7. přednáška

Vyhledávací tabulky I.

Rozpis přednášky

- Klasifikace metod.
- Sekvenční vyhledávání v souboru, poli, seznamu.
- Použití zarážky.
- Využití seřazení podle pravděpodobnosti vyhledávání,
- Adaptivní řazení podle četnosti vyhledávání v poli seřazeném podle klíče.
- Binární vyhledávání,
- Dijkstrova metoda, uniformní binární vyhledávání,
- Sharova metoda, Fibonacciho vyhledávání.

Základní pojmy a klasifikace

- přístupová doba (access time)
- minimální doba vyhledání, maximální doba vyhledání, průměrná doba vyhledání
- doba úspěšného a neúspěšného vyhledání
- klasifikace:
  - vyhledávání v tabulce implementované datovou strukturou s přímým přístupem
  - vyhledávání v tabulce implementované datovou strukturou se sekvenčním přístupem

Základní struktura vyhledávacího algoritmu

Nasel:= false;
while not nasel and <množina prvků není vyčerpána> **do begin**
    < prozkoumej další prvek, a je-li to hledaný, nastav Nasel na true >
end; (* while *)
Search:= Nasel

Pozor na ošetření konce cyklu

```
(* Pole *)
i:=1;
while (K<> Pole[i]) and (i <= max) do begin
   i:=i+1
end;
Search:= K= Pole[i]
```

V případě neexistence dojde k pokusu o referenci neexistujícího prvku Pole[max+1]!!!

Řešení:

```
Nasel:=false; i:=1;
while not Nasel and (i<=max) do begin
    if K=Pole[i]
    then Nasel:=true
    else i:=i+1
end;
Search:=Nasel
```

Řešení se zkratovým vyhodnocováním Booleovského výrazu.

B1 and B2 and B3 and .... and BN
V případě, že B1 je false, vše je false

B1 or B2 or B3 or .... or BN
V případě, že B1 je true, vše je true

Zkratové vyhodnocování se nastavuje překladači V rámci IAL ho budeme používat jen výjimečně!!

```
i:=1;
(* zkratové vyhodnocení Booleovského výrazu *)
while (i <= max) and (K<> Pole[i]) do
begin
   i:=i+1
end;
Search:= i<=max
```

Podobná situace nastává v seznamové nebo souborové struktuře:

```
Uk:=UkZac;
while (K<>Uk^.Klic) and (Uk<>nil) do begin (* chybná
   Uk:=Uk^.Puk
end;
```

Řešení zkratovým vyhodnocením

```
.....while (Uk<>nil) and (K<>Uk^.Klic) do ...
```

nebo raději:

```
Nasel:=false;
while not Nasel and Uk<>nil do begin
    if K=Uk^.Klic
    then Nasel:= true
    else Uk:= Uk^.Puk
end;
```

Pozn:

v cyklu while má složený Booleovský výraz
většinou tvar konjunkce, zatím co
v cyklu repeat má za until většinou tvar
disjunkce!!!

V opačných případech bývá zápis – byť správný –
méně srozumitelný!!!

K procvičení:
Zapište obdobný příklad chybného a správného
cyklu vyhledávání pro tabulku implementovanou
sekvenční strukturou soubor.

Metody implementace tabulky

- Sekvenční vyhledávání v neseřazeném poli
- Sekvenční vyhledávání v neseřazeném poli se zarážkou
- Sekvenční vyhledávání v seřazeném poli
- Sekvenční vyhledávání v seřazeném poli se zarážkou
- Sekvenční vyhledávání v poli seřazeném podle pravděpodobnosti vyhledání klíče

- Sekvenční vyhledávání v poli s adaptivním uspořádáním podle četnosti vyhledání
- Binární vyhledávání v seřazeném poli
  - normální binární vyhledávání
  - Dijkstrova varianta binárního vyhledávání
- Uniformní binární vyhledávání
- Fibonacciho vyhledávání
- Binární vyhledávací stromy (BVS)
- AVL stromy
- Tabulky s rozptýlenými položkami (Hashing tables) - TRP

Sekvenční vyhledávání v poli

datové typy:

```
const
   max=...;
type   (* typ položky tabulky *)
 TPol=record
      Klic:TKlic;
      Data:TData
end;
TTab=record
    Tab: array[1..max] of TPol; (* pole tabulky *)
  N: integer; (* aktuální počet prvků v tabulce*)
 (* typ tabulka implementovaná polem*
end;
```

Pozn.

