Data och Informationsteknik / Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
Erland Holmström Göteborg 6 nov 2018

Laboration 1 i Datastrukturer IT2

Laborationer är obligatoriska och måste lämnas in.

Anvisningar:

Om det står tex "40R" vid en uppgift så betyder det att programmet, inklusive eventuell main, kan skrivas på ca 40 rader. Det är ingen tävling men om du har många fler rader i ditt program så är det troligen inte välskrivet (och det är ju ett skäl till retur). Observera dock att jag undviker parenteser ensamma på en rad tex i början av metoder (men inte i slutet)

Inlämning:

Uppg 1: En pdf-fil döpt till `x.lab1.1.pdf` där x är ert gruppnummer med analys och resultat enligt nedan.

> Ni skall presentera och motivera de olika komplexitetsanalyserna samt för varje metod presentera ett diagram innehållande kurvor för den experimentella och teoretiska tidsåtgången som funktion av indatafältets storlek samt diskutera kring resultatet. Speciellt bör ni ta upp frågan:
> Hur väl stämmer verkligheten med teorin och om inte, vad beror det på?

Uppg 2: se Uppg 2 som finns i en separat fil på hemsidan.

Note:

- Namn och grupp skall finnas först i ALLA filer, även källtext.
  (Vi vill lämna rätt kommentar till rätt student.)
- Tänk på att namnge filerna rätt och tag bort eventuella paketdeklarationer som Eclips eller andra IDEs skapar och lägger in först i era filer.
  (Annars kan vi inte enkelt provköra era program)
- Använd helst max 80-100 tecken på en rad i källkodsfilerna.
  (Förutom att det blir svårläst så bör du kanske tänka igenom din struktur om du har längre rader.)
- Skicka inte in onödiga filer tex ”tilde” filer eller ”.class” filer.

Uppgift 1:

Övar *komplexitetsberäkningar*.
Vi går igenom komplexitet på föreläsning 3 och övar lite efter föreläsning 4. Vänta gärna med uppgiften till efter det.
Här är 3 metoder som gör samma sak men på lite olika sätt.

```
public final class MaxSumTest {
  static private int seqStart = 0;
  static private int seqEnd = -1;
  /**
  * contiguous subsequence sum algorithm.
  * seqStart and seqEnd represent the actual best sequence.
  * Version 1
  */
  public static int maxSubSum1( int[] a ) {
    int maxSum = 0;
    for( int i = 0; i < a.length; i++ )
      for( int j = i; j < a.length; j++ ) {
        int thisSum = 0;
        for( int k = i; k <= j; k++ ) {
          thisSum += a[k];
        }
        if( thisSum > maxSum ) {
          maxSum   = thisSum;
          seqStart = i;
          seqEnd   = j;
        }
      }
    return maxSum;
  }

  // Version 2
  public static int maxSubSum2( int[] a ) {
    int maxSum = 0;
    for( int i = 0; i < a.length; i++ ) {
      int thisSum = 0;
      for( int j = i; j < a.length; j++ ) {
        thisSum += a[j];
        if( thisSum > maxSum ) {
          maxSum = thisSum;
          seqStart = i;
          seqEnd   = j;
        }
      }
    }
    return maxSum;
  }
```

```
    // Version 3
    public static int maxSubSum3( int[] a ) {
      int maxSum  = 0;
      int thisSum = 0;
      for( int i = 0, j = 0; j < a.length; j++ ) {
        thisSum += a[j];
        if( thisSum > maxSum ) {
          maxSum = thisSum;
          seqStart = i;
          seqEnd   = j;
        }
        else if( thisSum < 0 ) {
          i = j + 1;
          thisSum = 0;
        }
      }
      return maxSum;
    }
  ...
```

Vad gör dom? Det är troligen lättast att förstå om man tittar på version 1 eller 2.

Om man själv skriver dessa algoritmer så är version 1 naturligast att komma på först. Version2 ”inser” man när man förstått version 1 ordentligt. Version 3 är inte helt lätt att komma på.

a) Analysera deras komplexitet. Analysera alla 3 dels med handviftning och dels med en matematiskt korrekt uppskattning (dvs sätt upp summorna och lös dem).

   Glöm inte motivera vad du gör. Största bristen med studentsvar är avsaknaden av motiveringar till vad ni gör.
   (se OH bilderna om komplexitet för en förklaring av pedantisk analys, matematiskt korrekt uppskattning och handviftning (”rough estimate”) samt hur man kan motivera matematiska beräkningar)

b) Det finns ett testprogram, `MaxSumTest.java`, på hemsidan som du kan använda som kör metoderna ovan och mäter tidsåtgången. Metoderna finns i klassen `MaxSum.java` som också finns på hemsidan.

   För varje metod rita en graf med de empiriskt uppmätta värdena. Rita också ut i graferna kurvor som motsvarar era teoretiska beräkningar av komplexiteten. Skala de teoretiska kurvorna i y-led så att de överlappar de experimentella så bra som möjligt.
   I `MaxSumTest.java` är fältens storlek satta rätt lågt för att det skall gå snabbt när du kör det första gången. *Öka storleken* så det blir stabila värden, utan att det tar halva dagen att köra förstås. Nu har du dina empiriska data för de tre funktionerna från testprogrammet och dina teoretiska data i form av en ekvation.

   Hur väl stämmer verkligheten med teorin och om inte, vad beror det på?

   (Referens: Jag och två av 2011 års IT studenter, Ludwig Kjellström & Daniel Olausson har skrivit programmet men du måste sätta dig in i hur det fungerar.)