Algoritmer och datastrukturer

Kursanvisningar VT 2012

Henrik Bergström

Historik

Detta är ett levande dokument som kommer att uppdateras kontinuerligt under kursens gång. Det är alltså ingen idé att skriva ut den här versionen. Veckouppgifterna kan uppdateras fram till och med introduktionsföreläsningen motsvarande vecka. Därefter kan de förtydligas, men inte egentligen ändras.

Eventuella fel, eller andra konstigheter rapporteras till `henrikbe@dsv.su.se`*. En uppdaterad version av dokumentet finns alltid i Moodle:* `https://ilearn.dsv.su.se/course/view.php?id=100`*.*

2012-02-13 Mindre uppdatering av grafuppgiften.

2012-02-09 Lade in förslag på pooluppgifter för träd, hashning och prioritetsköer.

2012-02-03 Uppdaterade uppgiftstexten för sorteringuppgiften.

2012-01-30 Uppdaterade uppgiftstexten för hashtabellsuppgiften vecka 5 och förslagen på pooluppgifter från vecka 3.

2012-01-23 Lade in förslag på poolfrågor hämtade från första veckans inlämningar.

2012-01-21 Uppdaterade kraven för träduppgiften vecka 4.

2012-01-18 Fixade en felaktig uppräkning i algoritmanalysuppgiften för första veckan.

2012-01-13 Den första versionen.

Innehåll

Del I

Kursen

1 Om kursen

Detta dokument innehåller kursanvisningar för kursen *Algoritmer och datastrukturer* (ALDA) för kursomgången vårterminen 2012. Kursen är obligatorisk på det datavetenskapliga programmet och på spelprogrammets konstruktionsspår samt valbar på flera andra program.

1.1 Kursmål

Studenten ska efter avslutad kurs kunna:

- analysera datastrukturer och algoritmer med avseende på korrekthet och effektivitet
- resonera kring lämplighet, effektivitet, etc. hos två eller flera lösningar av samma problem
- implementera och dokumentera vanliga datastrukturer och algoritmer som en del av ett klassbibliotek

1.2 Litteratur

Kurslitteraturen är *Data Structures and Algorithm Analysis in Java* av Mark Allen Weiss, Addison-Wesley, 2007 [11].

Kursen bygger på boken och vi kommer inte att ägna någon egentlig föreläsningstid åt sådant som står i den; det förutsätts du kunna läsa in på egen hand. Du måste alltså ha boken från dag ett. Ämnet är dock ganska standardiserat, så andra böcker kan fungera; men vi lämnar inga garantier.

1.3 Förkunskapskrav

Kursen förutsätter att du kan programmera ganska väl. De officiella förkunskapskraven är *2*7,5 hp objektorienterad programmering, t.ex. kurserna OOP och PROG2*, och vi förutsätter att du tagit till dig dessa kurser, eller motsvarande.

1.4 Kontaktinformation

Kursansvarig lärare är Henrik Bergström. Han träffas enklast i samband med undervisning, men ett besök på rum 7421 på plan 7, eller ett mejl till `henrikbe@dsv.su.se` kan också fungera.

2 Upplägg

Kursen är uppbyggd kring veckoteman som motsvarar ett eller två kapitel i kursboken. Varje vecka inleds med en kort introducerande föreläsning på en timme. Resten av veckan ägnas åt eget arbete. Som stöd finns det frågestunder inlagda tisdag, onsdag och torsdag varje vecka. Dessa kan ses som en sorts studentdrivna minilektioner. Vi kommer inte att presentera något nytt material, utan diskutera de frågor som finns. Finns det inga frågor; ja då blir det *väldigt* korta tillfällen.

Till skillnad från de flesta kurser på DSV så är examinationen inte uppdelad i flera moment utan består av en muntlig examination, eller munta, på 7,5hp. Det enda du *behöver* göra för att bli godkänd är alltså att klara muntan.

2.1 Inspelning

Såväl introduktionerna som frågestunderna kommer att spelas in[1], men det är värt att tänka på att frågestunderna är avsedda att vara liveevent. Går man inte på dem och ställer frågor kommer de att ge mycket lite.

2.2 Muntaförfarandet

Totalt består kursinnehållet av ett nio olika teman, se figur 2.1, som motsvarar ett eller ett par kapitel i kursboken. Under kursens gång kommer vi att tillsammans bygga upp en pool av frågor för dessa olika teman. Vid muntan kommer du att ges en slumpmässigt vald fråga ur poolen för var och en av dessa teman, och du måste klara alla[2] för att bli godkänd på muntan. Varje deltagare ges max en timmes muntatid. Är man inte godkänd då så är man underkänd och får gå upp vid nästa muntatillfälle.

	Tema	Kapitel	Vecka
1	Linjära datastrukturer: lista, stack och kö	1 & 3	3
2	Algoritmanalys	2	3 & 4
3	Träd	4	4
4	Hashtabeller och hashning	5	5
5	Prioritetsköer	6	5
6	Sortering	7	6
7	Grafer	9	7
8	Algoritmdesigntekniker	10	8
9	Avancerade datastrukturer	12	9

Figur 2.1: Teman

[1] Givet att utrustningen fungerar, vi kommer ihåg det, etc.

[2] Egentligen bara nästan alla eftersom tema nio inte är ett krav för att få ett E.

Vid muntan får man ha med sig kursboken, men inga andra hjälpmedel förutom det som krävs för att redovisa en eventuell programmeringsuppgift.

Låter det mycket med ett tiotal frågor man måste klara på, i praktiken, strax under en timme? Det är det. Som tur är finns det två möjligheter att minska på osäkerheten: programmeringsuppgifter och veckouppgifter.

2.3 Programmeringsuppgifter

Frågepoolen till muntan kommer, av naturliga skäl, främst att innehålla teorifrågor. Eftersom vi vill uppmuntra till praktiskt programmering finns det också en separat pool med programmeringsuppgifter kopplade till varje tema. Om du vill får du göra upp till två sådana uppgifter ur olika teman och redovisa dem vid muntan istället för de normala frågorna.

Denna möjlighet ger dig inte mer tid på muntan eftersom uppgifterna redovisas där, men däremot goda möjligheter att styra betyget i rätt riktning.

2.4 Inlämningsuppgifterna

Kursledningen har en stark tilltro till kontinuerlig examination och erbjuder därför en möjlighet att klara hela, eller delar av, muntan innan själva muntatillfället. Denna möjlighet ges i form av inlämningsuppgifter och peer reviews varje vecka.

Om man inte lämnar in en uppgift i tid, eller inte blir godkänd, finns det ingen som helst möjlighet att komplettera. Då får man klara av detta tema på muntan istället.

2.5 Betygskriterier

Varje fråga i frågepoolen för muntan, liksom majoriteten av de olika programmeringsuppgifterna, har tre olika betygssteg: underkänt, godkänt och väl godkänt. Veckouppgifterna kan bara ge ett godkänt, men kan kompletteras på muntan. Det slutgiltiga betyget på kursen vägs samman från dessa olika betyg i enlighet med kriterierna nedan.

För samtliga betyg gäller att kraven för alla lägre betyg också måste vara uppfyllda. Detta gäller även om det inte explicit står i betygskriterierna.

E: För detta betyg måste du uppvisa en grundläggande förståelse för samtliga teman utom nummer nio: avancerade datastrukturer. Du visar detta genom att bli godkänd på motsvarande inlämningsuppgifter, programmeringsuppgifter eller muntafrågor; eller någon kombination av dessa. Vid redovisningen av eventuell programmeringsuppgifter ska du kunna förhålla dig självständig till koden. Det är alltså fullständigt okej att du har samarbetat med någon annan om lösningen, men det är *du* som ska visa att *du* förstått lösningen och skulle kunna klara av att lösa en motsvarade uppgift på egen hand.

D: För detta betyg måste du uppvisa en grundläggande förståelse för samtliga teman och dessutom visa att du klarar av att implementera en lösning för något av problemen i programmeringsuppgiftspoolen. Förutom kravet på självständighet vid redovisning av programmeringsuppgiften så ska implementationen uppfylla grundläggande krav på god kodkvalitet. Detta inkluderar indentering, god och konsekvent namngivning, korrekta skyddsnivåer, korrekt användning av typer, etc. Minst en, icketrivial, metod ska vara ordentligt dokumenterad med hjälp av JavaDoc [8] eller motsvarande.

Eftersom de extra kraven på implementationen inför en viss grad av subjektivitet i bedömningen kan det ges *en* möjlighet till komplettering av dessa delar.

C: För detta betyg måste du uppvisa en djupare förståelse för minst två teman. Minst ett av dessa måste vara en fråga ur frågepoolen. Tema ett: linjära datastrukturer, kan normalt inte bidra till detta. Den ses som alltför grundläggande. Djupare förståelse visar du genom att klara av uppgifter på VG-nivå och genom att kunna diskutera uppgiften vidare. Att bara kunna ett svar på VG-nivå är alltså inte tillräckligt. Du måste vara självständig.

B: För detta betyg måste du uppvisa en djupare förståelse för minst tre teman, inklusive minst ett av de tre sista temana: grafer, algoritmdesigntekniker eller avancerade datastrukturer. Lösningen på den eller de programmeringsuppgifter du gjort måste vara fullständig, och av god kvalitet. Det får alltså inte finnas några egentliga tveksamheter, vare sig i implementationen eller i dokumentationen. Normalt kan du inte få detta betyg om dina programmeringsuppgifter kräver komplettering.

A: För detta betyg måste du uppvisa en djupare förståelse för minst fyra teman, inklusive minst två av de tre sista. Du kan inte få detta betyg om dina programmeringsuppgifter kräver komplettering.

2.6 Samarbete, plagiat och fusk

De hederskodex som finns på DSV (`http://dsv.su.se/utbildning/studentinfo/hederskodex`) och SU (`https://pp1.it.secure.su.se/pub/jsp/polopoly.jsp?d=970&a=2979`) gäller naturligtvis även på denna kurs.

Alla inlämnade uppgifter kommer att kontrolleras för plagiat. De plagiatkontrollerade uppgifterna kommer också att sparas i en databas över inlämnade uppgifter så att de kan användas som underlag för plagiatkontroll i framtiden. Alla fall av plagiat, eller andra försök att fuska, kommer obönhörligen att anmälas, och följden kan i värsta fall bli avstängning från studierna.

