《算法设计与分析》上机报告

姓名：	廖洲洲	学号：	PB17081504	日期：	2019.12.3
上机题目：	最长公共子序列问题和调研报告				

实验环境：
CPU：Intel Core i7-8550U; 内存:8G ; 操作系统：Win 10 ; 软件平台: JetBrains CLion ;

一、算法设计与分析：

题目一: 求解两个字符串 s1,s2 的最长公共子序列。 s1，s2 均是随机生成的字符(大、小写字母和数字，共 62 种不同字符)。n,m 的规模分别为 2^6,2^8。

(一)算法步骤

1. LCS 最优解的结构特征
   定义 X 的 i^{th} 前缀：Xi=(x1,x2,…,xi)，i=1~m
   设序列 X=(x1,x2,…,xm)和 Y=(y1,y2,…,yn)，Z=(z1,z2,…,zk)
   是 X 和 Y 的任意一个 LCS，则

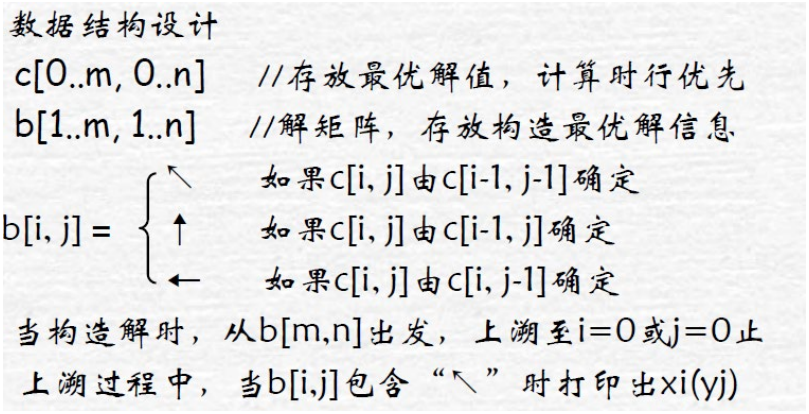
   (1)若 $x_m=y_n$, ==> $z_k=x_m=y_n$ 且 Z_{k-1} 是 X_{m-1} 和 Y_{n-1} 的一个LCS;
   (2)若 $x_m<>y_n$ 且 $z_k<>x_m$, ==> Z 是 X_{m-1} 和 Y 的一个LCS;
   (3)若 $x_m<>y_n$ 且 $z_k<>y_n$, ==> Z 是 X 和 Y_{n-1} 的一个LCS;

2. 子问题的递归解
   将 X 和 Y 的 LCS 分解为：

   (1)if $x_m=y_n$ then　　//解一个子问题
   　找 X_{m-1} 和 Y_{n-1} 的LCS;
   (2)if $x_m<>y_n$ then　　//解二个子问题
   　找 X_{m-1} 和 Y 的LCS 和 找 X 和 Y_{n-1} 的LCS;
   　取两者中的最大的;
   - c[i,j]定义为 X_i 和 Y_j 的LCS长度, i=0~m, j=0~n;

$$c[i,j]=\begin{cases}0 & i=0\ or\ j=0\\ c[i-1,j-1]+1 & i,j>0\ and\ x_i=y_j\\ \max\{c[i,j-1],c[i-1,j]\} & i,j>0\ and\ x_i<>y_j\end{cases}$$

3. 计算最优解值



题目二：调研求解最长公共子串算法（KMP，QuickSearch，Karp-Rabin）。
并利用 K-R 算法，求解字符串匹配问题。文本串 T 的长度为 n，对应的模式串 P 的长度为 m，字符串均是随机生成的字符 (大、小写字母和数字，共 62 种不同字符)。n,m 的规模为（2^8,8）（2^11,16）(2^14,32)。

(一)Karp-Rabin 算法

1. 简介

   Rabin-Karp 算法需要对字符串和模式进行预处理，其预处理时间为 O (m) ，在最坏情况下的运行时间为 O ((n-m+1) m) ，但基于某种假设，它的平均情况下的运行时间还是比较好的。为了便于说明，假设 ∑ = { 0，1，2.....9 }，这样每个字符都是一个十进制数字。（对于更一般的情况，可以假设每个字符都是基数为 d 的表示法中的一个数字，d = | ∑ | ）可以用一个长度为 k 的十进制数来表示由 k 个连续字符组成的字符串。因此，字符串 31415 就对应于十进制数 31415 。

