Exempeltentafrågor i TDDD86 Datastrukturer, algoritmer och programmeringsparadigm

5 december 2014

1. Bevisa eller motbevisa följande påståenden genom att använda definitionerna.
   (a) $(n+1)^2 \in \mathcal{O}(n^3)$
   (b) $(n-1)^3 \in \mathcal{O}(n^2)$
   (c) $2^{n+1} \in \mathcal{O}(2^n)$
   (d) $3^{n-1} \in \mathcal{O}(2^n)$
   (e) $(n+1)^2 \in \Omega(n^3)$
   (f) $(n-1)^3 \in \Omega(n^2)$
   (g) $2^{n+1} \in \Omega(2^n)$
   (h) $3^{n-1} \in \Omega(2^n)$
   (i) $(n+1)^2 \in \Theta(n^3)$
   (j) $(n-1)^3 \in \Theta(n^2)$
   (k) $2^{n+1} \in \Theta(2^n)$
   (l) $3^{n-1} \in \Theta(2^n)$
2. Låt $f(n)$ och $g(n)$ vara funktioner sådana att $f(n) \geq 0$, $g(n) \geq 0$ för alla n. Visa:
   (a) $\mathcal{O}(f(n) + g(n)) = \mathcal{O}(\max(f(n), g(n)))$
   (b) $f(n) \cdot g(n) \in \mathcal{O}(f(n) \cdot g(n))$
3. Är det sant att om $\log f(n) \in \mathcal{O}(\log g(n))$ så gäller $f(n) \in \mathcal{O}(g(n))$? Motivera svaret.
4. Bevisa att om $f \in \mathcal{O}(g)$ och $g \in \mathcal{O}(h)$ så gäller $f \in \mathcal{O}(h)$. Vilka värden på n_0 och c får f och h i termer av dem för f och g samt för g och h?
5. Visa att $f \in \Theta(g)$ om och endast om $g \in \Theta(f)$. (Alltså partitionerar Θ klassen av alla funktioner i ekvivalensklasser.)
6. Algoritmerna A och B har värstafallskomplexitet $\mathcal{O}(n^2)$ respektive $\mathcal{O}(n \log n)$. På vissa indata har A kortare exekveringstid än B. Ge tre möjliga förklaringar till att detta fenomen kan uppstå. Är fenomenet fortfarande möjligt om värstafallskomplexiteterna är $\Theta(n^2)$ och $\Theta(n \log n)$?
7. Ge motiverade svar på följande frågor:
   (a) Tidskomplexiteten i *värsta fallet* för en algoritm är $\Omega(n)$. Är det möjligt att algoritmen använder tid $T(n) \in \Omega(n^2)$ för *något* indata?

(b) Tidskomplexiteten i *värsta fallet* för en algoritm är $\Omega(n)$. Är det möjligt att algoritmen använder tid $T(n) \in \Omega(n^2)$ för *alla* indata?

(c) Det *bästa indatat* gör att algoritmen använder $\Theta(n)$ tid. Är det möjligt att algoritmen använder tid $T(n) \in \Omega(n^2)$ för *något* indata?

(d) Det *bästa indatat* gör att algoritmen använder $\Theta(n)$ tid. Är det möjligt att algoritmen använder tid $T(n) \in \Omega(n^2)$ för *alla* indata?

8. Analysera tidskomplexiteten för följande algoritmer:

(a)
```
procedure Mystery(integer n):
for i from 1 to n − 1 do
  for j from i + 1 to n do
    for k from 1 to j do
      en instruktion som exekveras i tid O(1)
```

(b)
```
procedure VeryOdd(integer n):
for i from 1 to n do
  if i is odd then
    for j from i to n do
      x ← x + 1
    for j from 1 to i do
      y ← y + 1
```

(c)
```
procedure Complicated(integer n):
for i from 1 to n do
  for j from n − 1 downto 3 do
    for k from 1 to 5 do
      anrop till en procedur som exekverar i tid O(log n)
```

(d)
```
procedure PrintStars(integer n):
a ← 1
for i from 1 to n do
  a ← a·i
  for j from 1 to a do
    print "*"
  new_line
```

9. Betrakta följande procedur:

```
procedure foo(integer n):
for i from 1 to 4 do
  x ← i
```

Vilka av följande påståenden stämmer för procedurens exekveringstid $T(n)$: $T(n) \in \mathcal{O}(0)$, $T(n) \in \mathcal{O}(7)$, $T(n) \in \mathcal{O}(n)$, $T(n) \in \Omega(7)$? Motivera svaren.

