Mittuniversitetet
MID SWEDEN UNIVERSITY

DT046G, DT064G Datastruktuer och algoritmer

Laboration:
Analys av sorteringsalgoritmer

Martin Kjellqvist

lab.tex 162 2016-10-06 13:55:19Z martin

Innehåll

1 Introduktion

Sortering kommer i två huvudsakliga varianter, sortering baserad på jämförelser, och sortering baserad på klassificering. I denna labb betraktar vi endast jämförelsebaserade sorteringmetoder.

Jämförelsebaserad sortering är vanligast och baseras på en enkelt beskriven ordning

Till jämförelsebaserad sortering hör Selection sort, Insertion sort, Quicksort, Mergesort, Introsort och många andra välkända sorteringsmetoder.

Algorithm 1 Ordna två element

if $a < b$ **then**
  a läggs före b i strukturen
else
  b läggs före a i strukturen
end if

Alternativet till en jämförelsebaserad sortering utnyttjar värdets interna representation för att analysera storleken hos det som ska sorteras. Vanliga metoder som inte använder jämförelser är bland annat Counting sort, Radix sort och Bucket sort. Bara speciella typer av data lämpar sig för dessa metoder.

2 Syfte

Vi kommer i labben att i detalj undersöka hur olika jämförelsebaserade sorteringsmetoder beter sig för olika typer av indata.

Undersökningen består i att samla och sammanställa data från tidsmätningar för sortering av olika sekvenser för olika soretringsmetoder.

En beskrivning av sorteringsalgoritmerna finner du i kurslitteraturen eller i referenserna.

Det finns olika sorteringsmetoder implementerade i standardbiblioteket, bland annat `std::sort`. Om du behöver sortera data i ett eget program är det troligast att du vill använda någon av de metoder som finns i standardbiblioteket. `std::sort`.

3 Uppgift

Du skall jämföra tre egenhändigt implementerade sorteringsalgoritmer med standardbibliotekets `std::sort`.

Din kod ska vara strukturerad så att funktioner är samlade i relaterade headerfiler med separata cpp-filer. Koden ska vara lättläst med förtydligande kommentarer både för funktionsdeklarationer och programkod. Var noggran med namngivningen av dina variabler. `a, i , x` är olämpliga namn. Namn ska vara beskrivande och hjälpa läsbarheten i din programkod.

Förslagsvis gör du en upppdelning i

- sorteringsalgoritmer
- tidsmätning
- datagenerering
- datarapportering

4 Datagenerering

Fyra olika dataserier ska behandlas.

- Slumpdata, ex `std::rand`.
- Monotont stigande
- Monotont fallande
- Konstant värde

Storleken av varje serie kommer bero på hur många element som krävs av sorteringen så att den ger en god tidsmätning. Datagenereringen ska inte inkluderas i tidsmätningarna. Se även 7

5 Sorteringsalgoritmer

Du skall utföra mätningar på följange metoder:

- Insertion sort Knuth [3]
- Selection sort Knuth [3]
- Quicksort. Pivot i partitioneringssteget är höger element. Hoare [2]
- Quicksort. Pivot enligt median-of-three. Hoare [2]
- std::sort, eller motsvarande standardalgoritm som gäller för det språk du valt.

6 Tidsmätningar

För lämligt stora dataserier mäter du tiden för varje sorteringssteg. Denna tid bokförs av 7.

Nogrannhet måste iakktas vid mätningar det som måste balanseras är noggrannhet och tiden det tar för en programkörning.

Använder du `C++` så har du två alternativ. Antingen använder du `std::clock()` ur `<ctime>` eller så använder du det modernare `std::chrono::` ur `<chrono>`. `<chrono>` ger bättre precision i tidsmätningarna.

En mätserie för varje indata måste bestå av minst 5 tidsmätningar. Fler är bättre. Ju fler mätningar desto mindre mätfel.

