Exempeltentafrågor i TDDC91 Datastrukturer och algoritmer

15 oktober 2015

1. Den här uppgiften handlar om algoritmanalys.

   (a) Vad gör följande algoritm? Analysera dess exekveringstid i värsta fallet.

   Require: två heltal, $a \neq 0$ och $n \geq 0$
   Ensure: ?
   function FOO(a, n)
   $k \leftarrow 0$
   $b \leftarrow 1$
   while $k < n$ **do**
   $k \leftarrow k + 1$
   $b \leftarrow b \cdot a$
   return b

   (b) Vad gör följande algoritm? Analysera dess exekveringstid i värsta fallet.

   Require: två heltal, $a \neq 0$ och $n \geq 0$
   Ensure: ?
   function BAR(a, n)
   $k \leftarrow n$
   $b \leftarrow 1$
   $c \leftarrow a$
   while $k > 0$ **do**
   if $k \bmod 2 = 0$ **then**
   $k \leftarrow k/2$
   $c \leftarrow c \cdot c$
   else
   $k \leftarrow k - 1$
   $b \leftarrow b \cdot c$
   return b

2. Låt T vara en minHeap som lagrar n nycklar. Beskriv en effektiv algoritm för att hitta alla nycklar i T som är mindre än eller lika med en given nyckel x (vilken inte nödvändigtvis förekommer i T). Observera att nycklarna inte behöver rapporteras i sorterad ordning. Exekveringstiden för din algoritm ska vara $O(k)$, där k är antalet nycklar som hittas.

3. Antag att du får givet ett diagram över ett telefonnätverk ritat som en graf G vars noder representerar telefonväxlar och vars bågar representerar kommunikationslinjer mellan två växlar. Bågarna är märkta med sin bandbredd. Bandbredden av en väg i G är bandbredden av den båge i vägen som har lägst bandbredd. Beskriv en algoritm som givet ett diagram och två telefonväxlar a och b returnerar den maximala bandbredden för en väg mellan a och b. (Returnera bara den maximala bandbredden; du behöver inte returnera den faktiska vägen.) Du kan anta att grafen i diagrammet är enkel och sammanhängande samt att alla bandbredder är ickenegativa. Analysera tidskomplexiteten hos din algoritm.

4. Ett polynom av grad n är ett uttryck på formen

$$a_0 + a_1 x + \ldots + a_n x^n$$

där $a_0, \ldots, a_n$ är konstanter, $a_n \neq 0$ och x är en variabel över $\mathbb{R}$.

   (a) Ge pseudokod för en algoritm som, utgående från uttrycket ovan, beräknar värdet av ett polynom av grad n för ett givet värde på x. Analysera tidskomplexiteten för din algoritm under antagandet att x^i inte är en primitiv operation utan måste beräknas genom multiplikation.

   (b) Beskriv en annan algoritm för att, under samma antaganden, beräkna värdet av ett polynom av grad n i tid $\Theta(n)$. Kan du koppla lösningen till ett av de algoritmiska paradigm som diskuterats i kursen?

5. Antag att du har fått uppgiften att implementera ADT Stack med enbart två köer Q_1 och Q_2 som instansvariabler.

   (a) Beskriv på svenska eller med pseudokod hur du skulle implementera metoderna push() och pop().

   (b) Karakterisera den asymptotiska exekveringstiden för dina implementationer av push() och pop() som funktioner av antalet element i stacken n.

6. I en sökning efter ett element x i ett binärt sökträd kallas listan av noder x jämförs med *nyckelsekvensen* för sökningen.

   Antag att vi har talen från 1 till 1000 insatta i ett binärt sökträd och att vi vill söka efter talet 363. Förklara varför följande *inte* kan vara nyckelsekvensen för sökningen: 925, 202, 911, 240, 912, 245, 363.

