Pracovní list 12: Řazení

Co už máme znát

- rozdělení řadicích metod;
- vlastnosti řadicích metod;
- principy řadicích metod;
- implementace programového modulu;
- experimentální stanovení časové složitosti;
- vliv implementačních konstant časové složitosti.

Kontrolní otázky

12.1 Na jaké kategorie lze rozdělit řadicí metody?

12.2 Jak je definována přirozenost řadicí metody?

12.3 K čemu lze využít stabilitu řadicí metody?

12.4 Kdy lze použít sekvenční řadicí metodu?

12.5 Jaký je princip řazení metodou Heap sort?

12.6 Jaký je princip řadicí metody Quick sort?

12.7 Které obecné metody (použitelné na obecná data) mají lineárně logaritmickou složitost?

12.8 Jak může být modifikována časová složitost stromového řazení v závislosti na řazených datech?

12.9 Za jakých okolností lze dosáhnout teoreticky nejlepší časové složitosti – lineární?

12.10 Jak se postupuje při experimentálním stanovení časové složitosti algoritmu?

12.11 Jak lze porovnat implementační konstanty dvou algoritmů stejné třídy časové složitosti?

Příprava na cvičení

Ve cvičení budeme potřebovat překladač jazyka C++, editor pro přípravu zdrojových textů a vybavení příkazového řádku. Pro jednotlivé úlohy jsou k dispozici soubory s daty, případně výsledné soubory v adresáři `/home/rybicka/vyuka/progt/cecko/cviceni/cv12` na serveru `akela`. Konkrétní jména těchto souborů jsou uvedena u jednotlivých úloh.

Řešené příklady

Příklad 12.1 Pro analýzu řadicích metod vytvořte modul, implementující pole s milionem složek typu `float`, které bude naplněno ze standardního vstupu určitým počtem hodnot a na které bude aplikována zvolená řadicí metoda. Tuto metodu modul získá jako parametr (metoda bude reprezentována datovým typem podprogram). Zároveň bude implementována možnost měření času, jak už bylo řešeno ve cvičení týkajícím se časové složitosti.

Řešení: Pro implementaci takového modulu potřebujeme navrhnout jeho základní způsob práce – jde o proceduru, která ze standardního vstupu přečte potřebný počet údajů a naplní pole, nastaví měření času, spustí řadicí metodu, znovu odečte čas a vypíše výsledek. Požadovaný počet zpracovávaných hodnot bude zadán jako parametr z příkazového řádku. Nebude-li parametr zadán, bude se jako výchozí hodnota brát milion hodnot připravených v souboru `nahcisla.txt`. Název modulu zvolíme `razeni`. Hlavičkový soubor pak bude mít následující tvar:

```
747 #ifndef RAZENI_H
748 #define RAZENI_H
749
750 const int Rozmer = 1000000; //max. počet údajů ve struktuře
751 typedef float TypPole[Rozmer]; //datový typ pracovního pole
752
753 typedef void (*Serazeni)(TypPole P, int Kolik); //řadicí metoda
754
755 void Zamena(TypPole P, int A, int B);
756     //často použitelná procedura pro záměnu prvků v pracovním poli
757 void Proces(TypPole P, int Kolik, Serazeni Metoda);
758     //hlavní procedura provádějící proces testu řadicí metody
759
760 #endif
```

Implementační část modulu v souboru `razeni.cpp` realizuje naznačené operace. Případné komentáře jsou uvedeny v kódu:

```
761 #include ”razeni.h”
762 #include <iostream>
763 #include <ctime>
764 using namespace std;
765
766 void Napln(TypPole P, int &Kolik){
767 //naplnění pole ze vstupu, test na přítomnost dostatku hodnot
768   int i;
769   for (i=0; (i<Kolik and cin>>P[i]); i++);
770   Kolik = i; //Kolik udává skutečný počet načtených čísel
771 }
772
```

```
773 void Zamena(TypPole P, int A, int B){
774 //pomůcka pro více řadicích metod
775   float Pom;
776   if (A!=B) {
777     Pom = P[A];
778     P[A]=P[B];
779     P[B]=Pom;
780   }
781 }
782
783 void Proces(TypPole P, int Kolik, Serazeni Metoda){
784   long start, konec; //časové údaje
785   Napln(P, Kolik);   //naplnění pole ze vstupu a kontrolní výpis
786   cout << ”Naplněno ”<< Kolik << ” položek.”<<endl;
787   start = clock();
788   Metoda(P, Kolik); //seřazení zvolenou metodou a měření času
789   konec = clock();
790   cout << ”Řazeno ”<<Kolik<<” položek, čas: ” <<konec-start << ” (”
791        << float(konec-start)/CLOCKS_PER_SEC << ” sec.)”<<endl;
792 }
```

Příklad 12.2 Využijte modul z příkladu 12.1 a testujte přirozenou variantu bublinového řazení.