Det är tillåtet att samarbeta i grupper om upp till tre personer med inlämningsuppgifterna och programmeringsuppgifterna. Större grupper än så är inte tillåtna, och inte heller att kopiera, eller i alltför hög grad basera, sina lösningar på någon annans arbete. Detta oavsett om denna andra person är en annan student, en författare, en bloggare, eller någon annan. Arbetet ska vara ert eget. Om du är det minsta tveksam om vad som gäller så chansa inte utan fråga!

Arbetar man i grupp är hela gruppen solidariskt ansvarig för inlämningen. Om en person i gruppen plagierar kommer alltså alla att råka illa ut.

2.7 Hur pass vanligt är plagiat?

Av ovanstående avsnitt att döma kommer kursledningen att göra sitt bästa för att sätta dit alla studenter på kursen för plagiat. Så är naturligtvis inte fallet. Den överväldigande majoriteten brukar sköta sig exemplariskt, men tyvärr finns det ibland någon eller några som missköter sig, och vi kommer att göra vårt bästa för att se till att det inte lönar sig. Examinationsmomenten är till för att ni ska kunna visa upp *era egna* kunskaper samtidigt som ni får övning och feedback, och det får ni inte om ni plagierar.

2.8 Vad gäller för gamla studenter?

Studenter från tidigare kursomgångar som ännu inte är godkända ombedes kontakta kursledningen för genomgång av vad som måste göras för att bli godkänd. Detta varierar från individ till individ beroende på hur pass stora delar man är klar med sen tidigare.

3 Praktiskt om muntan

De två sista veckorna på kursen är helt vikt åt muntan. Läs igenom detta avsnitt noga, och se till att ta upp eventuella frågetecken i god tid. Vid själva muntatillfället kommer det inte att finnas tid för några egentliga frågor.

3.1 Bokning av tid

Bokning av muntatid görs i Daisy, och det är först till kvarn som gäller. Varje grupp rymmer tre personer, och det totala antalet platser är större än det förväntade antalet studenter, så alla bör ha en rimlig chans att hitta en tid som inte krockar med examinationen på den andra kursen man går.

Vilka tider de olika grupperna i Daisy motsvarar syns i schemat nedan:

	Tid	Måndag	Tisdag	Onsdag	Torsdag	Fredag	Lördag
V. 10	9-12	1	4	7	10	13	15
	13-16	2	5	8	11	14	16
	17-20	3	6	9	12		
V. 11	9-12	17	20	23*	26	29	31
	13-16	18	21	24*	27	30	32
	17-20	19	22	25*	28		

Figur 3.1: Muntagrupper

3.2 Lokal

Samtliga muntatillfällen förutom onsdagen vecka 11 är i seminarierum 6202. Onsdagspasset vecka 11, markerat i tabellen ovan, är istället i lektionssal 513.

3.3 Extra muntatider

Under vecka åtta och nio, alltså veckorna innan de normala muntatillfällena, kommer det att ligga ett par extra muntatillfällen. Dessa är framför allt avsedda för studenter från tidigare kursomgångar som ännu inte är godkända på kursen, men inget hindrar studenter från årets omgång att gå på dessa om de vill. Observera dock att man inte får gå på något av dessa tillfällen, och sen direkt på ett av de ordinarie om man skulle köra. Man får vänta tills nästa muntatillfälle.

3.4 Om man inte blir godkänd

Varje muntatillfälle är helt och hållet separat från alla andra. Om du inte blir godkänd så tar du inte med dig något till nästa tillfälle, förutom de avklarade

veckouppgifterna naturligtvis. Dessa förs vidare eftersom vi vill uppmuntra folk till att göra dem.

Anledningen till att man inte får ta med sig avklarade teman till nästa muntatillfälle är enkel: det skulle ge möjliggöra en alltför effektiv strategi för att få höga betyg.

3.5 Kom i tid!

Varje muntapass är tre timmar, och är avsett för tre studenter. Man har alltså max en timme på sig.

En annan sak som är viktig att känna till är att vi inte väntar mer än tio minuter på någon deltagare. Inklusive den första. Har ingen dykt upp då är hela detta muntatillfälle kört för hela gruppen, precis som om ingen dykt upp till en tenta. Tiden vi får vänta dras också av från muntatiden för den som kommer försent. Du bör alltså planera att vara på plats i god tid innan starttiden.

Varje munta är individuell, så två personer i gruppen kommer alltid att få vänta. Detta är olyckligt, men nödvändigt för att få ett hanterbart schema. Jämfört med att skriva en vanlig tenta så är dock inte väntetiden speciellt lång, så vi har inte så dåligt samvete.

3.6 Ta med legitimation

Muntan är en lika formell examination som en tenta. Du måste alltså kunna legitimera dig.

3.7 Tillåtna hjälpmedel

Du får ha med dig kursboken på muntan om du vill. Räkna dock inte med att det är till någon större hjälp. Du kan använda den för att slå upp någon enstaka sak, men du kommer inte att ha tid att läsa på i någon egentlig omfattning.

I kursboken får rimliga mängder anteckningar finnas. Med rimliga menas att det är okej med understrykningar, förtydliganden etc., men inte med långa lösningsförslag, extra blad etc.

Tips!

Förutom boken får du inte ha med dig några ytterligare hjälpmedel förutom det som krävs för att redovisa eventuella programmeringsuppgifter. Vår erfarenhet är att det bästa och mest effektiva sättet att redovisa dessa är att ha dem utskrivna på papper, eventuellt med ett UML-diagram eller andra illustrationer som stöd. Du får ha med dig en dator om du vill, men risken att något går snett eller tar längre tid ökar betydligt.

3.8 Frågeurval

Frågorna på muntan kommer att utgöras av ett slumpmässigt urval, en för varje tema, ur den frågepool som byggs upp under kursen. Du bestämmer själv ordningen du vill besvara dem i. Ett tips är att börja med de frågor som

krävs för att bli godkänd. Information om vilka teman du redan är godkänd på kommer att finnas tillgänglig vid muntatillfället om du är osäker.

4 Praktiskt om veckouppgifterna

Veckouppgifterna finns i detta kompendium på sidan 35 och lämnas in via Moodle absolut senast fredag klockan 8:59 enligt Moodle-serverns klocka. Efter denna tidpunkt finns ingen som helst möjlighet att lämna in. Tänk därför på att lämna in med viss marginal. *Obs, viktigt!*

Klockan 9:00, även detta enligt serverns klocka, slumpas de inlämnade uppgifterna ut och varje person som skickat in får tre andra personers svar att genomföra en så kallad peer review på. En peer review är en granskning, och denna granskning ingår också i examinationen.

4.1 LaTeX och PDF

Inlämning sker i form av en rapport. Denna rapport ska vara i PDF-format och ska göras i LaTeX. För mer information om LaTeX se sidan 18. En mall för rapporten finns i Moodle.

4.2 Det är det ni lämnar in som ni blir bedömda på

Det ges inga möjligheter att komplettera veckouppgifterna, och det är det ni lämnar in som ni blir bedömda på. Inget annat. Tänk därför på att rapporten ska vara möjlig att läsa, och att den tydligt ska visa vad ni har gjort.

Här är det kanske också läge att påminna om kravet på att allt arbete ska vara ert eget. Se sidan 5 för mer information.

4.3 Programspråk

Veckouppgifterna innehåller normalt både teorifrågor och programmeringsuppgifter. Programmeringsuppgifterna ska lösas i Java. För de programmeringsuppgifter som görs till muntan gäller andra regler. Se sidan 47.

4.4 Hur gör man med programkoden?

Programkoden ska ingå i rapporten. Hur ni får in den på ett bekvämt sätt diskuteras på sidan 26.

När det gäller programkod är det extra viktigt att tänka på den som ska läsa koden. Använd rubriker för att leda läsaren. Om det bara är en bit av koden som är intressant så inkludera bara den. Lägg sedan all kod som bilaga i slutet för dem som vill ha det fulla sammanhanget.

4.5 Grupper

Det är rekommenderat att göra veckouppgifterna i grupp. Den maximala gruppstorleken är tre personer, och ni behöver inte anmäla vilken grupp ni tillhör på något sätt förutom att skriva upp allas namn på inlämningen. Ni får växla grupp mellan olika veckor.

4.6 Alla i gruppen skickar in

Som tidigare sagts så är det tillåtet att göra inlämningsuppgifterna i grupper om upp till tre personer. Detta gäller dock bara själva lösandet av uppgifterna. Granskningsförfarandet är helt individuellt. Detta betyder att alla medlemmar i gruppen måste skicka in sin egen version av uppgiften och delta i granskningen.

4.7 Granskningen är en del av examinationen

Första gången vi testade granskningsmodellen på ALDA så fanns det några studenter som tyckte att de inte fick tillräckligt bra feedback. Vi vill därför trycka på att granskningen är en viktig del av examinationen, och att man måste bli godkänd på den också för att bli godkänd på uppgifterna. Du kan alltså göra en perfekt inlämningen och fortfarande bli underkänd om granskningen inte är tillräckligt bra gjord. Räkna med att du kommer att behöva lägga ner en till två timmar per vecka på granskningarna. Så pass noga ska det vara.

Ibland kommer du att stöta på uppgifter som är perfekt lösta, eller uppenbart underkända, och då är det bara att säga det och gå vidare. Men: detta är undantag. De flesta lösningsförsök brukar ligga någonstans mitt emellan dessa extremer.

Tänk också på att du ska granska *alla* uppgifterna. Detta gäller även om det är uppenbart redan från den första att det inte kommer att kunna godkännas.

Det finns en checklista för granskning i Moodle. Använd den! För den som inte har något emot att läsa väldigt liten text finns den också på sidan 12.

4.8 Det går inte att lämna in på något annat sätt!

Den överväldigande majoriteten av alla studenter brukar sköta inlämningen exemplariskt, men det finns alltid några undantag. Vi vill därför redan nu säga att veckouppgifternas upplägg är väldigt fyrkantigt. Lämnar du inte in i rätt tid, på rätt sätt, i rätt format, etc. så finns det inga möjligheter att fixa det på något annat sätt. Då är dessa uppgifter körda för dig. Detsamma gäller om du inte deltar i, eller inte gör ett tillräckligt bra jobb, i granskningen.

