Vyhledávání ve stromově uspořádaných seznamech

Jan M Honzík

Ústav informatiky a výpočetní techniky FEI VUT v Brně, Božetěchova 2, 612 66 Brno 12, Česká republika

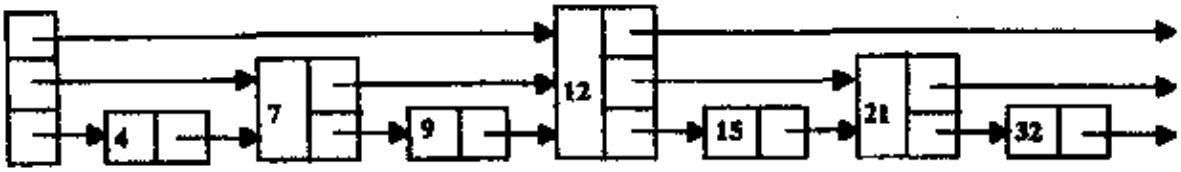
Abstrakt
Stromově uspořádané seznamy, nebo také "přeskakovací" seznamy (skip lists) jsou snadno implementovatelné datové struktury určené k vyhledávání. V rychlosti vyhledávání položky se zadaným klíčem se vyrovnají binárním stromům, vkládání a rušení položky je však snadnější, než u stromů, které provádějí znovuustavení vyváženosti.

1. Úvod

Ve sbornících předcházejících konferencí byly uvedeny příspěvky uvádějící většinu nejznámějších implementací a algoritmů pro vyhledávání. Vyhledávání založené na využití binárních stromů [3] zajišťuje dobu vyhledávání časovou složitostí řádu $\ln_2(n)$. Nevhodnou posloupností vkládaných (rušených) položek může strom degenerovat na lineární seznam. Výškově vyvážený (AVL) strom přináší pojem výšky. Je to *strom, pro jehož každý uzel platí, že výška levého a pravého podstromu je shodná, nebo se liší o jednu.* Každý uzel pak může být ve stavu rovnováhy, nebo je vpravo či vlevo *těžký*. Přidá-li se do pravého podstromu uzlu vpravo těžkého nový uzel tak, že se zvýší jeho výška, dojde k porušení rovnováhy. Symetricky tomu může být v levém podstromě. AVL stromy se v tom případě vyvažují prostřednictvím dvou typů operací zvaných *rotace*, které existují ve stranově symetrických dvojicích. Prostřednictvím vybrané rotace se znovuustaví vyváženost AVL stromu rekonfigurací několika uzlů v okolí tzv. *kritického uzlu*. Zápis operace insert a delete pro AVL strom není jednoduchý a delete navíc vyžaduje rekurzi nebo zásobník. Výsledná účinnost vyhledávání v takovém stromu je však natolik účinné, že vyspělejší programové vybavení používá pro vyhledávání v paměti většinou binárních AVL stromů. Náročnost zápisu operací AVL stromů vedla k dalším modifikacím. Mezi ně patří především *červeno-černé binární stromy* (red-black search binary tree) [5]. Princip binárního vyhledávání zůstává stejný. Rotace používané ke znovuustavení pravidel rozložení *černých* a *červených* uzlů, která zajišťují vyváženost stromu, jsou však jednodušší. *Šikmé stromy* (splay-tree) [6] je zvláštní typ binárního stromu, který po každém přístupu k uzlu provádí rekonfiguraci stromu takovou, že často vyhledávané uzly se hromadí v okolí kořene, zatím co nevyhledávané uzly se hromadí na obvodu větví stromu. To zvyšuje účinnost vyhledávání v případě nerovnoměrně rozložení pravděpodobnosti vyhledávání jednotlivých položek. *Přeskakovací seznamy* nebo *stromově uspořádané seznamy* jsou další variantou na stromy s logaritmickou efektivností vyhledávání a jednoduchou implementací operací insert a delete.

2. Princip přeskakovacího seznamu

Schema struktury ideálního přeskakovacího seznamu je uvedeno na obr.1. Seznam sestává z uzlů, které mají kromě vlastního informačního obsahu (vyhledávacího klíče a dat) 1,2,4,8,..., 2^n ukazatelů. V ideálním případě má každý druhý uzel dva ukazatele, každý čtvrtý tři ukazatele a každý 2^n má (n-1) ukazatelů. Uzly jsou seřazeny podle velkosti klíče.



