KSI 2012
Úloha 2-2: Informatický pětiboj

Jan Horáček
Gymnázium, Brno, Vídeňská 47; jan.horacek@seznam.cz

2. listopadu 2012

1 Úvod

Když někdo řekne "nejefektivnější řadící algoritmus", tak se mi okamžitě vybaví quicksort. Přesto, že jsem ho nikdy osobně neimplementoval, programátoři říklají, že na opravdu náhodná data v nekritických situacích je jeho aplikace velmi vhodná.

2 Zdrojový kód

K tomuto algoritmu není přiložen žádný spustitelný kód, pouze zde bude slovně popsán algoritmus, kterým lze daný problém efektivně vyřešit.

Tento popis řešení obsahuje úryvky zdrojového kódu. Tento zdrojový je psán syntaxí jazyka C.

3 Popis řešení

Hlaví jádro programu je založeno na quicksortu. Pak se zde nachází pomocné podprogramy, které se starají o obstarání dat pro řadící algoritmus a o výpis výsledků.

Následuje kompetní slovní popis funkce programu.

3.1 Parsing vstupních dat

Aby byl algoritmus nějakým způsobem použitelný, musíme mu dodat vstupní data.

Pro jednoduchost si představme vstupní data parsovaná ze souboru. To je samozřejmě jen příklad, do programu můžeme data zadávat třeba přes standarní vstup, ostatně podle koncepce UNIXu je všechno soubor.

Ve vstupním souboru bude na každém řadku jeden informatik (resp. jeho startovní číslo) a pak jeho výsledky v daných disciplínách oddělené například středníkem. Jelikož později budeme provádět součet bodů, na pořadí disciplín nezáleží. Vstupní data by tedy mohla vypada například takto:

```
1015;8;5;10;5;6
5963;9;2;3;9;10
```

Při načítání (parsingu) dat provedeme hned sečtení bodů za všechny závody a získání celkového počtu bodů daného závodníka. Spolu s jeho startovním číslem budeme data ukládat do jednoduché datové struktury, která může vypadat například takto:

```
typedef struct{
    int startNum;
    int body;
} tInformatik;
```

Listing 1: Definice datové struktury závodníka

Jelikož bodů bude maximálně 50, můžeme je ukládat do datového typu int, s kterým dnešní procesory pracují nejrychleji.

Dále zde bude dynamické pole právě těchto datových struktur.

3.2 Quicksort

Celý princip quicksortu je datailně popsaný na wikipedii a já budu psát spíše o tom, jak ho modifikovat pro tuto úlohu.

Vztahujme tedy následující problémy k algoritmu na wikipedii (verze v jazyce C):

1. **Volba pivotu**

   Na wikipedii se detailně píše o tom, jak je pivot důležitý zejména pro časovou náročnost algoritmu.

   Já osobně bych zvolil pivota jako medián 3 dosažených bodů na pseudonáhodných indexech polí daného řazeného rozsahu. Tato cesta mi přijde relativně rychlá, jednoduchá a hlavně dostaneme relevantní výsledek. Výběr mediánu ze 3 bych provedl nějakým řadícím algoritmem: zde postačí obyčejný bubble sort, ale pokud by někdo stál o zlepšení časové náročnosti, může samozřejmě implementovat libovolný řadící algoritmus.

2. **Přístup k poli struktur**

   Jediná změna oproti kódu na wikipedii by byla ta, že místo $array[x]$ by v programu bylo $array[x].body$. Prohození 2 prvků mezi sebou by šlo provádět stejným způsobem, ale pom by musel být datového typu $tInformatik$.

3. **Nestabilita algoritmu**

   Pokud budeme tento algoritmus pouštět na slušných zařízeních, které odpovídají 21. století, nemělo by dojít k přeplnění zásobníku. Samozřejmě, pokud do programu vložíme například informatiky, jejichž počet bude přesahovat int, algoritmus na tom bude špatně a asi pravděpodobně skončí na "Stack overflow". V takovém případě bych doporučoval upravit Quicksort na stabilnější, ale více prostorově náročnou verzi.

3.3 Výstup

Při výpisu výstupu (ať už kamkoliv) je zapotřebí myslet na to, že v zadání se píše, že závodníci se stejným počtem bodů mají být na stejné pozici. To je zapotřebí uvažovat.

Tedy potřebujeme zjišťovat, jestli informatici mají stejný počet bodů a pouze, pokud má další informatik menší počet bodů, tak zvýšit počítadlo umístění. Zdrojový kód může vypadat třeba takto:

```
int poscnt = 1,bodtmp = 51;   //zaciname na 1. miste, max 50 bodu
for (int i=(count-1);i>=0;i--)   //count = pocet prvku pole
{
   if (zavodnici[i].body < bodtmp)
   {
      //dalsi misto v poradi
      bodtmp = zavodnici[i].body;
      poscnt++;
      printf("%2d.: %2d %d",poscnt,
      zavodnici[i].body,zavodnici[i].startNum);
   } else {
      printf("    : %2d %d",zavodnici[i].body,zavodnici[i].startNum);
   }
}
```

Listing 2: Výpis výstupu

4 Závěr

Při průměrných vstupních hodnotách lze říci, že časová náročnost algoritmu je $O(NlogN)$, ba i lepší, protože jsme pivota zvolili relativně sofistikovaně a přesně.

Z popisu řešení jednoznačně vyplývá, že je algoritmus správný.

Reference

[1] Wikipedia *Quick sort*
http://cs.wikipedia.org/wiki/Quicksort