 1st position of P) ➔ shift 1

How to run KMP *manually*

a	b	a	b	y	a	x	b		a	a	b	a	b	c	a

a b a b c — 5 comp ; case A : shift 2 for matching

a b a b c — 1 comp (with **y**); case B: shift 2 to align P with c

a b a b c — 1 comp (with **y**); case C: shift 1

a b a b c — 2 comp(a-a, x-b); case B: shift 1

a b a b c — 1 comp (with **x**); case C: shift 1

a b a b c — 1 comp (with **b**); case C : shift 1

a b a b c — 1 comp (with **' '**); case C: shift 1

a b a b c — 2 comp(a-a, a-b); case B: shift 1

a b a b c — 5 comp; found

a	b	a	b	y	a	x	b		a	a	b	a	b	c	a

Notes on the KMP algorithm

The KMP algorithm has 2 phases:

- **Phase 1**: Pre-computing the Array F[] where F[i] = longest prefix-suffix length if mismatch happens at position i of the pattern P. Note that F[0]= -1 to fit case C.

  Example:

Position:	0	1	2	3	4
Pattern:	a	b	a	b	c
F[]=	-1	0	0	1	2

- **Phase 2**: Searching the Text:
  - Compare the pattern P with the text T from left to right
  - When a mismatch occurs at position i in the pattern P, shift P to the right by i-F[i] and resume the comparison from the current character in the text T

    This lets us avoid re-checking characters already known to match.

  - Overall complexity: **O(n)** (≈ **2n** character comparisons in the worst case).

Understand the BMH algorithm

First align the pattern P with the text T.

Loop:

Start from the *right end* of the pattern, and work to the left, comparing P[i] with the corresponding character in T

if mismatched:

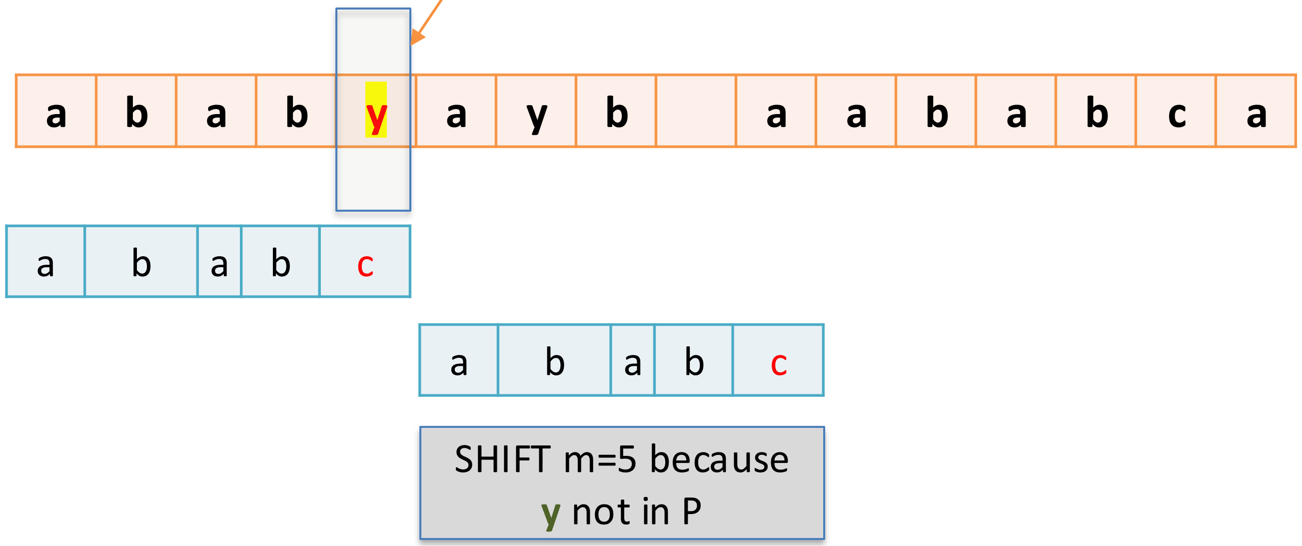
shift pattern P to the right

else

FOUND

return NOTFOUND

- The shifts depends on the rightmost-examined T, here y
- Shift pattern P to the right at least 1 position and until that y matches with a character in P (here: shift 5 because no match is possible)