Typ relace nad typem TKlic:

a) rovnost

b) relace uspořádání

Dohoda: pro typ klíče budeme používat nejčastěji identifikátor TKlic, pro název klíčové složky položky tabulky identifikátor Klic a pro hodnotu vyhledávaného klíče identifikátor K.

Sekvenční vyhledávání

```
function Search(T:Tab; K:TKlic):Boolean;
(* N je konstanta vyjadřující počet prvků v poli *)
var
i:integer; Nasel:Boolean;
begin
Nasel:=false;
  with T do begin
     i:=1;
     while not Nasel and (i<=N) do begin
         if K = Tab[i].Klic
         then Nasel:=true
         else i:=i +1
     end; (* while *)
     Search:=Nasel
   end (* with *)
end; (* function*)
```

Varianta operace Search za účelem vrácení polohy (indexu) nalezeného:

```
procedure SearchIns(T:TTab; K:TKlic; var
Nasel:Boolean;              var Kde:integer);
var i:integer;
begin
  Nasel:=false;
  with T do begin
     i:=1;
     while not Nasel and (i<=N) do begin
         if K = Tab[i].Klic
         then Nasel:=true
         else i:=i +1
     end; (* while *)
     kde:=i; (* pro Nasel=false je i nedefinováno*)
  end (* with *)
end; (* procedure *)
```

Operace insert:

```
procedure INSERT(var T:TTab; Pol:TPol; var
OverFlow:Boolean);
begin
    Overflow:=false; (* počát. nastavení příznaku plné tabulky *)
  SearchIns(T, K, Nasel,Kde);(* vyhledání za účelem vkládání *)
  if Nasel
    then T.Tab[Kde]:= Pol; (* přepsání staré položky *)
  else begin (* má se vložit nový prvek *)
   Kde:=T.N+1;
       if Kde <=max (* je v  tabulce místo ? *)
   then begin (* lze  vložit *)
    T.Tab[Kde]:=Pol;  (* vkládání *)
          T.N:=Kde;   (* aktualizace  počitadla *)
   end else Overflow := true (* nelze vložit – přeteč. *)
   end;
end; (* procedure *)
```

Operace Delete:

```
procedure Delete(var T:TTab; K:TKlic);
var
   kde:integer;
   Nasel:Boolean;
begin
   with T do begin
     SearchIns(T, K, Nasel, Kde);
     if Nasel then begin
        Tab[Kde]:=Tab[N]  (* rušený je přepsán
posledním *)
        N:=N-1;
     end; (* if *)
  end; (* with *)
end; (* procedure *)
```

Delete lze také implementovat zaslepením: Klíč rušené položky se přepíše hodnotou, která se nikdy nebude vyhledávat. Tím se ale snižuje aktivní kapacita tabulky !

Hodnocení metody

- minimální čas úspěšného vyhledání = 1
- maximální čas úspěšného vyhledání = N
- průměrný čas úspěšného vyhledání = N/2
- čas neúspěšného vyhledání = N
- Nejrychleji jsou vyhledány položky, které jsou na počátku tabulky

Vyhledávání se zarážkou – tzv. rychlé sekvenční vyhledávání

Zarážka (angl. sentinel, guard, stop-point) dovoluje vynechat test na konec pole. Efektivní kapacita tabulky se sníží o jednu položku. Vynecháním testu na konec se alg. zrychlí.

```
function SearchG(T:TTab; K:TKlic):Boolean;
var
  i:integer;
begin
   i:=1;
   T.Tab[T.N+1].Klic := K; (* vložení zarážky *)
  while K<>T.Tab[i].Klic do i:=i+1;
    Search:=i<>(T.N+1) (* když našel až zarážku, tak vlastně nenašel...*)
end; (* function *)
```

