▲ **Def:** 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*.

▲ 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?

▲ Algoritmus binárního vyhledávání:

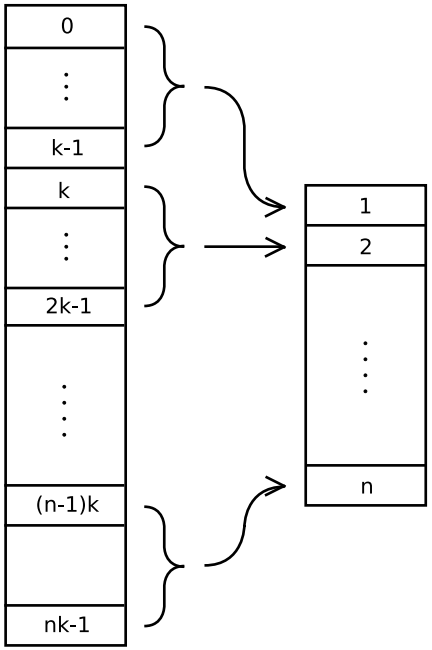
```
type Elem = Integer;
     Pole = array [1..999] of Elem;

function bSearch(k:Elem; var D: Pole; n: Integer): Integer;
{Posl. D je rostoucí. Když D[i]=k, tak bSearch(k)=i, jinak bSearch(k)=-1}
  function bs (l, r: Integer):Integer;
  begin
    if l>r then bs := -1 {nenalezeno}
           else
             begin
               m := (l+r) div 2;
               if k < D[m] then bs := bs(l,m-1);
                 else if k > D[m] then bs := bs(m+1,r);
                 else    {k=D[m]}    bs := m;
             end
  end
begin  bSearch := bs (1,n);  end
```

▲ Jaká je složitost binárního vyhledávání?

▲ 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 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 $1, ..., n$. Převod klíčů na čísla realizuje hašovací funkce $h : K \to \{1, ..., n\}$. Výpočet hodnot hašovací funkce musí být efektivní, nejlépe složitosti $\Theta(1)$.

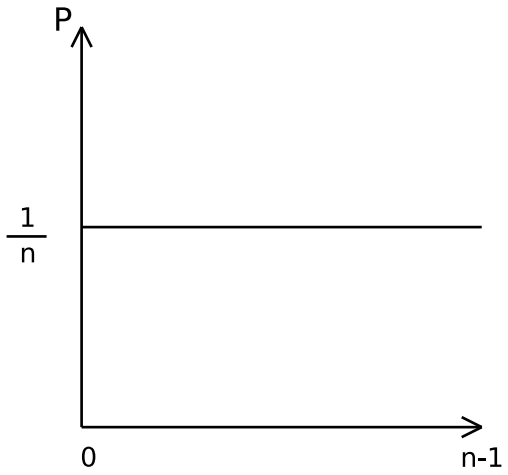
Příklad:



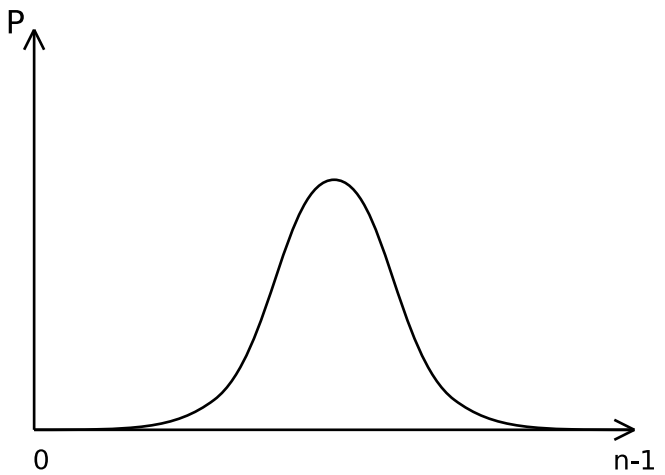
▲ Jaké hašovací funkce jsou použity v následujících případech?

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

▲ 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.



(a) Uniformní



(b) Gaussián

▲ Mějme uniformní hašovací funkci, která mapuje interval $0..19$ na $1,\ldots,$ $80..99$ 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.

