```
Rebuild(x):
    A=empty array
    inorder(x,A)
    return buildBalanced(A,1,size_of(A))

inorder(x,A):
    if x = NULL return
    inorder(x.left,A)
    push(A,(x.key,x.val))
    inorder(x.right,A)
```

```
buildBalanced(A,from,to):
   mid = (from+to)/2
   root = new node(A[mid].key,A[mid].val)
   root.size = 1
   if (from<mid)
     // build left subtree
     left = buildBalanced(A,from,mid-1)
     root.left = left
     left.parent = root
     root.size += left.size
   if (to>mid)
     // build right subtree
     right = buildBalanced(A,mid+1,to)
     root.right = right
     right.parent = root
     root.size += right.size
   return root
```

```
Insert(key,val,root):
  if root = NULL
    x = new node(key,val)
**  x.size = 1
**  return (x,x,0)

  if key < root.key
**  (root.left,x,depth) = Insert(key,val,root.left)
    root.left.parent = root
**  root.size += 1
  else
**  (root.right,x,depth) = Insert(key,val,root.right)
    root.right.parent = root
**  root.size += 1

**return (root,x,depth+1)
```

```
Scapegoat_Insert(key,val,root):
  (root,x,depth) = Insert(key,val,root)
  if depth>maxDepth(root.size)
     scapegoat = findScapegoat(x)
     parent = scapegoat.parent
     y = Rebuild(scapegoat)
     if (parent = NULL)
        return y
     else
       if (parent.left = scapegoat)
          parent.left = y
       else
          parent.right = y
       y.parent = parent
   return root
```

```
findScapegoat(x):
  if (x.left and x.left.size>(2/3)*x.size)
    return x
  if (x.right and x.right.size>(2/3)*x.size)
    return x
  return findScapegoat(x.parent)
```

Slovníky: Zhrnutie

Metóda	Insert	Search	Delete	Analýza
Spájaný zoznam	$\Theta(1)$	$\Theta(n)$	$\Theta(n)$	worst-case
Utriedené pole	$\Theta(n)$	$\Theta(\log n)$	$\Theta(n)$	worst-case
Hašovanie s reťazením	$\Theta(1)$	$\Theta(n)$	$\Theta(n)$	worst-case
	$\Theta(1)$	$\Theta(1+\alpha)$	$\Theta(1+\alpha)$	expected
— s otvorenou adresáciou	$\Theta(n)$	$\Theta(n)$	N/A	worst-case
	$\Theta(\frac{1}{1-\alpha})$	$\Theta(\frac{1}{\alpha}\ln\frac{1}{1-\alpha})$ or $\Theta(\frac{1}{1-\alpha})$	N/A	expected
Binárne vyhľ. stromy	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$	worst-case
	$\Theta(\log n)$	$\Theta(\log n)$	$\Theta(\log n)$	average
AVL stromy	$\Theta(\log n)$	$\Theta(\log n)$	$\Theta(\log n)$	worst-case
Scapegoat stromy	$\Theta(\log n)$	$\Theta(\log n)$	$\Theta(\log n)$	amortized
Tries (string of length m)	$\Theta(m)$	$\Theta(m)$	$\Theta(m)$	worst-case

```
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 = (10*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
```

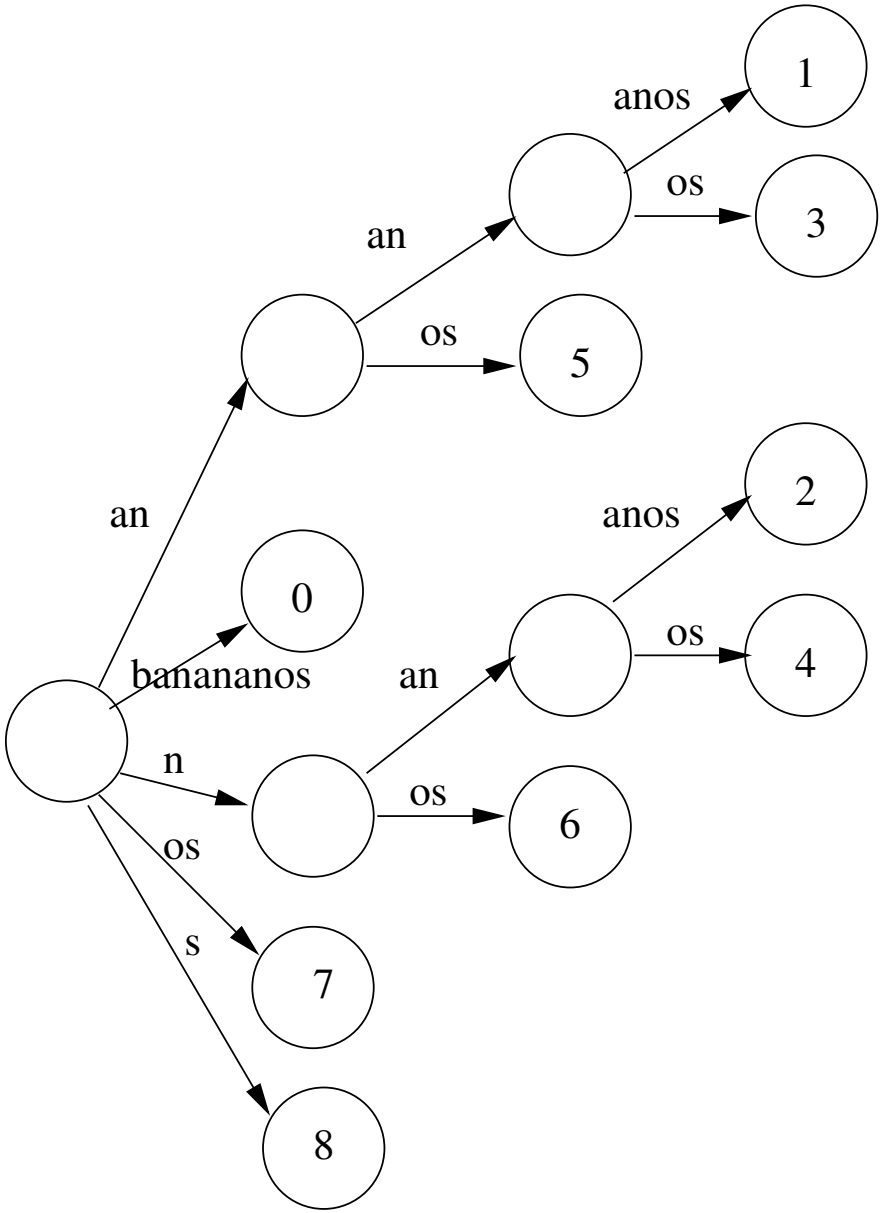
```
DFA_STRING_MATCHING(T[1..n],tr):
  state:=0;
  for i:=1 to n
    state:=tr[state,T[i]]
    if (state = m)
      output shift i-m
```

```
CONSTRUCT_TRANSITION_FUNCTION(P[1..m]):
  for i:=1 to m
    for all characters c in alphabet Sigma
      suffix := P[1..i].c
      drop := 0
      while suffix is not prefix of P
        drop first character of suffix
        drop := drop + 1
      tr[i,c] := i+1-drop
```

```
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
```

```
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