Vanliga misstag är

- Sorteringen tar för vissa små indata för kort tid för att tidsmätningen ska ge ett riktigt värde. Ofta är klockan inte tillräckligt noggrann för att mäta korta tidsintervall.
- Man försöker sortera indata som redan sorterats. Detta påverkar påtagligt flera av metodernas tidsmätningar.
- Man mäter inte endast sorteringssteget utan även initierande programkod. En vanlig fallgrop är att kopiera dataserien till sorteringsfunktionen istället för att ge pekare eller iteratorer till serien.

7 Datarapportering

Din programkörning ska lagra programkörningsresultatet på en enkel och lättillgänglig textform. Du kommer för redovisningen och rapporteringen att använda ett grafritningsverktyg som illustrerar alla dina resultat på ett överskådligt sätt.

Du har två alternativ för att åstadkomma det:

1. Spara samtliga datapunkter från programmet. Metod, N, tidsåtgång, serie. Du behöver då efterbehandla mätserierna.
2. Spara medelvärdet och standardavvikelsen för varje mätpunkt. Du behöver även spara antal mätningar och annan relevant data.

Ett förslag för alternativ två.

```
Insertion sort
N           T[ms]       Stdev[ms]    Samples
20000       0.0654      0.0032       5
40000       0.3284      0.0092       5
60000       1.0541      0.0032       5
80000       2.2210      0.3662       5
100000      3.0654      0.6019       5
...
```

Listning 1: insertionsort.data

N ska täcka åtminstone en tiopotens; här 20000 → 200000.

Det sammanställda datat innehåller åtminstone

- Sorteringsmetod
- Varje mätpunkt

8 Redovisning

Redovisning av uppgiften görs med

- Förklarad och välstrukturerad programkod.
- Grafer som illustrerar uppmätta serier med felgränser. Graferna ska också innehålla en passning till den algoritmiska komplexitet som ges av litteraturen, eller standardbiblioteket för `std::sort`.
- En förklaring till varje graf som antingen bekräftar teoretisk gräns eller beskriver varför teori och verklighet skiljer sig.
- Bifogade genererade data.

9 Tips och Kodsnuttar

Din programkod får inte innehålla direkta avskrifter av kodsnuttarna här.

9.1 Tidsmätningar

Den största svårigheten med tidsmätningen är att sortera lämpligt mycket data. Sorterar du för lite data får du inga rimliga tidsmätningar. Sorterar du för mycket data får du inte godtagbara feluppskattningar.

Ett skelett för att kalibrera sorteringsmängderna kan ske via en repetition i olika intervall i grunddatat.

```
// Enough intervals so timers provides
// adequate precision
int INTERVALS = 10;

// method: sorting method to call
// input: N size of sorting buffer
// data: stuff to sort.
double time(FUNC sort, const int N, std::vector<value_type>
    data){
  // Make sure we have enough data.
  assert(source.size() >= N * INTERVALS);

  Timer timer;
  timer.start();
  intervals_left = INTERVALS;
  for(
    auto interval = data.begin(),
    auto end = interval + N;
             --intervals_left;
              interval = end, end += N){
      sort(interval, end);
  }

  timer.stop();
    // Absolutely inadequate resolution check.
  assert( timer.time() >= 0 );

  return timer.time();
}
```

9.2 Felanalys

Utgångspunkten för felanalysen är du tar fram en standardavvikelse för varje mätpunktserie. Desto fler mätpunkter, desto lägre standardavvikelse. Standardavvikelsen är enkel att beräkna i programkod

$$s = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \overline{x})^2} \tag{1}$$

