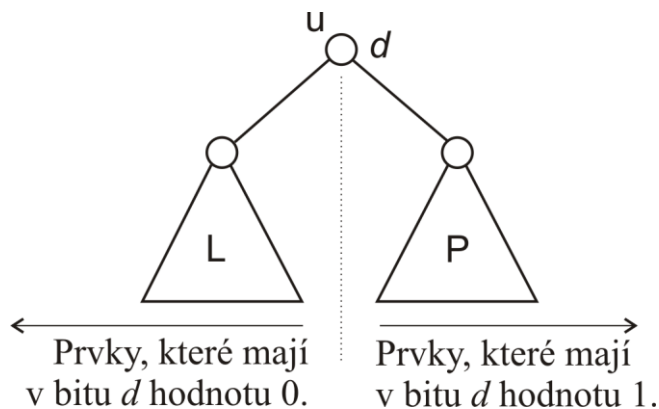


Vyhledávací stromy *Trie*

Číslicové vyhledávací stromy na rozdíl od ostatních vyhledávacích stromů nemají prvky uspořádané dle velikosti (nejsou setříděné), pokud procházíme uzly stromu zleva-doprava (do hloubky). To může být v některých použitích nevýhodné. Ale mnohem výraznější nevýhodou těchto stromů je skutečnost, že při vyhledávání je v každém uzlu nutné srovnávat hledaný prvek s prvkem uloženým v procházeném uzlu (zejména, je-li datový charakter prvků takový, že srovnání dvou prvků je časově náročnější).

Číslicové vyhledávací stromy nazývané *trie* mají prvky setříděné zleva-doprava obdobně jako jiné vyhledávací stromy. Název *trie* je odvozen od slova *retrieval*, které vystihuje použití těchto stromů (vyhledávání údajů).

Stromy *trie* mají prvky uložené jen v listových uzlech. Opět v nich platí, že prvek, který má v daném bitu hodnotu 0, je umístěn v levém podstromu daného uzlu a naopak prvek, který v daném bitu má hodnotu 1, je umístěn v pravém podstromu daného uzlu.



Ve stromu dále platí, že v každém listovém uzlu je právě jeden prvek.

Vyhledání prvku

1. Počáteční krok

Uzel, který je v daném okamžiku vyhledávání aktuální, budeme označovat u . Na začátku jím bude kořen stromu.

Hledaný prvek nechť je x .

Aktuální index (pořadí) bitu označíme d , první bit má index 0.

2. Průběžný krok

Je-li aktuální uzel nelistový, zjistíme hodnotu bitu d prvku x .

- Je-li tato hodnota 0, ověříme, zda uzel u má levého následovníka. Pokud ano, učiníme ho novým aktuálním uzlem, zvýšíme hodnotu d o 1 a opět

provedeme krok 2. Pokud uzel levého následníka nemá, vyhledávání končí, hledaný prvek není ve stromu obsažen.

- Je-li hodnota bitu d rovna 1, pokračujeme obdobně ve vyhledávání v pravém podstromu, pokud aktuální uzel u má pravého následníka.

Je-li aktuální uzel u list, srovnáme, zda je hledaný prvek x roven prvku, který je uložen v tomto uzlu. Pokud ano, hledaný prvek byl v tomto uzlu nalezen, jinak vyhledávání končí neúspěšně.