   已知一个模式 P[1.. m]，设 p 表示该模式所对应的十进制数的值（如模式 P = "31415" ，数值 p = 31415）。对于给定的文本 T [1.. n]，用 ts 来表示其长度为 m 的子字符串 T [s+1.. s+m] （s = 0，1，.. n-m）相对应的十进制数的值。ts = p 当且仅当 T [s+1.. s+m] = P[1.. m] ，因此 s 是有效位移当且仅当 ts = p 。

2. 预处理

   应用霍纳法则在 O (m) 的时间内计算 p 的值：

   p = P[m] + 10(P[m-1] + 10 (P[m-2] + .. + 10(P[2] + P[1]) ...))

   类似的，也可以在 O (m) 时间内根据 T[1.. m] 计算出 t0 的值。

   为了在 O (n - m) 的时间内计算出剩余的值 t1，t2，...，t (n - m)，可以在常数时间内根据 ts 计算出 t (s+1)。因为

   t (s + 1) = 10 (ts - 10^(m-1) T [s+1]) + T [s + m +1]　　(1)

   事实上，就是去掉最高位，然后左移了一位，在加上 T [s + m +1] ，就得到了 t (s + 1) 。

   预处理的时间为 O (m)

3. 字符串匹配

   我们只需要将 ti (i = 0 , 1 , ... n-m) 与 p 进行比较，相等则为合法匹配，否则为非法匹配。整个匹配过程的时间为 O (n -m + 1)。

然而，上述问题对于模式 p 的长度较小时，比较方便。当 p 和 ts 的值很大时，p 的结果会太大，以至于不能很好的处理这类问题 。所以需要改进。

4. 增加模运算

对一个合适的模 q 来计算 p 和 ts 的模。每个字符是一个十进制数，因为 p ，t0 以及递归式（1）计算过程都可以对模 q 进行，所以可以在 O (m) 时间内计算出模 q 的 p 值。在时间 O(n-m+1) 计算出模 q 的所有 ts 值。通常选模 q 为一个素数，使得 10q 正好为一个计算机字长。

在一般情况下，采用 d 进制的字母表 { 0 ，1，... ，d - 1 } 时，所选取的 q 要满足使 dq 的值在一个计算机字长内，并调整递归式 (1) 以使对模 q 进行运算，使其成为

t (s + 1) = (d (ts - T [s+1] h) + T [s + m +1]) mod q

其中 h ≡ d ^ (m-1) (mod q) 。

5. 伪匹配

加入模 q 后，我们已经不能通过 ts ≡ p (mod q) 并不能说明 ts = p 。当 ts ≡ p (mod q) 不成立时，则肯定 ts != p 。因此，当 ts ≡ p (mod q) 时我们还需要进一步进行测试，看看 ts 是否等于 p ，因为 ts 可能是匹配的也有可能是伪匹配的。

6. 具体代码与分析在之后会给出。

(二) QuickSearch 算法

1. 算法思想

在匹配过程中，模式串并不被要求一定要按从左向右进行比较还是从右向左进行比较,它在发现不匹配时,窗口尽可能向后移动，从而提高了匹配效率。算法实际上只是使用 BM 算法的坏字符规则，但是在平均效率上，可以获得比 BM 算法更好的效率。

2. QuickSearch 算法主要在比较的时候，如果发现匹配失败，则使用类似于 BM 算法的坏字符规则让窗口向后移动。所不同的是，坏字符不是当前正在比较字符，而是窗口后的第一个字符。如果不匹配，窗口需要向后移动；显然可以得知窗口后的第一个字符(不妨设为 c)要参加下一轮的比较。既然 c 要参与比较，为了尽可能使下一次匹配成功，因此首先就要移动窗口使 c 能够成功匹配。如果 c 出现在模式串 P 中，则移动两者距离之差；否则，则窗口需要移动|P|+1 的距离。因此，公式可以定义如下：

$$Qs\text{-}Bc[c] = \begin{cases} \min\{i:\ 1 \le i \le m \text{ and } P[m+1-i] = c\} \\ m+1 \qquad\qquad\qquad\qquad \text{otherwise} \end{cases}$$

