Institut for Matematik og Datalogi
Syddansk Universitet, Odense

6. marts 2014
RF

DM507 – Opgaver uge 11

Eksaminatorier I

1. Cormen et al. øvelse 8.2-1 (side 196).
2. Cormen et al. øvelse 8.2-4 (side 197).
3. Eksamen juni 2008, opgave 1a.
4. Cormen et al. øvelse 8.3-1 (side 199).
5. Cormen et al. øvelse 8.3-4 (side 200).
6. Cormen et al. øvelse 8.3-2 (side 200). Hint til sidste del: udvid elementers nøgler.
7. Implementer Quicksort i Java ud fra bogens pseudokode (side 171). Test at din kode fungerer ved at generere arrays med forskelligt indhold og sortere dem. Tilføj tidtagning af din kode (kun selve sorteringen, ikke den del af programmet som genererer array'ets indhold).

   Kør derefter din kode med input, som er random `int`'s. Gør dette for mindst 5 forskellige værdier af n (antal elementer at sortere), vælg værdier som får programmet til at bruge fra ca. 100 til ca. 5000 millisekunder. Gentag hver enkelt kørsel tre gange og find gennemsnittet af antal millisekunder brugt ved de tre kørsler. Dividér de fremkomne tal med $n \log_2 n$ og check derved hvor godt analysen passer med praksis – de resulterende tal burde ifølge analysen være konstante.

   Sammenlign med dine køretider for det tilsvarende forsøg med Mergesort fra opgaverne i uge 8. Er Quicksort eller Mergesort hurtigst?

   [Ekstraopgave: prøv en let optimeret variant af Quicksort, hvor pivotelementet x i PARTITION vælges ved at se på de tre elementer på første, midterste og sidste plads i den del af array'et, som skal partitioneres. Disse tre elementer sammenlignes indbyrdes, og det ordningsmæssigt midterste (medianen) af disse tre bruges som pivot-element. Gør dette for kald til PARTITION, hvor der er 16 elementer eller mere, men ikke

for kald på mindre instanser. Kører denne version af Quicksort (lidt) hurtigere end standardversionen?]

Gentag derefter eksperimenterne med Java's (optimerede) sorteringsmetode `sort` fra klassen `java.util.Arrays`, og sammenlign køretiderne med dine implementationer af Quicksort og Mergesort.

8. [Udfordrende] Cormen et al. problem 7-4 (side 188). Hint til del c: vælg hvilken del man vil kalde rekursivt på, i stedet for at bruge den venstre del altid.

Eksaminatorier II

1. Cormen et al. øvelse 12.3-3 (side 299). Bemærk at opgaven mener ubalancerede søgetræer (kapitel 12). Besvar bagefter opgaven igen, men nu med rød-sorte træer i stedet for ubalancerede binære søgetræer.
2. Cormen et al. øvelse 13.1-2 (side 311).
3. Cormen et al. øvelse 13.1-6 (side 312).
4. Cormen et al. øvelse 13.2-4 (side 314).
5. Cormen et al. øvelse 13.3-3 (side 322).
6. Eksamen juni 2011, opgave 1.
7. Eksamen jan 2005, opgave 1.
8. Cormen et al. øvelse 13.3-2 (side 322).
9. Spørgsmål til projektet, del I.

Studiegrupper

Forslag til fokus for arbejde i studiegrupper:

Genfortæl for hinanden ideen i rød-sorte træer: hvad er strukturkravet, hvorfor giver dette $O(\log n)$ højde, hvordan er rebalancering efter henholdsvis indsættelse og sletning bygget op, hvorfor virker det (dvs. fjerner overtrædelser af strukturkravet), og hvorfor tager det $O(\log n)$ tid?

Forbered dele af opgaverne til eksaminatorietimer, f.eks. på nedenstående måde.

- Lav eksamensopgaverne individuelt på forhånd, og ret hinandens i gruppen.
- I opgave I.7, lav programmeringen og programkørsel (gerne i par) før gruppemøde. På mødet, sammenlign køretider.
- Forsøg at løse de mere kreative opgaver (I.5, I.6, I.8, II.3) i fællesskab. Arbejd både med at få ideer på skitseplanet til de ønskede algoritmer og argumenter, og med at få dem formuleret præcist til sidst. I kan evt. dele dise opgaver op imellem delgrupper, som senere forsøger at formidle de fundne løsninger til hinanden så klart og præcist som muligt.
- Forsøg at lave resten af opgaverne hver især på forhånd, og sammenlign svar i studiegruppen.