7. Följande kodsnutt upphittades i en datorsal:

```
public Object foo (Sequence S, Comparator C, int k, int a, int b)
{
  int c = partition(S,C,a,b);
  if (c == k) { return S.elemAtRank(k); }
  else if (c < k) { return foo(S,C,k,c+1,b); }
  else { return foo(S,C,k,a,c-1); }
}

public int partition (Sequence S, Comparator C, int left_bound, int right_bound)
{
  Object pivot = S.atRank(right_bound).element();
  int left_index = left_bound, right_index = right_bound-1;
  while (left_index <= right_index) {
    while ((left_index <= right_index) &&
           C.isLessThanOrEqualTo(S.atRank(left_index).element(),pivot)) {
      left_index++;
    }
    while ((right_index >= left_index) &&
           C.isGreaterThanOrEqualTo(S.atRank(right_index).element(),pivot)) {
      right_index--;
    }
    if (left_index < right_index) {
      S.swap(S.atRank(left_index),S.atRank(right_index));
    }
  }
```

```
  S.swap(S.atRank(left_index),S.atRank(right_bound));
  return left_index;
}
```

Olyckligtvis verkar inte författaren vara DALG-student, eftersom det inte finns några kommentarer och vissa variabelnamn är olyckligt valda. Kodens andra del känns lätt igen som partitioneringssteget i Quicksort, men den första delen är lite mystisk. Din uppgift är att lista ut vad den här algoritmen gör genom att göra följande saker:

(a) Stega genom en exekvering av `foo(`S`,`C`,5,0,7)`, där S är sekvensen (C, H, B, E, J, G, D, A) och C är en jämförelseoperator för bokstäver som ordnar dem i alfabetisk ordning. Det första steget visas nedan; fortsätt genom att visa alla anrop till `foo` och `partition` och deras returvärden samt när platsbyten sker och hur de förändrar S. Vilket värde returneras till slut av anropet `foo(`S`,`C`,5,0,7)`?

```
foo(S,C,5,0,7)                      // S is (C,H,B,E,J,G,D,A)
  partition(S,C,0,7)
    swap elements at ranks 0 and 7 // S is now (A,H,B,E,J,G,D,C)
    return 0
  foo(S,C,5,1,7)                    // c is 0; choose c < k case since 0 < 5
```

(b) Vad gör `foo(`S`,`C`,`k`,`a`,`b`)` i allmänhet? Antag att k, a och b har giltiga värden.

8. Antag att Tommy har valt tre heltal och placerat dem på en stack i slumpmässig ordning. Skriv en snutt pseudokod (utan loopar eller rekursion) som bara använder en jämförelse och en variabel x, men ändå kan garantera att sannolikheten att variabeln x innehåller det största av Tommys tre heltal är 2/3. Motivera varför din metod fungerar.

9. Lisa och Sluggo bygger en robot i en projektkurs de läser och har upptäckt att de behöver passa in två legobitar sida vid sida i en öppning. Öppningen är x centimeter bred och summan av de två bitarnas bredder måste vara precis lika med öppningens bredd, annars faller roboten sönder under demonstrationen. De två konstruktörerna har tillgång till en enorm mängd legobitar med varierande bredder och måste välja två bitar som uppfyller ovanstående villkor.

   Eftersom vi på IDA tycker att allt ska göras enligt väldefinierade algoritmer är din uppgift att förse Lisa och Sluggo med en asymptotiskt effektiv algoritm för att lösa deras problem!

   Mer specifikt får du en mängd S bestående av n reella tal och ytterligare ett reellt tal x. Beskriv en algoritm, med exekveringstid $\mathcal{O}(n \log(n))$, som avgör huruvida det finns två element i S vars summa är exakt x eller ej.

10. Ett forensiskt laboratorium får en leverans av n prover. Proverna ser likadana ut, men faktum är att vissa av dem har olika kemisk sammansättning. Laboratoriet har tillgång till en maskin som kan användas för att ta reda på om två prov har olika sammansättning eller inte. Redan i förväg vet man att de flesta (mer än 50%) av proverna är identiska. Hitta ett av dessa identiska prover genom att använda jämförelsemaskinen som mest n gånger. (Se upp: det är fullt möjligt att två prov är identiska utan att tillhöra den stora majoriteten av identiska prover.)

11. (a) Rita ett binärt sökträd T sådant att
    - varje nod i T lagrar en bokstav,
    - en traversering i *preorder* av T ger ordet EXAMFUN och
    - en traversering i *inorder* av T ger ordet MAFXUEN.

    (b) Ge en algoritm med exekveringstid $\mathcal{O}(n)$ för att beräkna djupet av varje nod i ett träd T, där n är antalet noder i T. Antag att det finns två metoder setDepth(v, d) och getDepth(v) som använder $\mathcal{O}(1)$ tid.