Obr. 1 Schéma struktury přeskakovacího seznamu

Vyhledávání začíná na linii nejvyšší úrovně ukazatelů. Je-li klíč následujícího prvku seznamu větší, než vyhledávaný klíč, pak je-li úroveň ukazatele větší než jedna, sníží se úroveň linie ukazatelů a pokračuje se tak dlouho, dokud nedojde k jedné ze dvou podmínek:

- nalezení hledaného klíče - *vyhledávání končí úspěšně*
- vyčerpání seznamu (ukazatel na další je roven nilu) - *vyhledávání končí neúspěšně*

Je-li klíč následujícího prvku seznamu větší, než vyhledávaný klíč a úroveň linie ukazatelů je rovna jedné, *vyhledávání končí neúspěšně.*

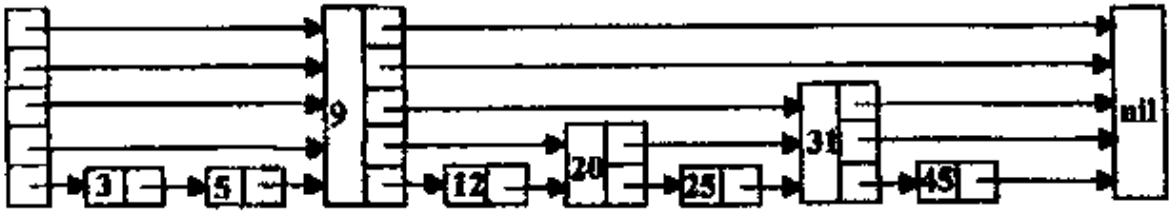
3. Přeskakovací seznam

Ideální přeskakovací seznam poskytuje nejlepší podmínky pro vyhledávání. Jeho udržování při dynamickém chování - v případě opakovaných operací insert a delete - by však bylo neúměrně náročné. V [2] byl uveden seznam, jehož přeskakovací uzly nebyly uspořádány ve struktuře pravidelně tak, jak je uvedeno na obr. 1. ale s využitím pravděpodobnostního rozložení.

Stojí za povšimnutí, že využití pravděpodobnostního rozložení hraje klíčovou roli v nejznámější a nejrychlejší metodě řazení - Quicksort [3]. Pro rozdělení pole čísel na část pole čísel větších a část pole čísel menších je nejvýhodnější medián, který pole rozdělí na dvě části o stejné velikosti a tím zajistí ideální výkon algoritmu. Hledání mediánu má však složitost, která by znehodnotila celý algoritmus. Genialita C.A.R. Hoara spočívala v nápadu nahradit medián *pseudomediánem*, za který je vzato číslo s náhodnou hodnotou, získané ze středu pole proto, aby se zabránilo nežádoucímu jevu v případě již seřazeného pole.

Na podobné úvaze je založena myšlenka, že přeskakovací uzly s mnohanásobnou úrovní ukazatelů nemusí být rozloženy s pravidelnosti uvedenou na obr.1. ale mohou být rozloženy na základě pravděpodobnostního rozložení tak, aby v průměru polovina uzlů měla 1 úroveň ukazatelů (jediný ukazatel ukazuje na následující prvek), čtvrtina by měla 2 úrovně, osmina 3 úrovně atd. Situaci naznačuje obr.2. Vkládání uzlu s náhodnou úrovní má za důsledek, že není zapotřebí rekonstruovat celý seznam. Zůstává skutečností, že v nejhorším případě se může stát, že seznam se dostane do stavu, kdy konstrukce jeho přeskakovacích uzlů způsobí při vyhledávání plné sekvenční prohlížení a přeskakovací seznam je degenerovaný na jednoduchý seznam. Podobné nebezpečí je však i u Quicksortu

nebo při nevhodné volbě rozptylovací funkce u tabulek s rozptýlenými položkami (Hashing tables). Takový případ nastává s velmi malou pravděpodobností a praxe prokázala úspěšnost Quicksortu i přeskakovacích stromů. Z inženýrského hlediska je nutné s tímto nejhorším případem počítat zejména v časově náročných sekcích programů pracujících v reálném čase.



