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

Göteborg 10 mars 2013

Examination in

Datastrukturer och Algoritmer IT TDA416

DAY: Tuesday DATE: 2013-03-12 TIME: 8.30-13.30 (5 tim) ROOM: V

Responsible teachers:	Erland Holmström 0708-710 600
	Birgit Grohe tel. 0708-11 75 31,
Results:	Are sent by mail from Ladok.
Solutions:	Are eventually posted on home page.
Inspection of grading:	The exam can be found in our study expedition after posting of results.
	Time for inspection of grading are announced on homepage after the result are published or mail Erland and we find a time.
Grade limits:	CTH: 3=28p, 4=38p, 5= 48p. Maxpoäng = 60.
	OBS ! A grade of 3 or better also requires you to have at least 10p on both part A and B individually.
Aids on the exam:	Only a summary of Javas API that is handed out on the exam or that you have printed beforehand.

Observe:

- Start by reading through all questions so you can ask questions when I come. I usually will come after appr. 2 hours.
- All answers must be motivated when applicable and not otherwise stated.
- Write legible! Draw figures. Solutions that are difficult to read are not evaluated!
- Answer concisely and to the point.
- The advice and directions given during course must be followed.
- Programs should be written in Java, indent properly, use comments and so on.
- Start every new problem on a new sheet of paper.

Good Luck!

Del A Teori

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

a) Höjden på ett binärt sökträd med n noder är alltid $O(\log n)$.

b) Binärsökning är en effektiv metod för att söka upp ett givet element i en sorterad länkad lista.

c) Antag att du pushar 20, 30, 50, 15 och 60 på en stack. Sedan tar du ut 4 element. Vad finns kvar på stacken?

d) Sätt in elementen 20, 30, 50, 15 och 60 i en kö. Sedan tar du ut 3 element, vad finns kvar i kön?

e) När man väljer storleken av en hashtabell, ska man helst välja jämna tal.

f) Man kan beräkna kortaste vägen från en startnod till alla andra noder med algoritmen BFS om alla kostnader på bågarna är antingen 1 eller 2.

g) Både Prims algoritm och Kruskals algoritm returnerar ett uppspännande träd i en sammanhängande graf. Båda algoritmerna returnerar alltid samma uppspännande träd för en given graf.

h) Bubble sort och quicksort har samma värstafallskomplexitet.

(1p per st) (8p)

Problem 2. *Testar: Sökträd*

a) Sätt in följande element ett efter ett i ett AVL-träd och rita det resulterade trädet. (Mellansteg behöver du inte rita upp.) [5,3,8,2,4,7,10,1,12,6,9,11] (2p)

b) Tag bort elementet 4 från AVL-trädet. Förklara steg för steg hur borttagningsalgoritmen fungerar genom att rita upp trädet efter varje steg. (3p)

c) Vad är skillnaden mellan binära sökträd och AVL-träd? Beskriv både skillnaderna i egenskapen av träden samt eventuella skillnader av metoder för insättning och borttagning. (3p)

(8p)

Problem 3. *Testar: POV träd, heapsort.*

a) Vilka egenskaper har ett partiellt ordnat och vänsterbalanserat (POV) binärt träd (en heap)? Definiera. (1p)

b) Sätt in följande element i en heap [5,3,8,2,4,7]. Rita trädet efter varje steg. (2p)

c) Trädet ovan representeras vanligen med ett fält när man utför heapsort. Beskriv hur trädet lagras i fältet. Utför nu heapsort på indatan i b). Här efterfrågas den något enklare varianten som tar $O(n)$ extra minne. Beskriv först idén i ord och utför sedan sorteringen steg för steg. Rita trädet *och* fältet efter varje steg. (4p)

d) Tag trädet från c) och utför en preorder traversal. Vad blir resultatet? (1p)

e) Resultatet av traverseringsmetoden preorder är inte sorterat. Beskriv en traverseringsmetod som returnerar en sorterad lista/ett sorterad fält givet ett träd som skapats av att man utfört en heapsort. Beskriv din metod först i ord och sedan i pseudokod. (4p)

(12p)

Problem 4. *Testar: Hashtabeller*

Antag att du vill lagra heltal så att du kan söka efter dem på ett effektivt sätt. För detta ändamål kan du använda dig av någon form av hashtabell.