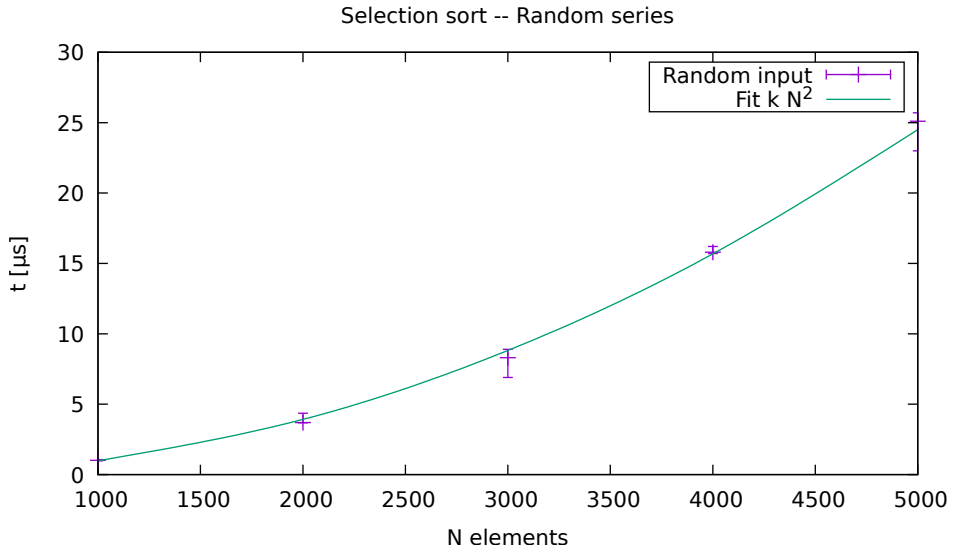
N är antal mätningar per mätserie för en punkt
$\overline{x}$ är medelvärdet av mätserien
x_i är värdet för mätpunkten
Blir i c++

```
value_type std_dev(std::valarray<value_type> measurements){
   value_type sum = std::accumulate(measurements.begin(),
                                    measurements.end(),
                                       value_type());
   auto n = measurements.size();
   auto avg = sum / measurements.size();

   std::valarray<value_type> dev_square =
       std::pow(measurements - avg, 2);

   value_type square_sum =
       std::accumulate(dev_squared.begin(),
                                    dev_squared.end(),
                                        value_type());

   return std::sqrt(square_sum * (1.0 / (N - 1)));

}
```

Standardavvikelsen ska presenteras på ett sådant sätt att det är tydligt att den innefattar den teoretiska passningen. Om du vill göra en automatisk passning till den teroetiska modellen finns metoden *minsta kvadratmetoden*[1]. Om man har ett grafritningsverktyg som uppdaterar kan uppdatera grafer i realtid, som Calc och Excel, är det möjligt att höfta en rimlig passning genon att gissa.

9.3 Grafer

Ett lämpligt utseende för ett diagram som illustrerar mätpunkter och teoretiska modeller innehåller mycket information. Kombinera olika mätserier så att du ger en balans mellan information i diagrammet och läsbarhet. Ett exempel på en graf för metoden selection_sort kan vara



En implementation för ovanstående graf i gnuplot ser ut som

```
set terminal pdf font 'cmr'
set output 'image.pdf'
#set size 1,2
set key box
#set samples 50, 50
#set style data lines
set title "Selection sort -- Random series"
set xlabel "N elements"
set ylabel "t [{/Symbol m}s]"
x = 0.0

plot "sel_sort.dat" title "Random input" with errorbars,
    "fit.dat" smooth csplines t "Fit k N^2"
#close output file
set output
set terminal pop
```

Referenser

[1] Björck, Å. *Numerical methods for Least Squeare problems.* Scociety for Industrial and applied mathematics, 1986. ISBN 978-0-89871-360-2.

[2] Hoare, C.A.R. Quicksort. *The Computer Journal*, 1962.

[3] Knuth, Donald Ervin. *The art of computer programming. Vol. 3, Sorting and searching.* Addison-Wesley, Reading, Mass., 2. ed. utgåvan, 1998. ISBN 0-201-89685-0.

[4] Williams, Thomas, Kelley, Colin, Bröker, Hans-Bernhard, och Merrit, Ethan A. *gnuplot 4.6*, 4.6.5 utgåvan, 2014. URL `http://www.gnuplot.info/docs_4.6/gnuplot.pdf`.