Obr. 2 Příklad reálného přeskakovacího seznamu

Algoritmy pro vkládání a rušení položky vyhledávací tabulky implementované přeskakovacím stromem jsou podstatně jednodušší než při implementaci vyvažovaných stromů, při zachování srovnatelné časové složitosti. V následujícím odstavci jsou uvedeny úseky programu operací přeskakovacího seznam. Algoritmy vytvořila, a odladila a testovala *Kateřina Němcová, studentka 2.r .2.stupně oboru IVT FEI VUT v Brně*. Vytvořila také programy pro výukovou demonstraci a zobrazování seznamové vyhledávací struktury a porovnala rychlost operací s týmiž operacemi nad AVL stromem. Tyto programy nejsou předmětem příspěvku.

4. Implementace přeskakovacího seznamu

Struktura byla odvozena z obyčejného lineárního zřetězeného seznamu přidáním více ukazatelů do každého uzlu seznamu. Počet ukazatelů v uzlu se určuje pomocí pravděpodobnostní funkce, která využívá bitových součinů náhodně generovaného čísla se zadanou maskou. Tato maska je určena tzv. pravděpodobnostním parametrem ve tvaru $p=1/2^x$, kde x je počet jedničkových bitů v masce. Parametr má vliv na průměrný počet ukazatelů v uzlu a tím i vlastnosti celé struktury. Optimální volba je p=1/4.

Seznam je tvořen hlavičkou, která obsahuje nejvyšší možný počet ukazatelů (je určen konstantou MAXLEVEL) a jednotlivými uzly, které obsahují (kromě klíče a datové položky nesoucí užitečnou informaci) různý počet ukazatelů na své následníky. Nejvyšší aktuální počet ukazatelů je uchováván v proměnné ACTLEVEL. To je nutné z důvodu zachování konzistence seznamu po operacích vkládání a rušení uzlů. Při vytvoření nového seznamu (inicializace) je vytvořena hlavička a všechny její ukazatele se naplní hodnotou NIL.

Základní operace implementované nad seznamem jsou INSERT, DELETE a SEARCH:

Search

Vyhledávání začíná od hlavičky a to na nejvyšší aktuální úrovni ukazatelů - proměnná ACTLEVEL. Dokud je vyhledávaná hodnota menší než hodnota v uzlu, na který se příslušný ukazatel odkazuje, posouváme se o uzel dál. Pokud je hodnota v následujícím uzlu menší, rovna nebo se ukazatel odkazuje na NIL, provede se snížení úrovně a

postup se opakuje. Hledání končí vždy až na nejnižší úrovni. Pokud se hodnota uzlu, na kterém jsme skončili, shoduje s hledanou, je hledání úspěšné, v opačném případě nikoliv.

Insert

Skládá se ze tří částí - vyhledání místa pro vložení, vytvoření uzlu a jeho vložení a aktualizace ukazatelů. V první části se používá algoritmus jako při vyhledání, přičemž se vytváří pomocné pole update, do něhož se na příslušné úrovně ukládají ukazatele na uzly, jejichž pole ukazatelů bude nutno aktualizovat. Jsou to vždy uzly, které budou na dané úrovni předchůdci vkládaného uzlu. Po vyhledání místa je uzel vytvořen - počet ukazatelů určí pravděpodobnostní funkce. Připouští se vždy zvýšení aktuální úrovně ukazatelů (ACTLEVEL) maximálně o 1, aby nevznikaly zbytečně "vysoké" uzly. Pochopitelně nesmí počet přesáhnout konstantu MAXLEVEL. Na závěr se pomocí pole update aktualizují příslušné ukazatele.

Pozn.: Pokud se vkládaný uzel již v seznamu nachází je potřeba místo vložení buď aktualizovat datovou část nebo hlásit chybu - dle aplikace.

Delete

Operace se skládá ze tří částí. Při vyhledání se vytváří pole update, potom se provede aktualizace potřebných ukazatelů (jen těch, které se odkazují na rušený uzel). Na závěr je uzel zrušen a je nutno aktualizovat proměnnou ACTLEVEL pro případ, že rušený uzel byl nejvyšší v seznamu.

