INFORMATIKA: NĚKOLIK ÚVODNÍCH TÉMAT NA ZÁVĚR

V úvodních textech k informatice (computer science) se objevují pojmy jako spojový seznam, strom, graf, to vše jako dynamické datové struktury. Jejich přesah do oblasti numerického modelování (scientific computing, computational science) není životně důležitý, skýtají však alternativu k ukládání dat pomocí polí a záznamů, o které je dobré vědět. Netriviální a přitom průzračné jsou i základní informatické algoritmy, ve kterých se záhy objevuje rekurze. Na pomezí informatiky a našeho oboru stojí problém třídění dat.
Všechny zmíněné struktury a algoritmy jsou podrobně popsány v Töpfer P., Algoritmy a programovací techniky (1995/2010), některé třídicí algoritmy také v Press W. H. et al., Numerical Recipes, Second Edition (1992), kap. 8 Sorting (www.nrbook.com/a/bookcpdf.php, www.nrbook.com/a/bookfpdf.php).

Dynamické datové struktury

Dynamické datové struktury zahrnují ve směru od jednoduchých ke složitějším: spojový seznam, strom a graf. Jejich společnou vlastností je realizace s explicitním výskytem ukazatelů v každém uzlu struktury a díky tomu možnost snadno spojovat a rozpojovat vazby mezi uzly. Jinou strukturou je seznam, např. zásobník a fronta, dalším pojmem je rozptýlená tabulka (hash table). Pěkným příkladem užití zásobníku a fronty jsou algoritmy prohledávání binárních stromů do hloubky a do šířky.

Spojové seznamy, stromy a grafy

Spojový seznam (linked list) je posloupnost záznamů téhož typu, obsahujících ukazatel na další uzel seznamu; lineární spojový seznam neobsahuje (obvykle nevítané) cykly. Spojové seznamy se mohou užít např. jako náhrada polí tam, kde je důležité rychle vkládat a vyjímat vnitřní prvky.
Deklarace typu pro spojový seznam (obsahuje odkaz na typ deklarovaný až následně):

jednosměrný seznam

```
Pascal: type pNode = ^tNode;
        tNode = record
                  Info : libovolny_typ;
                  Next : pNode;
                end;
```

obousměrný seznam

```
type  pNode2 = ^tNode2;
      tNode2 = record
                 Info : libovolny_typ;
                 Next, Prev : pNode2;  // ukazatele na sousední uzly
               end;
```

Alokace uzlu a inicializace položek

```
var p0,p : pNode;        // ukazatel na první a na průběžný uzel
New(p0); p0^.Info:=0; p0^.Next:=nil; p:=p0;
New(p^.Next); p:=p^.Next; with p^ do begin Info:=1; Next:=nil end;
```

Výpis položky Info (pokud byl předchozí řádek proveden třikrát)

```
writeln(p0^.Info, p0^.Next^.Info, p0^.Next^.Next^.Info, p0^.Next^.Next^.Next^.Info);
```

Přidání uzlu na pozici ukazatele p (s pomocným ukazatelem pNew)…

```
p:=p0; New(pNew); with pNew^ do begin Info:=0.5; Next:=p^.Next end; p^.Next:=pNew;
```

Vyjmutí uzlu z pozice za ukazatelem p (s pomocným ukazatelem pDel)

```
p:=p0; pDel:=p^.Next; p^.Next:=p^.Next^.Next; Dispose(pDel); pDel:=nil;
```

Srovnání s polem

- výhoda – snadné přidávání uzlů kamkoliv a vyjímání uzlů odkudkoliv s konstantní časovou složitostí, O(1)
- nevýhoda – pomalý přístup k uzlům s lineární časovou složitostí, O(n)

Strom (tree) je posloupnost záznamů vycházejících z jediného uzlu (kořen). Uzly obsahují ukazatele (větve) na potomky, uzly bez potomků se mohou nazývat listy. Binární strom (binary tree) je strom, z jehož uzlů vedou větve k nejvýše dvěma potomkům; obecný strom s odkazy na více potomků lze reprezentovat binárním stromem přesměrováním odkazů na prvního syna (dceru) a na sousedního sourozence (kanonická reprezentace stromu). Ve stromech nejsou přípustné cykly. Lineární spojový seznam je strom, z jehož každého uzlu pokračuje nejvýše jedna větev. Užitečnost stromů vyvstává například při vyhledávání údajů (vyhledávací stromy).
Deklarace typu pro binární strom:

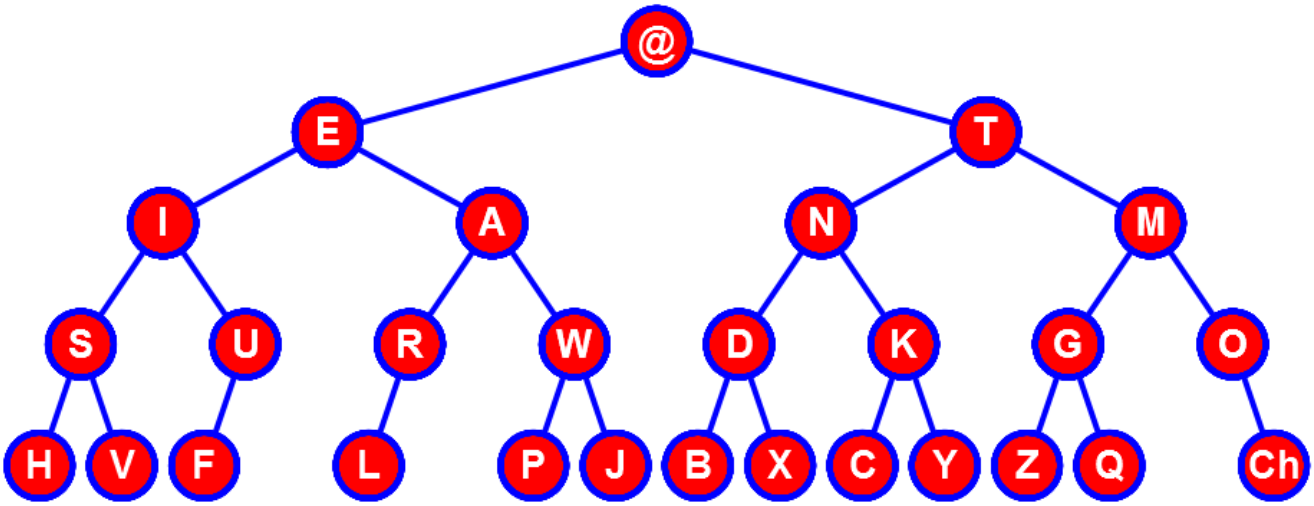
```
Pascal: type pNode = ^tNode;
        tNode = record
                  Info : libovolny_typ;
                  Left, Right : pNode;      // ukazatele na potomky
        end;
```

Binární halda (binary heap) je binární strom splňující požadavek na téměř vyvážený tvar (zaplňováním souvisle po hladinách a zleva) a na relaci uspořádanosti mezi uzly a potomky. Halda je užitečná k okamžitému poskytnutí extrému (ležícího v kořenu) a potažmo ke třídění dat (heapsort). Paradoxně, binární halda se snáze ukládá do pole než do struktury pro binární strom (protože $1+2+4+8+... = 2^n-1$, potomci i-tého uzlu mají indexy 2i a 2i+1).
Graf je soustava bodů (uzlů, vrcholů) se spojnicemi (hranami); graf může být orientovaný nebo neorientovaný (mají-li spojnice orientaci), ohodnocený nebo neohodnocený (mají-li spojnice ohodnocení), cyklický nebo acyklický (vytváří-li některá z cest mezi uzly cyklus). Strom je acyklický zakořeněný graf. K oblíbeným algoritmům patří hledání nejkratší cesty mezi vrcholy grafu a sestrojení minimální kostry grafu.

