```
for i := 0 to n-m
  valid := true;
  for j := 1 to m
    if P[j] != T[i+j]
      valid := false; break loop

  if (valid) then output shift i
```

```
Rabin-Karp(T[1..n],P[1,,m]):
  hashp = hash(P,1)
  hasht = hash(T,1)

  for i:=0 to m-n
    // inv: hasht = hash(T,i+1)
    if (hashp = hasht)
      // check whether T[i+1..i+m] matches the pattern
      valid = true
      for j = 1 to m
        if P[j] != T[i+j]
          valid = false; break loop
      if valid then output i
    hasht = shift_hash(T,i+1,hasht)
```

– veľkosť abecedy: k

– hašovacia funkcia: $(S[i]S[i+1]...S[i+m-1]) \bmod q$

– predpočítaná hodnota: $ktmm1 = k^{m-1} \bmod q$

```
hash(S,i):
  // compute hash of S[i]S[i+1]...S[i+m-1]
  result = 0
  for j = 0 to m-1
    result = (k * result + S[i+j]) mod q
  return result


shift_hash(S,i,oldhash):
  // compute hash of S[i+1]S[i+2]...S[i+m]
  // given that oldhash is a hash of S[i]S[i+1]...S[i+m-1]
  return ((oldhash + q - (S[i]*ktmm1 mod q))
          * k + S[i+m]) mod q
```

```
     0  1  2  3  4  5  6  7  8  9 10 11
  T: b  a  n  a  n  a  n  o  b  a  n  o

i=0: X
i=1:    X
i=2:       n  a  n  X
i=3:          X
i=4:             n  a  n  o
i=5:                X
i=6:                   n  X
i=7:                         X
i=8:                            X
i=9:                               n  X
i=10:                                 X
```

```
DFA_STRING_MATCHING(T[1..n],tr):
  state:=0;
  for i:=1 to n
    state:=tr[state,T[i]]
    if (state = m)
      output shift i-m
```

```
for i := 0 to m
  prefix := P[1..i];

  for all symbols c from the alphabet
    current := prefix + c;
    for j := i+1 downto 0
      if current[i+1-j+1..i+1] = P[1..j]
        tr[i,c] := j;
        break the loop
```

```
KMP_STRING_MATCHING(T[1..n],pi):
  // P[m+1] = some character outside alphabet
  state := 0
  for i := 1 to n
    while state>0 and T[i]<>P[state+1]
      state := pi[state]
    if T[i] = P[state+1]
      state := state + 1
    if state = m
      output shift i-m
```

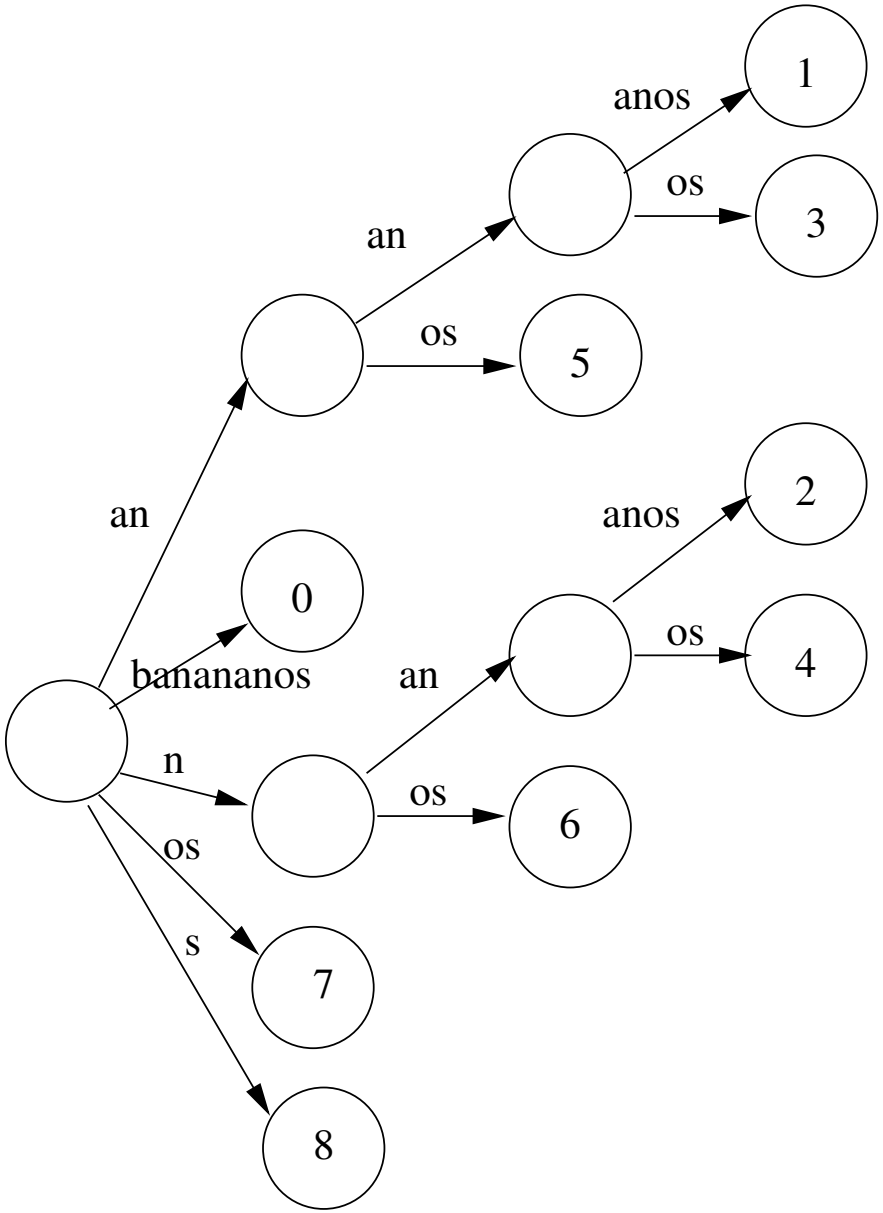
```
for all values of q such that
  P[1...q] is suffixo-prefix of P[1...i-1]:
  (the values are listed from largest to the smallest)
  if P[q+1]=P[i] then
    pi[i]:=q+1;
    break loop;
```

vypíš všetky sufixo-prefixy reťazca $P[1..i]$:

```
q:=i;
while q>0
  q:=pi[q]
  print P[1..q]
```

```
CONSTRUCT_PREFIX_FUNCTION(P[1..m]):
  pi[0]:=0;
  for i:=1 to m
    q:=i-1; pi[i]:=0
    while q > 0
      q:=pi[q]
      if P[q+1]=P[i] then
        pi[i]:=q+1;
        break loop;
```

```
CONSTRUCT_PREFIX_FUNCTION(P[1..m]):
  pi[0]:=0; pi[1]:=0;
  q:=0;
  for i:=2 to m
    while q > 0 and P[q+1]<>P[i]
      q := pi[q];
    if P[q+1] = P[i]
      q := q+1
    pi[i] := q
```


1
anos
os
3
an
os
5
an
2
anos
0
bananos
an
os
4
n
os
6
os
s
7
8

Vyhľadávanie v texte: Zhrnutie

Algoritmus	Predpočítanie	Vyhľadávanie	
Naivný algoritmus		$O(mn)$	worst-case
Rabin-Karp		$O(m+n+\frac{mn}{q})$	expected
Konečný automat	$O(m^3)$	$O(n)$	worst-case
Knuth-Morris-Pratt	$O(m)$	$O(n)$	worst-case
Sufixový strom	$O(n)$	$O(m)$	worst-case