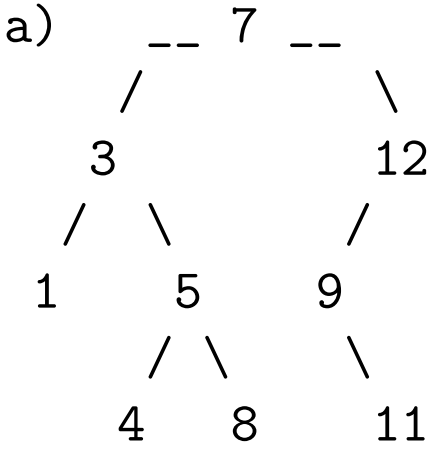
▲ Předpokládejme hašovací funkci h, která mapuje n různých hodnot do pole T délky m. Jaký je předpokládaný počet kolizí uvažujeme-li uniformní hašování? To jest, u kolika z uložených n prvků lze očekávat, že budou na stejné pozici s jinými prvky?

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

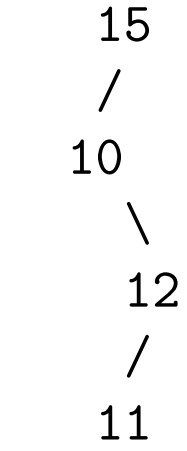
- hodnoty uzlů v podstromu `left(t)` jsou menší než `rootval(t)` a hodnoty uzlů v podstromu `right(t)` jsou větší než `rootval(t)`

Rozhoďte, zda následující stromy jsou BVS. Odpověď zdůvodněte.

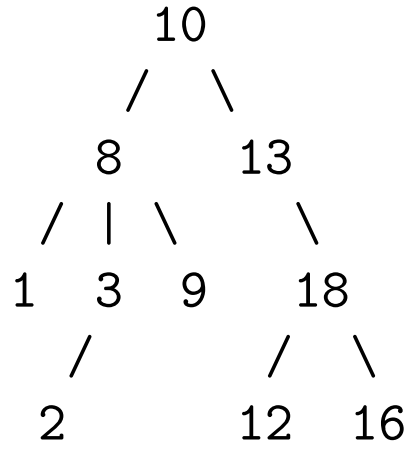
a)



b)

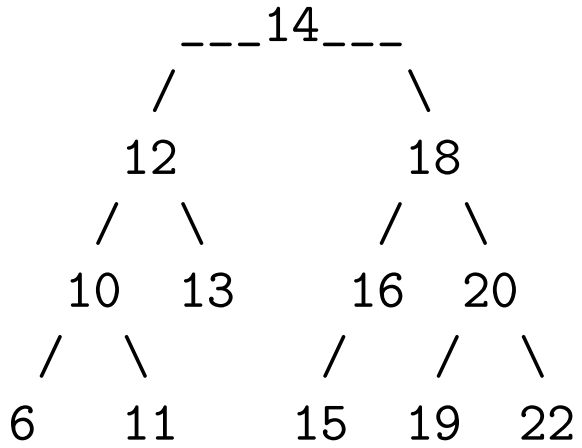


c)



▲ 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
```



Nechť `Node20` označuje ukazatel na uzel s hodnotou klíče 20. 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`

▲ 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.

▲ Předpokládejme, že máme čísla mezi 1 a 1000 v BVS a hledáme číslo 363. Která z následujících sekvencí nemůže být sekvencí uzlů při hledání této hodnoty?

a) 2, 252, 401, 398, 330, 344, 397, 363
b) 924, 220, 911, 244, 898, 258, 362, 363
c) 925, 202, 911, 240, 912, 245, 363
d) 2, 399, 387, 219, 266, 382, 381, 278, 363
e) 935, 278, 347, 621, 299, 392, 358, 363

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

```
Node = record                 |    function Init(T)
   key : Integer              |    begin
   left,right : ^Node         |       T.root = nil
   parent : ^Node             |    end
end                           |
-------------------------------------------------------------------
                              |    function Search(T,k)
BVS = record                  |    begin
   root : ^Node               |      x := T.head
end                           |      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             |    if (y.parent = nil) then
      y := z                           |        T.root := x
    else                               |    else
      y := Successor(T,z)              |       if (y = (y.parent).left) then
                                       |          (y.parent).left := x
   if (y.left <> nil) then             |       else
      x := y.left                      |          (y.parent).right := x
   else                                |
      x := y.right                     |    if (y <> z) then
                                       |       z.key := y.key
   if (x <> nil) then                  |
      x.parent := y.parent             |    return y
                                       | end
```

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

▲ Určete časovou složitost jednotlivých operací nad BVS.

▲ **Def:** *Černobílý strom* je binární vyhledávací strom, jehož každý uzel je obarven černou nebo bílou barvou. Musí splňovat tyto podmínky:

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

Zkonstruujte černobílý 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.

Pěkný applet pro vyzkoušení obou operací nad černobílými stromy lze najít na `www.ececs.uc.edu/~franco/C321/html/RedBlack/redblack.html`

▲ Jaká je složitost jednotlivých operací nad BVS, pokud uvažujeme černobílé stromy?