Sekvenční vyhledávání v seřazeném poli

a) Podmínka: nad typem klíč je definována relace uspořádání (dříve stačila relace rovnosti).
b) Pole je seřazeno podle velikosti klíče.
c) operace Search skončí neúspěšně, jakmile narazí na položku s klíčem, který je větší, než je hledaný klíč!
d) operace Search se urychlí jen pro případ neúspěšného vyhledávání. To je jediný význam vyhledávání v seřazeném poli!! Jinak se věci jen komplikují...!
e) Operace Insert musí najít správné místo, kam vloží nový prvek, aby zachovala seřazenost pole. Segment pole od nalezeného místa se musí posunout o jedničku. (N se zvýší o jedničku).
f) Operace Delete posune segmentem pole napravo od vyřazovaného doleva o jednu pozici a tím se vyřazovaný přepíše. (Počet prvků N se sníží o jedničku.)

a) Pozor: Posun segmentu doprava se dělá cyklem zleva a naopak!!!

Posun segmentu [Dolni..(Horni-1)] o jednu pozici doprava:

```
for i:= Horni to (Dolni + 1) do Pole[i] := Pole[i-1]
```

Vyhledávání v poli seřazeném podle četnosti vyhledání

- Nejvýhodnější by bylo uspořádání pole podle četnosti vyhledání tak, aby nejčastěji vyhledávané položky byly na počátku pole. To lze realizovat občasným seřazením položek podle počitadla, které se aktualizuje po každém přístupu k položce.
- Jinou variantou je vyhledávání s adaptivním rekonfigurací pole podle četnosti vyhledání. Po každém přístupu k položce se položka vymění se svým levým sousedem, pokud sama již není na první pozici:

Součástí vyhledávávacího cyklu této metody je příkaz:

```
if Kde>1
then T.Tab[Kde]:=:T.Tab[Kde-1]
```

kde zápis **„A:=:B“** označuje zkratku pro operaci výměny, kterou jinak zapisujeme trojicí příkazů:

```
Pom:=A;
   A:=B;
   B:=Pom;
```

Operaci :=: budeme někdy používat v písemném zápisu pro zkrácení a usnadnění zápisu – i např. u písemné zkoušky (nelze ji použít v programu).

Binární vyhledávání

- Binární vyhledávání se provádí nad seřazenou množinou klíčů s náhodným přístupem (v poli). Metoda připomíná metodu půlení intervalu pro hledání jediného kořene funkce v daném intervalu. Hlavní vlastností binárního vyhledávání je jeho složitost, která je v nejhorším případě logaritmická $\log_2(n)$.

- K samostatné úvaze. Nechť N je 1000. Porovnejte zaokrouhlenou hodnotu pro nejhorší případ binárního vyhledávání $\log_2(N)$ s hodnotou N pro nejhorší případ sekvenčního vyhledávání.

Pro pole tabulky implementované binárním vyhledávání platí:

Tab[1].Klic < Tab[2].Klic < ...< Tab[N].Klic

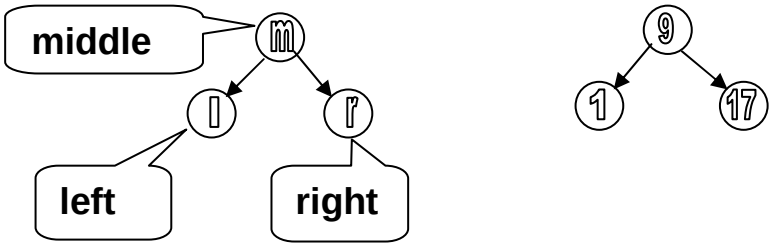
a pro vyhledávaný klíč musí platit:

(K >=Tab[1].Klic) and (K<=Tab[N].Klic

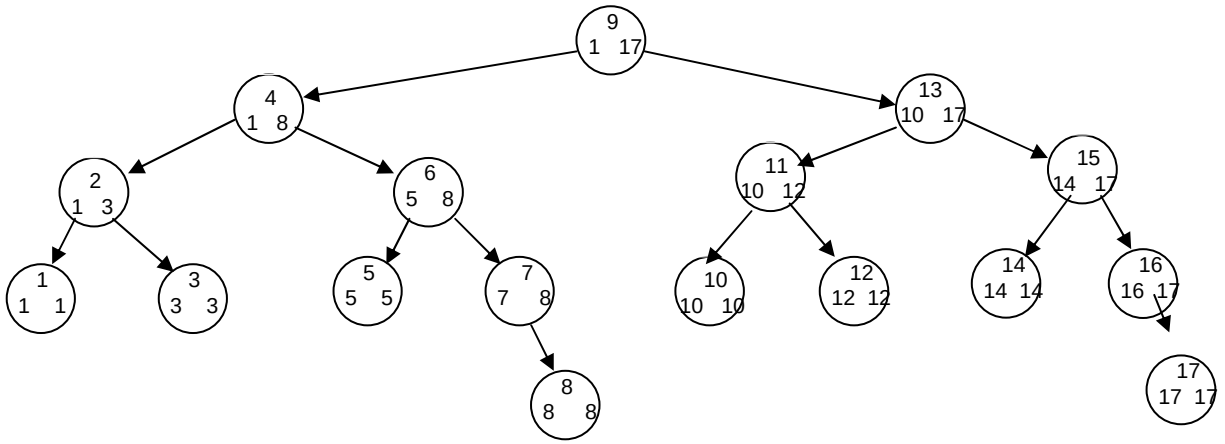
pak algoritmus vyhledávání tvoří sekvence příkazů:

```
...
left:=1;  (* levý index *)
right:=n;  (* pravý index *)
repeat
  middle:=(left+right) div 2;
  if K < Tab[middle].Klic
  then right:=middle - 1   (* hledaná položka je v levé polovině *)
 else left:= middle +1;   (* hledaná položka je v pravé polovině *)
until (K=Tab[middle].Klic) or (right < left);
Search:= K=Tab[middle].Klic;
```

Mechanismus výpočtu středu je ⇨(levý + pravý) div 2



Rozhodovací strom binárního vyhledávání popisuje proces vývoje výpočtu středu.



Dijkstrova varianta binárního vyhledávání

- E.W.Dijsktra – významný teoretik programování druhé poloviny minulého století.
- Dijsktrova varianta binárního vyhledávání vychází z předpokladu, že v poli může být více položek se shodným klíčem. (To se neočekává u vyhledávací tabulky. Tato varianta se používá pro jiné účely – a to pro účely řazení.)
- Je-li v seřazeném poli více klíčů se stejnou hodnotou, polohu kterého z nich má vrátit mechanismus Search? Obvyklým požadavkem je některý z krajních, a nejčastěji poslední ze stejných. Tomu odpovídá algoritmus, který nekončí tím, že najde shodu s klíčem, ale tím, že se dělením dostane na dvojici sousedních prvků:

Pro tuto variantu (hledání nejpravějšího) pak platí:

Tab[1].Klic <= Tab[2].Klic <= ... <= Tab[N-1].Klic **<** Tab[N].Klic
a také

(K >= Tab[1].Klic) and (K **<** Tab[N].Klic)

Pak Dijkstrova varianta má podobu sekvence příkazů:

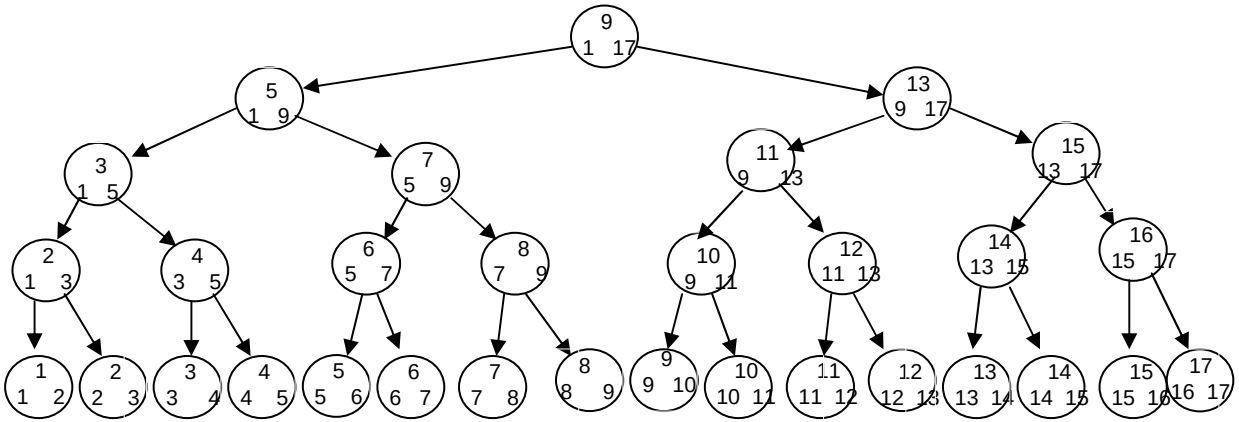
....

```
left:=1;
right:=n;
while right <> (left+1) do begin
  middle:=(left+right) div 2;
  if Tab[middle].Klic <= K
  then left:=middle
  else right:=middle
end;
Search:= K=Tab[left].Klic;
```

