v.0.03 16.02.2014

Náplň

- Jednoduché příklady na práci s poli v C
- Vlastnosti třídění
- Způsoby (algoritmy) třídění

Spojení dvou samostatně setříděných polí

```
void Spoj(double aPole1[], int aDelka1, double aPole2[],
          int aDelka2, double aPole[], int aDelka)
  {
  int i1=0, i2=0, iv=0; // inicializace proměnných
  if(aDelka1 + aDelka2 > aDelka)
      return 0; // špatná délka výsledného pole
  aDelka = aDelka1 + aDelka2; // délka pole po spojení

  while(iv < aDelka) // dokud není naplněno celé pole
     {
     if(aPole1[i1] < aPole2[i2]) // vlož menší prvek
         aPole[iv++] = aPole1[i1++];
     else
         aPole[iv++] = aPole2[i2++];
     }

  return aDelka;
  }
```

Otočte pořadí prvků v daném intervalu pole

```
int OtocVIntervalu(double aPole[], int aDelka, int aPocatek,
                   int aKonec)
  {
  double pom;
  if(aKonec > aDelka)
      aKonec = aDelka;

  --aKonec;

  for( ; aPocatek < aKonec; ++aPocatek, --aKonec)
      {
      pom = aPole[aPocatek];
      aPole[aPocatek] = aPole[aKonec];
      aPole[aKonec] = pom;
      }

  return 1;
  }
```

Třídění – sort

- patří k základním, často využívaným algoritmům
- existuje několik algoritmů, které se liší svou obtížností, rychlostí ...
- některé algoritmy jsou vhodnější pro určitý typ dat,
- rozdílnost, výhody jednotlivých algoritmů se projeví při velkých datových souborech
- možné typy vstupních dat: neseřazená, částečně seřazená, po částech seřazená, spojení dvou seřazených sad
- je možné přímé třídění originálních dat (jejich přeskládání).
  V případech, že je nutné zachovat pořadí původních dat (záleží nám i na čase, posloupnosti pořízených dat) potom je nutné použít pomocný datový prvek (indexy, klíče), pomocí kterého data „seřadíme“ i když původní data zůstanou neseřazena.
- pro řazení složitějších dat (struktur) podle jedné z hodnot může být důležité i zjistit zda při třídění zůstane dříve vložená struktura se stejnou hodnotou před druhou, nebo je možné, že se vymění

Třídění - pokračování

- data je možné předtřídit po částech,
- je možné celý soubor třídit v paměti, nebo využít externí úložiště (disk, pásku)
- algoritmy se liší i nároky na paměť - některé algoritmy pracují s pamětí původních dat, jiné potřebují paměť pomocnou (na mezivýsledky, pomocné výpočty (tabulky), na výsledné pole).
- řadit se dá sestupně nebo vzestupně (algoritmy jsou podobné)

Třídění – výpočetní náročnost

- není možné určit výpočetní náročnost absolutně, pouze statisticky
- na základě náročnosti a charakteru dat můžu určit vhodnost algoritmu
- náročnost může být paměťová, časová, výpočetní …
- pro každou metodu existují „lepší“ a „horší“ data (seřazená, náhodná, uspořádaná tak, že výhody metody jsou eliminovány). Existuje tedy nejlepší případ (například již seřazené pole), nejhorší pole (pro každý algoritmus jiná varianta), a „průměrný“ případ

Třídění – výpočetní náročnost

- u výpočetní náročnosti (složitosti algoritmu) se určuje, zda a jakým způsobem se bude zvětšovat výpočetní čas algoritmu v závislosti na zvyšování počtu vstupních dat.
  Máme-li počet vstupních dat N:
  Závislost je konstantní – značí se O(1), např. indexace pole
  Závislost lineárně narůstá – značí se O(N), například součet prvků pole o délce N
  Kvadraticky narůstá O(N^2)
  Exponenciálně narůstá O(N^N)...