12. (a) Antag att vi får en sekvens S innehållande n element, där varje element är färgat med en av k färger. Antag att S representeras med en array och ge in **in-place**-algoritm för att ordna S så att alla element med samma färger ligger i följd i arrayen. Algoritmen ska ha exekveringstid $\mathcal{O}(nk)$ i värsta fallet.

    (b) Antag att vi får en sekvens S innehållande n element, där varje element är ett heltal från intervallet $\left[0, n^2 - 1\right]$. Beskriv ett enkelt sätt att sortera S i $\mathcal{O}(n)$ tid. (*Tips*: fundera på olika sätt att representera elementen.)

13. Betrakta följande giriga strategi för att hitta kortaste vägen från *start* till *goal* i en given sammanhängande graf.

    i. Initialize *path* to *start*.
    ii. Initialize $VisitedVertices$ to $\{start\}$
    iii. If $start = goal$, return *path* and exit. Otherwise, continue.
    iv. Find the edge $(start, v)$ of minimum weight such that v is adjacent to *start* and v is not in $VisitedVertices$.
    v. Add v to *path*.
    vi. Add v to $VisitedVertices$.
    vii. Set *start* equal to v and go to step iii.

    Hittar den här giriga strategin alltid kortaste vägen från *start* till *goal*? Ge antingen en översiktlig förklaring till varför det fungerar eller visa ett motexempel.

14. Omöjligt? Din kompis Egbert är känd för att komma med fantastiska påståenden. Vilka av Egberts påståenden nedan är möjliga respektive omöjliga? Svar utan motivering ger inga poäng.

    (a) Egbert säger: Jag har en ny algoritm som kan sortera listor i linjär tid, förutsatt att varje elements position i den osorterade listan inte är mer än 1000 steg bort från rätt position i den sorterade listan.

    (b) Egbert säger: Jag har en ny algoritm som kan hitta det största talet i en given osorterad array i logaritmisk tid.

    (c) Egbert säger: Jag har en ny algoritm som kan sortera vilken lista som helst i linjär tid, bara jag får en lämplig jämförelsefunktion.

15. Om `a` är en array av längd m, där varje element `a[i]` är ett heltal mellan 0 och $n - 1$, vad är värstafallstiden för en körning av följande kodsnutt?

```
k = 0;

for ( int i = 0; i < m; ++i ) {
   for ( int j = 0; j < a[i]; ++j ) {
      b[k] = i;
      ++k;
   }
}
```

    Om det också är känt att summan av alla element i `a` är n, vad blir då exekveringstiden för kodsnutten ovan?

16. Träd

    (a) Den *totala längden* av ett träd T är summan av avstånden från roten av T till alla andra noder i T. Beskriv en algoritm som får som indata roten r till ett binärt träd och som returnerar den totala längden av trädet. Analysera tidskomplexiteten hos din algoritm.

    (b) En grupp biologer lagrar information om DNA-strukturer i ett AVL-träd där de använder strukturens specifika vikt (ett heltal) som nyckel. Biologerna behöver hela tiden svara på frågor av typen "Finns det några strukturer i trädet med specifik vikt mellan a och b (inklusivt)?" och de hoppas på så snabba svar som möjligt.

    Beskriv en algoritm som, givet heltal a och b, returnerar sant om det finns en nyckel x i trädet, sådan att $a \leq x \leq b$, och falskt om ingen sådan nyckel existerar i trädet. Vad har din algoritm för tidskomplexitet?

17. En array är *bitonisk* om den innehåller en strikt ökande sekvens av nycklar omedelbart följd av en strikt avtagande sekvens av nycklar. Beskriv en algoritm som hittar den största nyckeln i en bitonisk array av längd N i tid proportionell mot $\log(N)$.

18. Skriv om följande påstående som en sats om oriktade grafer och bevisa satsen. Antag att om A är vän till B, så är B vän till A och att för varje A gäller att A inte är vän till A. "I varje grupp av $n \geq 2$ personer finns det två personer med samma antal vänner i gruppen."