Příklad:
V poli: 1,2,3,4,5,5,6,6,**6**,8,9,13 najde algoritmus Klíč K=6 na 8. pozici:

V poli: 1,1,1,1,1,1,1,1,1,**1**,2 najde algoritmus klíč K=1 na 10. pozici.

Rozhodovací strom Dijsktrovy varianty pro pole [1..17] má tvar:



Dijkstrova varianta končí vždy za stejnou dobu, určenou hodnotou dvojkového logaritmu počtu prvků.

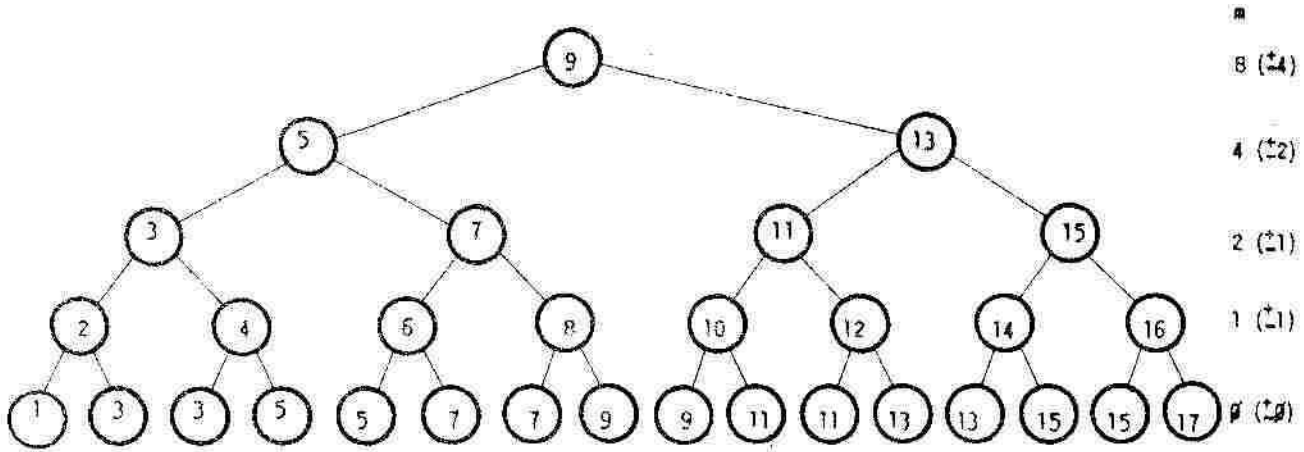
Závěr hodnocení binárního vyhledávání

a) Vyhledávání má logaritmickou složitost
b) Je zvlášť výhodné pro statické tabulky, kde není nutný potenciálně časově náročný posun segmentu pole
c) Operace Insert a Delete mají stejný charakter jako u sekvenčního vyhledávání v seřazeném poli.

Uniformní binární a Fibonacciho vyhledávání v seřazeném poli

- V některých počítačových systémech je multiplikativní operace (i půlení) časově složitá činnost a při mnohačetném opakování se ztrácí výhoda rychlosti.
- Uniformní binární vyhledávání je založeno na principu určení hranic intervalu odchylkou od středu. Pro danou tabulku se spočítá tabulka odchylek

Z následujícího obrázku je vidět, že levý i pravý okraj je od středu vzdálen o stejnou hodnotu. Na dané úrovni stromu jsou tyto odchylky stejné (uniformní). Tyto odchylky lze pro danou tabulku spočítat dopředu a vyhnout se operaci půlení.



Obr. 6.3: Rozhodovací binární strom pro uniformní binární vyhledávání

Nevýhodou metody by bylo, že při každé změně počtu prvků je třeba odchylky přepočítat. Ukazuje se však, že stromy, jejichž počet prvků je celou mocninou dvou mínus jedna, mají stejné tabulky odchylek.