Det finns acceptabla skäl att få mer tid på sig, till exempel dyslexi, men detta kommer i sådana fall att hanteras indivudellt, och nästan alltid vara kopplat till muntan och inte till veckouppgifterna.

PROGRAMMERINGSUPPGIFTER			
Är den huvudsakliga funktionaliteten korrekt implementerad enligt uppgiften?	Ja		Nej
Är all funktionalitet korrekt implementerad?	Ja	Nej	
Följer koden gängse kodkonventioner för Java?	Ja	Nej	
Är koden dokumenterad på ett sådant sätt att den är förståelig för målgruppen? Detta kan göras i koden i form av kommentarer, eller direkt i rapporten.	Ja	Nej	

ALGORITMANALYSUPPGIFTER			
Är resultatet av analysen korrekt?	Ja, för alla deluppgifter	Ja, för de flesta deluppgifterna	Nej
Går det att följa resonemanget i analysen?	Ja, för alla deluppgifter	Ja, för de flesta deluppgifterna	Nej

SKRIVUPPGIFTER OCH TEORETISKA UPPGIFTER			
Är hela uppgiften besvarad?	Ja	Ja, nästan	Nej
Är det som står korrekt?	Ja	Ja, nästan	Nej
Är svaret läsbart av den tilltänkta målgruppen? I vissa uppgifter finns det en explicit målgrupp. För övriga uppgifter är det studenter som går kursen.	Ja		Nej

POOLUPPGIFTER			
Behandlar frågan rätt ämne för temat?	Ja		Nej
Går frågan att besvara?	Ja		Nej
Uppfylls eventuella restriktioner på frågan?	Ja		Nej
Är svårighetsgraden på frågan lämplig för kursen?	Ja	Nja	Nej
Finns det en tydlig uppdelning i vad som krävs för att bli godkänd och vad som krävs för att få ett VG?	Ja		Nej
Är frågan av lämplig längd för muntan? Den bör normalt sett inte vara mer än max en halv A4, gärna betydligt mindre.	Ja	Nej	

GENERELLT			
Har alla uppgifter gjorts?	Ja		Nej
Är rapporten läsbar?	Ja	Nja	Nej
Är rapporten gjord i Latex och följer mallen?	Ja		Nej

Figur 4.1: Checklista för granskning

5 JUnit

Ett problem vi har haft med de praktiska uppgifterna på kursen är att alla inte testar sina lösningar tillräckligt innan de skickas in. Vi har därför bestämt att alla uppgifter där det är möjligt ska ha en uppsättning automatiska testfall som *måste* gå igenom innan man får skicka in koden. Dessa testfall kommer att använda sig av JUnit, ett ramverk för enhetstestning som är mer eller mindre standard för Java-programmerare.

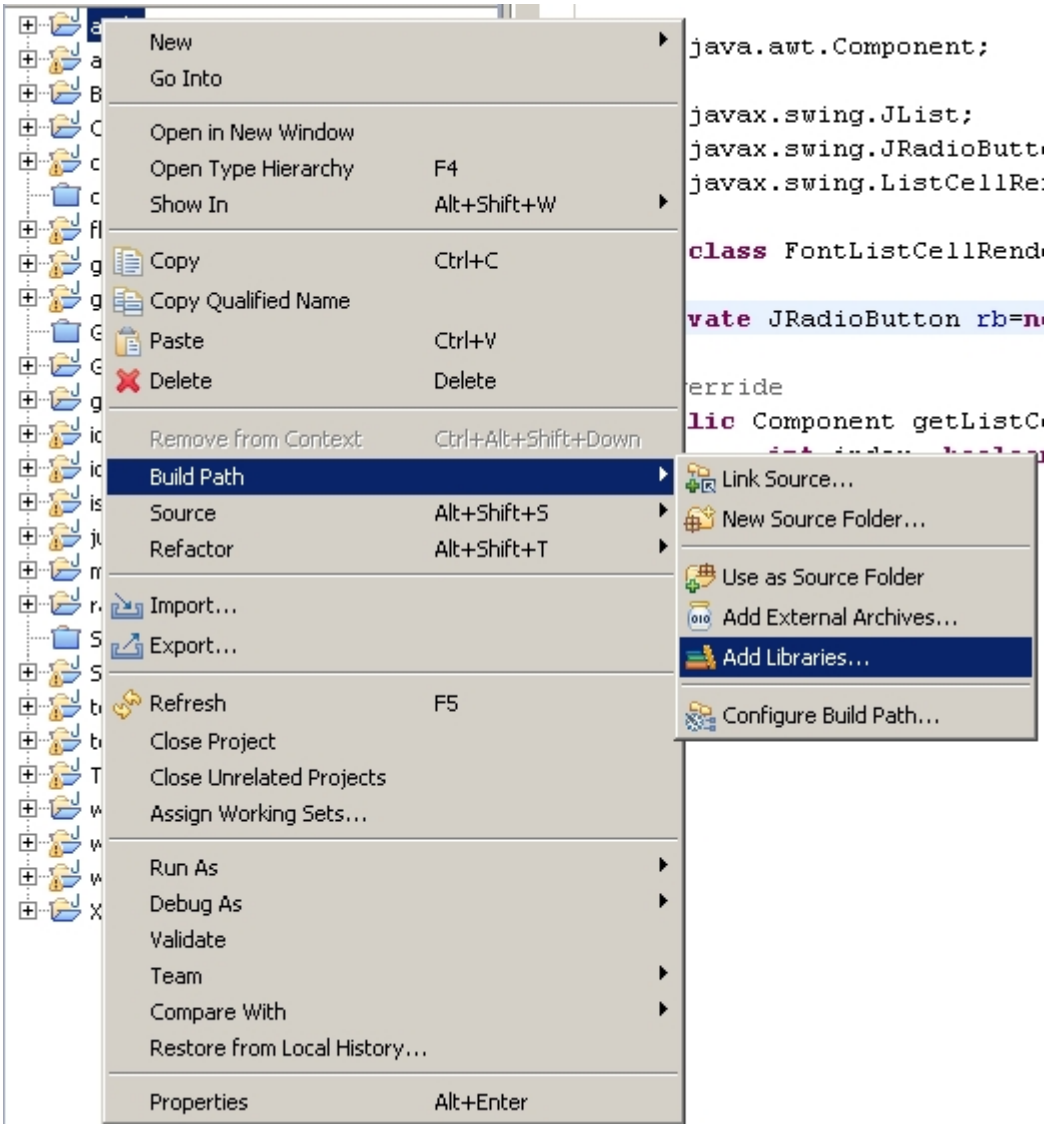
Ni kommer inte att behöva göra några (större) ändringar i testfallen, så för kursens räkning räcker det att ni kan köra JUnit-testfallen som vi förser er med. Instruktioner för detta finns nedan. För den som vill veta mer finns det hur mycket information som helst på webben. Observera dock att vi kör den ”nya” versionen av JUnit, version 4, som skiljer sig ganska mycket från de tidigare versionerna. Den officiella hensidan för JUnit är `http://www.junit.org/`

5.1 Miljön

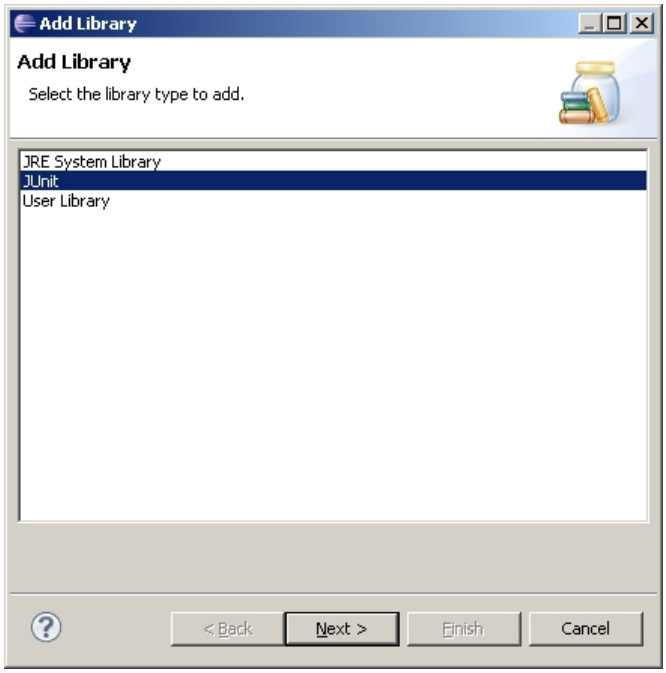
När det gäller programmering kommer vi att förutsätta att ni använder Eclipse. Det här är inget krav, JUnit kan köras på många olika sätt, inklusive från kommandoprompten, men Eclipse kommer att vara den enda miljö som vi officellt stödjer. Vi kan eventuellt hjälpa dig även med andra miljöer, men där lämnar vi inga som helst garantier.

5.2 Förberedelser

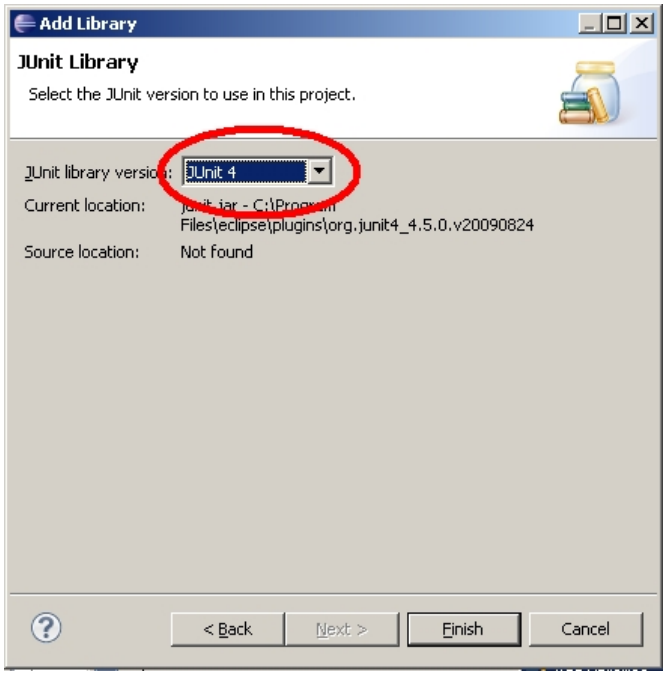
Figur 5.1, 5.2 och 5.3 visar hur man lägger till JUnit-biblioteken till ett projekt i Eclipse. Börja med att högerklicka på projektet och välj ”Build Path” och ”Add Libraries...” Du får då en fråga om vilket bibliotek du vill lägga till. Den exakta listan kan se olika ut beroende på vilken version av Eclipse du kör, men ”JUnit” bör alltid finnas med. I den sista dialogrutan gäller det sen att vara noga att man väljer JUnit 4, och inte JUnit 3. Gör man fel så plockar man helt enkelt bort biblioteket från projektet och börjar om.