Illustrera hur denna datastruktur fungerar genom att lagra följande tal i angiven ordning: {40,16,90,38,7,72,82,28,12,59} enligt nedan. Hashkoden är själva talet, tabellstorleken ska vara 11. I b) börjar du om med en tom tabell.

a) Först ska talen lagras i en hashtabell med hjälp av chaining (kedjning). Rita hashtabellen. (1p)

b) Använd nu open adressing med linear probing. Rita hashtabellen. (2p)

c) Tag bort talet 82 och 18 ur hashtabellen från b). Beskriv hur borttagningen (remove) fungerar och rita upp resultatet efter varje borttagning. (2p)

d) Använd nu resultatet från c) för att söka efter talet 12. Beskriv kortfattat hur sökning (get) fungerar. (1p)

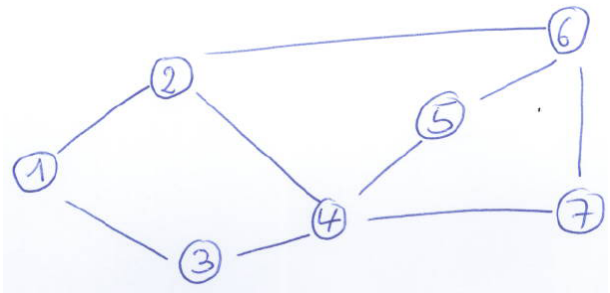
(6p)

Del B Implementeringar

Problem 5. *Testar: grafer.*

Diametern i en oriktad, viktad graf definieras som det största avståndet mellan två noder i grafen, och avståndet definieras som antal bågar på den kortaste vägen mellan noderna.

a) Givet följande graf, vad är diametern? (2p)



b) Hur kan man ta reda på diametern i en graf? Beskriv din idé i ord. För full poäng på b) och c) krävs att du väljer en algoritm med optimal asymptotisk komplexitet. Mindre effektiva algoritmer ger lägre poäng i b) och c).
Om du använder dig av algoritmer som ingick i kursen, så behöver du inte skriva pseudokod för dem. (2p)

c) Skriv pseudokod för din idé i b). (3p)

d) Komplexitet? Använd n för antal noder och m för antal bågar. (3p)

(10p)

vänd

Problem 6. *Testar: länkade strukturer*

Man kan representera polynom som en ordnad lista med termer, där termerna är ordnade efter sina exponenter. För att addera två polynom traverserar man bägge listorna och undersöker termerna på de aktuella positionerna. Vi antar att vi har pekare för att hålla reda på aktuell position i listorna.

Om exponenten för den ena är mindre än för den andra så skapar vi en term som är lika med den större och sätter in den i resultatlistan och flyttar fram positionen för listan som innehåller den större termen. Om exponenterna är lika så skapar vi en ny term med den exponenten och summan av koefficienterna och flyttar fram bägge positionerna. Exempel

$3x^4+2x^2+3x+7$ adderat med $2x^3+4x+5$ blir $3x^4+2x^3+2x^2+7x+12$.

(I den här uppgiften skall du inte använda Javas klasser för listor eller polynom utan skall själv skapa de strukturer som behövs)

a) Definiera en klass Term som innehåller en exponent och en koefficient. Vi kan för enkelhets skull anta att bägge är heltal. Klassen skall implementera Comparable interfacet genom att jämföra värdet på exponenter. Den skall strax vara en inre klass i polynomklassen.

b) Definiera en klass, Poly, för att hantera polynom. Ett polynom skall implementeras med hjälp av en enkellänkad liststruktur som använder sig av klassen Term. Det räcker med datastruktur för listan och en toString metod här.

   - behöver du en konstruktor senare så lägg till den då
   - toString skapar en sträng enligt [2x3+4x1+5x0].
   - du behöver inte implementera en komplett klass för listor här, en enkel datastruktur räcker ju (make it simple!)

   Tips: det blir enklare algoritmer om du använder en "dummy" nod (kallas även ”sentinel”) i början av listan)

c) Skriv metoden

```
public Poly addPoly(Poly p1)
```

som givet ett polynom adderar detta och det aktuella objektet och returnerar det resulterande polynomet (som ett nytt polynom). Du kan anta att de givna polynomen är ordnande efter exponenten.

(16p)