Tentamen
Datastrukturer (DAT037)

- Datum och tid för tentamen: 2016-01-09, 14:00–18:00.
- Ansvarig: Nils Anders Danielsson. Nås på 0700 620 602 eller anknytning 1680. Besöker tentamenssalarna ca 15:00 och ca 17:00.
- Godkända hjälpmedel: Ett A4-blad med handskrivna anteckningar.
- För att få betyget n (3, 4 eller 5) på tentan måste man få betyget n eller högre på minst n uppgifter.
- En helt korrekt lösning av en uppgift ger betyget 5 på den uppgiften. Lösningar med enstaka mindre allvarliga misstag kan *eventuellt* ge betyget 5, och sämre lösningar kan ge lägre betyg.
- Om en viss uppgift har deluppgifter med olika gradering (t ex "För trea:", "För fyra:") så behöver man, för att få betyget n på den uppgiften, få betyget n eller högre på alla deluppgifter med gradering n eller lägre.
- Betyget kan i undantagsfall, med stöd i betygskriterierna, efter en helhetsbedömning av tentan bli högre än vad som anges ovan.
- 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. Om inget annat anges får pseudokod 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 (sådant som har gåtts igenom på föreläsningarna), men däremot motivera deras användning.
- Efter rättning är tentamen tillgänglig vid expeditionen på plan 4 i EDIT-huset, och kan granskas där. Den som vill diskutera rättningen kan, inom tre veckor efter att resultatet har rapporterats, kontakta examinatorn och boka tid för ett möte. Notera att tentamen i så fall inte ska tas med från expeditionen.

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

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

   Använd kursens uniforma kostnadsmodell, och gör följande antaganden:

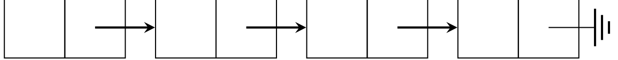
   - Att `n` är ett positivt heltal, och att typen `int` kan representera alla heltal.
   - Att `t` är ett AVL-träd som till att börja med innehåller `n` heltal, alla olika.
   - Att `q` är en leftistheap som till att börja med är tom.
   - Att den vanliga ordningen för heltal ($\ldots < -1 < 0 < 1 < 2 < \ldots$) används vid jämförelser.

   Svara med ett enkelt uttryck (inte en summa, rekursiv definition, eller dylikt). Onödigt oprecisa analyser kan underkännas.

2. (a) *För trea:* Implementera en metod/funktion `size` som beräknar storleken av (antalet element i) en enkellänkad lista, representerad av följande klass:

```
public class SinglyLinkedList<A> {
    private class Node {
        public A contents;  // Nodens innehåll.
        public Node next;   // Nästa nod; null för
                            // sista noden.

        public Node() {}
    }

    private Node first;  // Första noden; null om
                         // listan är tom.

    public SinglyLinkedList() {}

    public int size() {
        // Din uppgift.
    }

    ...
}
```

Exempel: För följande lista (där endast `Node`-objekt och `next`-pekare ritas ut) ska svaret bli 4:

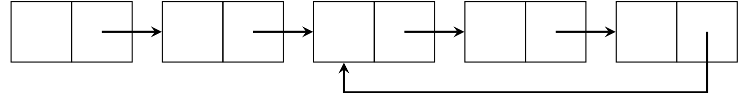


Endast detaljerad kod (inte nödvändigtvis Java) godkänns. Du får inte anropa andra procedurer e d om du inte implementerar dem själv som en del av din lösning, med undantag för enkla heltalsoperationer (som addition), samt följande datastrukturer: binärheapar, leftistheapar, hashtabeller, AVL-träd, prefixträd och skipplistor.

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

(b) *För fyra:* Som för deluppgift (a), men du ska också kunna hantera ”listor” med cykler. Dessutom ska du visa att din metod har tidskomplexiteten $O(n)$, där n är listans storlek.

Exempel: För följande ”lista” ska svaret bli 5:



3. Uppgiften är att konstruera en datastruktur för en mängd-ADT med följande operationer:

   `empty()` eller `new Set()` Konstruerar en tom mängd.