Seznamy a tabulky

Seznam (list) je posloupnost prvků se stanoveným pořadím. V praxi se velmi často používají zásobník a fronta.
Zásobník (stack): má neprostupné dno (nultý index), k přidávání (push) i vybírání (pop) prvků slouží druhý konec (vrchol zásobníku). Realizuje přístupovou metodu LIFO (last in, first out). Příklady: ukládání lokálních dat při volání procedur, ukládání mezivýsledků při vyhodnocování výrazů, narvaný autobus s jedněmi dveřmi.
Fronta (queue) má jeden konec vstupní, druhý výstupní. Realizuje přístupovou metodu FIFO (first in, first out). Příklady: vstup z klávesnice, zápis do souboru, instrukční fronta, narvaný autobus s dvojími dveřmi.
Rozptýlená tabulka (hash table) je pole velikosti N indexované prostřednictvím transformační (hash) funkce. Slouží k ukládání a následnému vyhledávání prvků, jejichž indexy (klíče) se prostírají ve velkém rozsahu (>> N), ale prvků samotných je obvykle méně (< N). Originální klíč K je transformován (nejsnáze modulárně, K mod N) na index v poli. Pro případ shody výsledného indexu pro více klíčů (kolize) musí být součástí algoritmu náhradní řešení, např. sekundární hash funkce nebo dynamická alokace pomocného místa.

Příklad. Binární strom s Morseovou abecedou (levé větve pro tečky, pravé pro čárky)
http://geo.mff.cuni.cz/~lh/NOFY056/cviceni-2015/#morseova-abeceda



Algoritmus prohledávání stromu do hloubky (průchod všemi uzly stromu)
průchod užitím rekurze: vypsat obsah kořenu; projít levý podstrom; projít pravý podstrom
průchod užitím zásobníku: vypsat obsah uzlu; uložit do zásobníku adresu obou podstromů, pokud existují; vybrat adresu ze zásobníku a posunout se na ni; to vše opakovat tak dlouho, dokud není zásobník prázdný
výsledek (pro 3 úrovně): kořen E I S U A R W T N D K M G O
Algoritmus prohledávání stromu do šířky
průchod užitím fronty: vypsat obsah uzlu; uložit do fronty adresu obou podstromů, pokud existují; vybrat adresu z fronty a posunout se na ni; to vše opakovat tak dlouho, dokud není fronta prázdná
výsledek (pro 3 úrovně): kořen E T I A N M S U R W D K G O

Poznámka k terminologii

Paměť programu rozvržená překladačem pro proměnné a další údaje bývá často členěna na části:
– statická paměť: pro statické globální proměnné a statické lokální proměnné hlavního programu,
– zásobník (stack): pro lokální proměnné procedur, návratové adresy, mezivýsledky při vyhodnocování výrazů,
– halda (heap): pro dynamicky alokované proměnné (např. dynamická pole).
Zásobník-paměť je spravován jako zásobník-seznam (metodou LIFO), halda-paměť nemá s haldou-stromem společného nic. Haldou je dynamická paměť nazývána pro nepořádek v ní obvykle panující, ostatně pořádek v ní se snaží udržovat mechanismus zvaný garbage collector.

Metody třídění dat

Tříděním dat (sorting) se rozumí řazení číselných a znakových dat podle velikosti. Vnitřní třídění se odehrává v paměti počítače, vnější třídění (pro velké soubory) odkládá data na pomalejší médium. Vnitřní třídění realizují jednodušší (přímé) pomalé metody, jejichž časová závislost na počtu dat je kvadratická, $O(n^2)$, složitější rychlé metody se závislostí O(n log n) a ještě rychlejší speciální metody. Sedm metod je zvlášť populárních; všechny jsou popsány včetně ukázek pro Pascal v Töpfer (1995), tři z nich také v Numerical Recipes, tam včetně zdrojových kódů pro C a Fortran. Příkaz sort je součástí Windows i Linuxu. Zdrojový kód pro třídění je přiložen i u těchto poznámek. Příbuznou úlohou je indexování dat (třídění pole integer indexů odkazujících na tříděná data podle klíčů odvozených z dat). Animace metod jsou ke shlédnutí na Wikipedii i jinde.

Pomalé metody

Společnou vlastností pomalých metod je kvadratická časová závislost $O(n^2)$ na počtu n vstupních dat, vyplývající z realizace pomocí dvou vnořených cyklů. Přiložený kód setřídí na běžném PC 10^5 náhodných integer dat pomocí insertion, selection a bubble sortu za 3, 7 a 36 s. Časová složitost insertion sortu se pro setříděná data zlepšuje na lineární (vnitřní cyklus degeneruje), bublinkové třídění je obecně pomalé pro příliš mnoho paměťových přesunů. Numerical Recipes předkládají také Shell sort (vícefázový insertion sort) s efektivní časovou složitostí $O(n^{1.25})$ až $O(n^{1.5})$.

