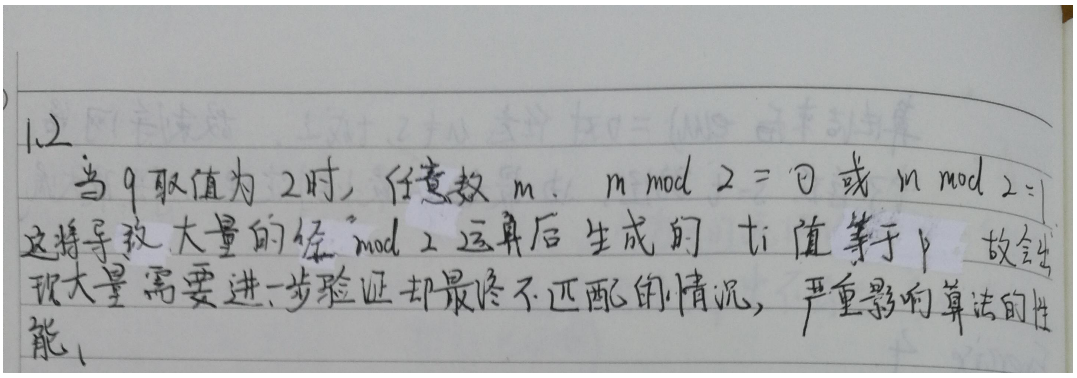
Exercise 1

1.1

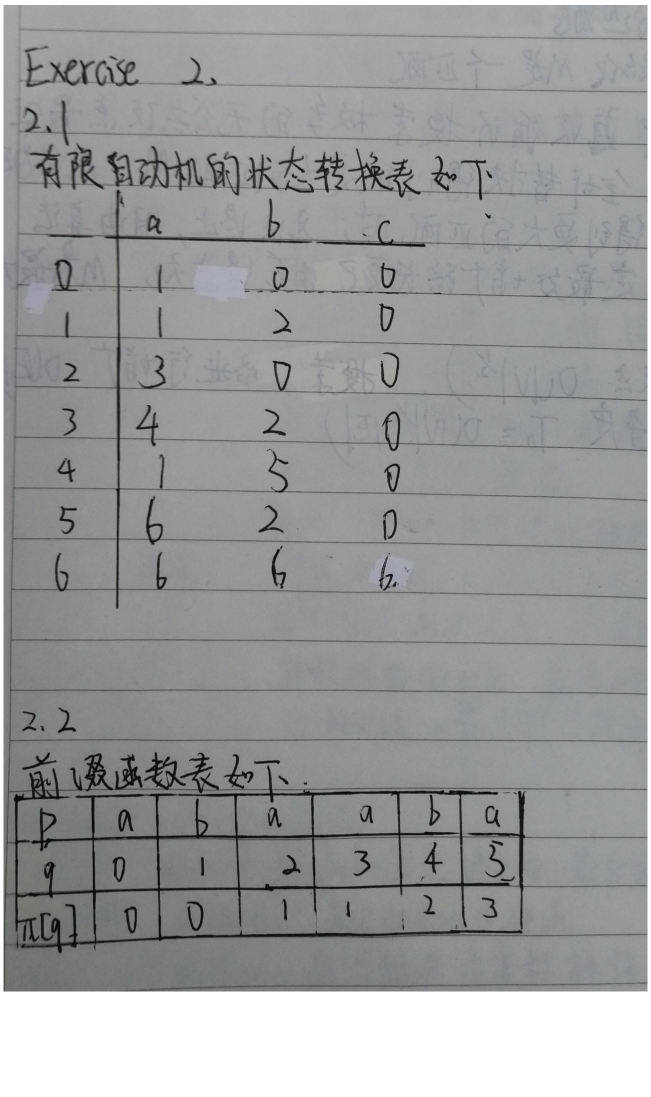
代码：

```
public class RabinKarp {

  public static void RabinKarpAlogrithm(char[] T, char[] P, int d, int q) {
    int n = T.length;
    int m = P.length;
    if (n < m)
      return;
    int h = 1;
    for (int i = 1; i <= m - 1; i++)
      h = h * d % q;
    // 预处理,计算 p， t0
    int p = 0, t = 0;
    for (int i = 0; i < m; i++) {
      p = ((d * p + P[i]) % q);
      t = ((d * t + T[i]) % q);
    }
    // 开始匹配
    for (int s = 0; s < n - m + 1; s++) {
      if (p == t) {
        int i = 0;
        for (i = 0; i < m; i++)// 进一步验证
          if (P[i] != T[s + i])
            break;
        if (i == m)
          System.out.println("Pattern occurs from:" + s);
      }
      if (s < n - m)
        t = (d * (t - T[s] * h % q) + T[s + m]) % q; // 计算 ts+1
    }
    System.out.println("String matching ends");
  }

  public static void main(String[] args) {
    String strT = "2359023141526739921";
    String strP = "31425";
    char[] T = strT.toCharArray();
    char[] P = strP.toCharArray();
    int d = 10; // 所有出现的字符均为数字 0-9，故字母表大小为 10.
    int q = 13;
    RabinKarp.RabinKarpAlogrithm(T, P, d, q);
    }

  }
```



1.2

当q取值为2时，任意数 m. m mod 2 = 0 或 m mod 2 = 1 这将导致大量的经 mod 2 运算后生成的 ti 值等于 p，故会出现大量需要进一步验证却最终不匹配的情况，严重影响算法的性能。



Exercise 2.