Figur 5.1: Lägga till JUnit-biblioteket, steg 1



Figur 5.2: Lägga till JUnit-biblioteket, steg 2



Figur 5.3: Lägga till JUnit-biblioteket, steg 3

5.3 Att köra testfallen

JUnit-testfallen består av vanlig Java-kod, och kan köras på samma sätt som ni kör ett vanligt Java-program. Till varje uppgift som kräver det kommer det att finnas en länk till en java-fil med alla testfall. För att kunna köra testfallen sparar ni ner, eller kopierar in, koden som en Java-klass i ert projekt. Därefter kör man den precis som vanligt, fast som "JUnit Test" istället för som "Java Application". (Se figur 5.4.)

Resultatet av att köra en samling JUnit-test visas som en lista över alla testfallen och om de gick bra eller inte. Figur 5.5 visar ett exempel där klassen `DateTester` innehåller sex stycken testfall. Två av dessa har gått igenom korrekt och är markerade med en grön bock. De övriga fyra har ett litet blått kryss som signalerar att något är fel. Klickar man på ett misslyckat testfall så får man se mer information om vad som gick fel. I figur 5.5 har vi klickat på testfallet `fraction2` och får då reda på att det förväntade resultatet var 1.4, men koden vi testar har gett svaret 1.0.

Om alla testfall går igenom så visas listan ihopslagen, och den röda förloppsindikatorn[1] blir grön. Det är så det ska se ut innan ni skickar in er kod. Observera dock att bara för att alla testfall går igenom så är ni inte automatiskt godkända. Testen är till för att undvika uppenbara fel, inte för att utgöra någon sorts facit.

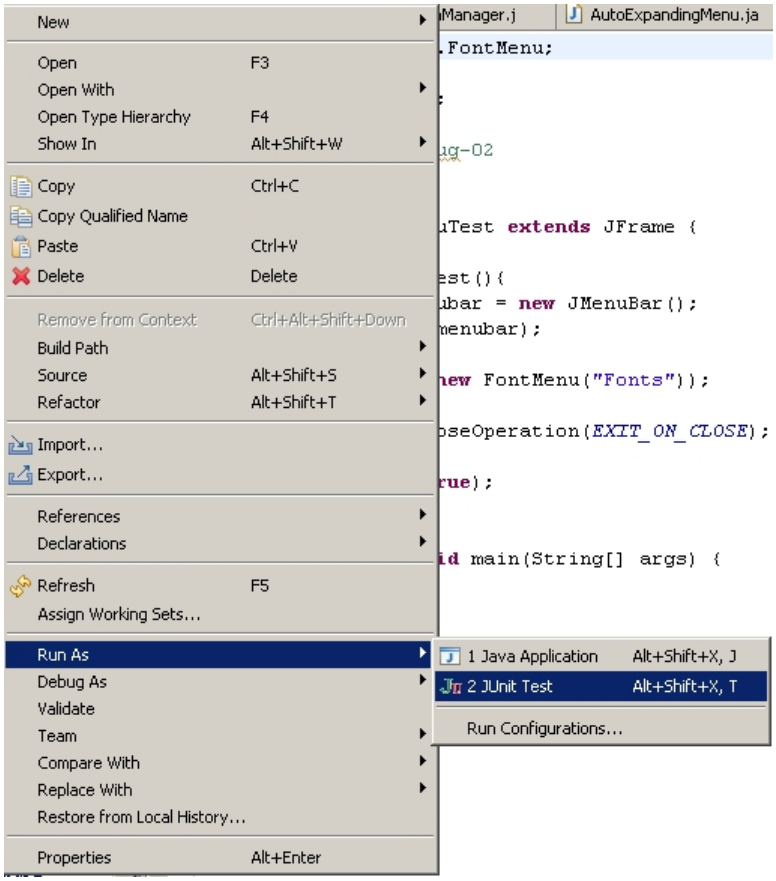
5.4 Egna testfall

De testfall som vi skickar med ska normalt sett aldrig ändras, och ni får absolut *inte* plocka bort några. Däremot är det helt okej att lägga till fler om ni vill det. I sin enklaste form så gör man det genom att skapa en metod som ser ut så här:

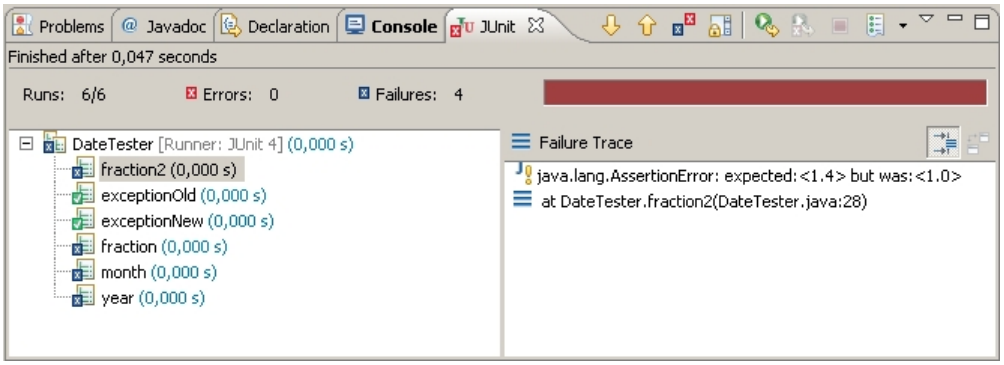
```
@Test
public void namnPåMetoden(){
   testkoden
}
```

De viktigaste kommandon man bör känna till är kanske `assertEquals` som kollar om två värden är lika, och `assertTrue` med sin motpol `assertFalse` som tittar om ett boolskt uttryck är sant respektive falskt.

[1] Jo, det verkar heta så på svenska.



Figur 5.4: Att köra JUnit-testfall i Eclipse



Figur 5.5: Resultatet av att köra några JUnit-testfall

6 LaTeX

LaTeX är ett typsättningssystem skapat av Leslie Lamport och bygger på TEX, som i sin tur skapades av Donald Knuth för att typsätta bokserien *The Art of Computer Programming* [5, 6, 7]. Knuths böcker är några av de vanligaste referenserna i vår kursbok, och är antagligen de mest kända algoritmböckerna som finns.

Nytt för i år är att det är ett krav att göra sina veckouppgifter i LaTeX. I början kan det kännas lite knöligt, men så fort man har kommit över den inledande tröskeln så blir det väldigt mycket enklare att göra dokument av den typ som ni kommer att göra på den här kursen.

Detta avsnitt ger en *kortfattad* introduktion till LaTeX; bara precis tillräckligt för att man ska kunna komma igång. Ett bra ställe att få mer information är `http://dsv.su.se/student/it/program/latex`. Här hittar du länkar till olika distributioner och mycket mer utförliga genomgångar än den här. Tveka inte heller att komma med eventuella problem till handledningen. Alla vi som jobbar på kursen använder det.

6.1 Hur gör man

Att skriva ett dokument i LaTeX påminner om att skriva ett i HTML. Man skriver sin text i en vanlig texteditor och sätter in markörer för olika typer av text. Till skillnad från HTML, som ju tolkas av webbläsaren, så måste man sen kompilera sin LaTeX-kod och får då ett dokument med layout.

6.2 Kompilering

Kompilering i LaTeX är en enkel procedur. Man skriver `latex filnamn.tex` och hoppas man slipper kompileringsfel. Kom bara ihåg att det är frågan om kompilering, och LaTeX-kompilatorn är lika kinkig som vilken annan kompilator som helst.

Det man får ut av LaTeX-kompilatorn är en dvi-fil som man kan titta på i en dvi-läsare vilket normalt följer med LaTeX-distributionen. Oftast vill man dock göra om den till PDF eller postscript. Det enklaste sättet att göra en PDF-fil är att använda en annan kompilator istället: pdflatex. Den fungerar precis som den vanliga kompilatorn[1], så för att kompilera skriver man `pdflatex filnamn.tex` och får en PDF-fil direkt.

Om ett PDF-dokument ser suddigt ut på skärmen, kan det hjälpa att plocka bort paketet `fontenc`. Avstavningen blir lite sämre dock.

Ytterligare en sak som kan vara bra att känna till är att LaTeX-kompilatorn bara läser igenom filen en gång. Det betyder att om man har referenser i texten som pekar framåt, till exempel en innehållsförteckning, så kan den inte skapas

[1] Nu ljuger vi, det finns en del skillnader. En man antagligen kommer att märka rör bilder. Mer om detta på sidan 23.

under en kompilering. Istället måste man köra LaTeX-kompilatorn två gånger. En gång för att generera en index-fil och en gång för att informationen i index-filen ska komma in i dokumentet.

6.3 Programvara

Ett tips är att skaffa sig en bra editor integrerad med en kompilator så man slipper skriva allt för hand. Länkar till ett par finns på DSVs LaTeX-sida.

För Windows är MikTex den mest populära LaTeX-distributionen. Den kan hämtas från `http://www.miktex.org/`. Om du inte vill kompilera för hand rekommenderas en bra miljö också. Det finns gott om alternativ. En helt okej är TeXnicCenter `http://www.texniccenter.org/`.

6.4 Ett minimalt exempel

Den grundläggande LaTeX-koden är väldigt enkel. Man skriver texten precis som vanligt, med några enkla kommandon insprängda:

`Här kommer \emph{lite} text.`
`Och lite till och lite till.`
 	
`Här kommer ett nytt stycke,`
`observera den tomma raden ovan.`
`Hur det ser ut beror på`
`stilmallen. I en tabell som den`
`här syns det knappt alls. På`
`andra ställen kan det markeras`
`med indrag eller med extra`
`utrymme. Allt beroende på vad`
`stilmallen säger.`
	Här kommer *lite* text. Och lite till och lite till.
	Här kommer ett nytt stycke, observera den tomma raden ovan. Hur det ser ut beror på stilmallen. I en tabell som den här syns det knappt alls. På andra ställen kan det markeras med indrag eller med extra utrymme. Allt beroende på vad stilmallen säger.