19. Vilka av följande påståenden är sanna och vilka är falska? Svar utan motivering ger inga poäng.

    (a) En lista L består av n heltal. Alla tal i listan ligger mellan 1 och n. Det går att sortera dessa n tal i tid $O(n)$.

    (b) $n^{1+\varepsilon/\sqrt{\log n}} \in O(n \log n)$, $\varepsilon > 0$.

20. Antag att du behöver generera en *slumpmässig* permutation av heltalen från 1 till N. Exempelvis är $\{4, 3, 1, 5, 2\}$ och $\{3, 1, 4, 2, 5\}$ giltiga permutationer, men $\{5, 4, 1, 2, 1\}$ är inte en giltig permutation eftersom ett tal saknas och ett tal förekommer två gånger. En sådan metod kommer ofta till användning i olika typer av simuleringar. Vi antar att vi har tillgång till en slumptalsgenerator, `r`, med metoden `randInt(i,j)`, som generar heltal mellan `i` och `j` med likformig sannolikhet. Här är tre algoritmer:

    1. Fyll arrayen `a` från `a[0]` till `a[N-1]` enligt följande: För att fylla i `a[i]`, generera slumptal till ett dyker upp som inte redan finns i `a[0]`, `a[1]`, ..., `a[i-1]`.
    2. Samma som algoritm (1), men håll reda på en extra array `used`. När ett slumptal `ran` först sätts in i `a`, sätt `used[ran]=true`. Detta betyder att när `a[i]` ska fyllas i med ett slumptal kan vi testa i ett steg om slumptalet redan använts.
    3. Fyll i arrayen så att `a[i]=i+1` håller. Gör sedan följande:

```
for ( int i = 1; i < N; i++ ) {
   swap( a[i], a[ randInt( 0, i )] );
}
```

    Det är inte svårt att se att alla tre algoritmer ovan genererar giltiga permutationer. De första två algoritmerna har tester som garanterar att inga dubletter genereras och den tredje algoritmen blandar om i en array som initialt är fri från dubletter. Det är också lätt att se att de två första algoritmerna är helt slumpmässiga och att varje permutation är lika sannolik. Den tredje algoritmen, konstruerad av R. Floyd, är inte lika lätt att förstå sig på, men det går att visa att även denna algoritm är helt slumpmässig med hjälp av induktion.

    (a) Gör en asymptotisk analys av den *förväntade* körtiden för varje algoritm.

    (b) Vad är exekveringstiden i värsta fall för respektive algoritm?

21. Vi använder en array med indexen 0 till 6 för att implementera en öppet adresserad hashtabell av längd 7 med följande tabell som hashfunktion:

nyckel	hashvärde
A	5
B	2
C	5
D	1
E	4
F	1
G	3

Antag att linjär sondering (*linear probing*) används för att hantera kollisioner.

(a) Visa hur arrayen ser ut efter att nycklarna har satts in i alfabetisk ordning: A, B, C, D, E, F, G.

(b) Vilken/vilka av följande skulle kunna vara innehållet i arrayen efter att nycklarna har satts in i någon ordning?

I.

0	1	2	3	4	5	6
A	F	D	B	G	E	C

II.

0	1	2	3	4	5	6
F	A	D	B	G	E	C

III.

0	1	2	3	4	5	6
C	A	B	G	F	E	D

22. Binära träd

(a) Låt T vara ett fullt binärt träd med rot r. Betrakta följande algoritm:

Input: Rot r av ett fullt binärt träd
Output: ?

```
function TRAVERSE(r)
    if r är ett löv then return 0
    else
        t ←TRAVERSE(vänster barn till r)
        s ←TRAVERSE(höger barn till r)
        return s + t + 1
```

Vad gör algoritmen?

(b) Låt T vara ett fullt binärt träd med 7 noder: a, b, c, d, e, f, g. En preordertraversering av T besöker noderna i ordningen b, g, d, a, c, f, e. En inordertraversering av T besöker noderna i ordningen d, g, c, a, f, b, e. Finns det några noder som är på avstånd 3 från roten av T? Vilka i så fall? (**Tips**: I trädet T är noden d vänster barn till g.)

(c) En mängd $K = \{k_1, \ldots, k_n\}$ av $n \geq 1$ nycklar ska lagras i ett AVL-träd. Vilket av följande påståenden är alltid sant?