Z toho vyplývá tzv. Sharova metoda, která umožňuje použití uniformního vyhledávání pro libovolný počet prvků:

První rozdělení pole se udělá na hodnotě vyhovující uniformnímu binárnímu vyhledávání (největší celé mocnině dvou menší než N).

- Je-li vyhledávaný klíč v levé části, vezme se jen levá část pole, která svou velikostí vyhovuje metodě.
- Je-li vyhledávaný klíč v pravé části, vezme se zprava jen tak velká část, jaká odpovídá vhodné metodě s tím, že se její levý index transformuje na „počátek“ pole.

Fibonacciho metoda je založena na Fibonacciho binárním stromu. Ten vychází z Fibonacciho posloupnosti, která je definována:

A[0]=0;
A[1]=1;
pro i>1 A[i]= A[i-1]+A[i-2]

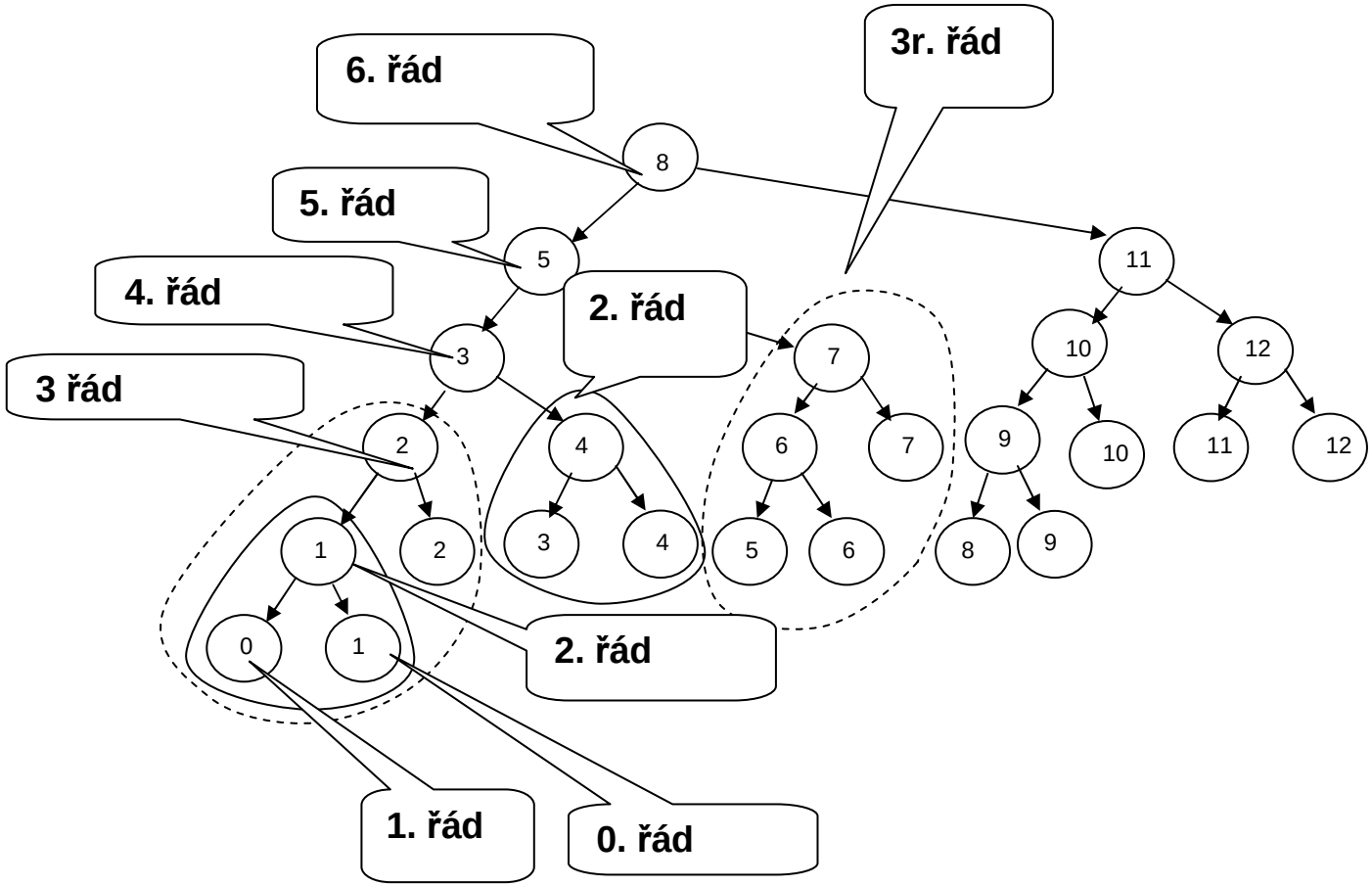
Následující prvek je dán součtem aktuálního prvku a jednoho jeho předchůdce.