   `insert(`s`)` Lägger till bitsträngen s till mängden. (Lämnar mängden oförändrad om s redan finns i mängden.)

   `someMemberStartsWith(`s`)` Avgör om det finns någon sträng i mängden som börjar med bitsträngen s (d v s om det finns någon sträng s' i mängden, och någon annan sträng s'', så att $s' = ss''$).

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

```
Set strings = new Set();
strings.insert("0000");
strings.insert("1111");
strings.insert("0101");
return strings.someMemberStartsWith("") &&
       strings.someMemberStartsWith("00") &&
       strings.someMemberStartsWith("010") &&
       strings.someMemberStartsWith("1111") &&
       (! strings.someMemberStartsWith("10")) &&
       (! strings.someMemberStartsWith("00000"));
```

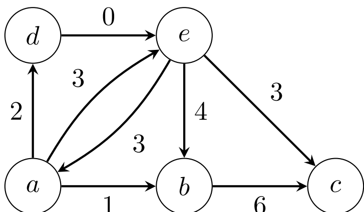
   Operationerna måste ha följande tidskomplexiteter (där n är antalet element i mängden, och ℓ är längden på strängargumentet som benämns s ovan):

   - *För trea:* `new`: $O(1)$, `insert`, `someMemberStartsWith`: $O(\ell n)$.
   - *För fyra:* `new`: $O(1)$, `insert`, `someMemberStartsWith`: $O(\ell \log n)$.
   - *För femma:* `new`: $O(1)$, `insert`, `someMemberStartsWith`: $O(\ell)$.

   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 dina algoritmer, så kanske du undviker onödiga fel. Halvfärdiga lösningar godkänns knappast; om du inte lyckas konstruera en *korrekt* datastruktur som uppfyller kravet för ett visst betyg så kan det vara en bra idé att istället försöka konstruera en enklare datastruktur som uppfyller kravet för ett lägre betyg.

4. Använd Dijkstras algoritm för att beräkna kortaste vägen från startnoden (nod a) till alla noder i följande graf:



Svara med en lista av par (n, d), där n är en nodetikett och d längden på den kortaste vägen från startnoden till nod n. Varje nod i grafen ska förekomma exakt en gång i listan, och noderna ska placeras i listan i den ordning som algoritmen hittar de kortaste vägarna. Första paret ska alltså vara $(a, 0)$.

5. (a) *För trea:* Använd LSD-radixsortering för att sortera följande lista: $[357, 310, 824, 794, 150]$. Använd talbasen 10, d v s sortera med avseende på en decimal siffra i taget. Svara med en sekvens av listor: listan efter sortering med avseende på första siffran, listan efter sortering med avseende på andra siffran, o s v.

   (b) *För fyra:* När man implementerar LSD-radixsortering kan man använda olika underliggande sorteringsalgoritmer för att sortera med avseende på en given nyckel (t ex en given siffra, som i (a)-uppgiften). För vilka av följande underliggande sorteringsalgoritmer ger LSD-radixsortering alltid rätt svar?

   - Heapsort.
   - Mergesort.

   Anta att ”normala” implementationer av algoritmerna används. Om svaret för en viss algoritm är nej, ge då en lista för vilken algoritmen ger fel svar, och förklara varför algoritmen inte fungerar.

6. Beskriv en algoritm som uppfyller följande krav, och analysera algoritmens tidskomplexitet:

   - Indata: En oviktad, riktad, acyklisk graf G, samt två noder a, b i G.
   - Algoritmen ska avgöra om det finns någon nod c i G som kan nås från både a och b (via vägar av längd 0 eller längre), och som dessutom uppfyller kravet att alla andra noder som kan nås från både a och b kan nås från c.

   Algoritmen behöver inte beskrivas med detaljerad kod, men lösningen måste innehålla så mycket detaljer att tidskomplexiteten kan analyseras. För högsta betyg krävs att algoritmen är (någorlunda) effektiv; onödigt långsamma lösningar kan inte få högre betyg än fyra.

   Exempel:

Graf	Korrekt svar
a b	Nej
a b	Ja
a b	Nej

Graf	Korrekt svar
a b	Ja
a b	Ja