(A) Om k_i är den minsta nyckeln i K så kommer vänster barn till noden som lagrar k_i i varje AVL-träd som lagrar K vara ett löv.

(B) Alla AVL-träd som lagrar K har exakt samma höjd oavsett vilken ordning nycklarna sätts in i trädet.

(C) En preordertraversering av ett AVL-träd som lagrar K besöker noderna i ökande nyckelordning.

(D) I varje AVL-träd som lagrar K är nyckeln som lagras i roten av trädet samma, oavsett ordningen nycklarna sätts in i trädet.

(E) Inget av påståendena ovan är alltid sant.

23. Givet två arrayer `a[]` och `b[]`, innehållandes M respektive N punkter i planet (med $N \geq M$), konstruera en algoritm som avgör hur många punkter som förekommer i båda arrayerna. Exekveringstiden för din algoritm ska vara proportionell mot $N \log M$ i värsta fallet och får bara använda en konstant mängd extra minne. Motivera att din algoritm klarar av dessa krav.

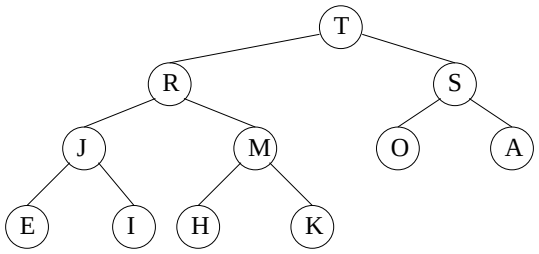
24. Kolumnen nedan till vänster innehåller strängar som ska sorteras, kolumnen nedan till höger är strängarna i sorterad ordning, övriga kolumner visar innehållet vid något mellanliggande steg i ett anrop till de fyra sorteringsalgoritmerna listade nedan. Para ihop varje algoritm med rätt kolumn. Använd varje algoritm exakt en gång.

```
COS ARC ARC ARC REL ARC
PHY CHE CHE ART PHY ART
ELE COS COS CEE PHY CEE
COS COS COS CHE ELE CHE
MAT ECO ECO CHM PHI CHM
MOL ELE EEB COS ORF COS
LIN GEO ELE COS ORF COS
ARC LIN ELE COS COS COS
ECO MAE ENG COS ELE COS
CHE MAT GEO COS EEB COS
MAE MOL LIN COS MUS COS
GEO PHY MAE ECO GEO ECO
ORF ORF MAT ORF ORF EEB
EEB EEB MOL EEB MAT EEB
ENG ENG ORF ENG LIN ELE
ELE ELE PHY ELE COS ELE
COS COS ART MOL COS ELE
ELE ELE CEE ELE ECO ENG
CEE CEE COS ELE CEE GEO
EEB EEB EEB EEB CHE LIN
ART ART ELE PHY ART MAE
MUS MUS MUS MUS MAT MAT
PHI PHI ORF PHI MAE MAT
ORF ORF PHI ORF ELE MOL
COS COS COS GEO COS MUS
PHY PHY PHY PHY MOL ORF
COS COS COS LIN COS ORF
MAT MAT MAT MAT EEB ORF
CHM CHM CHM MAT CHM PHI
ORF ORF ORF ORF ENG PHY
COS COS COS MAE COS PHY
REL REL REL REL ARC REL
--- --- --- --- --- ---
1:a 2:  3:  4:  5:  6:b
```

(a) Indata
(b) Sorterad sekvens
(c) Selection-sort
(d) Insertion-sort
(e) Merge-sort
(f) Heap-sort (in-place)

25. Nycklarna i den här uppgiften om binära heapar är stora bokstäver och vi använder bokstavsordning för att ordna dessa nycklar.

    Betrakta följande max-heap representerad som ett binärt träd.



Max-heapen ovan är resultatet av en sekvens av `insert`- och `removeMax`-operationer. Antag att sista operationen i denna sekvens var ett anrop till `insert`. Vilken/vilka nyckel/nycklar skulle kunna vara de(n) som sattes in sist?

26. Randomiserade prioritetsköer.

    Beskriv, gärna med pseudokod, hur man kan lägga till metoderna `sample()` och `removeRandom()` till en implementation av ADT PRIOKÖ. De två metoderna returnerar en nyckel som väljs ut slumpmässigt med likformig sannolikhet bland nycklarna som för tillfället är insatta i prioritetskön, där den senare metoden också tar bort den nyckeln ur prioritetskön.