How to run BMH *manually*

a b a b y a y b   a a b a b c a

a b a b c

1 comp until mismatch
no matter where mismatch happens, the shift is totally decided by the rightmost examined char of T y

Shift P until having the first of P's character with that y
(here, no match found)

SHIFT m=5
because y not in P

a b a b y a y b   a a b a b c a

a b a b c

1 comps, SHIFT 2 = until a matches a

a b a b y a y b   a a b a b c a

a b a b c

1 comps, SHIFT 1 = until b matches b

a b a b y a y b   a a b a b c a

a b a b c

1 comps, SHIFT 2 = until a matches a

a b a b c

5 comps, all matched, found

Notes on the BMH algorithm

The BMH algorithm has 2 phases:

- **Phase 1:** Pre-computing the Shift Array S[] *for the whole alphabet*, where S[c]= number of positions to shift when mismatch happens at character c

  Example Pattern:

a	b	a	b	c

  shift array S[]: S['a']= 2, S['b']= 1, S[any-other-symbol]= 5

- **Phase 2:** Searching the Text:
  1. Align the pattern P with the start of text T.
  2. Compare P with the corresponding part of T from right to left
  3. If all characters match, return the match position.
  4. Otherwise, let c be the character in T aligned with the last character of P.
  5. Shift P to the right by the amount of S[c].
  6. Repeat until the end of T is reached; if no match is found, return NOTFOUND.

- Complexity O(mn) but practically very fast

String Searching Review

Algorithms & Efficiency:

- Naïve: brute force (when mismatch, shift the pattern only 1 position)

  complexity O(nm)

  max number of character comparisons = (n-m+1)*m
- BMH: also O(mn) but practically fast especially when m<<n
- KMP: O(n+m)

Remember

- KMP and BMH gain from long shifts, but
- KMP compares pattern (with text) from *left to right,*
- BMH compares pattern (with text) from *right to left.*

😀 Why?

String matching: KMP & BMH

- `KMP` compare pattern (with text) from *left to right.*
- `BMH` compare pattern (with text) from *right to left,* 😀 Why?

because the letters `K,M,P` *are in alphabetic order* 😀

but the letters `B,M,H` *are not* 😀

Assignment 1 or 7.16 + exercises from lec06.pdf

7.16 (word frequencies): Given a program that outputs all distinct words in an input text. Modify the program so that it also outputs the frequency of each distinct word.

Exercise 1 Write a function `is_subsequence(char *s1, char *s2)` that returns 1 if the characters in `s1` appear within `s2` in the same order as they appear in `s1`. For example, `is_subsequence("bee", "abbreviate")` should be 1, whereas `is_subsequence("bee", "acerbate")` should be 0.

Exercise 2 Ditto arguments, but determining whether every occurrence of a character in `s1` also appears in `s2`, and 0 otherwise. For example, ", `is_subset("bee "rebel")` should be 1, whereas `is_subset("bee", "brake")` should be 0.

Exercise 3 Write a function `is_anagram(char *s1, char *s2)` that returns 1 if the two strings contain the same letters, possibly in a different order, and 0 otherwise, ignoring whitespace characters, and ignoring case. For example, `is_anagram("Algorithms", "Glamor Hits")` should return 1.

Exercise 4 Write a function `next_perm(char *s)` that rearranges the characters in a string argument and generates the lexicographically next permutation of the same letters. For example, if the string `s` is initially `"51432"`, then when the function returns s should be `"52134"`.

Exercise 5 If the two strings are of length `n` (and, if there are two, `m`), what is the asymptotic performance of your answers to Exercises 1–4?

Additional Slides

BMH Algorithm Review

The task: Seaching for a pattern `P` (such as “HELL” that has length m=5) in a text `T` (such as “SHE SELLS SEA SHELLS”, having length n=20).