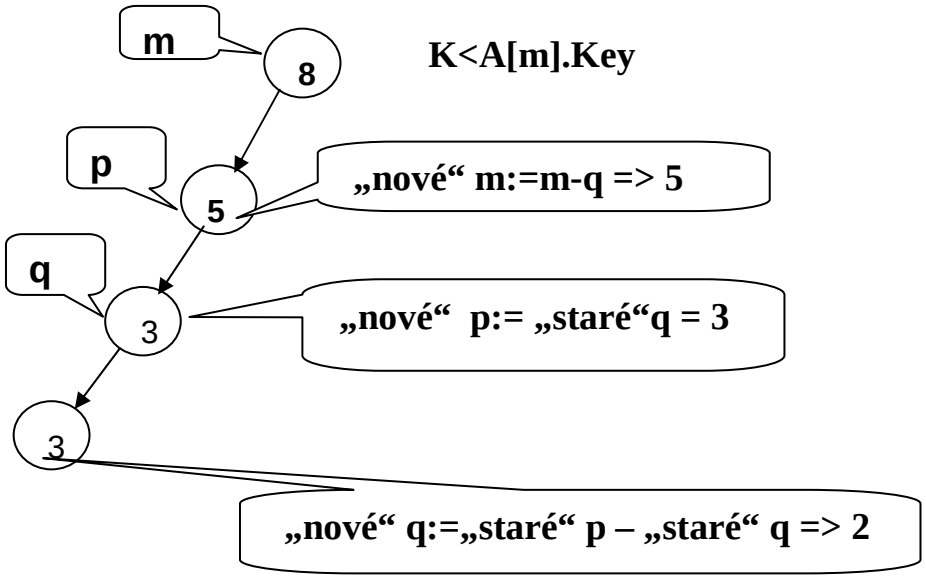
Pro počáteční hodnoty 0, a 1 má posloupnost tvar:

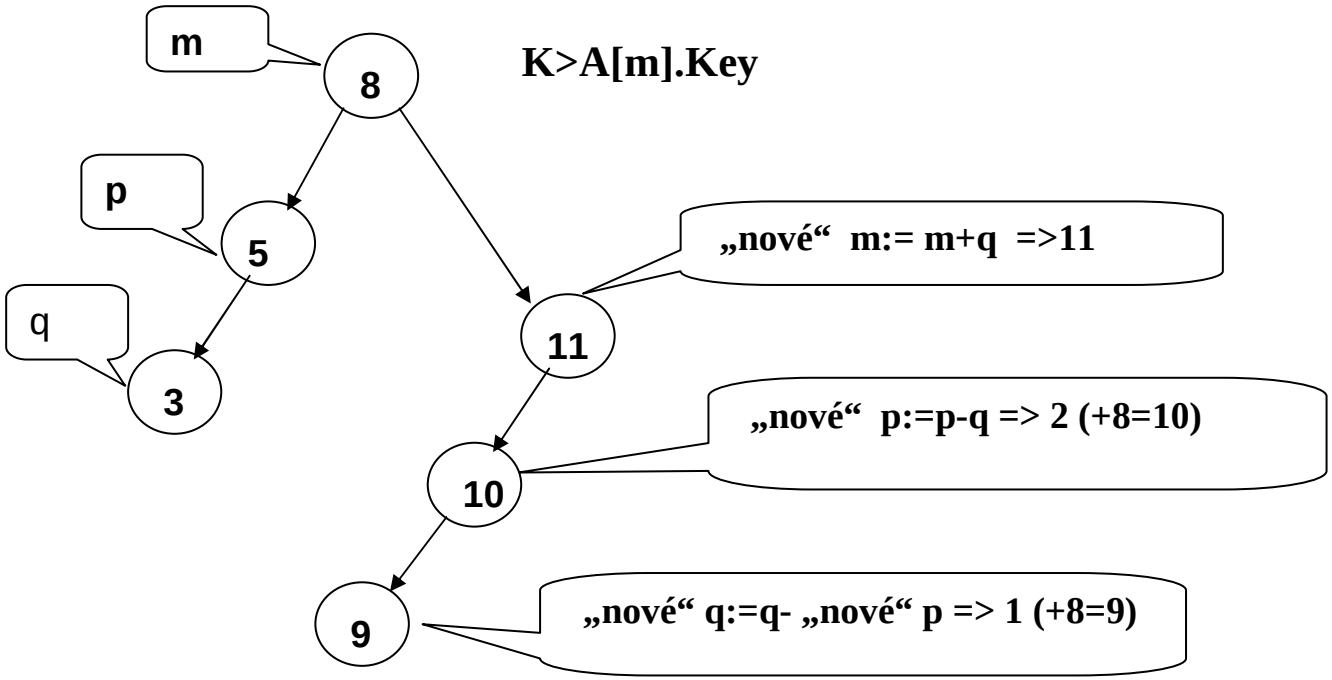
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55 atd.

