Tentamen
Datastrukturer (DAT036)

- Datum och tid för tentamen: 2012-12-18, 8:30–12:30.
- Ansvarig: Nils Anders Danielsson. Nås på 0700 620 602 eller anknytning 1680. Besöker tentamenssalarna ca 9:30 och ca 11:30.
- Godkända hjälpmedel: Ett A4-blad med handskrivna anteckningar.
- För att få betyget n (3, 4 eller 5) måste du lösa minst n uppgifter. Om en viss uppgift har deluppgifter med olika gradering så räknas uppgiften som löst om du har löst *alla* deluppgifter med gradering n eller mindre.
- För att en uppgift ska räknas som ”löst” så måste en i princip helt korrekt lösning lämnas in. Enstaka mindre allvarliga misstag kan *eventuellt* godkännas (kanske efter en helhetsbedömning av tentan; för högre betyg kan kraven vara hårdare).
- Lämna inte in lösningar för flera uppgifter på samma blad.
- Skriv din tentakod på varje blad.
- Lösningar kan underkännas om de är svårlästa, ostrukturerade eller dåligt motiverade. Pseudokod får gärna användas, men den får inte utelämna för många detaljer.
- Om inget annat anges så kan du använda kursens uniforma kostnadsmodell när du analyserar tidskomplexitet (så länge resultaten inte blir uppenbart orimliga).
- Om inget annat anges behöver du inte förklara standarddatastrukturer och -algoritmer från kursen, men däremot motivera deras användning.

1. Analysera nedanstående kods tidskomplexitet, uttryckt i `n`:

```
for (int i = 0; i < n; i++) {
    t.insert(i);
}
```

   Använd kursens uniforma kostnadsmodell, och gör följande antaganden:

   - Att `n` är ett ickenegativt heltal, och att typen `int` kan representera alla heltal.
   - Att `t` är ett obalanserat binärt sökträd som till att börja med är tomt.
   - Att den vanliga ordningen för heltal ($... < -1 < 0 < 1 < 2 < ...$) används vid insättning i sökträdet.

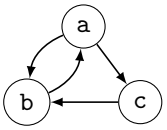
   Onödigt oprecisa analyser kan underkännas; använd gärna Θ-notation.

2. (a) *För trea:* Implementera en algoritm som, givet en riktad, oviktad graf, byter riktning på grafens alla kanter, och analysera algoritmens tidskomplexitet. Använd följande grafrepresentation:

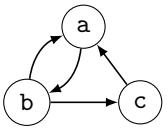
```
public class Graph<A> {
    // Antal noder.
    // Noderna numreras från 0 till nodes - 1.
    private int nodes;

    // Array med nodinnehåll.
    private A[] contents;

    // Array med grannlistor.
    private List<Integer>[] adjacent;

    ...

    // Din uppgift.
    public void reverse() {
        ...
    }
}
```

   Endast detaljerad kod (inte nödvändigtvis Java) godkänns. Du får inte anropa några andra metoder/procedurer, om du inte implementerar dem själv. *Rättelse (inte med i den ursprungliga tesen): Standardmetoder och -konstruerare för listor får användas.*

   Exempel: Om algoritmen appliceras på grafen



så ska resultatet vara följande graf:



Tips: Testa din kod, så kanske du undviker onödiga fel.

(b) *För fyra:* Beskriv en grafrepresentation som möjliggör en $O(1)$-implementation av `reverse`. (Svaret måste motiveras ordentligt.)

3. Beskriv en algoritm som, givet en lista med antalet hårstrån på huvudet hos olika personer, avgör om minst två av de här personerna har samma antal hårstrån på huvudet. Algoritmen måste vara linjär (eller bättre) i antalet personer. Visa att så är fallet.

   Du får anta att det finns en övre gräns för antalet hårstrån på ett huvud (oberoende av algoritmens indata), t ex 200 000. Om du använder antagandet, var i så fall tydlig med hur det används.

4. Uppgiften är att konstruera en datastruktur för en avbildnings-ADT med följande operationer:

   `new Map()` Konstruerar en tom avbildning.

   `insert(`k`, `v`)` Lägger till paret (k, v) till avbildningen. Om det redan finns ett par $p = (k, v')$ så tas p bort.

   `member(`k`)` Avgör om det finns något par (k, v) i avbildningen.

   `nth-smallest(`i`)` Får endast köras om i är ett positivt heltal och avbildningen innehåller minst i element. Om avbildningen innehåller paren $(k_1, v_1), (k_2, v_2), ...$ där $k_1 < k_2 < ...$ så ska operationens resultat vara v_i. Operationen ska alltså ta fram värdet v_i som svarar mot den i-te minsta nyckeln k_i.

   Du kan anta att alla nycklar och värden är heltal.

   Exempel: Följande kod, skriven i ett Javaliknande språk, ska ge `true` som svar:

```
Map m = new Map();
m.insert(5, 5);
m.insert(5, 3);
m.insert(6, 0);
return m.member(5) && (! m.member(1)) &&
       m.nth-smallest(1) == 3 && m.nth-smallest(2) == 0;
```

   Operationerna måste ha följande tidskomplexiteter (där n är antalet par i avbildningen):

   - *För trea:* `new`: $O(1)$, `insert`, `member`: $O(\log n)$, `nth-smallest`: $O(n)$.

- *För fyra:* `new`: $O(1)$, `insert`, `member`, `nth-smallest`: $O(\log n)$.

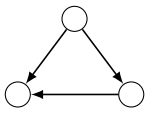
Visa att så är fallet. Implementationen av datastrukturen behöver inte beskrivas med detaljerad kod, men lösningen måste innehålla så mycket detaljer att tidskomplexiteten kan analyseras.

Tips: Konstruera om möjligt inte datastrukturen från grunden, bygg den hellre m h a standarddatastrukturer. Testa din algoritm, så kanske du undviker onödiga fel.

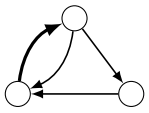
5. Visa att tidskomplexiteten för insättning av n element i en tom dynamisk array är $O(n)$. Du kan anta att varje element sätts in sist i arrayen, att arrayens längd till att börja med är 1, och att arrayens längd fördubblas vid insättning i en full array.

6. Beskriv en effektiv algoritm som avgör om en riktad graf kan göras starkt sammanhängande[1] genom att lägga till (som mest) en enda kant, och analysera dess tidskomplexitet.

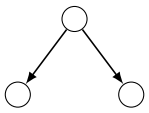
   Exempel: Grafen

   är inte starkt sammanhängande, men blir starkt sammanhängande om vi lägger till en kant:

   Grafen

   är inte heller starkt sammanhängande, och kan inte göras starkt sammanhängande genom att lägga till en enda kant.

   Tips: Börja med att lösa problemet för riktade acykliska grafer.

[1] En graf är starkt sammanhängande om det finns vägar från varje nod till varje annan nod.