INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ

Číslo projektu	CZ.1 .07/1.1 .36/02.0066
Autor	Ladislav Kašpárek
Předmět	Programování a vývoj aplikací
Téma	Kvadratické řadící algoritmy
Metodický pokyn	výkladový text s ukázkami

Kvadratické řadící algoritmy

Problém

Vstup: Máte zadáno pole čísel, nebo čehokoliv jiného, co se dá a co má smysl řadit.
Úkol: Máte najít algoritmus, který seřadí podle zadaného pravidla prvky v poli.

Nyní podrovněji

Máme zadáno pole - tzn. homogenní datovou strukturu, u které máme přístup k libovolné její položce v konstantním čase na základě indexu.
Máme najít algoritmus, který seřadí prvky podle zadaného pravidla, (u čísel obvykle velikost), ale obecně by nám měla postačit jakákoliv funkce, která pro dva prvky určí jejich pořadí.

Co nás zajímá?

U řadících algoritmů nás zajímá, podobně jako u jiných, jejich složitost. Čili jak roste výpočetní čas (doba trvání algoritmu), když zvyšuji požadavky (rozsah vstupních dat).
Složitost řazení se zapisuje jako funkce, která nám říká, kolik musíme udělat elementárních operací nad zadanými daty, abychom mohli uživateli s jistotou říci, že nyní je celé pole seřazeno.
U řazení to bývá trochu problém, protože mohu měřit dvě elementární operace:
Jednak porovnání dvou prvků (abych zjisitl zda jsou ve správném pořadí) - to musím provést vždy.
A jednak přesun prvků - to se někdy provádět nemusí (protože třeba už jsou ve správném pořadí).

Bubble sort

Základní myšlenka:

Prvním řadícím algoritmem je tzv. bublinkové řazení - bubble sort.
Princip spočívá v tom, že se postupně prochází celé pole a porovnávají se vždy dva sousední prvky.
Pokud jsou v nesprávném pořadí, vymění si pozice.
Jakmile takto projdeme celé pole, máme jistotu, že nejméně jeden prvek je na svém místě.

Pole o tento prvek zkrátíme a celý postup opakujeme znovu.

Kód:

```
public static void BubbleSort(int[] pole) {
    for (int i = 0; i < (pole.length - 1); i++) {
        for (int j = 0; j < (pole.length - 1 - i); j++) {
            if (pole[j] > pole[j + 1]) {
            int tmp = pole[j];
                pole[j] = pole[j + 1];
                pole[j + 1] = tmp;
            }
        }
    }
}
```

Složitost :

```
PorovnaniBubble = Sum[N - i, {i, N}]
```

$$\frac{1}{2}\left(-N+N^2\right)$$

```
PresunyBubbleMin = 0;
PresunyBubbleMax = PorovnaniBubble;

PresunyBubbleAvg = (PresunyBubbleMin + PresunyBubbleMax) / 2
```

$$\frac{1}{4}\left(-N+N^2\right)$$

Zhodnocení :

U porovnání i u přesunů nám vyšly velmi podobné výrazy. Z hlediska asymptotické složitosti nás zajímá “nejsložitější” funkce a to je N^2. Musíme tedy konstotovat, že složitost bubble sortu je kvadratická.

Select sort

Základní myšlenka:

Princip spočívá v tom, že se v poli najde buď maximum, nebo minimum a tento prvek se prohodí s prvkem na posledním, resp. prvním místě.
Poté postup opakujeme, s vynecháním tohoto posledního / prvního prvku.
Po proběhnutí každé jedné fáze máme vždy nejméně jeden prvek na správné pozici.

Kód:

```
public static void SelectSort(int[] pole) {
    for (int i = 0; i < pole.length - 1; i++) {
        int max = pole.length - 1 - i;
        for (int j = 0; j < pole.length - i; j++) {
            if (pole[j] > pole[max]) {
            max = j;
            }
        }
        int pom = pole[pole.length - 1 - i];
        pole[pole.length - 1 - i] = pole[max];
        pole[max] = pom;
    }
}
```

Složitost :

```
PorovnaniSelect = Sum[N - i, {i, N}]
```

$$\frac{1}{2}\left(-N+N^2\right)$$

```
PresunySelect = N - 1;
```

Zhodnocení :

Z asymptotického hlediska je “nejhorší” funkcí z obou výrazů opět N^2, takže opět musíme konstatovat, že i select sort má kvadratickou složitost.
Ale je doufám patrné, že tím že jsme prohazování prvků dali do vnějšího for-cyklu jsme dokázali snížit počet přesunů na lineární.

Insert sort

Základní myšlenka:

Princip spočívá v tom, že se postupně prochází prvky a každý nesetříděný se zařadí na správné místo.
K tomu je nutné mít pole již částečně setříděné, to se řeší tak, že se první prvek považuje za setříděný a začne se až od druhého.
Všechny prvky se v průběhu jednotlivých fází postupně posouvají blíže a blíže svým správným místům.
Dokud ale neproběhne všech N-1 fází nemůžeme s jistotou říci, že je nějaký prvek na svém místě, nebo že je posloupnost setříděná.

Kód:

```
public static void InsertSort(int[] pole) {
    for (int i = 1; i < pole.length; i++) {
        int ins = pole[i];
        int j = i - 1;
        while ((j >= 0) && (pole[j] > ins)) {
            pole[j + 1] = pole[j];
            j--;
        }
        pole[j + 1] = ins;
    }
}
```

Složitost :

```
PorovnaniInsertMin = N - 1
```

$$-1+N$$

```
PorovnaniInsertMax = Sum[i, {i, N - 1}]
```

$$\frac{1}{2}\,(-1+N)\,N$$

```
PorovnaniInsertAvg = Simplify[(PorovnaniInsertMin + PorovnaniInsertMax) / 2]
```

$$\frac{1}{4}\left(-2+N+N^2\right)$$

```
PresunyInsertMin = PorovnaniInsertMin * 2
```

$$2\,(-1+N)$$

```
PresunyInsertMax = PorovnaniInsertMax + 2
```

$$2 + \frac{1}{2} (-1 + N) N$$

```
PresunyInsertAvg = Simplify[(PresunyInsertMin + PresunyInsertMax) / 2]
```

$$\frac{1}{4} N (3 + N)$$

Zhodnocení :

Z asymptotického hlediska je “nejhorší” funkcí z obou výrazů opět N^2, takže i zde musíme konstatovat, že insert sort má kvadratickou složitost.
Ale je zjevné že nyní ani porovnání nemusí být kvadratický počet, takže šetříme i na nich.

Závěrečné porovnání :

Z asymptotického hlediska jsou si tyto algoritmy rovny.
Všechny mají kvadratickou složitost.
Nic méně je vidět, že na reálných datech by byl bubble nejpomalejší, select sort by byl druhý a insert sort by běžel nejrychleji.

Zdroje

http://reference.wolfram.com/mathematica/ref/Sum.html