`PQ()`	skapa en tom prioritetskö
`void insert(Key key)`	sätt in en nyckel i prioritetskön
`Key min()`	returnera den minsta nyckeln
`Key removeMin()`	ta bort och returnera den minsta nyckeln
`Key sample()`	returnera en nyckel vald med likformig sannolikhet
`Key removeRandom()`	ta bort och returnera en nyckel vald med likformig sannolikhet

    Antag att du har tillgång till en slumptalsgenerator `Random.uniform(N)` som returnerar slumptal mellan 0 och $N-1$ med likformig sannolikhet. Din implementation av `sample()` får bara använda konstant tid, medan `removeRandom()` får använda tid proportionell mot $\log N$, där N är antalet nycklar i datastrukturen.

27. Låt A och B vara två mängder av heltalsnycklar. Vi säger att A *separerar* B om det finns tre nycklar $x < y < z$, sådana att x och z finns i B och y finns i A. Antag att alla nycklar är distinkta, att nycklarna i A lagras i ett balanserat binärt sökträd T_A av lämplig sort och att nycklarna i B lagras i ett balanserat binärt sökträd T_B. Konstruera en algoritm som tar T_A och T_B som indata och avgör ifall A separerar B. Din algoritm ska ha exekveringstid $O(\log n)$, där n är det totala antalet nycklar i A och B, i värsta fallet. Notera att din algoritm får använda sig av ev. interna metoder sökträden har och inte behöver begränsa sig till `insert`, `remove` och `find`. Beskriv din algoritm med pseudokod eller naturligt språk och förklara varför dess värstfallskomplexitet är $O(\log n)$.

28. Den här uppgiften handlar om grafer.

    Betrakta två noder x och y som båda finns samtidigt på anropsstacken (den som håller reda på alla rekursiva anrop) vid någon tidpunkt i exekveringen av en djupetförstsökning från noden s i en *riktad* graf. Vilka av följande måste vara sanna utsagor?

    I. Det finns *både* en riktad stig från s till x *och* en riktad stig från s till y i grafen.

    II. Om det *inte* finns någon riktad stig från x till y så finns det en riktad stig från y till x.

    III. Det finns *både* en riktad stig från x till y *och* en riktad stig från y till x.

(a) Endast I.
(b) Endast I och II.
(c) Endast I och III.
(d) I, II och III.
(e) Ingen av utsagorna.

29. Analysera den asymptotiska exekveringstiden för nedanstående kodsnuttar.

(a)
```
sum = 0;
for (i = 0; i < n; i++)
    for (j = 0; j < i * i; j++)
        for (k = 0; k < j; k++)
            sum++;
```

(b)
```
sum = 0;
for (i = 1; i < n; i++)
    for (j = 1; j < i * i; j++)
        if (j % i == 0)
           for (k = 0; k < j; k++)
               sum++;
```

30. Stackar

(a) Föreslå en datastruktur som stödjer stackoperationerna `push` och `pop` samt en tredje operation `findMin` som returnerar det minsta elementet i datastrukturen, samtliga i $O(1)$ värstafallstid.

(b) Visa att om vi lägger till en fjärde operation `deleteMin` som hittar och tar bort det minsta elementet, så måste minst en av de fyra operationerna använda $\Omega(\log n)$ tid.

31. Sant eller falskt? Givet en max-heap med n distinkta nycklar blir resultatet av att sätta in en nyckel som är större än de n nycklarna i heapen och sedan anropa `removeMax` exakt samma heap som den vi ursprungligen hade.

32. En sammanhängande oriktad graf $G = (V, E)$ kallas *2-färgbar* om och endast om det finns en funktion $f : V \to \{\mathsf{red}, \mathsf{blue}\}$ sådan $f(v) \neq f(w)$ när det finns en båge mellan v och w; med andra ord, inga grannar tilldelas samma färg. Konstruera en algoritm med polynomisk värstafallstid som avgör ifall en sammanhängande oriktad graf är 2-färgbar eller ej. Förklara noggrannt varför din algoritm fungerar och varför den bara använder polynomisk tid. Tips: använd en girig approach.