The Algorithm:

need to do a pre-processing of the pattern before performing the search

normally, $|P| \ll |T|$, this step doesn’t affect the overall complexity

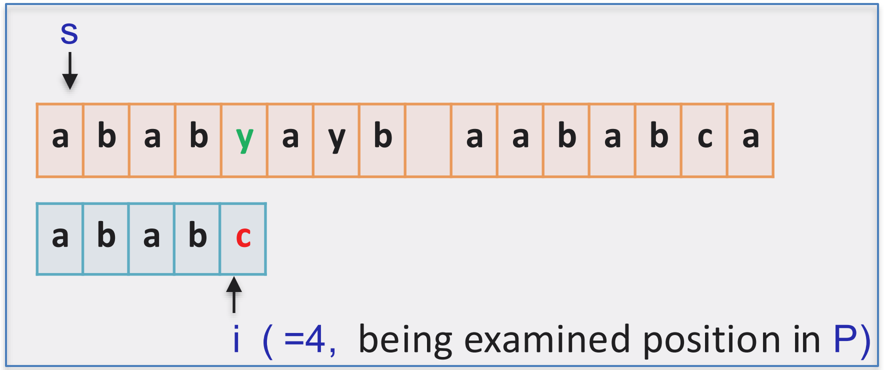
BMH: pre-processing

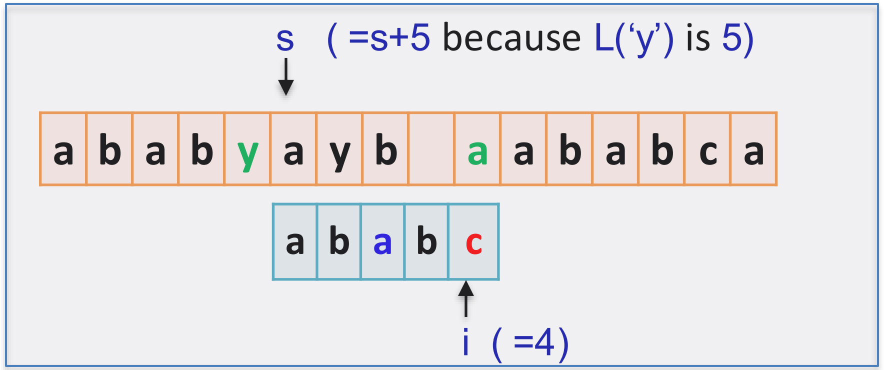
Pre-processing*:* build `L[x]` for every possible character `x`, ie. for all x from the alphabet (in the lecture, the alphabet has σ symbols), by:

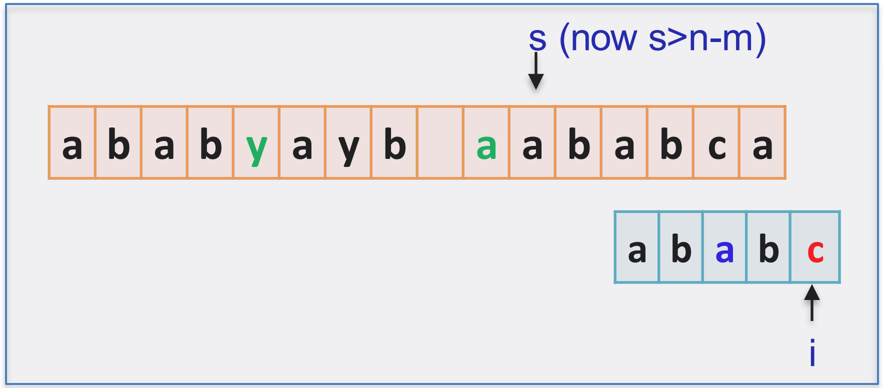
1. first, set `L[x]= m` for all `x`, then
2. for each character `x` in `P`, *except for the last one*: `L(x)` = distance from the last appearance of `x` to the end of `P`

$$
\begin{array}{l}
\textbf{for } v \leftarrow 0 \textbf{ to } \sigma - 1 \\
\qquad L[v] \leftarrow m \\
\textbf{for } i \leftarrow 0 \textbf{ to } m - 2 \\
\qquad L[P[i]] = m - i - 1
\end{array}
$$

BMH - searching








```
s=0;        // current start of P in T
i= m-1;     // current position in P
c= T[s+m-1];   // the pilot character

while (s+i not passing the end of T){
   if (mismatched at P[i]) {
      s = s + L[c];
      i = m-1;
      c= T[s+m-1];
   } else {
      if (i==0)
         return s;
      else
         i--;
   }
}
return NOTFOUND;
```

