Tentamen
Datastrukturer, DAT037

- Datum och tid för tentamen: 2018-08-23, 8:30–12:30.
- Ansvarig: Fredrik Lindblad. Nås på tel nr. 031-772 2038. Besöker tentamenssalarna ca 9:30 och ca 11:00.
- Godkända hjälpmedel: Ett A4-blad med handskrivna anteckningar (fram- och baksida).
- Tentan innehåller 6 uppgifter. Varje uppgift får betyget U, 3, 4 eller 5.
- 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.
- Betyget kan i undantagsfall, med stöd i betygskriterierna för DAT037, 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 ansvarig lärare och boka tid för ett möte. Notera att tentamen i så fall inte ska tas med från expeditionen.

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

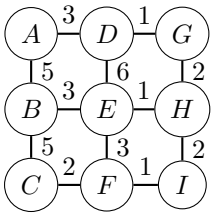
```
for (int i = 0; i < n; i++) {
    int x = q.dequeue();
    if (s.contains(x))
     l.addLast(x);
    else
     l.addFirst(x);
}
```

   Använd kursens uniforma kostnadsmodell och gör följande antaganden:

   - Att `int` kan representera alla heltal och att `n` är ett positivt heltal.
   - Att `q` är en kö av heltal med från början `n` element och att kön är implementerad med en länkad lista.
   - Att `s` är en mängd av heltal implementerad med ett AVL-träd och att mängden innehåller max $2n$ element.
   - Att `l` är en från början tom lista med heltalselement och att listan är implementerad med en dynamisk array.

   Analysen ska bestå av en matematisk uträkning av tidskomplexiteten och ska hänvisa till programkoden. För datastrukturernas operationer kan du hänvisa till standardimplementeringarnas komplexitet. Svara med ett enkelt uttryck (inte en summa, rekursiv definition eller dylikt). Onödigt oprecisa analyser kan underkännas.

2. Visa hur Dijkstras algoritm beräknar närmsta vägen från nod A till alla andra noder i följande graf.



   Redovisa genom att beskriva algoritmens tillstånd steg för steg.

3. Konstruera en datastruktur som representerar en mängd av heltal.

   Datastrukturen ska ha följande operationer:

   `empty()` Skapar en tom mängd.

   `add(`x`)` Lägger till heltalet x till mängden. Om x redan finns i mängden förändras den inte.

   `remove(`x`)` Tar bort heltalet x från mängden. Om x inte finns i mängden förändras den inte.

   `member(`x`)` Returnerar `true` om x finns i mängden, annars `false`.

   `ithsmallest(`i`)` Returnerar det i:te minsta talet i mängden. Om mängden innehåller talen $x_0, x_1, \ldots, x_{n-1}$ och $x_0 < x_1 < \cdots < x_{n-1}$ så returneras x_i. i kan antas vara icke-negativt och mindre än antalet element i mängden.

   Låt n vara antalet tal i mängden. Operationerna ska ha följande tidskomplexitet:

   - *För trea:* `empty`: $O(1)$, övriga operationer: $O(n)$
   - *För fyra eller femma:* `empty`: $O(1)$, övriga operationer: $O(\log n)$

   Oavsett betyg ska du motivera att komplexitetskraven är uppfyllda.

   Du kan använda eller utgå från standarddatastrukturer och -algoritmer utan att beskriva dem i detalj. Implementationen av datastrukturen behöver inte beskrivas med detaljerad kod, men lösningen måste innehålla så mycket detaljer att tidskomplexiteten kan analyseras.

4. Uppgiften handlar om binära heapar. Du behöver bara lösa en av följande två delar.

   För trea: Följande array representerar en binär min-heap.

3	5	8	10	8	13	16	12	13	15	20	20	14	

   Hur ser arrayen ut efter att talet 2 har lagts in i heapen?

   För fyra eller femma: Implementera operationerna för att skapa en tom binär min-heap och att lägga till ett element till heapen. Elementen ska vara heltal och prioriteten bestäms av den vanliga ordningen. Gör detta genom att komplettera följande klass:

```
class BinHeap {
    // att göra

    BinHeap() {
        // att göra
    }
    void add(int x) {
        // att göra
    }
    int deleteMin() {
        ....
    }
}
```

   Du behöver redovisa vilka fält som används för representationen av heapen samt hur konstruktorn och metoden `add` implementeras.

   Ange och motivera tidskomplexiteten för `add`.

   Endast detaljerad kod godkänns. Med undantag av dynamisk array får inga standarddatastrukturer eller -algoritmer användas utan att även implementera dessa. Hjälpmetoder som du också implementerar är tillåtet.

5. Följande klass implementerar heltalsmängd med ett binärt obalanserat sökträd.

```
class BST {
    class Node {
        int contents;  // nodens innehåll
        Node left, right;   // vänster och höger delträd
    }
    Node root;  // roten
    void add(int x) { .... }
    void remove(int x) { .... }
    ....
}
```

Implementera en av följande metoder:

- *För trea:* `add(int x)` som lägger till talet `x` till mängden om det inte finns där.
- *För fyra eller femma:* `remove(int x)` som tar bort talet `x` från mängden om det finns där.

Du kan bortse från de problem som kan uppstå då en rekursiv metod används på ett obalanserat, och därmed potentiellt högt, träd. Det är alltså tillåtet att implementera metoderna rekursivt.

Endast detaljerad kod godkänns, ej pseudokod. Att använda hjälpmetoder är tillåtet, men du får inte anropa några andra metoder utan att visa hur de implementeras.

6. Konstruera en datastruktur som representerar en mängd av heltalspar, (x, y). En mängd har ett speciellt talpar associerat till sig som kallas mittpunkt. Mittpunkten räknas inte till själva mängden av talpar.

   Datastrukturen ska ha följande operationer:

   `empty()` Skapar en mängd med mittpunkten $(0, 0)$.

   `setcenter(`x`, `y`)` Ändrar mittpunkten till (x, y).

   `add(`x`, `y`)` Lägger till talparet (x, y) till mängden. Om talparet redan finns i mängden (bortsett från mittpunkten) så förändras den inte.

   `member(`x`, `y`)` Returnerar `true` om (x, y) finns i mängden (bortsett från mittpunkten), annars `false`.

   `removeclosest()` Tar bort ett av de talpar som ligger närmast mittpunkten. Avståndet mellan två punkter (x_1, y_1) och (x_2, y_2) definieras som $|x_1 - x_2| + |y_1 - y_2|$. Om mängden är tom (bortsett från mittpunkten) så förändras den inte.

   Låt n vara antalet talpar i mängden. Operationerna ska ha följande tidskomplexiteter:

   - *För trea:* `empty`: $O(1)$, övriga operationer: $O(n)$.
   - *För fyra eller femma:* `empty`: $O(1)$, `setcenter`: $O(n)$, `add`: $O(\log n)$, `member`: $O(1)$, `removeclosest`: $O(\log n)$.

   Oavsett betyg ska du motivera att komplexitetskraven är uppfyllda.

   Du kan använda eller utgå från standarddatastrukturer och -algoritmer utan att förklara hur de fungerar. Implementationen av datastrukturen behöver inte beskrivas med detaljerad kod, men lösningen måste innehålla så mycket detaljer att tidskomplexiteten kan analyseras.