Innan man kan börja skriva sin text måste man dock först sätta upp ett dokument för den, ungefär som man måste ha en klass och en main-metod för att kunna skriva ett program i Java. Figur 6.1 visar ett minimalt sådant dokument. Figur 6.2 på sidan 21 visar resultatet efter kompilering. Den egna texten är i det här fallet de två raderna:

```
\section{Introduktion}          % En rubrik
Detta är ett litet exempel.     % Vanlig text
```

som kommer att ge en rubrik med texten *Introduktion* och under det början på ett stycke med texten *Detta är ett litet exempel.* Procenttecknen är kommentarer motsvarande Javas `//`, dvs allt efter dem på en rad plockas bort av kompilatorn.

```
\documentclass[a4paper,11pt]{article} % bestämmer vilken
                                      % stilmall som ska användas

\usepackage[swedish]{babel}    % Dessa tre rader ger svensk
\usepackage[T1]{fontenc}       % avstavning etc
\usepackage[latin1]{inputenc}

\title{Ett pytteexempel}       % Dokumentets titel
\author{Henrik Bergström}      % och författare

\begin{document}               % Här börjar själva dokumentet
\maketitle                     % Formaterar titel på dokumentet
                               % enligt stilmallen

\section{Introduktion}         % En rubrik
Detta är ett litet exempel.    % Vanlig text

\end{document}                 % Här avslutas dokumentet
```

Figur 6.1: Ett minimalt latex-dokument

6.5 Specialtecken

Obs viktigt!

Som nämndes i föregående stycke så är procenttecknet ett specialtecken i LaTeX och ger en kommentar. Vill man ha in ett procenttecken i texten skriver man `\%`, alltså på samma sätt som om man vill få in specialtecken i strängar i Java. Det finns flera såna tecken i LaTeX och de vanligaste är `&` och `\`. `&` fungerar precis som `%`, man skriver `\&`. `\` däremot är lite annorlunda. Det används för att indikera början på ett kommando, och vill man ha ut ett sådant får man problem. `\backslash` finns som kommando, men det fungerar bara i matteläget, så det får man skriva `$\backslash$`. `\\` är något helt annat som vi ska återkomma till i beskrivningen av vanliga fel.

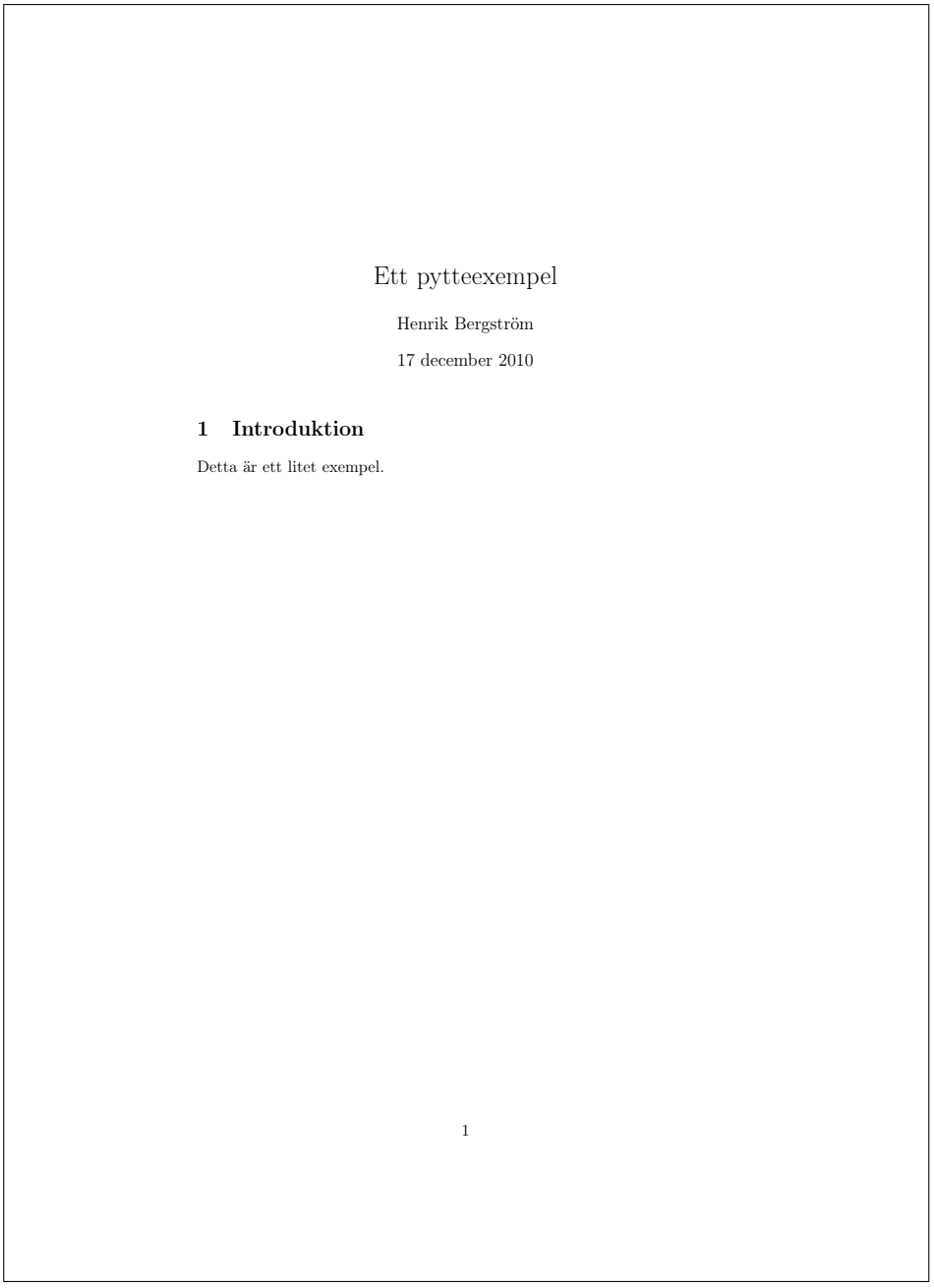
6.6 Vanliga kommandon

Det finns tusentals kommandon i LaTeX, men i praktiken är det bara ett par stycken man måste kunna.

Nytt stycke: Ett nytt stycke fås genom att skriva en tom rad mellan två stycken.

Rubriker: `\chapter{...}`, `\section{...}` och `\subsection{...}` ger olika typer av rubriker. Vilka rubriktyper som finns beror på vilken typ av dokument det är. I *article* finns till exempel inte `\chapter`.

Markerad text: Vill man markera text så används med fördel `\emph{...}` som står för emphasis. De flesta stilmallar gör detta till kursiv text, men det är inte säkert. Det skulle lika gärna kunna vara fetstil eller understruken text. Allt beroende på vad som passar mallen.



Figur 6.2: Dokumentet som koden i figur 6.1 genererar

Avstavning: Avstavning för vanliga ord fungerar helt automatiskt, men vill man lägga in en manuell avstavning så skriver man `\-` där man vill ha avstavningspunkterna så här: `av\-stavnings\-punkterna`.

6.7 Det vanligaste felet

Det absolut vanligaste felet man som nybörjare gör är att man slåss med systemet. Är man van vid en ordbehandlare så vill man gärna ha mer kontroll och göra saker på sitt eget sätt. *Fall inte för frestelsen!* När du har blivit varm i kläderna kan du börja ändra på saker, men mer troligt är att du då insett att systemet oftast gör rätt. Visst finns det tillfällen när det blir fel, självklart, men det allra mesta görs väldigt bra.

Ett typiskt exempel är nya stycken. Det finns (åtminstone) tre olika sätt att avsluta en rad i LaTeX. Förutom det vi redan har lärt oss: en blank rad mellan stycken så kan kan man avsluta den med två bakvända snedstreck: `\\`, eller med ett `\newline`. *Använd inte dessa!* För vissa uppgifter är de korrekta, men *aldrig* för att göra ett nytt stycke. De är till för om man vill bryta en rad men inte börja ett nytt stycke.

Många får för sig att använda manuella radbrytningar därför att de inte gillar indraget i början på stycken utan istället vill ha ett avstånd mellan styckena. Det korrekta sättet att få det utseendet är inte att klottra ner texten med manuella radbrytningar, utan att byta till en stilmall som ser ut så. En enda ändrad rad stället för hundratals.

Bilder och tabeller är ett annat område där folk gärna slåss mot systemet, liksom marginalerna. I samtliga fall gäller det att systemet har ”rätt” betydligt oftare än dem som slåss mot det, och att om man absolut vill ändra på beteendet så finns det normalt ett enklare sätt att göra det.

6.8 Paket

Den riktiga styrkan med LaTeX är alla paket man kan använda sig av. Det finns bokstavligen tusentals som gör nästan allt man kan tänka sig.

För att kunna använda sig av ett paket så måste det importeras med kommandot: `\usepackage{...}`. Detta kommando måste stå i den så kallade ”preamble”, alltså den del av dokumentet som står innan `\begin{document}`. I exemplet i figur 6.1 på sidan 20 så importerade vi tre paket: `bable`, `fontenc` och `inputenc`.

I inlämningsmallen är några vanliga paket redan importerade, bland annat `graphicx` för grafik, `url` för att typsätta url:er och `listings` som kan typsätta programkod. Exempel på hur man använder de två sistnämnda finns i mallen.

6.9 Fler kommandon

Resten av detta avsnitt är en ”kokbok” med exempel på olika konstruktioner som är bra att känna till. De visas i sin vanligaste form med korta noter om vad man kan göra mer.

Listor

Listor är väldigt enkla. `\begin{typ av lista}` och `\end{typ av lista}` börjar och slutar listan. Varje punkt är sedan ett `\item`. Undernivåer skapas genom att man lägger in en ny lista i listan.

```
\begin{itemize}
  \item Första punkten
  \item Andra pukten
  \item Tredje punkten
\end{itemize}
```

- Första punkten
- Andra pukten
- Tredje punkten

```
\begin{enumerate}
  \item Första punkten
  \item Andra pukten
  \item Tredje punkten
\end{enumerate}
```

1. Första punkten
2. Andra pukten
3. Tredje punkten

Bilder

Bilder och figurer är ett av de ämnen som nybörjare brukar svära mest över. Problemen brukar vara två: bildformat och placering.