3. 算法伪代码

1) Pre-Bc 算法

```
Pre-Bc(P, m, Qs-Bc)
1    for i←1 to ASIZE
2        do  Qs-Bc[i] ← m+1;
3    for i←1 to m
4        do  Qs-Bc[ P[i] ] = m+1 - i;
```

2) QS 算法

```
void QS(char *P, int m, char *T, int n) {
        int j, Qs-Bc[ASIZE];
        Pre-Bc(P, m, Qs-Bc);          //* Preprocessing *//
        //* Searching *//
        j = 1;
        while (j ≤ n – m+1) {
            if (memcmp(P, T+ j -1, m) == 0)   //*P,T下标从1开始*//
                    OUTPUT( j );
            j = j+ Qs-Bc[ T[ j + m] ];                  //* shift *//
        }
}
```

4. QuickSearch 是 BM 算法的简化，仅使用坏符号规则，易于实现。
5. 算法运行时间分析
   1) 预处理需要 O(m+σ)时间复杂和 $O(\sigma)$空间复杂度
   2) 查找时间为 O(mn)时间复杂度

(三)KMP 算法

1. 算法思想
   当遇到一个不成功匹配后，充分利用已经得到的有关前一部分匹配的信息，避免多余的测试，加快匹配过程。
2. 算法步骤
   1) 在匹配过程过程中，文本串中的当前指针只会向右移(增加)，不会向左倒退；
   2) 在测试文本串的 T[i+1..i+m]这一段时，得到一个部分匹配：P[1..j] = T[i+1..i+j], 但 P[j+1] ≠ T[i+j+1];
   3) 设 k 是使得 P[1..k] = P[(j-k+1)..j]的最大整数（即 P[1..j]中最长的与真前缀相同的真后缀的长度）；
   4) 定义: Next[j+1]= max{ k+1|P[1..k]是 P[1..j]的后缀，j>k≥0 }；
   5) Next 函数值的计算只与模式串有关，与文本串内容无关，可以在进行匹配前预先计算得到 Next[1..m]。
   6) 注：在《算法导论》一书中定义了另一种类似的函数 π [i]，两种函数之间的关系为： π [i] = Next[i+1] -1
3. 算法伪代码

1) Next 函数算法

```
Next(P[1..m])
1.  j ← 0;
2.  m ← Length(P) ;
3.  For i ← 1 to m do
4.        Next[i] ← j ;
5.        While j > 0 and P[i]≠P[j] do
6.              j ← Next[j] ;
7.        j ← j + 1;
```

2) KMP 算法

```
KMP(T, P)
1   j ← 1;
2   For i ← 1 to n do
3       while j > 0 and T[i] ≠ P[j]  do
4             j ← Next[j] ;
5       if j = m then              // 找到一个成功匹配
6             return (i-m+1) ;
7       j ← j +1 ;
8   return   (none)
```

4. 算法运行时间分析

1) NEXT 算法运行时间：O(m)

- 由于 i -j ≥ 0，而且 i -j 单调增；
- i －j 不变的次数不超过 m;
- i －j ≤ m, 其增加的次数≤ m。

2) KMP 算法运行时间：O(n)

- 由于 i -j ≥ 0，而且 i -j 单调增；
- i －j 不变的次数不超过 n;
- i －j ≤ n, 其增加的次数≤ n。

3) 总的算法运行时间 O(m+n)

二、核心代码：

题目一:

```
int LCS_LENGTH(){//自底向上计算LCS的长度，利用b数组帮助构造最优解
    int i,j;
    for(i=1;i<=m;i++)//初始化
        c[i][0]=0;
    for(j=0;j<=n;j++)
        c[0][j]=0;
    for(i=1;i<=m;i++)
        for(j=1;j<=n;j++){
            if(s1[i]==s2[j]){//若s1 s2最后字符相同，这个最后字符是它们LCS的一个字符，LCS长度加1
                c[i][j]=c[i-1][j-1]+1;
                b[i][j]=0;//Z(k-1)是s1(i-1) s2(j-1)的LCS
            }
            else if(c[i-1][j]>=c[i][j-1]){//若s1(i-1) s2(j)的LCS长于s1(i) s2(j-1)
                c[i][j]=c[i-1][j];
                b[i][j]=1;//记录Z是s1(i-1) s2(j)的LCS
            }
            else{
                c[i][j]=c[i][j-1];
                b[i][j]=2;
            }
        }
    return 1;

}
```

```
void PRINT_LCS(int i,int j){
    if(i==0||j==0)
        return ;
    if(b[i][j]==0){
        PRINT_LCS( i: i-1, j: j-1);
        printf( _Format: "%c",s1[i]);
    }
    else if(b[i][j]==1)
        PRINT_LCS( i: i-1,j);
    else
        PRINT_LCS(i, j: j-1);
    return ;
}
```