10. Analysera värstafallskomplexiteten för tid och minne hos följande algoritm, där indata består av ett heltal i och en sorterad tabell av heltal av längd n.

```
function (integer i, table A[1..n]): boolean
first ← 1
last ← n
found ← false
while first ≤ last and not found do
  index ← ⌊(first + last)/2⌋
  if i = A[index] then found ← true
  elsif i < A[index] then last ← index − 1
  else first ← index + 1
return found
```

11. Skriv en rekursiv ekvation som karakteriserar tidskomplexiteten hos följande program:

```
function Factorial(integer n): integer
if n = 0 then return 1
else return n· Factorial(n − 1)
```

12. Analysera tids- och minneskomplexitet för följande funktion (som antar att $n > 1$):

```
procedure PrintC(integer n):
array A[1..n]
for i from 1 to n do
  A[i] ← 0
loop1:
  i ← 1
  for j from 1 to n do
    print A[j]
  loop2:
    A[i] ← A[i] + 1
    if A[i] = 2 then
      A[i] ← 0
      i ← i + 1
    else
      exit loop2
    if i = n then
      exit loop1
```

13. Sekvensen X Y U V W läses en gång, tecken för tecken från vänster till höger. Varje tecken som lästs kan först placeras i temporärt minne eller skickas direkt till utskrift. Läsoperationerna kan blandas med operationer som tar bort tecken från minnet och skickar dem till utskrift. På det här sättet får vi på utskriften en permutation av indatasekvensen. Betrakta permutationerna X U W V Y och Y U W X V. Kontrollera, för var och en av dem, om permutationen går att få till om det temporära minnet är:

    (a) en stack

    (b) en kö

    Om svaret är "ja" visa sekvensen av operationer för att få till permutationen. Om svaret är "nej" förklara varför ingen sekvens av operationer kan generera den önskade permutationen.

14. Förklara hur man kan implementera två stackar i en vektor $T[0..n]$ på så sätt att ingen stack får overflow så länge inte båda stackarna sammanlagt innehåller $n + 1$ element. Stackoperationerna ska köra i $\mathcal{O}(1)$ tid.

15. En sträng är ett *palindrom* om den blir samma sträng baklänges (t.ex. strängen "na-

turrutan". Konstruera en algoritm som använder en stack för att kontrollera om en sekvens av n inlästa symboler utgör ett palindrom eller ej. Vad är körtiden för din algoritm?

16. Beskriv hur man kan implementera ADT QUEUE med hjälp av två stackar.

    (a) Förklara idén.

    (b) Beskriv den med pseudokod.

    (c) Antag att du sätter in och tar ut n element i kön. Sammantaget gör vi alltså n anrop till enqueue och n anrop till dequeue. Ordeningen på dessa operationer är sådan att kön barablir tom efter den sista dequeue-operationen. Vad är den värsta exekveringstiden för en enskild enqueue-operation ocn en enstaka dequeue-operation?

    (d) Vad är den amorterade tiden för samtliga operationer?

17. En dubbeländad kö *double-ended queue (deque)* är en sekvens som kan modiferas genom att lägga till och ta bort element i både fram- och bakänden av sekvensen. Vi har med andra ord följande fyra abstrakta operationer:

    - $addFront(E, D)$ lägger till elementet E först i deque D,
    - $deleteFront(D)$ returnerar första elementet i deque D och tar bort det från D.
    - $addEnd(E, D)$ lägger till elementet E sist i deque D,
    - $deleteEnd(D)$ returnerar sista elementer i deque D och tar bort det från D.

    Förklara hur en dequeue kan representeras i sammanhängande minne så att var och en av de fyra operationerna endast använder konstant tid.

18. 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

19. Låt A och B vara två sekvenser av heltal. Antag att båda sekvenserna har längd n. Beskriv en algoritm med exekveringstid $O(n \log n)$ som givet ett heltal m bestämmer om det finns ett heltal a i A och ett heltal b i B sådana att $m = a + b$.

20. 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.

21. 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?

22. 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.

23. 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 DALP-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.

24. Algoritmkonstruktion
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.

25. 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.

26. 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?

27. 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?

28. 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 $\log(N)$.

29. 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?

30. 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) 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

31. 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.

32. 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.

33. 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 slumptalsgenereator `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.

34. 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)

35. 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 $O(\log n)$.

36. 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++;
```

37. En sammanhängaande 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.

38. Beskriv en effektiv implementation av ADT Dictionary för att lagra n element som har en associerad mängd av $r < n$ nycklar som kommer från en totalordning. Alltså, mängden mycklar är mindre än mängden element. Din implementation ska kunna utföra operationen `getAll` i $O(\log(r) + s)$ förväntad tid, där s är antalet element som returneras, operationen `entrySet()` i $O(n)$ tid och resterande operationer i ADT Dictionary i $O(\log(r))$ förväntad tid.

39. ADT Map
Antag att en applikation endast använder tre operationer, `insert()`, `find()`, and `remove()`.

    (a) Under vilka omständigheter skulle du använda ett splayträd istället för en hashtabell?

    (b) Under vilka omständigheter skulle du använda ett $(2, 4)$-träd istället för ett splayträd?

    (c) Under vilka omständigheter skulle du använda en osorterad array istället för ett $(2, 4)$-träd?

40. Konstruera en algoritm för att hitta det minsta antalet bågar som behöver tas bort från en given oriktad graf så att den resulterande grafen är acyklisk.