Om vi börjar med bildformaten så är det så att den vanliga LaTeX-kompilatorn *bara* accepterar bilder i eps-format. Pdflatex å andra sidan accepterar jpeg, png och pdf, men *inte* eps.

Så länge man bara använder en av kompilatorerna så är det inget större problem, men vissa gillar att växla. Vill man kunna göra det så kan man göra två versioner av sin bild, en i eps och en i något av formaten som pdflatex accepterar. När man sen lägger in bilden så låter man bli att specificera filändelsen. Kompilatorn letar då reda på rätt version av bilden. I exemplet i figur 6.3 nedan så ligger till exempel de bägge filerna *image.eps* och *image.png* i katalogen *img*.

För att kunna lägga in en bild måste man importera paketet `graphicx`. Kommandot för att lägga in bilden är sedan `\includegraphics{sökväg}`. Om man vill kan man också skicka med parametrar som skalar om eller roterar bilden: `\includegraphics[parametrar]{sökväg}`. Vanliga parametrar är `width`, `height` och `angle`. De två förstnämnda tar en storlek som kan uttryckas i ett antal olika enheter. Lättast är oftast centimeter, till exempel `width=15cm`. `angle` slutligen tar ett gradtal och vänder bilden så många grader *moturs*.

Det andra problemet vi nämnde att många har rör placeringen. Om man bara använder `includegraphics` så betraktas bilden mer eller mindre som ett tec-

ken. Ett stort och konstigt tecken, men i övrigt inte så annorlunda. Det här ställer till en del problem, bland annat med hur mycket utrymme som det ska finnas runt bilden och med bildtexter. Det ”korrekta” sättet att lägga in bilder i ett LaTeX-dokument är därför något mer komplicerat. Figur 6.3 ger ett fullständigt exempel.

```
\begin{figure}[htbp]
  \centering
  \includegraphics[width=3cm, angle=180]{img/image}
  \caption{Bildtext}
  \label{bildidentifierare}
\end{figure}
```

Figur 6.3: Kod för att inkludera en bild

Koden i figur 6.3 ger bilden i figur 6.4. Som du kan se så är kartan 3 cm bred och vänd 180 grader. Att den ser lite sned ut beror på orginalbilden. Kartbilden var lätt lutad.

Figur 6.4: Bildtext

Ovanstående kod kan se avskräckande ut, men är egentligen ganska enkelt. `\centering` placerar bilden centrerat på sidan, `\caption{...}` sätter bildtexten. Att bildtexten hamnar under bilden beror på att `\caption{...}` står efter `\includegraphics{...}`. Ville vi istället ha bildtexten ovanför bilden hade vi bytt ordningen mellan dem.

`\label{...}` återkommer vi till i nästa avsnitt. Det används för att lägga in korsreferenser. Det enda som är viktigt här är att det står *efter* `\caption{...}`.

Det vi har kvar nu är de två raderna i början och slutet: `\begin{figure}[htbp]` och `\end{figure}`. Dessa skapar en så kallad *float*. En del av dokumentet som kan placera sig själv på en lämplig plats. Det är nu problemen brukar uppkomma. Vad floaten tycker är lämpligt och vad den som skriver dokumentet tycker är inte alltid riktigt samma sak. Floaten försöker ta hänsyn till en massa typografiska regler vilket ibland kan vara lätt frustrerande. Vet man hur den är tänkt att fungera så brukar det dock inte vara något olösligt problem.

Det första man ska veta är att man *inte* kan säga åt en float exakt var den ska ligga. Det spelar ingen roll hur gärna man än vill, det går inte. Istället får man ge den önskemål om hur man vill att den ska placera sig. Det är det som `[htbp]` efter `\begin{figure}` gör. Varje bokstav representerar ett önskemål

om placering, och de testas i tur och ordning efter hur de står. Går ett önskemål att uppfyllas så är det där figuren hamnar, annars går man vidare till nästa. De olika alternativen betyder:

h står för ”here”, alltså på det ställe där koden står. Detta fungerar inte om det till exempel inte finns tillräckligt med utrymme kvar på sidan.

t ”top of page”, högst upp på en sida, inte nödvändigtvis den du står på just nu. Flera figurer med detta önskemål kan placeras under varandra. I skrivandets stund är detta fallet med figur 6.3 och 6.4 som bägge ligger efter varandra högst upp på föregående sida. När du läser dokumentet kan detta ha ändrats eftersom andra ändringar i dokumentat kan ha gjort en annan placering önskvärd.

b ”bottom of page”, längst ner på en sida, inte nödvändigtvis den du står på just nu.

p ”page of floats”, en separat sida med bara figurer och andra floats.

En figur-float behöver inte innehålla något `\includegraphics` om man inte vill. Dess uppgift är att placera vad som helst som om det vore en figur. Koden i figur 6.1 på sidan 20 är ett `verbatim`-block (se avsnittet om programkod nedan) placerat inom `\begin{figure}[htbp]` och `\end{figure}`.

Hänvisningar och korsreferenser

I avsnittet om bilder och figurer ovan så fanns det en rad som vi lovade återkomma till: `\label{bildidentifierare}`. Detta kommando sätter ut en osynlig markör i texten som kan användas för att referera tillbaka till den aktuella positionen.

Vill man referera till en viss bild eller rubrik skriver man `\ref{id}`. Till exempel så ger texten `figur \ref{bildidentifierare}` följande resultat: figur 6.4. Om vi nu sätter in en figur till innan figur 6.4 i texten så kommer numreringen automatiskt att ändras. Siffran som ges beror på var man skriver labeln. Skrivs den i vanlig text så skulle den referera till numret på den senaste rubriken.

Vill man istället ha sidnummret så använder man `\pageref{id}` istället. Texten `sidan \pageref{bildidentifierare}` ger i det här dokumentet resultatet: sidan 24.

Obs viktigt!

När man använder korsreferenser i sitt dokument måste man normalt komma ihåg att kompilera det två gånger. Annars kommer referenser till labels som ligger senare i texten inte att komma med.

Programkod

Det enklaste sättet att inludera programkod är ett verbatim-block:

```
\begin{verbatim}
          Här skriver du
texten
  precis som du vill ha
  den...

Specialtecken funkar också:
!"'#%&/\½§
\end{verbatim}
```

```
          Här skriver du
texten
  precis som du vill ha
  den...

Specialtecken funkar också:
!"'#%&/\½§
```

Texten i ett verbatim-block sätts med en font med fast bredd och alla specialtecken kommer ut precis som de ska. Utseendet blir dock inte speciellt roligt. Ett bättre alternativ för programkod är paketet listings. Lägg till följande rader i din preamble:

```
\usepackage{listings}
\lstset{language=Java, tabsize=3, numbers=left, frame=line}
```

och ersätt Java med namnet på det språk du använder dig av. Nu kan du använda dig av ett lstlistings-block istället[2]:

```
\begin{lstlisting}
class Exempel{
        private int i = 25;

        int method(int p){
                return i*p;
        }
}
\end{lstlisting}
```

```
class Exempel{
    private int i = 25;

    int method(int p){
        return i*p;
    }
}
```

Ett annat alternativ är att läsa in koden direkt från en fil:

```
\lstinputlisting{sökväg/filnamn.java}
```

Om man inte vill ha med hela filen går det att plocka ut en bit:

```
\lstinputlisting[firstline=5, lastline=9]{sökväg/filnamn.java}
```

[2]Radnumren är borplockade här på grund av problem med ramen runt koden.

Tabeller

Tabeller består precis som figurer av två delar: själva tabellen och en float. Floaten fungerar precis som för figurer förutom att det står `\begin{table}` och `\end{table}` i början och slutet och att rubriken kommer att inledas med *Tabell* istället för *Figur*. För mer information om placering, rubriker, etc. se avsnittet om bilder.

För att skapa själva tabellen använder man ett `tabular`-block:

```
\begin{tabular}{kolumninformation}
  ...
\end{tabular}
```

Det viktigaste här är kolumninformationen. Den består av ett antal bokstäver, en för varje kolumn som talar om hur kolumnen ska vara formaterad:

l vänsterjusterad. Bredden sätts av den längsta texten på någon rad.

c centrerad. Bredden sätts av den längsta texten på någon rad.

r högerjusterad. Bredden sätts av den längsta texten på någon rad.

p{bredd} vänsterjusterad med fast bredd och radbrytning. Bredden kan anges i flera olika enheter, till exempel så här: `p{5cm}`.

Ett vertikalt streck (`|`)mellan två bokstäver i kolumninformationen betyder att det ska ritas ett vertikalt streck mellan dem i tabellen.

Själva tabelldata skrivs in i tabular-blocket med & mellan kolumnerna och \\ efter varje rad. Vill man ha en horisontell linje mellan rader använder man `\hline`.

Tabell 6.1 till 6.4 på sidan 30 ger några exempel på hur tabeller kan byggas upp. Det första exemplet i tabell 6.1 är en vanlig standardtabell med inramade celler; en av varje typ. Koden för denna syns nedan:

```
\begin{table}[p]
  \centering
    \begin{tabular}{|l|c|r|p{5cm}|}
      \hline
      AAA & BBB & CCC & DDD \\
      \hline
      A   & B   & C   & D   \\
      \hline
    \end{tabular}
  \caption{En standardtabell med enkel ram}
  \label{tabell:medram}
\end{table}
```

Notera att vi i floaten bara angett *p* som placeringsönskemål. Detta har vi gjort i alla tabellexemplen därför att vi vill placera allihopa på en i övrigt tom sida. Ville vi istället ha dem längst ner på den här sidan, om möjligt, så hade vi angett *b* innan *p*.