题目二：将字符串视为 d 进制数，求出模式串对 q 求模的值，然后将模式串的值与要比较的字符串的值比较，匹配后再对伪命中点进行再检测

```
int h=1;//最高位数为1时的值
int p=0;
int t=0;
int s=0;//偏移量
for(int i=1;i<=m-1;i++)
    h=(h*d)%q;//即d的m-1次方模q的值
printf("d=%d,q=%d,h=%d\n",d,q,h);
for(int i=1;i<=m;i++){
    p=(d*p+P[i]-'0')%q;//求得模式串对应的值
    t=(d*t+T[i]-'0')%q;//求得t0
}
for(s=0;s<=n-m;s++) {
    if (p == t) {//伪命中点
        if (STR_CMP(P, T + s, m)) {
            printf("FOUND:s=%d\n", s);
            printf("String in T:%s\n", T_found + 1);
        }
    }
    if (s < (n - m)) {
        t = (d * (t - (T[s + 1] - '0') * h) + T[s + 1 + m] - '0') % q;
        if(t<0)
            t+=q;
    }
}
return 1;
```

三、结果与分析：

题目一:

```
Please input the length of s1:
m=15
Please input the length of s2:
n=15
s1:
GKwnfipky8TIA64
s2:
I61Qxu2BqFlHMYV
LSC:
I6
```

```
Please input the length of s1:
m=50
Please input the length of s2:
n=50
s1:
d1nMSe74B172fC4qm7s242bcjKkcXPpWddIuagEO6vNF9ZQi1c
s2:
9W4QGtW7xVHi3YdAW1h773Jzb3242wMWQ8H2bwg3DA9SYBgdl2
LSC:
d1772422bg9
```

(一)分析：

1. 对 LCS_LENGTH 算法，运行时间为 Θ(mn)，因为每个表项的计算时间为 Θ(1)。
2. 对于 PRINT_LCS 算法，过程的运行时间为 O(m+n)，因为每次调用 i 和 j 至少会有一个减 1。

题目二：

```
Please input the length of T:
n=100
Please input the length of P:
m=5
T:
6q9istTy3ui3ORg3qiRp8wjrsU8q6oU1iDN35721zc07ZZ277hD52FI00196P9rqf0sz93cnBtNAaKC1p590uz6sDY8bdUBk9XR7
P:
Ty3ui
d=75,q=79,h=19
FOUND:s=6
String in T:Ty3ui
```

```
Please input the length of T:
n=1000
Please input the length of P:
m=3
T:
1PRTsvdd2y329RaQ9sZ946wr0r2f3tCLv2979ydLSaUIclN97Hp5EuOx6oXK6zBy9GvMi7RwUdr0WrS5NpBT1Ksp0DyxgK027NpZyz658Eq49O9ubkGJb2tV
P91z95elJlUq62y57a8l9RVjyVa151N6fmxhMxKx83kzw6vS834QsL52j4hHZ6en4w77Y2EWL9blPCGUPacz4SrbMdb0YnHqvmB3vU8aIIV597Yr6H8GlrU3
VPAPH7hZ28c5u4Kl2R6dANHTlyHuA6yfet4Uk79MyHH05SZoxQQh6kBMlw8ifvLnEH0K59bGCbqNg1G2lhio17bg38FUxB262CE37J5oijJDkK6XFHBbbvq6
364Zjhk2PD0P1lL44SuMQTJx6J10f64dAu0VTpBNRP1SBQ69JdMY6Ap1a7o1563roj2U7q1Y4fY09jNo4p2Sc033ppZDBkmQV60LlsQExCNJ2o2DV30M711W
500d017Vffk7G162KI54CK06cP5dABavO3e3zF19Ey1O32Hc2ky9OlzS60i0pMHoE245S8bk6kwsxwmAKQ41N7PBav9p0CGhYfk2uPF9ceHbEtj6w3u00gKR
8WIQu7tx45s1Ft128cmH6Z0l6Pr8c0aY0bhh0IkZ8e70bes3RdG0SQw8bEOxYf5Kz131EC4Hr559KQxRpBeNk4CTdz0X36CfiCXVYB2n27P3UYiz0oAoDFEn
kpsQ5R9y12jYr52CHR6WLNH98ts5Ve22MzLjwt85GlW0aP5Q0K93o11zN7BjANPG3zdQZ68rylv5FPYs25HtpA1o3a25SUwWT9fjPE264m0wS2ZDyHp3wnVR
Z0x39UvN3LZdIr88Ehvb67o884qielBI32cB36AUtU3NMf5Pc4Kds8IQFYWBKg8r0HSE1AEQknkj44oNtgG305ruhALn8kdL6ZiDL6eC7g7A6554 6ssBen9X
AN2A7a25J0aGw1i1iyYxHW2T1u0dJyy5fPfwn5HZ
P:
9Ra
d=75,q=79,h=16
FOUND:s=12
String in T:9Ra
```