Přímé vkládání (insertion sort): vezmi nesetříděný, vytvářej místo v setříděných, vhodně vlož tříděný
```
př. for i:=2 to n do begin x:=a[i]; j:=i; while x<a[j–1] do begin a[j]:=a[j–1]; {zmenši j} end; a[j]:=x end;
```
Přímý výběr (selection sort): najdi nejmenší nesetříděný, vlož za setříděné
```
př. for i:=1 to n–1 do begin j:=minloc(a,i,n); swap(a[i],a[j]) end;
```
Bublinkové třídění (bubble sort): posouvej nejmenší z nesetříděných směrem k setříděným
```
př. for i:=2 to n do begin for j:=n downto i do if a[j–1]>a[j] then swap(a[j–1],a[j]) end;
```

Rychlé metody

Elegantní algoritmy se superlineární časovou závislostí O(n log n). Quicksort a mergesort užitím rekurzivního volání pro data rozdělená na dvě části aplikují techniku „rozděl a panuj", heapsort řadí tříděná data do binární haldy. Metody se liší vnitřní složitostí, paměťovou náročností a citlivostí na seřazení vstupních dat. Přihrádkové třídění je ještě rychlejší, s lineární složitostí, lze však provést jen na datech s omezeným množstvím hodnot (integer, char). Přiložený kód třídí 10^7 náhodných integer dat následujícími metodami za 2, 7, 9 a 1 s. Profesionální kódy používají kombinace metod, např. quicksort přepínající v závěrečných fázích (pro málo dat) na insertion sort.

Quicksort: vyber pivot a zaměňuj větší než pivot zleva za menší zprava, pak rekurzivně totéž s oběma částmi
```
př. i:=1; j:=n; while i<j do begin {zaměňuj}; inc(i); dec(j) end; quicksort(1..i–1); quicksort(i..n);
```
Klíčový je výběr pivotu (srovnávací prvek): se štěstím, tj. je-li blízký mediánu (prostřední prvek), je časová závislost O(n log n), při smůle budou získané části nestejně velké a časová závislost klesne až k $O(n^2)$; přesto má třídění quicksortem nejlepší průměrnou rychlost. Potřeba zásobníku o velikosti O(log n).

Třídění sléváním (merge sort): setřiď rekurzivně obě poloviny, pak slévej z obou do pomocného pole
```
př. mrgsort(a_left); mrgsort(a_right); for i:=1 to n do {b[i] := a[j]<a[k] ? a[j] : a[k]; posouvej j, k}; b:=a;
```
Rychlost jen málo závislá na řazení vstupních dat, díky četnému kopírování menší než u ostatních algoritmů. Potřeba pomocného pole o velikosti O(n).

Třídění haldou (heapsort): sestav a rozeber binární haldu (speciální binární strom, viz výše)
```
př. for i:=1 to n do EnlargeHeap(a,i); for i:=n downto 1 do ReduceHeap(a,i);
```
Rychlost jen málo závislá na řazení vstupních dat, quicksort bývá mírně rychlejší. Žádný pomocný prostor, halda se vytváří v tříděném poli.

Speciální metoda pro celočíselná data

Přihrádkové třídění (bucket sort): rozmísti tříděná data do jimi indexovaných „přihrádek" a ty pak rozeber
```
př. for i:=1 to n do inc(p[a[i]]); k:=0; for j:=1 to np do for i:=1 to p[j] do begin inc(k); a[k]:=j end;
```
Časová složitost je lineární, O(n + np), kde np = maxval(a) – minval(a) + 1 je rozsah hodnot tříděných dat. Potřeba pomocného pole přihrádek indexovaných hodnotami vstupních (obvykle integer) dat.

Odkazy

animace třídění

www.sorting-algorithms.com

cs.wikipedia.org/wiki/Insertion_sort, .../Bubblesort, .../Quicksort, .../Mergesort, .../Heapsort

více metod a porovnání vlastností

en.wikipedia.org/wiki/Sorting_algorithm