§10.2 Rabin\_Karp算法（找文本T中所有和P相同的子串）

一．思想：Rabin\_Karp算法的思想是在朴素算法的基础上，用函数值来比较子串和。

二．函数值

设字母表为，。我们把字符串表示成d进制数

也可以表示成

的函数值可定义为

，其中q为一个尽可能大的素数。

由模运算的性质，一个表达式的值取模，可以把模运算作用于其中任意子表达式，最后运算结果相同。(对于不同的字符串最后的函数值有可能相同)

用类似以上的方法，可求的函数值。串和串的比较可化为函数值与的比较。

当时，我们将比较和，而的函数值可按如下方法计算：

。

三．算法：

PROCEDURE RK\_Matcher (T, P, d, q);（文本，子串，字母表，模数）

BEGIN

1. n := length(T);
2. m := length(P)；
3. h := ;
4. p := 0;
5. := 0;
6. FOR := 1 TO m DO

BEGIN

1. := (d);
2. := (d);

END;

1. FOR s := 1 TO nm+1 DO

BEGIN

1. IF THEN
2. IF THEN
3. WRITE(“Pattern occurs at”, s);
4. IF THEN
5. ;

END

END;

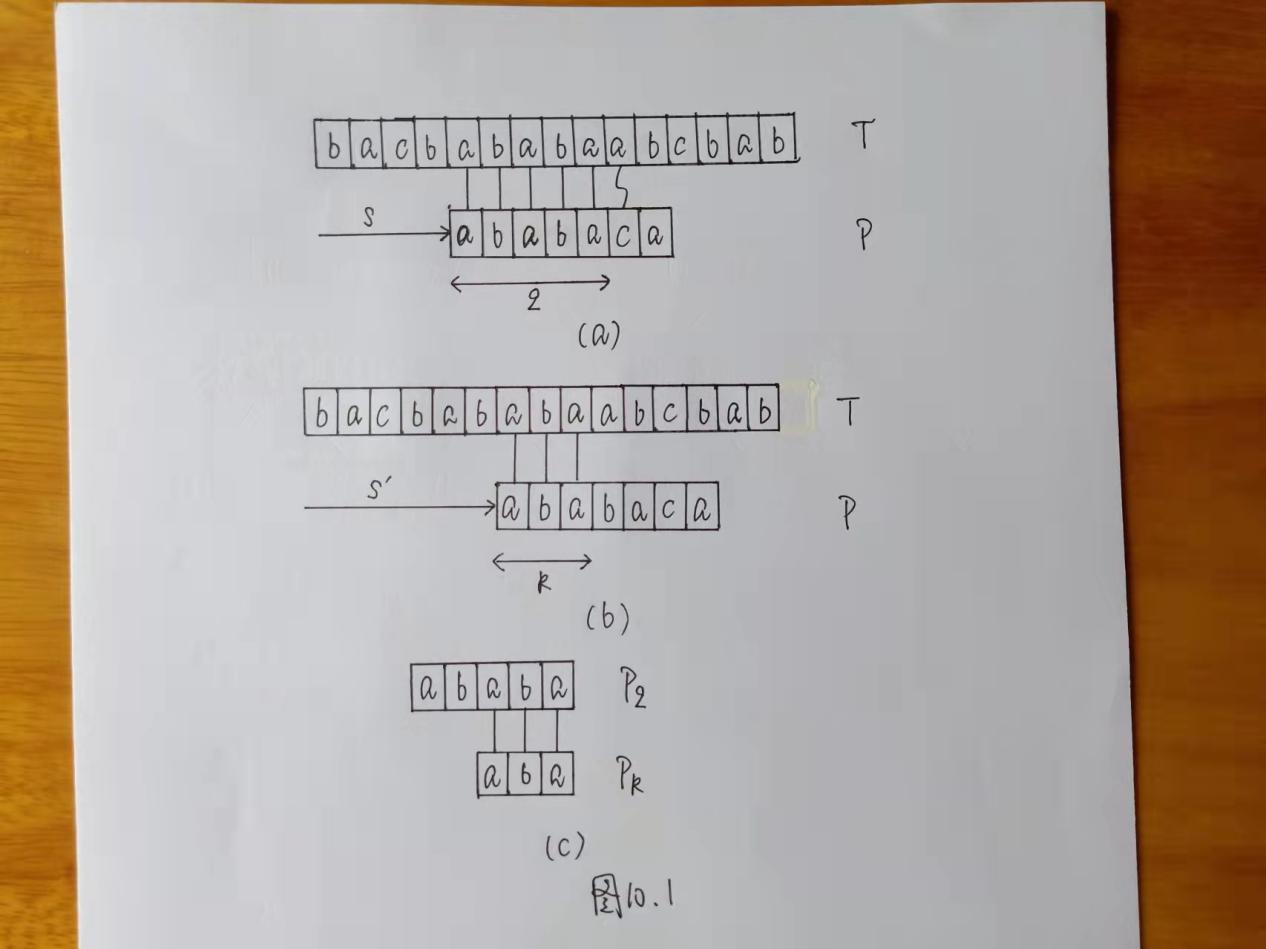
注意不同的字符串可能有相同的函数值，故后，我们还要判断是否等于。但由于是充分大的素数，故函数值的冲突是很少的。