Eftersom man explicit måste markera var man vill ha ramar så är det enkelt att plocka bort dem. Detta syns i tabell 6.2 som genereras av denna kod:

```
\begin{table}[p]
  \centering
    \begin{tabular}{lcrp{5cm}}
      AAA & BBB & CCC & DDD \\
      A   & B   & C   & D   \\
    \end{tabular}
  \caption{Samma tabell utan ram}
  \label{tabell:utanram}
  Observera att tabellen fortfarande är centrerad på sidan.
  Den fjärde kolumnen är lika bred som i tabell \ref{tabell:medram},
  det är ramen som inte syns.
\end{table}
```

Här har vi plockat bort de vertikala strecken mellan kolumndefinitionerna och alla `\hline`. Om du tittar på tabellen så ser den lite konstig ut. Den borde väl vara längre till höger för att vara centrerad? Nej, det borde den inte. Den sista kolumnen är ju fem cm bred, och vi utnyttjar bara ungefär en cm av dessa för texten. Resten av kolumnen finns dock där och tar upp plats vilket man ser om man jämför med figuren ovanför.

Vill vi markera vissa delar av ramen så finns det inget som hindrar oss från att sätta in flera vertikala eller horisontella streck mellan kolumner eller rader. Ett exempel på detta syns i tabell 6.3 som genereras av denna kod:

```
\begin{table}[p]
  \centering
    \begin{tabular}{|l||l||l|}
      \hline
      AAA & BBB & CCC  \\
      \hline
      \hline
      AAA & BBB & CCC  \\
      \hline
      \hline
      AAA & BBB & CCC  \\
      \hline
    \end{tabular}
  \caption{En tabell där vi lekt med ramen}
  \label{tabell:varieranderam}
\end{table}
```

Till sist kan det vara bra att se ett exempel på skillnaden mellan en *p*-kolumn och en *l*-kolumn. Den förstnämnda har en fast bredd och radbryts därför. Den

sistnämnda anpassas efter bredden på den längsta raden, oavsett vad detta betyder. Detta illustreras av tabell 6.4 som ges av följande kod:

```
\begin{table}[p]
  \centering
    \begin{tabular}{|p{5cm}|l|}
      \hline
      En lång text som radbryts när den blir mer än fem cm bred. &
      En lång text som inte radbryts eftersom kolumnen inte har fast bredd \\
      \hline
    \end{tabular}
  \caption{Skillnaden mellan en p-kolumn och en l-kolumn}
  \label{tabell:lp}
\end{table}
```

I koden kan vi också se att man inte behöver strukturerar tabellkoden efter hur den ska se ut på skärmen. De två långa raderna med text som i koden ligger under varandra visas brevid varandra. Det är `&` och `\\` som bestämmer var kolumner respektive rader slutar, inget annat.

Det finns väldigt mycket mer att säga om tabeller i LaTeX. Vi har inte berört hur man får saker och ting att sträcka sig över flera rader och kolumner, hur man får tabeller som sträcker sig över flera sidor, etc. etc. För den som vill veta mer rekomenderas en titt på `http://en.wikibooks.org/wiki/LaTeX/Tables`.

AAA	BBB	CCC	DDD
A	B	C	D

Tabell 6.1: En standardtabell med enkel ram

AAA	BBB	CCC	DDD
A	B	C	D

Tabell 6.2: Samma tabell utan ram

Observera att tabellen fortfarande är centrerad på sidan. Den fjärde kolumnen är lika bred som i tabell 6.1, det är ramen som inte syns.

AAA	BBB	CCC
AAA	BBB	CCC
AAA	BBB	CCC

Tabell 6.3: En tabell där vi lekt med ramen

En lång text som radbryts när den blir mer än fem cm bred.	En lång text som inte radbryts eftersom kolumnen inte har fast bredd

Tabell 6.4: Skillnaden mellan en p-kolumn och en l-kolumn

Formler

Ett område där LaTeX är väldigt bra är matematiska formler. Korta formler som ska stå direkt i texten skrivs mellan dollartecken: `$...$`. Som ett exempel ger `$\sqrt{\frac{a^{57}}{3-b}}$` följande formel: $\sqrt{\frac{a^{57}}{3-b}}$. `\sqrt{...}` tar roten ur argumentet, `\frac{...}{...}` ger ett horisontellt dividsionstecken och `a^{57}` ger a upphöjt till 57.

Om vi ville hade vi kunnat använda ett vanligt divisionstecken istället för `\frac`: `$\sqrt{a^{57}/3-b}$` och då fått detta utseende: $\sqrt{a^{57}/3-b}$.

Formler som ska numreras kan skrivas inom ett equation-block enligt nedan. Här ser vi också exempel på många av de funktioner som ni kan ha intresse av att skriva under kursen.

Exemplen insprängda i texten ovan kan se lite avskräckande ut eftersom det är många nivåer. Här ser vi ett betydligt enklare, renare, och mer vanlig formel:

```
\begin{equation}
a = b + c^3
\end{equation}
```

$$a = b + c^3 \tag{6.1}$$

Specialtecken finns det gott om. Här ser vi två grekiska bokstäver, α och π, samt en prick istället för en stjärna för multiplikation.

```
\begin{equation}
\alpha = \pi \cdot r^2
\end{equation}
```

$$\alpha = \pi \cdot r^2 \tag{6.2}$$

De vanligaste matematiska funktionerna har fördefinerade kommandon, till exempel logaritmen:

```
\begin{equation}
O(n \log n)
\end{equation}
```

$$O(n \log n) \tag{6.3}$$

Här ser vi ett lite renare exempel på skillnaden mellan `\frac{}{}` och vanlig division:

```
\begin{equation}
\frac{1}{2} + 1/2 = 1
\end{equation}
```

$$\frac{1}{2} + 1/2 = 1 \tag{6.4}$$

Hur man skriver a^i har vi ju redan sett (`a^i`), men ibland vill man istället ha ett index. Då används ett understreck :

```
\begin{equation}
a_i
\end{equation}
```

$$a_i \tag{6.5}$$

Summatecken finns naturligtvis också. Observera skillnaden i skrivsätt mellan det som står under summatecknet (`_{i=1}`) och det som står över (`^n`). Måsvingar kan användas på bägge, men är bara nödvändiga om det är mer än ett tecken som ska tas med.

```
\begin{equation}
\sum_{i=1}^n(a_i)
\end{equation}
```

$$\sum_{i=1}^{n}(a_i) \tag{6.6}$$

Skulle vi strunta i måsvingarna runt `_{i=1}` skulle vi få något som ser väldigt konstigt ut:

```
\begin{equation}
\sum_i=1^n(a_i)
\end{equation}
```

$$\sum_{i} = 1^n(a_i) \tag{6.7}$$

De sista två exemplen visar ett problem med paranteser. Vanliga paranteser fungerar utmärkt så länge man inte använder något som tar mer utrymme i höjdled än ett vanligt tecken.

```
\begin{equation}
(1+(1/2)) = (1+(\frac{1}{2}))
\end{equation}
```

$$(1+(1/2)) = (1+(\frac{1}{2})) \tag{6.8}$$

Vill man istället ha paranteser som automatiskt ändrar storlek kan man använda `\left(` och `\right)` som vi gör här:

```
\begin{equation}
\left( 1 + \left( 1/2
\right) \right)
=
\left( 1 + \left( \frac{1}{2}
\right)\right)
\end{equation}
```

$$\left(1+\left(1/2\right)\right) = \left(1+\left(\frac{1}{2}\right)\right) \tag{6.9}$$

Del II

Veckouppgifter

7 Veckouppgifter

Här hittar du de veckouppgifter som gäller i år. Målsättningen är att uppgifterna för en vecka ska ta ungefär en dag heltid att göra, alltså någonstans runt fem till tio timmar. Detta är dock inget som går att säga säkert eftersom det beror på så många olika faktorer: förkunskaper, gruppstorlek, etc. Dessutom är det naturligtvis så att vissa veckor är lättare än andra.

Tema 2: algoritmanalys har ingen egen vecka utan måste göras på muntan. Flera av veckorna har dock uppgifter som anknyter till detta tema och som övar.

Veckouppgifterna kan uppdateras fram till och med introduktionsföreläsningen motsvarande vecka. Därefter kan de förtydligas, men inte egentligen ändras. *Viktigt!*

7.1 Krav för godkänt

För att bli godkänd på veckouppgifterna så måste man ha:

- lämnat in individuellt innan deadline
- bli godkänd på alla deluppgifterna
- göra en tillräckligt bra granskning

7.2 Förslag på pooluppgift

Den sista uppgiften varje vecka är alltid densamma. Kom med ett förslag på en lämplig muntauppgift för veckans tema. Självklart får ni komma med flera förslag om ni vill. Så länge minst en av dem är tillräckligt bra så är ni godkända på uppgiften. För att en uppgift ska bli godkänd måste den:

- anknyta till veckans tema
- vara av lämplig svårighetsgrad för kursen
- vara möjlig att sätta graderade betyg på

Rena faktafrågor är acceptabla, men det är ofta mer intressant att försöka använda datastrukturer och algoritmer för att lösa problem.

7.3 Vecka 3: linjära datastrukturer

Denna vecka är programmeringsmässigt vikt åt repetition. Det allra mesta som tas upp i uppgifterna har ni sett i tidigare kurser.

Lazy deletion

Uppgift 3.20 från boken.

I Moodle hittar du en implementation av en länkad lista. Implementationen liknar, men är inte identisk med den i boken. Skriv om klassen så att den använder *lazy deletion* istället. Alla metoder i klassen ska fortfarande fungera.

Två stackar i en array

Uppgift 3.24 från boken.

Designa och implementera en klass som representerar två stycken stackar med hjälp av en enda array.

Algoritmanalys

Uppgift 2.7 från boken.

I figur 7.1 på sidan 37 hittar du sex stycken kodsnuttar. Uppgiften är att göra en ordoanalys (Big-Oh) för dessa. Dessutom ska du implementera dem och mäta tiden för olika värden på N, samt jämföra resultatet med din resultatet av din analys. Det är naturligtvis okej att kopiera koden från PDF-filen.

Ett enkelt sätt att mäta tiden är med `System.nanoTime`. Tänk dock på att körtiden kan variera kraftigt från gång till gång, så man bör genomföra varje experiment flera gånger och använda den lägsta tiden.

Förslag på pooluppgift

Denna vecka finns det en restriktion på vad förslaget på fråga måste handla om. Föreslår ni flera frågor räcker det med att en av dem uppfyller villkoret, men det ska i så fall tydligt markeras, och frågan måste vara seriös. Den får inte bara vara påhittad för att uppfylla detta krav.

