Definice: Nechť $(K, \leq)$ je úplně uspořádaná množina tzv. *klíčů*, V je libovolná množina tzv. doplňujících údajů. Nechť $U = K \times V$ je množina dvojic (k, v), v níž se každý klíč k vyskytuje nejvýše jednou (tj. každý záznam z množiny U je určen jednoznačně svým klíčem). Problému nalézt k danému klíči k záznam $(k, v) \in U$ se říká *vyhledávací problém.*

▲ **1.1** Mějme pole hodnot A. Hodnoty jsou v poli A rozmístěny náhodně. Budeme-li předpokládat, že v tomto poli budeme hledat pouze k-krát, kde k je malé přirozené číslo (například 3), jak byste postupovali při hledání prvku v tomto poli? Jak byste postupovali v případě, že by bylo vyžadováno časté hledání prvků v tomto poli?

▲ **1.2** Jaká je složitost binárního vyhledávání?

▲ **1.3** V poli $A = [1, 4, 5, 6, 11, 13, 17, 18]$, nalezněte binárním vyhledáváním index prvku s hodnotou klíče 17. Ve stejném poli nalezněte index prvku s hodnotou klíče 2.

Hašovací tabulky

Definice: *Hašovací tabulka* je datová struktura, pomocí níž lze prakticky efektivně realizovat „slovníkové“ operace vyhledání, přidání a zrušení položky.

Prakticky efektivně znamená, že operace mají příznivou průměrnou časovou složitost (tedy ne nutně časovou složitost v nejhorším případě).

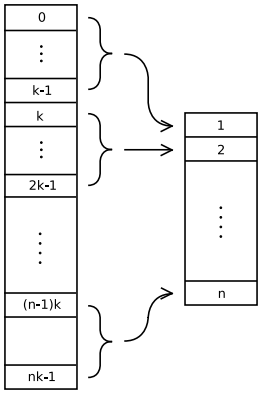
Hašovací tabulka je jednorozměrné pole H indexované čísly $1, \ldots, n$. Převod klíčů na čísla realizuje hašovací funkce $h : K \to \{1, \ldots, n\}$. Výpočet hodnot hašovací funkce musí být efektivní, nejlépe složitosti $\Theta(1)$.

Příklad hašovací tabulky je zobrazen na obrázku 1.

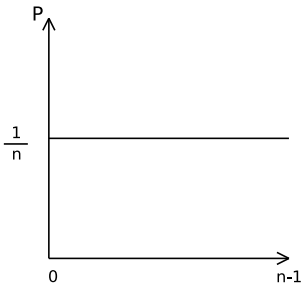
▲ **1.4** Jaké hašovací funkce jsou použity v následujících případech?

- Vyhledávání v telefonním seznamu.
- Psaní písmen na mobilním telefonu.
- Rozřazování mužstva s pomocí pravidla „první, druhý, první, druhý, ...“.

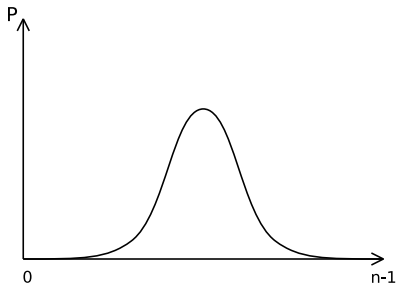
▲ **1.5** Navrhněte hašovací funkci pro data, která mají pravděpodobnost výskytu svých prvků danou níže uvedenými funkcemi. Funkci navrhněte tak, aby hodnoty byly v hašovací tabulce distribuovány rovnoměrně.



Obrázek 1: Příklad hašovací tabulky



(a) Uniformní



(b) Gaussián

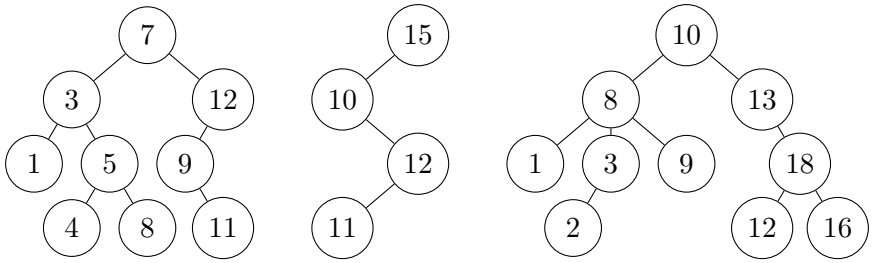
▲ **1.6** Mějme uniformní hašovací funkci, která mapuje interval 0..19 na 1,. . . , 80..99 na 5. Vložte do hašovací tabulky postupně čísla [57, 60, 74, 35, 16, 61, 7, 49, 86, 98]. Pro řešení kolizí použijte jednosměrně zřetězený seznam.