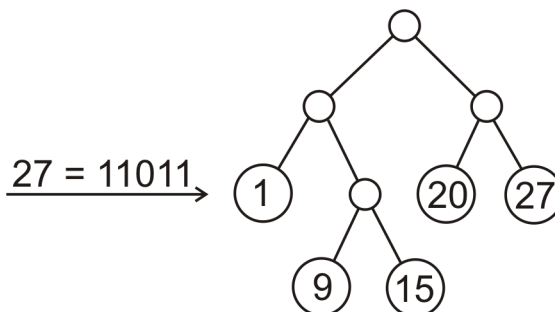
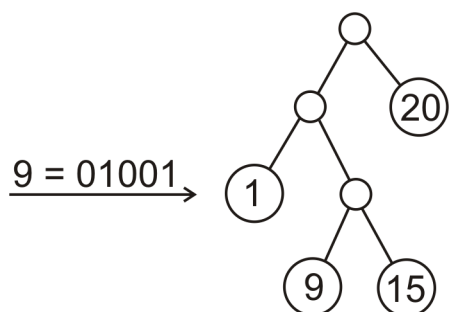
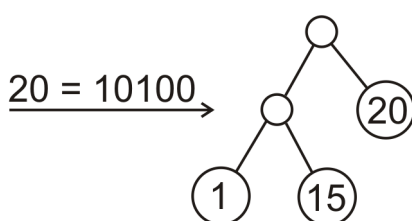
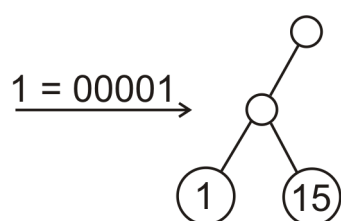
Přidání prvku

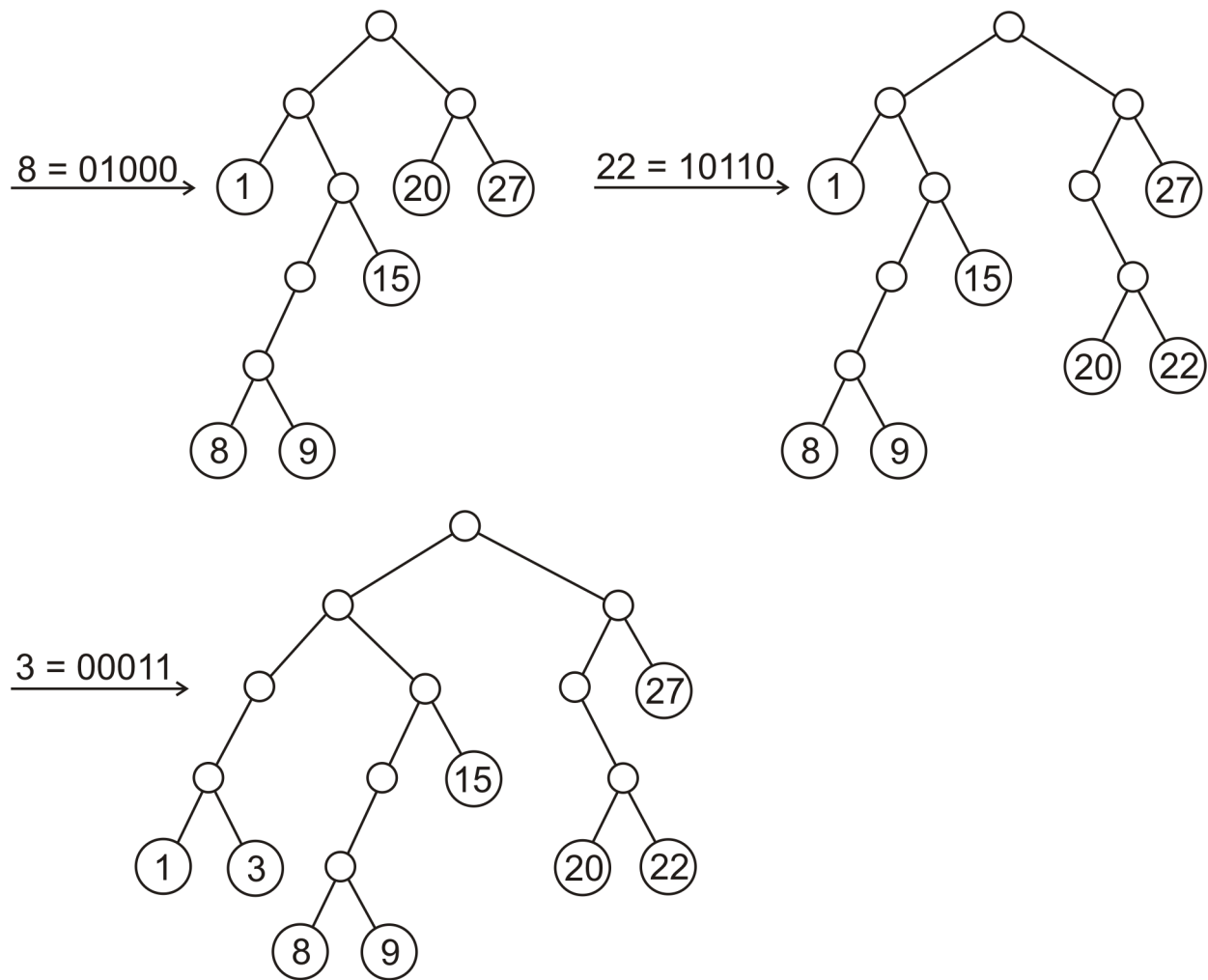
Přidávaný prvek x nejprve vyhledáme. Pokud nebyl nalezen, mohou nastat dva případy:

- ♦ Vyhledávání skončilo v nelistovém uzlu, protože v něm nebyl příslušný následník, aby vyhledávání mohlo pokračovat. Zde tohoto následníka vytvoříme a přidávaný prvek x do něho vložíme.
- ♦ Vyhledávání skončilo v listovém uzlu, ve kterém je uložen prvek c . Před tento list přidáváme nelistové uzly pro všechny bity až po bit (včetně tohoto bitu), jehož hodnoty jsou v prvcích x a c různé. Následně vytvoříme nový uzel, do kterého dáme prvek x a tento uzel učiníme následníkem posledního přidaného nelistového uzlu. Druhým následníkem tohoto uzlu je uzel s prvkem c .

Příklad. Do stromu budeme ukládat pětibitová čísla.

$15 = 01111 \rightarrow$ (15)





Pseudokód vyhledání:

IsLeaf(u)

return u.left=NIL and u.right=NIL

Search(T, x)

u ← T.root

d ← 0

while u ≠ NIL

if IsLeaf(u)

if x = u.item

return u

return NIL

if Bit(x,d) = 0

u ← u.left

else

u ← u.right

d ← d+1

```
return NIL
```

Pseudokód přidání:

```
NewNode(x)
```

```
u ← new Node
```

```
u.item ← x
```

```
u.left ← u.right ← NIL
```

```
return u
```

```
Insert(T, x)
```

```
if T.root = NIL
```

```
    T.root ← NewNode(x)
```

```
    return true
```

```
u ← T.root
```

```
if IsLeaf(u)
```

```
    if x = u.item
```

```
        return false
```

```
    T.root ← Split(x, u, 0)
```

```
    return true
```

```
d ← 0
```

```
while true
```

```
    if Bit(x, d) = 0
```

```
        if u.left = NIL
```

```
            u.left ← NewNode(x)
```

```
            return true
```

```
        if IsLeaf(u.left)
```

```
            if x = u.left.item
```

```
                return false
```

```
            u.left ← Split(x, u.left, d)
```

```
            return true
```

```
        u ← u.left
```

```
else
```

```
    if u.right = NIL
```

```
        u.right ← NewNode(x)
```

```
        return true
```

```
    if IsLeaf(u.right)
```

```

        if x = u.right.item
            return false
        u.right ← Split(x, u.right, d)
        return true
    u ← u.right
    d ← d+1
Split(x, v, d)
    z ← w ← new Node
    while true
        b ← Bit(x,d)
        if b = Bit(v.item,d)
            u ← new Node
            if b=0
                z.left ← u
                z.right ← NIL
            else
                z.right ← u
                z.left ← NIL
            z ← u
            d ← d+1
        else
            if b=0
                z.left ← NewNode(x)
                z.right ← v
            else
                z.left ← v
                z.right ← NewNode(x)
    return w

```