32.1-2

NAIVE-STRING-MATCHER(T,P) ：

s = 0

While s <= n-m :

If P[1..d] == T[s+1..s+d] and P[d+1] != T[s+d+1] and d <= m-1:

s = s + d + 1

else :

print "Pattern occurs with shift" s

s = m + 1

//可以发现以上算法的s指针一直不会回退，而且至少会前进1个字符，所以运行时间复杂//度在O(n-m+1)=O(n )，故达到了O(n)。

32.1-4

假设如果文本中存在题中所述模式P，那么对于在第一个间隔符之前的子字符串，文本中存在一个位置i使得子字符串与之匹配，如果位置i之前还有位置j (j<i)可以与子字符串匹配，则完全可以将子字符串的匹配从位置i前移到位置j，且对于模式P的匹配依然成立，因此可以让j取最小（第一个使得该子字符串匹配的位置）。所以由上述分析，有以下算法：

考虑第一个间隔符之前的子字符串，文本查找第一个满足匹配该子字符串的位置（如果文本中间没有找到匹配的子字符串，则返回0），文本中剔除该子字符串以及之前的字符。

再考虑第一个间隔符和第二个间隔符之间的子字符串，文本查找第一个满足匹配该子字符串的位置（如果文本中间没有找到匹配的子字符串，则返回0），文本中剔除该子字符串以及之前的字符。再考虑第二个间隔符和第三个间隔符之间的子字符串。。。如此操作一直到所有间隔符之间的子字符串全部匹配为止，返回1。中途文本中没有找到匹配的子字符串则返回0。

每一次找的都是间隔符之间子字符串的第一个位置，所以最差情况是所有间隔符之间的子字符串都要查找一遍。获得模式中间隔符之间子字符串时间复杂度为O(m)，对于间隔符之间子字符串的查找最多为，故最差复杂度为：

32.2-3

func match(T,P,d,q) :

n = T.len

m = P.len

t[0][0] = 0

h = d\*\*((i+1)\*m)%q for i range(m)

temp\_h = d\*\*(m-1)%q

for i in range(n-m) :

for j in range(n-m) :

if p == t[i][j] :

if P[0..m-1][0..m-1] == T[i+..i+m-1][j..j+m-1] :

print("Pattern occurs with shift" s)

if j == n-m :

t[i+1][0] = t[i][0]

temp = 1

for k in range(m) :

t[i+1][0] = h[0]\*(t[i+1][0]-T[i][k]\*h[m-1]\*temp+T[i+m][k]\*temp)%q

temp = temp\*d

else :

t[i][j+1] = t[i][j]

for k in range(m) :

t[i][j+1] = d\*(t[i][j+1]-T[i+k][j]\*temp\_h\*h[m-k-1]+T[i+m][k]\*h[m-k-1])%q

32.3-6

将模式P首尾添上间隔符，成为，转化成为和文本T的字符串进行匹配。

设为P，构造的有限自动机，最后将有限自动机串行连接即可得到最后有限自动机。可知时间开销为O(n)。

32.4-1

a b a b b a b b a b b a b a b b a b b

0 0 1 2 0 1 2 0 1 2 0 1 2 3 4 5 6 7 8

32.4-5

考虑势函数为q，每一次循环只能将势函数至多提高1，无论势函数怎么减少，其值始终大于等于0，循环到第k次时势函数减少次数不超过k，然而循环次数固定为n，所以时间开销是O(n)。