2.1

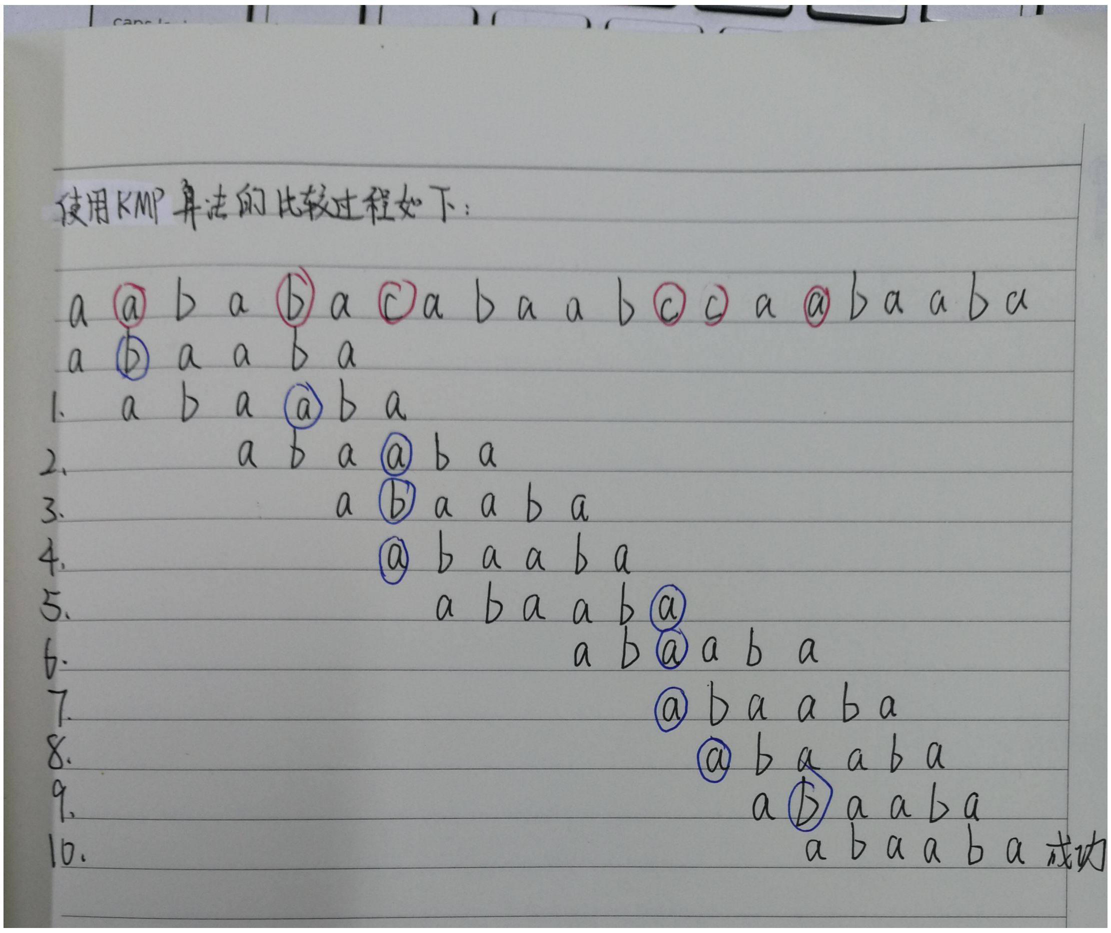
有限自动机的状态转换表如下

	a	b	c
0	1	0	0
1	1	2	0
2	3	0	0
3	4	2	0
4	1	5	0
5	6	2	0
6	6	6	6

2.2

前缀函数表如下：

P	a	b	a	a	b	a
q	0	1	2	3	4	5
π[q]	0	0	1	1	2	3



使用KMP算法的比较过程如下：

```
    a a b a b a c a b a a b c c a a b a a b a
    a b a a b a
1.    a b a a b a
2.        a b a a b a
3.            a b a a b a
4.              a b a a b a
5.                a b a a b a
6.                      a b a a b a
7.                          a b a a b a
8.                            a b a a b a
9.                              a b a a b a
10.                               a b a a b a 成功
```