▲ **1.7** Předpokládejme hašovací funkci h, která mapuje n různých hodnot do pole T délky m. Jaký je předpokládaný počet kolizí uvažujeme-li uniformní hašování? To jest, u kolika z uložených n prvků lze očekávat, že budou na stejné pozici s jinými prvky?

Binární vyhledávácí stromy

Definice: *Binární vyhledávácí strom* (BVS) je binární strom nad úplně uspořádanou množinou (tzv. klíčů) $(K, \leq)$ takový, že pro každý jeho podstrom t platí:

- hodnoty uzlů v levém podstromu t jsou menší než hodnota v kořenu t,
- hodnoty uzlů v pravém podstromu t jsou větší než hodnota v kořenu t.



Obrázek 2: Rozhodněte zda jsou BVS

▲ **1.8** Rozhodněte, zda jsou stromy na obrázku 2 BVS. Svou odpověď zdůvodněte.

▲ **1.9** Při praktické implementaci BVS lze každý jeho uzel reprezentovat pomocí struktury `Node` obsahující čtyři atributy: hodnotu klíče `Node.key`, ukazatel na rodiče `Node.parent`, ukazatel na levého `Node.left` a pravého `Node.right` potomka.

```
Node = record
   key : Integer
   left,right : ^Node
   parent : ^Node
end
```

```
          ___14___
         /        \
        12         18
       /  \       /  \
      10  13     16  20
     /  \       /   /  \
    6   11     15  19  22
```

Nechť `NodeX` označuje ukazatel na uzel s hodnotou klíče X. V daném BVS určete čemu se rovnají následující výrazy:

a) `(((Node20.parent).left).left).key`

b) `((Node13.parent).parent).parent`

c) `((Node14.left).left).right`

d) `((((Node12.parent).right).right).left).key`

▲ **1.10** Zkonstruujte BVS postupným vkládáním uzlů 16, 5, 9, 28, 2, 20, 18, 29, 24, 26, 22. Poté postupně odstraňte uzly 9, 5, 20. Nakonec vyhledejte uzly 24 a 9.

▲ **1.11** Mějme definován binární vyhledávací strom a operace nad ním následujícím způsobem:

```
Node = record
   key : Integer
   left,right : ^Node
   parent : ^Node
end
```

```
function Init(T)
begin
   T.root = nil
end
```

```
BVS = record
   root : ^Node
end
```

```
function Search(T,k)
begin
  x := T.head
  while (x <> nil) AND (x.key <> k) do
     if (k < x.key) then
        x := x.left
     else
        x:= x.right
  return x
end
```

```
function Minimum(z)
begin
   while (z.left <> nil) do
      z := z.left

   return z
end
```

```
function Maximum(z)
begin
   while (z.right <> nil) do
      z := z.right

   return z
end
```

```
function Successor(z)
begin
  if (z.right <> nil) then
     return Minimum(z.right)

  y := z.parent

  while (y <> nil AND
         z = y.right) do
     z := y
     y := y.parent

  return y
end
```

```
function Delete(T,z)
begin
   if (z.left = nil OR
       z.right = nil) then
      y := z
    else
      y := Successor(T,z)

   if (y.left <> nil) then
      x := y.left
   else
      x := y.right

   if (x <> nil) then
      x.parent := y.parent
```

```
   if (y.parent = nil) then
       T.root := x
   else
      if (y = (y.parent).left) then
         (y.parent).left := x
      else
         (y.parent).right := x

   if (y <> z) then
      z.key := y.key

   return y
end
```

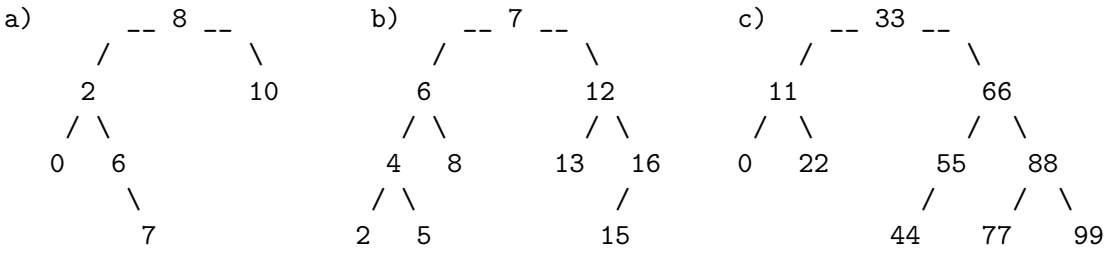
Navrhněte implementaci funkce `Insert(T,z)`, která vloží do stromu `T` uzel `z`.