Uvedený kód byl odladěn v Borland Pascalu.

(* Deklarace konstant a typů a objektů *)

```
const
  Maximum = 16;                 (* Max. počet úrovní *)
  BitsInRandom = 31;            (* Max počet bitů náhodného čísla*)
  MaxRandom = 65535;            (* Maximální náhodné číslo *)

  NoKey = 0;                    (* Pomocná konstanta *)
  NoValue = 0;                  (* Pomocná konstanta *)

  head = chr(255);              (* Pomocná konstanta *)
  null = chr(0);                (* Pomocná konstanta *)

type
  TKey = integer;               (* typ klíč *)
  TValue = word;                (* data uzlu *)
  PTValue = ^TValue;            (* Ukazatel na data *)
  PNode = ^TNode;               (* Ukazatel na uzel *)
```

```
TNode = Object                    (* Abstraktní datový typ uzel *)
   key : TKey;                    (* klíč *)
   value : TValue;                (* hodnota*)
   valid : byte;
   forwardArr : array [0..Maximum-1] of PNode;        (* Pole ukazatelů *)
   constructor Init(K:TKey;V:TValue;Num:byte);
   destructor Done;
  end;


 PList = ^TList;                  (* Abstraktní datový typ přeskakovací seznam*)


 TList = Object
   ActLevel : byte;               (* Úroveň uzlu*)
    header : PNode;               (* Hlavička seznamu *)
    randomBits,randomsLeft:word;
    constructor Init;
    destructor Done;
    procedure Insert(Klic:TKey;Data:TValue);          (* Operace vkládání *)
    procedure Delete(Klic:TKey);                      (* Operace rušení*)
    function Search(Klic:TKey;var Data:TValue):boolean;  (* Operace vyhledávání *)
  end;


var                               (* Globální proměnné *)
 param,                           (* pravděpodobnostní parametr - maska*)
 MaxNumberOfLevels,               (* Maximální počet úrovní * )
 MaxLevel:byte;                   (* Nejvyšší úroveň *)


(* Implementace metod *)

constructor TNode.Init(K:TKey;V:TValue;Num:byte);
(* Inicializace vytvářeného uzlu *)
  var i:byte;
begin
   key:=K;              (* Vložení klíče *)
   value:=V;            (* Vložení dat *)
   valid:=Num;          (* Počet platných úrovní ukazatelů *)
   for i:=MaxLevel downto (MaxLevel-valid) do (* Nilování nepoužitých horních úrovní *)
     forwardArr[i]:=nil;
end; (* TNode.Init *)


destructor TNode.Done;
begin
end; (* TNode.Done *)
```

```
constructor TList.Init;
(* Inicializace seznamu *)
begin
   Randomize;
   randomBits:=Random(MaxRandom);
   randomsLeft:=BitsInRandom div 2;
   header:=new(PNode,Init(NoKey,NoValue,MaxLevel));
   Actlevel:=0;
end; (* TList.Init *)


destructor TList.Done;
var
  p,q : PNode;
begin
   p:=header;
   repeat
     q:=p^.forwardArr[0];
     Dispose(p,Done);
     p:=q;
   until (p=nil);
end; (* TList.Done *)


procedure TList.Insert(Klic:TKey;Data:TValue);
(* Operace vkládání položky do vyhledávací tabulky*)
var
  k : integer;
  update : array[0..Maximum] of PNode;
  p,q : PNode;
  count : word;


function randomLevel:byte;
(* Funkce určuje počet úrovní vkládaného uzlu *)
var
  level : byte;
  b     : word;
begin
  level:=0;
   repeat
     b:=randomBits AND param;
     if (b=0) then level:=level+1;
     randomBits:=randomBits shr 2;
     randomsLeft:=randomsLeft-1;
     if (randomsLeft=0)
     then  begin
       randomBits:=random(MaxRandom);
       randomsLeft:=BitsInRandom div 2;
```

```
    end; (* if *)
  until (b <> 0);