Fibonacciho strom (F-tree) je definován pravidly:
F-tree i-tého stupně sestává z F(i+1)-1 non-terminálních uzlů a z F(i+1) terminálních uzlů.

Je-li i=0 nebo i=1 je strom reprezentován pouze kořenem a současně terminálním uzlem [0].
Je-li i>1, pak je kořen stromu reprezentován hodnotou F(i), jeho levý podstrom je F-tree řádu(i-1) a pravý podstrom je F-tree (i-2) řádu , v němž všechny hodnoty uzlů jsou zvýšeny o hodnotu kořene F(i).


6. řád
3r. řád
8
5. řád
5
11
4. řád
2. řád
3
7
10
12
3 řád
2
4
6
7
9
10
11
12
1
2
3
4
5
6
8
9
0
1
2. řád
1. řád
0. řád


m
8
K<A[m].Key
p
5
„nové“ m:=m-q => 5
q
3
„nové“ p:= „staré“q = 3
3
„nové“ q:=„staré“ p – „staré“ q => 2


m
8
K>A[m].Key
p
5
„nové“ m:= m+q =>11
q
3
11
„nové“ p:=p-q => 2 (+8=10)
10
9
„nové“ q:=q- „nové“ p => 1 (+8=9)

Vyhledávací algoritmus má tvar:

```
m:=F(l);
p:=F(l-1);
q:=F(l-2);
TERM:=false;
while (K<>A[m].Key) and not TERM do begin
  if K<A[m].Key
 then (* hledání pokračuje v levém podstromu *)
  if q=0
    then TERM:=true
                (* search končí na nulovém terminálu *)
```

```
else begin (* posun levého syna po diagonále *)
   m:=m-q;  p1:=q;
      q1:=p-q;  p:=p1;
      q:=q1;
    else (* search pokračuje v pravém podstromu *)
  if p=1
    then TERM:=true (* search končí na pravém
      jedničkovém terminálu prvního řádu *)
  else begin (* nastavení nových hodnot m, p a q v
                    pravém podstromu *)
   m:=m+q;
      p:=p-q;
      q:=q-p
    end; (* if
end; (* while *)
Search:= not TERM;
```

Kontrolní otázky

- Co je to strukturální ekvivalence?
- Co je to vyhledávání?
- Co je to přístupová doba?
- Má smysl použít cyklus FOR pro vyhledávání?
- Co je to zkratové vyhodnocování?
- Vysvětlete pojem zaslepení.
- Co je to zarážka?
- Jaká je výhoda vyhledávání v seřazené posloupnosti.

- Jak pracuje adaptivní sekvenční vyhledávání v neseřazené sekvenci
- Jak se konstruuje Fibonacciho strom?
- K čemu se používá Dijkstrova metoda binárního vyhledávání
- Co je to Sharova metoda