Rabin\_Karp算法在最坏情况下的时间复杂度仍是, （每一次的之都相同但字串不同）但它的平均时间复杂度为。

§10.3 Knuth\_Morris\_Pratt算法

\*在朴素串匹配算法中，有些比较是不必要的，利用已经匹配的部分子串所获得的信息，可以避免一些不必要的匹配。(见图10.1)

如果已经匹配的字符中，前k个字符和后k个字符相同，那么直接向后移m-k位（k尽可能大）

 图10.1

1. 模式的前缀函数

假设模式中前个字符已经匹配了子串，我们可以避免一些不必要的比较，直接算出(且尽可能小)，使得

（P的前k个字符和文本从s’开始的k个字符相同，并且第k+1个位置是之前匹配不成功过的位置）

其中，。

就是在位置s匹配不成功时的下一个应该进行比较的位置。

设是P的前个字符组成的前缀。表示与的后个字符匹配。

例：(见图10.1)

计算的问题就化为找最大的，使得。则。为此，我们定义模式前缀函数：

。

二．算法：

PROCEDURE KMP\_Matcher (T, P)

BEGIN

1. n := length(T);
2. m := length(P);
3. := Compute\_Prefix\_Function (P);
4. := 0;
5. FOR := 1 TO DO

BEGIN

1. WHILE () AND () DO

7. := ;

8. IF THEN

9. := ;

10. IF THEN

BEGIN

WRITE(“Pattern occurs at”, );

:= ;

END

END

END;

PROCEDURE Compute\_Prefix\_Function (P);

BEGIN

1. m := length(P);
2. := 0;
3. := 0;
4. FOR := 2 TO m DO

BEGIN

1. WHILE () AND () DO

6. := ;

7. IF THEN

8. := ;

9. := ;

END;

10. RETURN();

END;

Knuth\_Morris\_Pratt算法在最坏情况下的运行时间仍是。例如：在串aaaaaaaaaab中找子串aaab。但是当模式P中很少自我匹配的子串时，该算法运行得很快，是时间的。

§10.4 Boyer\_Moore算法

一．算法：

PROCEDURE BM\_Matcher (T, P, );

BEGIN

1. n := length(T);
2. m := length(P);
3. := Compute\_Last\_Occurence\_Function (P, m, );
4. := Compute\_Good\_Suffix\_Function (P, m);
5. s := 1;
6. WHILE s DO