Řešení: Program využívající již implementovaný testovací modul musí obsahovat řadicí metodu vyhovující datovému typu `Serazeni`, jejímiž parametry jsou pracovní pole a okamžitý počet naplněných a zpracovávaných hodnot. Pro zjištění aktuálního počtu požadovaných hodnot je potřebné zpracovat parametr z příkazového řádku, pokud je zadán. Není-li zadán, automaticky se použije konstanta definovaná v modulu, tj. `Rozmer`. Implementace metody a celý hlavní program již nepotřebuje další podrobnější komentáře:

```
793 #include <iostream>
794 #include <cstdlib>
795 #include ”razeni.h”
796 using namespace std;
797
798 void bubble(TypPole P, int Kolik){
799   bool jeste=true;
800   int i=1;
801   while (jeste) { //dokud nepřestanou výměny
802     jeste = false;
803     for (int j=0; j<Kolik - i; j++)
804       if (P[j]>P[j+1]) {
805         Zamena(P, j, j+1);
806         jeste=true;
807       }
```

```
808       i++;
809     }
810   }
811
812   int main(int pocet, char *param[]) {
813      int Pozadovano=Rozmer;
814      if (pocet>1) Pozadovano=strtol(param[1],NULL,10);
815     //zpracování parametru
816      float *Pole = new float[Pozadovano]; //alokace prostoru
817      Proces(Pole, Pozadovano, bubble);  //naplnění, seřazení, změření
818      return 0;
819   }
```

Příklady

Příklad 12.3 Stanovte experimentálně časovou složitost implementované metody bublinového řazení z příkladu 12.2. Využijte připraveného souboru `nahcisla.txt` s milionem hodnot.

Řešení: Pozor, nechcete-li čekat na seřazení všech hodnot celý týden, použijte jen úměrné množství. Do sta tisíc se ještě dočkáte...

Příklad 12.4 Implementujte metodu řazení v poli přímým vkládáním v základní variantě. Porovnejte časovou složitost a implementační konstantu s předchozí metodou.

Příklad 12.5 Využijte faktu, že přirozená varianta bublinového řazení je sekvenční, a implementujte ji v lineárním jednosměrném dynamickém seznamu. Zjistěte, zda se časová složitost liší od řazení v poli. Zjistěte také, zda se liší implementační konstanta od řazení v poli. K tomuto účelu modifikujte potřebným způsobem modul z příkladu 12.1.

Příklad 12.6 Implementujte metodu řazení pomocí uspořádaného binárního stromu a zjistěte její časovou složitost.

Příklad 12.7 Modifikujte metodu řazení přímým vkládáním tak, aby se vyhledávání vhodného místa pro umístění nové položky realizovalo metodou půlení intervalu. Zjistěte časovou složitost.

Příklad 12.8 Implementujte řadicí metodu Heap Sort. Stanovte časovou složitost a porovnejte s implementací z úloh 12.3 a 12.4.

Příklad 12.9 Implementujte řadicí metodu Quick Sort. Opět zjistěte časovou složitost.

Příklad 12.10 Porovnejte implementační konstanty všech čtyř implementovaných lineárně logaritmických metod (příklad 12.6, 12.7, 12.8 a 12.9).

Příklad 12.11 Implementujte řazení (multi)množinou pro seřazení celých čísel ze souboru `celacisla.txt`. Rozsah hodnot v tomto souboru je $\langle -9999; 9999 \rangle$. Opět stanovte časovou složitost a porovnejte se všemi předchozími metodami. Z implementovaného modulu využijte část týkající se měření času.

Příklad 12.12 Implementujte řazení hešováním pro reálná čísla (soubor `nahcisla.txt`). Pro návrh hešovací funkce využijte faktu, že jde o čísla se čtyřmi desítkovými řády před desetinnou čárkou. Zjistěte časovou složitost. Zároveň zjistěte maximální počet synonym.

Co máme po cvičení umět

- kvadratické řadicí metody,
- lineárně logaritmické řadicí metody,
- lineární řadicí metody,
- určení časové složitosti,
- určení vztahu implementačních konstant dvou algoritmů téže třídy časové složitosti.

Kontrolní otázky

12.12 Jaká je nevýhoda řadicí metody Quick sort?

12.13 Je řazení v lineárním jednosměrném seznamu přirozenou variantou bublinového řazení zásadně pomalejší než řazení stejnou metodou v poli?

12.14 Je řadicí metoda Heap sort stabilní?

12.15 Jakou výhodu má stromové řazení?

12.16 Jak vypadá monotónní hešovací funkce pro řazení hešováním?

12.17 Jak lze podle implementační konstanty seřadit lineárně logaritmické metody implementované v úlohách tohoto pracovního listu od nejrychlejší po nejpomalejší?

12.18 Jak se principiálně liší řazení množinou a řazení hešováním v příkladech 12.11 a 12.12?