$s, i \leftarrow 0, m-1$

while $s \leq n-m$

  if $T[s+i] \neq P[i]$

    $s, i \leftarrow s + L[T[s+m-1]], m-1$

  else if $i = 0$

    return s

  else

    $i \leftarrow i - 1$

return *not_found*

Marking Rubric: The importance of Style & Structure

Stage	Presentation	Structure	Execution	max accumulated mark
1	+6	+4	+2	12
2	+1	+1	+2	16
3	+1	+1	+2	20
all	8	6	6	

Failed code could even get 14

Absolutely correct program could get only 6

Correct and good Stage 1+2 alone could get 16

Example of using diff

```
[user@sahara ~]$ ./myass1 < test1.txt > test1-myout.txt

[user@sahara ~]$ diff test1-myout.txt test1-out.txt
4,7c4,7
< read 5 candidates and 8 votes
< voter 7 preference...
<     rank  1: Allyx
<     rank  2: Chenge
---
> read 5 candidates and 7 votes
> voter 7 preferences...
>     rank  1: Cheng
>     rank  2: Allyx
13a14,77
> round 1...
>     Allyx           :   2 votes,  28.6%
>     Breyn            :   1 votes,  14.3%
```

not matched

line starting with < is from the left file (ie. our output)
here: our line is not identical to the expected

line starting with > is from the right file (ie. expected output)
here: we missed 1 blank line in our output

not found

NO corresponding lines in the left (our output)

Case Study & Ex 7.16 – Understanding The Task

Design and implement a program that reads text from stdin, and writes a list of the distinct words that appear, together with their frequencies.

Sample texts:

```
A cat in a hat!
+-abc 10e12 e 1abc #e#abc.abcdefghijklm=xyz
```

Input = ?

- How to get the input text?

Output= ?

- How to store output, which data structure?
- And how to produce output?

Assumptions/limits:

- What’s a *word*?
- Other assumptions?

Case Study & Ex 7.16 – Alistair's getword

```
int getword(char W[], int limit) {
  int c, len=0;
  /* first, skip over any non alphabetics */
  while ((c=getchar()) != EOF && !isalpha(c))
      /* input:  12+34 aWord ???  : skips  12+34    /
  }
  if (c==EOF)  return EOF;

  /* ok, first character of next word has been found */
  W[len++] = c;
  while (len<limit && (c=getchar())!=EOF && isalpha(c)) {
    /* input:  12+34 aWord ???  :  aWord should be the first word */
    W[len++] = c;
  }

  /* now close off the string */
  W[len] = '\0' ;   // W is the string aWord
  return 0;
}
```

Alistair's words.c

```
#define MAXCHARS 10
  /* Max chars per word */
#define MAXWORDS 1000
  /* Max distinct words */


typedef char word_t
            [MAXCHARS+1];
  /* word_t word; now is
     equivalent to
  char word [MAXCHARS+1];
  */

int getword(word_t W,
            int limit);

#include "getword.c"


int
main(int argc,
     char *argv[]) {
  word_t one_word, all_words[MAXWORDS];
  int numdistinct=0, totwords=0, i, found;

  while (getword(one_word, MAXCHARS) != EOF) {
    totwords = totwords+1;
    /* linear search in array of previous words...*/
    found = 0;
    for (i=0; i<numdistinct && !found; i++) {
      found = (strcmp(one_word, all_words[i]) == 0);
    }
    if (!found && numdistinct<MAXWORDS) {
       strcpy(all_words[numdistinct], one_word);
       numdistinct ++;
    }
    /* NB - program silently discards words after
       MAXWORDS distinct ones have been found */
  }

  printf("%d words read\n", totwords);
  for (i=0; i<numdistinct; i++) {
    printf("word #%d is \"%s\"\n", i, all_words[i]);
  }
  return 0;
}
```

Program arguments

Write a program sum that accept two numbers and print out their sum. Example of execution:

```
$ ./sum 12 5
12.00 + 5.00 = 17.00
```

```
?: int main(int argc, char *argv[])
```