BEGIN

1. := ;
2. WHILE () AND () DO
3. := ;
4. IF THEN

BEGIN

1. WRITE(“Pattern occurs at”, s);
2. := ;（移动位数）

END

1. ELSE := ;

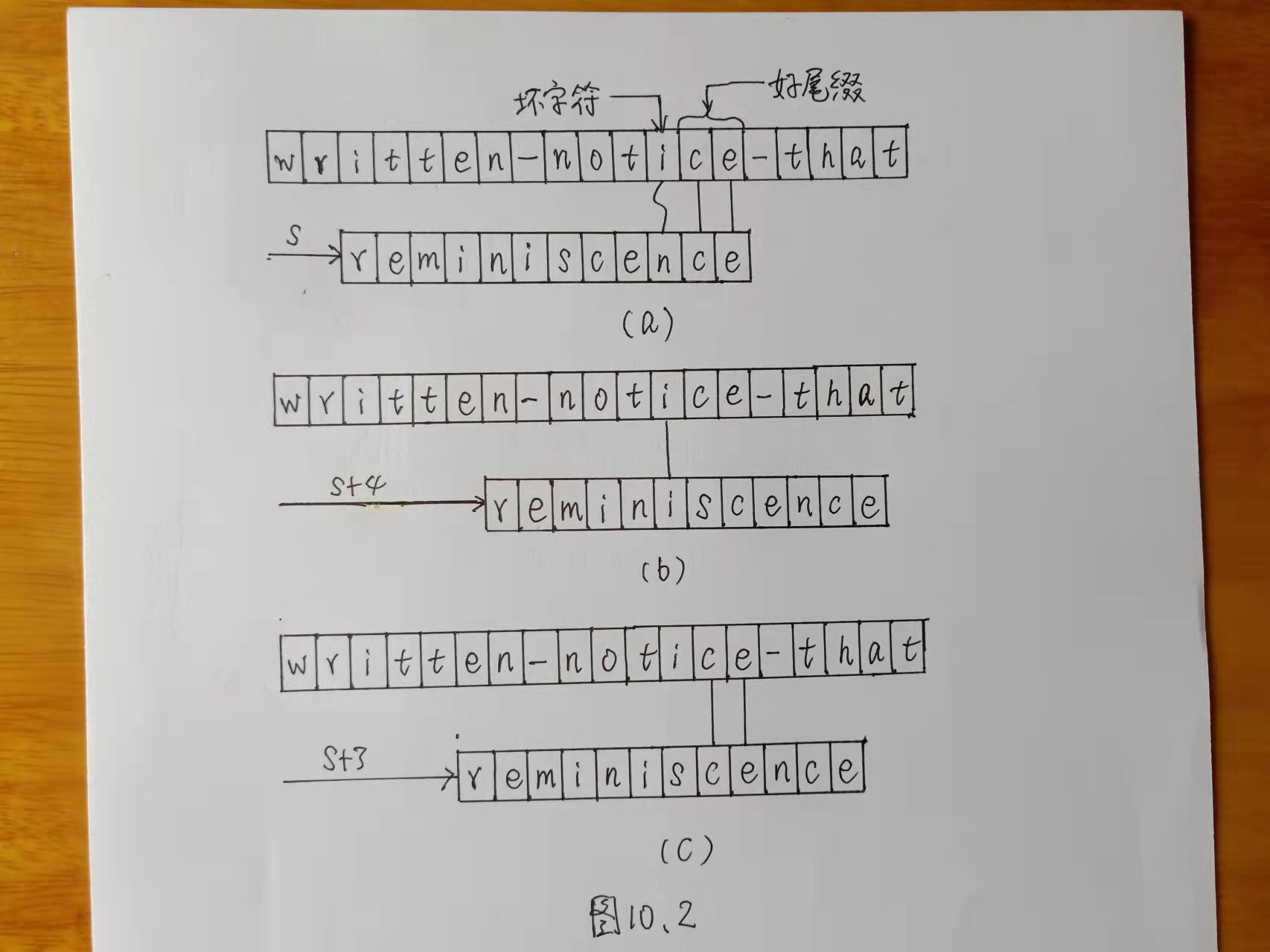
END

END;

Boyer\_Moore算法比较文本和模式时采用从后向前比较的方式。它采用“坏字符”和“好尾缀”的启发式技术来修改s的值。

二．“坏字符”和“好尾缀”启发式技术

例：

图10.2

算法：

PROCEDURE Compute\_Last\_Occurence\_Function (P, m, );

BEGIN

1. FOR each character DO
2. := ;
3. FOR := 1 TO m DO
4. := ;
5. RETURN();

END;

PROCEDURE Compute\_Good\_Suffix\_Function (P, m);

BEGIN

1. := Compute\_Prefix\_Function (P);
2. := reverse (P);
3. := Compute\_Prefix\_Function ();
4. FOR := 0 TO DO
5. := ;
6. FOR := 1 TO DO

BEGIN

1. := ;
2. IF THEN
3. ;

END;

1. RETURN();

END;

三．算法的时间复杂性

在最坏情况下，Compute\_Last\_Occurence\_Function的时间复杂度为, Compute\_Good\_Suffix\_Function的时间复杂度为。而整个算法的时间复杂度为。然而当较大时， Boyer\_Moore算法是目前已知的运行速度最快的算法。该算法是实际应用中常用的算法。

作业23：

1. 计算模式ababbabbababbababbabb的前缀函数。假定字母表 {a, b}。
2. 给定模式和字母表，计算函数和。