Data och Informationsteknik / Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
Erland Holmström

Göteborg 16 mar 2015

Examination i

Datastrukturer och Algoritmer IT, TDA416

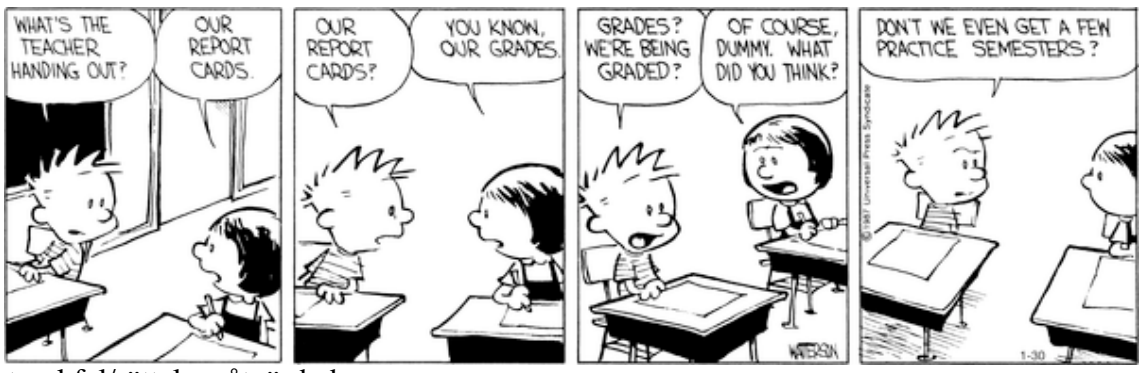
Dag: Måndag Datum: 2015-03-16 Tid: 8.30-13.30 (OBS 5 tim) Rum: V

Ansvarig lärare:	Erland Holmström tel. 1007, mobil 0708-710 600
Resultat:	Skickas med mail från Ladok.
Lösningar:	Läggs eventuellt på hemsidan.
Tentagranskning:	Tentan finns efter att resultatet publiserats på vår expedition. Tid för klagomål på rättningen annonseras på hemsidan efter att resultatet är publicerat (eller maila mig så försöker vi hitta en tid).
Betygsgränser:	CTH: 3=28p, 4=38p, 5= 48p, max 60p För betyg 3 skall man ha ca 10 poäng på både del A och del B. Dock kan gränsen variera beroende på tentans upplägg.
Hjälpmedel:	Bara en sammanfattning av delar av Javas API som kan lånas på tentan eller skrivas ut i förväg, se hemsidan.

Observera:

- Börja med att läsa igenom alla problem så du kan ställa frågor när jag kommer. Jag brukar komma efter ca 1-2 timmar (men det är många salar så det kan bli senare).
- ***Alla svar måste motiveras*** där så är möjligt/lämpligt.
- Skriv läsligt! Rita figurer. Lösningar som är svårlästa bedöms inte.
- Skriv kortfattat och precist.
- Råd och anvisningar som getts under kursen skall följas.
- Program skall skivas i Java, skall vara indenterade och lämpligt kommenterade.
- Börja nya problem på ny sida, dock ej korta delproblem.
-
- *Lämna inte in tesen med tentan utan behåll den*. Lämna inte heller in kladdpapper/skisser.

Lycka till!



Vissa tryckfel/rättelser åtgärdade

Del A Teori

(Tänk på att detta är del A, jag vill tex ha idébeskrivning som förklarar hur och varför algoritmer fungerar och eventuellt pseudokod om du vill men inte körbar Java kod här. Beskriv kortfattat" == steg 1. ***Du måste också motivera dina svar. Alltid.***

Problem 1. *Småfrågor:* Avgör om vart och ett av följande påståenden är sant eller falsk eller svara på frågan. För att få poäng måste du ange en kortfattat *motivation* till ditt svar.

a) Det stod ju i remove metoden för tex ArrayList att:

"The behavior of an iterator is unspecified if the underlying collection is modified while the iteration is in progress in any way other than by calling this method."

Hur hanterar/löser man det i Java?

b) Vad är skillnaden mellan teoretisk komplexitet och empirisk komplexitet?

c) Antag att vi har en metod som sorterar n tal på $O(n^2)$ tid. Hur mycket längre tid tar det att sortera om vi dubblerar indata till $2n$?

d) Urvalsorterings komplexitet beror på hur elementen är sorterade i fältet.

e) Maximala antalet bågar i en oriktad graf utan självloopar är n^2 där n är antalet noder i grafen.

f) I en hashtabell tar det alltid konstant tid att slå upp värdet till en nyckel.

g) Värsta falls komplexiteten för find-operationen i ett splay-träd är $O(n)$.

h) I en godtycklig graf kan kortaste vägen alltid hittas med Dijkstras algoritm.

(10p)

Problem 2. *Testar: hashtabeller*

a) Beskriv utförligt de *2* vanligaste sätten att hantera kollisioner i hashtabeller. Beskriv hur dom fungerar och deras för och nackdelar.

b) Hur ser hashfunktionerna för strängar och heltal ut i Java?