▲ **1.12** Určete časovou složitost jednotlivých operací nad BVS.

AVL stromy

Definice: *AVL strom* je binární vyhledávácí strom, u kterého se hloubka levého a pravého podstromu libovolného uzlu liší nejvýše o jedničku. Příkladem AVL stromu je Fibonacciho strom. Vyhledávání v AVL stromu se neliší od vyhledávání v běžném vyhledávacím stromu. Při přidávání položky do AVL stromu se musí kontrolovat (a případně spravit) vyváženost. Podobně při rušení položky.

▲ **1.13** Rozhodněte, zda jsou následující stromy AVL.



▲ **1.14** Postupným vkládáním klíčů 8, 3, 5, 0, 2, 4, 1, 6, 9, 7 vybudujte AVL strom.

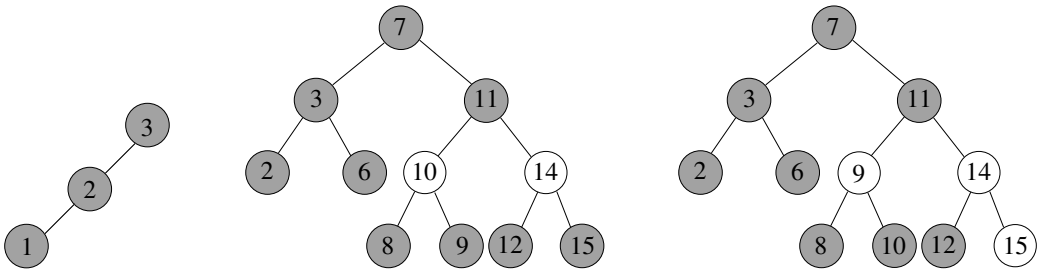
▲ **1.15** Z výsledného AVL stromu z předchozího příkladu odeberte postupně uzly s hodnotami 7, 5, 6 a 1.

▲ **1.16** Jaká je složitost operací vyhledání, přidání a odebrání uzlu v AVL stromu?

Červenočerné stromy

Definice: *Červenočerný strom* je binární vyhledávací strom, jehož každý uzel je obarven černou nebo červenou barvou. Musí splňovat tyto podmínky:

- kořen stromu je černý,
- je-li vnitřní uzel červený, jeho následníci (pokud existují) jsou černí,
- všechny větve obsahují stejný počet černých uzlů.

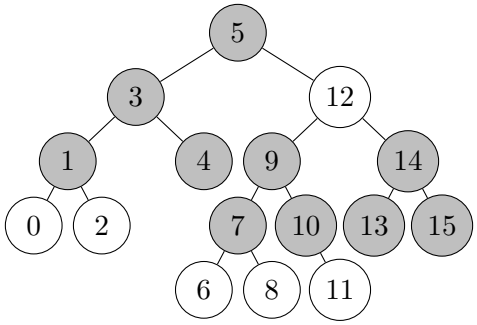


Obrázek 3: Strom pro červenočerné obarvení

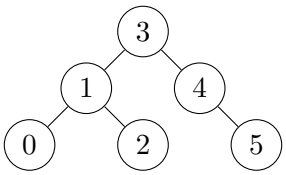
▲ **1.17** Rozhodněte, zda jsou stromy z obrázku 3 červenočerné.

▲ **1.18** Zkonstruujte červenočerný strom postupným vkládáním uzlů 12, 5, 9, 18, 2, 15, 13, 19, 17. Poté postupně odstraňte uzly 9, 5, 15. Nakonec vyhledejte uzly 17 a 9.

▲ **1.19** Z následujícího červenočerného stromu odstraňte uzly s hodnotou 10, 13.



▲ **1.20** Kolika způsoby je možné obarvit BVS z obrázku 4 na červenočerný?



Obrázek 4: Strom pro červenočerné obarvení

▲ **1.21** Jaká je složitost jednotlivých operací nad BVS, pokud uvažujeme červenočerné stromy?