- V reálných úlohách výpočetní náročnost závisí na charakteru vstupních dat. Proto je výhodné uvádět dvě hodnoty výpočetní složitosti: Horní odhad (nejhorší případ) a průměrnou hodnotu složitosti (určenou jako statistický průměr pro velkou množinu vstupních dat majících různý charakter)

Select sort (třídění výběrem)

Algoritmus

1) najděte maximální prvek (v poli)
2) vyměňte maximální prvek s prvním prvkem v poli
3) opakujte body 1,2,3 s polem o jedno kratším (zepředu) dokud je pole delší než jeden prvek

Náročnost pro N prvků je $O(N^2) = \frac{N^2}{2}$

Insert sort (třídění vkládáním)

Algoritmus
Vychází z toho, že pole má dvě části, setříděnou (na začátku nulová délka, nebo první prvek) a nesetříděnou (na začátku zbytek pole (N nebo N-1))

1) vezmi první prvek z nesetříděné části
2) posuň (konečnou) část setříděné části tak, aby vzniklo místo pro správné zařazení vyjmutého prvku
3) zařaď prvek
4) opakuj pro všechny prvky pole

Náročnost pro N prvků je $O(N^2) = \frac{N^2}{2}$

Bubble sort (bublinkové třídění)

- probublávání prvků,
- lze vytvořit různé varianty pro oba směry

Algoritmus

1) Postupně srovnávej sousední prvky (zprava). Pokud je první prvek (ten vlevo) menší, pak prvky vyměň (otoč). Tímto „probublá“ nejmenší prvek na konec (a v dalším už není nutné s ním pracovat).
2) Proveď bod 1) s polem o jedno kratším, pokud je (zbylá část) pole delší než jeden prvek

Algoritmus lze zrychlit například tím, že si pamatujeme místo, kde nastala poslední výměna – pole potom nezkracujeme o jeden prvek ale po místo poslední výměny

Merge sort (Třídění slučováním)

- slučování předtříděných dat
- slučování po částech předtříděných dat
- potřebuje pomocné pole
- rekurentní postup (algoritmus se provádí na (pod)části rozdělených dat)

Algoritmus

1) Pole rozdělíme na poloviny (je-li délka větší než jeden prvek) a opakujeme bod 1) pro všechny části
2) spojíme dvě předřazená pole do jednoho pole seřazeného

Náročnost pro N prvků je O(N.logN)

Quick sort (třídění rozdělováním)

- náročnost O(N.logN)
- rekurzivní algoritmus

Algoritmus

1) Náhodně se vybere prvek z pole (tzv. pivot)
2) pole rozdělíme na tři části - prvky větší než pivot, na pivot, a prvky menší než pivot
3) Na první a třetí část aplikujeme body (1,2,3) – pokud je délka větší než jeden prvek. Pivot je na svém místě.

Shell sort (shellův algoritmus)

- funguje jako insert sort, ale řadí prvky vzdálené od sebe o určitou vzdálenost
- vzdálenost byla původně n/2, nakonec jako nejlepší je zaokrouhlený násobek čísla 2.2
- tento algoritmus rychleji přesunuje vzdálené prvky

Algoritmus

1) zvol podle počtu prvků pole maximální krok (vzdálenost) pro porovnání
2) vyber z pole prvky vzdálené o daný krok a seřaď je insert sortem
3) krok 2 proveď pro všechny prvky ve vzdálenosti kroku (tj. proveď krok 2 pro všechny prvky – je-li vzdálenost 4, potom postupně tak aby začátek byl na prvním, druhém, třetím prvku (=4-1), čtvrtý prvek už byl zpracován při práci s prvním prvkem)
4) zkraťkrok a opakuj body 2 a 3

Heap sort (třídění pomocí haldy)

Patří k nejlepším řadícím algoritmům
Pro jeho realizaci se využívá halda (heap)
Realizace je pomocí složitějších struktur - stromů
Využívá binárního stromu – umožňuje rychlé vyhledání hodnoty v malém počtu kroků

Algoritmus je složitější na pochopení a vyžaduje pokročilejší programovací techniky – proto ho zde nebudeme uvádět.

Literatura

Kniha, ze které bylo čerpáno, která se věnuje algoritmům a mezi nimi třídění
Prokop, J.: Algoritmy v jazyku C a C++, Grada

Na následujících stránkách lze nalézt "pracující" příklady různých druhů třídění
http://www.algoritmy.net/article/154/Shell-sort