(一)分析

1. R_K 算法的预处理的时间为 Θ(m),在最坏情况下，它的匹配时间是 Θ((n-m+1)m)，因为 R_K 算法和朴素字符串匹配算法一样，对每个有效偏移进行显式验证。
2. 但是，如果期望的有效偏移量很少，而选取的素数 q 大于模式的长度，可以估计 R_K 算法的匹配时间为 O(m+n),由于 m<n,这个算法的期望匹配时间为 O(N)。

四、备注：

有可能影响结论的因素：

(一)在 R_K 算法实现中

- 我们在求 h ≡ d ^ (m-1) (mod q)的时候，不能像伪代码一样直接先求出 d ^ (m-1) 然后取模，因为当字符表种类很多，即 d 很大，而模式串也很长时，d ^ (m-1)很容易溢出。因此我们要采取循环相乘取模的方式

```
for(int i=1;i<=m-1;i++)
    h=(h*d)%q;
```

- 我们再求 t (s + 1) = (d (ts - T [s+1] h) + T [s + m +1]) mod q 时，有可能计算到 t (s + 1)为负值的情况，这时要对它进行修正

```
if (s < (n - m)) {
    t = (d * (t - (T[s + 1] - '0') * h) + T[s + 1 + m] - '0') % q;
    if(t<0)
        t+=q;
}
```

总结：

(一)字符串匹配是文本处理中一个重要领域，并且在其他很多应用中发挥着重要作用。通过本次实验，我调研了不同的字符串匹配算法，学习了这些算法的基本思想，分析了它们的运行时间。同时实现了 R_K 算法，加深了我对字符串匹配的理解。

(二)通过实现 LCS 算法，让我更加了解了动态规划这一问题解决策略。

附录（源代码）

算法源代码（C/C++/JAVA 描述）

LCS 算法：

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

char *s1;
char *s2;
int **c;//c 是一个二维数组，存储 LCS 的长度
int **b;//b 是一个二维数组，存储计算 c 时所选择的子问题的最优解
```

```
int m;//s1 的长度
int n;//s2 的长度

void Generate_Str(char *s,int len){
    s[0]='0';
    s[len]='\0';
    int flag;
    for(int i=1;i<len;i++){
        flag=rand()%3;
        if(flag==0){
            s[i] = rand()%26 +'a';
        }
        else if(flag==1){
            s[i] = rand()%26 +'A';
        }
        else{
            s[i] = rand()%10 +'0';
        }
    }
}
int LCS_LENGTH(){//自底向上计算 LCS 的长度，利用 b 数组帮助构造最优解
    int i,j;
    for(i=1;i<=m;i++)//初始化
        c[i][0]=0;
    for(j=0;j<=n;j++)
        c[0][j]=0;
    for(i=1;i<=m;i++)
        for(j=1;j<=n;j++){
            if(s1[i]==s2[j]){//若 s1 s2 最后字符相同，这个最后字符是它们 LCS 的一个字符，LCS 长度加 1
                c[i][j]=c[i-1][j-1]+1;
                b[i][j]=0;//Z(k-1)是 s1(i-1) s2(j-1)的 LCS
            }
            else if(c[i-1][j]>=c[i][j-1]){//若 s1(i-1) s2(j)的 LCS 长于 s1(i) s2(j-1)
                c[i][j]=c[i-1][j];
                b[i][j]=1;//记录 Z 是 s1(i-1) s2(j)的 LCS
            }
            else{
                c[i][j]=c[i][j-1];
                b[i][j]=2;
            }
        }
```