(7p)

Problem 3. *Testar: sortering*

Antag att vi har ett fält med längd n som innehåller unika heltal i intervallet *0* till n. Beskriv en algoritm som hittar heltalet som saknas som har komplexitet $O(n)$.

Ex: Om vi har ett fält med längd *5* så innehåller det tal mellan *0..5* tex talen *3 0 5 1 4*. Vi skall då "hitta" *2*an.

(3p)

Problem 4. Testar: träd

a) Beskriv kortfattat egenskaperna hos träden 2-3 träd, splay-träd, binärt sökträd, AVL-träd, partiellt ordnat och vänsterbalanserat träd (pov)

b) Sätt in talen 50, 30, 70, 10, 40, 5 i given ordning i nedanstående träd. Rita träden efter varje operation, ev. även stegen i varje operation.

1. ett binärt sökträd.
2. ett AVL-träd.
3. ett partiellt ordnat och vänsterbalanserat träd (pov) implementerad med en min-heap. Rita det träd den representerar.

(12p)

Del B Implementeringar och algoritmer

Problem 5. Testar: algoritmer, sortering, träd

Man kan hitta dubletter i en osorterad följd av *n* tal genom att jämföra varje tal med de efterföljande. För följden *{4,5,6,4,7,4,7}* skulle vi rapportera *4* dubletter; *(4,4)*, *(4,4)*, *(4,4)* och *(7,7)* och göra följande jämförelser för att komma dit

(4-5), (4-6), (4-4), (4-7), (4-4), (4-7), (5-6), (5-4), (5-7), (5-4), (5-7), (6-4), osv

Komplexiteten blir: först jämförs första talet med de *n-1* efterföljande, sedan andra med de *n-2* efterföljande osv. så antalet jämförelser blir

$$n\text{-}1 + n\text{-}2 + n\text{-}3 + \ldots + 1 = \sum_{i=1}^{n-1} i = (n^2 - n)/2 = O(n^2)\ .$$

Om (nästan) alla talen är lika eller olika spelar ingen roll, det är svårt att utnyttja den kunskapen i den här algoritmen men det påverkar antalet utskrifter, är alla lika så blir det lika många utskrifter som jämförelser dvs wc för utskrifter.

Konstruera ytterligare två algoritmer för att finna alla dubletter i en osorterad följd av tal. En algoritm skall använda sortering och en skall använda ett binärt sökträd. Du behöver alltså inte beskriva tex sorteringen om den är ”standard” utan hur du använder sorteringen. Gör du en specialsortering så måste du beskriva mer detaljer såklart.

Det räcker om du beskriver din ide och kort hur algoritmen går till, ungefär som jag gör ovan eller tex med hjälp av kortfattad pseudokod. Motivera dina val tex av sorteringsmetod, rätt val kanske kan snabba upp algoritmen?

För varje algoritm skall du ange hur algoritmen fungerar och hur komplexiteten påverkas om (nästan) alla tal är lika, om alla tal är olika samt om det är ”några dubletter” tex som följden jag har ovan. I det sista fallet räcker det med ”komplexiteten är mellan x..y” eller medelkomplexiteten. (Tänk på att ”alla olika” kan vara nästan sorterat och att kostnad för utskrifter också skall redovisas …)

Jag vill alltså ha en tabell med 2 uppsättningar av

Algoritm **med xxxxx**:

Alla lika:

Alla olika:

Några dubletter:

(16p)

Problem 6. *Testar: labben, grafer, algoritmer*

Man skall hålla en Europeisk konferens om ”Independent Europe” och vill hitta en så central plats som möjligt för konferensen Man definierar ”central” som den plats som har minsta sammanlagda kostnaden för alla deltagarna för att åka till konferensen. Man har en riktad viktad graf över alla möjliga orter i Europa (vikten= avstånd=kostnad mellan dem) och en lista med orter där deltagarna bor. Från varje ort som skall deltaga kommer det bara en person.
Du skall skriva en metod,

```
int findMostCentralPlace(List<Integer> thePlaces),
```

som hittar den optimala platsen i form av dess nummer. (Man kan slå upp namnet om man vill givet ortens nummer men det är inte så intressant här). Metoden har listan med orter (`thePlaces`) för deltagarna som parameter, dessa antas finnas med i grafen.
Du kan anta att det finns en klass för grafer som den i labben (se sist) som du kan använda dig av. Idé (tex som text eller pseudokod), komplexitet och kod krävs.

(12p)

```
public class DirectedGraph<E extends Edge> {
  private List<E>[] neighbours;
  public DirectedGraph(int noOfNodes) {...}
  // ====================================================
  public void addEdge(E e) {….finns….}
  // ====================================================
  public Iterator<E> shortestPath(int from, int to) {….finns….}
  // ====================================================
  public Iterator<E> minimumSpanningTree() {….finns….}
  // ====================================================
  public int  findMostCentralPlace(List<Integer> thePlaces)
                                        {….ditt jobb....}
  // ====================================================
```