   if (level > MaxLevel)
   then randomLevel:=MaxLevel
   else randomLevel:=level;
end; (* function randomLevel *)


begin (* Tělo metody  *)
  p:=header;
  k:=Actlevel;
  repeat       (* Vyhledávací cyklus *)
    q:=p^.forwardArr[k];
    while ((q <> nil) and (q^.key < Klic)) do begin
       p:=q;
       q:=p^.forwardArr[k];
     end; (*while*)
     update[k]:=p;
     k:=k-1;
   until (k < 0);


   if (q^.key = Klic)
   then q^.value:=Data   (* Klíč se našel - Přepis dat *)
   else  begin           (* Vytvoření a vložení nového uzlu *)
     k:=randomLevel;
     if (k > ActLevel) then  begin
       k:=ActLevel+1;
       ActLevel:=k;
       update[k]:=header;
     end;  (* if k>ActLevel *)


      q:=new(PNode,Init(Klic,Data,k));
      repeat   (* Aktualizace ukazatelů; *)
        p:=update[k];
        q^.forwardArr[k]:=p^.forwardArr[k];
        p^.forwardArr[k]:=q;
        k:=k-1;
      until (k < 0);
    end;
end; (* TList.Insert *)


procedure TList.Delete(Klic:TKey);
var
  k,m : integer;
  update : array[0..Maximum] of PNode;
  p,q : PNode;
begin
```

```
  p:=header;
  k:=ActLevel;
  m:=k;
  repeat  (* Vyhledávací cyklus *)
    q:=p^.forwardArr[k];
    while ((q <> nil) and (q^.key < Klic)) do
      begin
        p:=q;
        q:=p^.forwardArr[k];
      end; (* while *)
    update[k]:=p;
    k:=k-1;
  until (k < 0);
  if (q^.key = Klic)
  then begin
      k:=0;
      p:=update[k];
      while ((k <= m) and (p^.forwardArr[k] = q)) do   (* Cyklus aktualizace ukazatelů *)
        begin
          p^.forwardArr[k]:=q^.forwardArr[k];
          k:=k+1;
          if (k <= m) then p:=update[k];
        end; (* while *)
      dispose(q,Done);
      while ((header^.forwardArr[m] = nil) and (m > 0)) do m:=m-1;
      ActLevel:=m;
    end; (* if (q^.key = Klic) *)
end; (* TList.Delete *)


function TList.Search(Klic:TKey;var Data:TValue):boolean;
(* Operace vyhledání prvku s klíčem Klic *)
var
  k : integer;
  p,q : PNode;
begin
    p:=header;
    k:=ActLevel;
    repeat
      q:=p^.forwardArr[k];
      while ((q <> nil) and (q^.key < Klic)) do  begin
          p:=q;
          q:=p^.forwardArr[k];
      end; (* while *)
      k:=k-1;
    until (k < 0);


    if (q^.key = Klic)
    then begin
```

```
    Data:=q^.value;
    Search:=true;
  end else
    Search:=false;


end; (*TList.Search*)
```

5. Závěr

Přeskakovací seznamy představují jednoduchou a snadno pochopitelnou implementační metodu pro vyhledávací tabulky. Rychlost jejich operací je srovnatelná se všemi vyvažovanými binárními stromy. Pokud nevadí nejhorší případ, který má vlastnosti sekvenčního vyhledávání v seznamu, lze její použití doporučit.

Literatura:

1. Schneider,B.: Skip Lists. Dr.Dobb's Journal, January1994, s50-52
2. Pugh,W.: Skip Lists: A Probabilistic Alternative to Balanced Trees. CACM,June 1990
3. Honzík,J.: Vyhledávání v binárních vyhledávacích stromech. Sborník konference Programování '85, DT ČSVTS Ostrava 1985, s 67-84
4. Hanzálková,M., Honzík,J.: Metoda řazení Quicksort.. Sborník knference "Algoritmy '87, Štrbské pleso, Jednota matematiků Bratislava, 1977
5. Honzík,J.: Vyhledávání ve stromových strukturách I.
6. Sborník konference Programování '93, DT ČSVTS Ostrava, 1993, s100-113
7. Honzík,J.: Vyhledávání ve stromových strukturách II. Sborník konference Programování '94, DT ČSVTS Ostrava, 1993, s 86-101