```
        return 1;

}
void PRINT_LCS(int i,int j){
        if(i==0||j==0)
                return ;
        if(b[i][j]==0){
                PRINT_LCS(i-1,j-1);
                printf("%c",s1[i]);
        }
        else if(b[i][j]==1)
                PRINT_LCS(i-1,j);
        else
                PRINT_LCS(i,j-1);
        return ;
}
int main() {
        int i;
        printf("Please input the length of s1:\nm=");
        scanf("%d",&m);
        printf("Please input the length of s2:\nn=");
        scanf("%d",&n);
        s1=(char *)malloc((m+2)*sizeof(char));
        s2=(char *)malloc((n+2)* sizeof(char));
        srand(time(NULL));
        Generate_Str(s1,m+1);
        Generate_Str(s2,n+1);
        printf("s1:\n%s\n",s1+1);
        printf("s2:\n%s\n",s2+1);
        c=(int **)malloc((m+1)*sizeof(int *));
        for(i=0;i<m+1;i++)
                c[i]=(int *)malloc((n+1)* sizeof(int));
        b=(int **)malloc((m+1)*sizeof(int *));
        for(i=0;i<m+1;i++)
                b[i]=(int *)malloc((n+1)* sizeof(int));
        printf("LSC:\n");
        LCS_LENGTH();
        PRINT_LCS(m,n);
}
```

R_K 算法：

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <time.h>
```

```
#include <math.h>
char *T;
char *T_found;
char *P;
int n;
int m;

void Generate_Str(char *s,int len){
	s[0]='0';
	s[len]='\0';
	int flag;
	for(int i=1;i<len;i++){
		flag=rand()%3;
		if(flag==0){
			s[i] = rand()%26 +'a';
		}
		else if(flag==1){
			s[i] = rand()%26 +'A';
		}
		else{
			s[i] = rand()%10 +'0';
		}
	}
}
int STR_CMP(char *s1,char*s2,int len){
	int i;
	for(i=1;i<=m;i++){
		T_found[i]=s2[i];
		if(s1[i]!=s2[i])
			return 0;
	}
	return 1;
}
int RABIN_KARP_MATCHER(int d,int q){//将字符串视为 d 进制数，求出字符串对 q 求模的值，然后将模式串的值与要比较的字符串的值比较
	//之后再对伪命中点进行再检测
	int h=1;//最高位数为 1 时的值
	int p=0;
	int t=0;
	int s=0;//偏移量
	for(int i=1;i<=m-1;i++)
		h=(h*d)%q;//即 d 的 m-1 次方模 q 的值
	printf("d=%d,q=%d,h=%d\n",d,q,h);
```

```
        for(int i=1;i<=m;i++){
            p=(d*p+P[i]-'0')%q;//求得模式串对应的值
            t=(d*t+T[i]-'0')%q;//求得 t0
        }
        for(s=0;s<=n-m;s++) {
            if (p == t) {//伪命中点
                if (STR_CMP(P, T + s, m)) {
                    printf("FOUND:s=%d\n", s);
                    printf("String in T:%s\n", T_found + 1);
                }
            }
            if (s < (n - m)) {
                t = (d * (t - (T[s + 1] - '0') * h) + T[s + 1 + m] - '0') % q;
                if(t<0)
                    t+=q;
            }
        }
        return 1;
}
int main() {
        int i;
        printf("Please input the length of T:\nn=");
        scanf("%d",&n);
        printf("Please input the length of P:\nm=");
        scanf("%d",&m);
        T=(char *)malloc((n+2)*sizeof(char));
        P=(char *)malloc((m+2)* sizeof(char));

        T_found=(char *)malloc((m+2)* sizeof(char));
        T_found[m+1]='\0';
        T_found[0]=0;
        srand(time(NULL));
        Generate_Str(T,n+1);
        //Generate_Str(P,m+1);
        for(i=1;i<=m;i++)
            P[i]=T[12+i];
        P[i]='\0';
        printf("T:\n%s\n",T+1);
        printf("P:\n%s\n",P+1);
        int d='z'-'0'+1;//d=75,75 进制
        int q=79;//选取的素数为 13
        RABIN_KARP_MATCHER(d,q);
}
```