Restriktionen kan verka lite löjlig, men finns till för att vi vill ha spridning på uppgiftsförslagen. För att få det måste vi ha en möjlighet att slumpa vilka restriktioner som gäller för vilken grupp. Vi skulle kunna ge varje varje grupp information om detta individuellt, men det skulle vara väldigt svårt att administrera, speciellt som grupper kan ombildas när som helst. Istället använder vi oss av en enkel formel baserad på första bokstaven i en av gruppmedlemmarnas användarnamn. Den gruppmedlem det rör sig om är denna vecka den som står först på framsidan, alltså den som kommer först när gruppmedlemmarna är sorterade på efternamn.

Så, sortera författarlistan enligt instruktionerna i mallen; tag sen första bokstaven i försteförfattarens användarnamn på DSV och för in det i tabell 7.1 på sidan 38 för att få reda på vilken av fölajnde restriktioner som gäller just er.

1. Frågan måste behandla listor i någon form.
2. Frågan måste behandla stackar.
3. Frågan måste behandla köer.
4. Frågan måste behandla länkade listor. Observerera att detta inte betyder att frågan måste handla om datatypen lista. Den kan lika gärna handla om implementation av någon av de andra linjära datatyperna som behandlats.

```
// Exempel 1
sum = 0;
for (i = 0; i < n; i++)
    sum++;

// Exempel 2
sum = 0;
for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
        sum++;

// Exempel 3
sum = 0;
for (i = 0; i < n; i++)
    for (j = 0; j < n * n; j++)
        sum++;

// Exempel 4
sum = 0;
for (i = 0; i < n; i++)
    for (j = 0; j < i; j++)
        sum++;

// Exempel 5
sum = 0;
for (i = 0; i < n; i++)
    for (j = 0; j < i * i; j++)
        for (k = 0; k < j; k++)
            sum++;

// Exempel 6
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++;
```

Figur 7.1: Kod för algoritmanalysuppgiften vecka 3

Första bokstav						Grp
A	F	K	P	U	Z	1
B	G	L	Q	V		2
C	H	M	R	W		3
D	I	N	S	X		4
E	J	O	T	Y		5

Tabell 7.1: Första bokstaven i en av gruppmedlemmarnas användarnamnnet används för att bestämma vilken av restriktionerna som gäller för er. Vilken gruppmedlem det är varierar från vecka till vecka.

5. Frågan måste handla om att använda någon av de linjära datastrukturerna.

I samtliga fall som börjar med *"Frågan måste behandla"* gäller att man får ta upp andra saker också. Om man till exempel tillhör restriktionsgrupp 1 så är det helt okej att föreslå en fråga som tar upp både lista, stack och kö.

7.4 Vecka 4: träd

De första uppgifterna denna vecka utgår från bokens implementation av AVL-träd. Denna finns att ladda hem från `http://users.cis.fiu.edu/~weiss/dsaajava2/code/AvlTree.java`.

Alla dessa uppgifter går ut på att du ska lägga till ytterligare metoder till AVL-trädet. De flesta av dessa är enkla och mest till för att kunna kontrollera att den sista av dem fungerar.

Bokens implementation av AVL-träd kräver att `UnderflowException` finns tillgängligt. Enklast är att skapa det själv genom att skapa en klass som ärver från `RuntimeException`.

Hjälpmetoder

De första uppgifterna, markerade med stjärnor nedan, är inte ett absolut krav att göra. Det finns dock flera viktiga anledningar att göra dem: dels gör dem det möjligt att i någon mån testa er borttagsimplementatition, dels är de avsedda att göra er bekanta med koden, och dels avser de att öva på rekursion. Det sistnämnda är inte minst viktigt. Vi kommer att använda rekursion om och om igen. Samtliga metoder nedan går att implementera med rekursion, och det är en bra övning att också göra dem med rekursion.

De testfall som finns i Moodle förutsätter att metoderna finns och fungerar. Om detta inte är sant får ni kommentera bort alla referenser till dem i testfallen. Priset man betalar är naturligtvis att risken för att fel slinker igenom ökar dramatiskt.

Antal noder*

Designa och implementera en metod som returnerar hur många noder det finns i trädet. Metoden ska heta `size` och returnera en `int`.

Maximal höjd*

Designa och implementera en metod som returnerar den maximala höjden i trädet. Metoden ska heta `maxHeight` och returnera en `int`.

Kontrollera höjdinformationen*

Uppgift 4.22 från boken.

Designa och implementera en algoritm som kontrollerar att höjdinformationen i trädet är korrekt för alla noder. Metoden ska heta `hasCorrectHeightInfo` och returnera en `boolean`. Bokens uppgift kräver att algoritmen ska gå i linjär tid. Detta är inget absolut krav, men bör vara uppfyllt.

Med korrekt höjdinformation menas inte bara att höjden är korrekt, utan också att balanseringskravet för AVL-träd är uppfyllt.

Kontrollera att det är ett sökträd*

Uppgift 4.32 från boken.

Designa och implementera en algoritm som kontrollerar att varje nod i trädet uppfyller ordningskraven för ett binärt sökträd. Metoden ska heta `isSearchTree` och returnera en `boolean`. Bokens uppgift kräver att algoritmen ska gå i linjär tid. Detta är inget absolut krav, men bör vara uppfyllt.

Borttag ur AVL-träd

Detta är den huvudsakliga uppgiften på AVL-trädet.

Implementera borttag ur ett AVL-träd. Du får själv välja om du vill använda lazy deletion eller om du vill plocka bort elementen direkt.

Algoritmanalys

Analysera tidskomplexiteten hos följande metoder i klassen `SimpleLinkedList` från förra veckan:

- `boolean add(E element`
- `void add(int index, E element)`
- `boolean addAll(Collection<? extends E> c)`
- `boolean addAll(int index, Collection<? extends E> c)`
- `int indexOf(Object o)`
- `int lastIndexOf(Object o)`
- `boolean retainAll(Collection<?> c)`

Utgå från den ursprungliga implementationen i Moodle, inte från den omskrivna versionen du lämnade in förra veckan.

Förslag på pooluppgift

Denna vecka finns inga speciella restriktioner på pooluppgifterna. Splayträd och B-träd är dock perifera ämnen i den här kursen, så väljer ni att göra frågor på dem så bör ni begränsa er till den huvudsakliga idén och inte gå in på implementationsdetaljer.

7.5 Vecka 5: hashtabeller och prioritetsköer

Förklara hashtabeller och hashning

En generell färdighet som man som student förväntas öva på är förmågan att skriva. Eftersom vi tycker detta är en viktig färdighet vill vi passa på att öva på den, och den första uppgiften denna vecka är därför att skriva en artikel på en till två sidor[1] som förklarar hashatbeller och hashning.

Målgruppen för din text är människor som har läst en grundläggande kurs i programmering, men som i övrigt inte har någon bakgrund eller förkunskaper inom algoritmer och datastrukturer. Du måste därför förklara så pass utförligtatt hashning blir begripligt för någon ur målgruppen. Viktigt är också att exten är läsbar. Det är en artikel du ska skriva, inte en punktlista, essä, ...

Begrepp som nog borde förklaras är bland andra hashfunktion, separate chaining, linear probing, quadratic probing och double hashing. Du bör också ange användningsområden för hashfunktioner och när det kan vara lämpligt att använda de olika varianterna på kollisionshantering.

Eftersom detta är den första skrivuppgiften på kursen så vill vi påminna om kravet på att arbetet ska vara ert eget.

d-heap

Implementera en *arraybaserad d-heap* enligt interfacet `MiniHeap` som finns i Moodle. En d-heap är en heap där antalet barn inte är fast utan variabelt, och antalet barn ska i det här fallet anges som en variabel till konstruktorn för heapen.

Enklast är att utgå från bokens heap, men tänk på att det är en binär heap (varje element kan ha två barn) och att det därmed krävs vissa förändringar för att den ska fundera med ett variabelt antal barn.

Testfall för heapen finns i Moodle. Observera att testfallen utgår från att heapen börjar på index 1 istället för index 0. Vill du använda 0 som startposition behöver du visa att din uträkning för att hitta föräldrar och barn kommer att fungera oavsett storlek på d.

Förslag på pooluppgift

Eftersom denna vecka har två teman måste ni självklart också komma in med två frågor: en om hashtabeller eller hashalgoritmer, och en om prioritetsköer.

[1] Sidkravet är inget speciellt viktigt. Vill du skriva 20 sidor får du det, men du riskerar en sur granskning...

7.6 Vecka 6: sortering

Jämförelse av sorteringsalgoritmer

Uppgiften är att implementera och jämföra olika sorteringsalgoritmer, både gentemot varandra och gentemot deras teoretiska beteenden. I Moodle hittar du ett testprogram och ett antal klasser som representerar olika sorteringsalgoritmer. Några av dessa sorteringsalgoritmer är redan implementerade, och din första uppgift är att implementera resten. De flesta av algoritmerna är i det närmaste identiska med de som ges i boken, *men de jobbar på en lista, inte en array.* Detta betyder att du måste fundera på hur detta påverkar deras effektivitet i olika situationer.

Observera att dina implementationer ska använda sig av den lista som skickas in. Du får alltså inte kopiera över datat i en array för att göra någon algoritm snabbare. Det enda undantaget från detta krav är mergesortalgoritmen som behöver sin temporära array.

När du är klar med implementationen använder du testprogrammet, eller skriver ett eget, och jämför de olika algoritmernas beteenden under olika förutsättningar. Testprogrammet varierar ganska många olika aspekter förutom sorteringsalgoritmen: storlek på listan som ska sorteras, datatyp som ska sorteras, datastruktur som ska sorteras, om datat redan är sorterat eller inte, om dubletter förekommer eller inte. Du behöver inte ta hänsyn till alla dessa, men åtminstone några ska behandlas utförligt. Tänk också på att jämföra det du får fram med algoritmernas teoretiska beteenden.

Tänk också på att körtiden kan variera kraftigt från gång till gång, så man bör genomföra experiment flera gånger och använda den lägsta tiden.

Det du ska lämna in på denna uppgift är koden för de sorteringsalgortimer som inte är implementerade från början, och din analys av de olika sorteringsalgoritmernas beteenden under olika förutsättningar.

Förslag på pooluppgift

Denna vecka finns det två restriktioner på vilka pooluppgifter man får föreslå. Föreslår man flera räcker det med att en av dem uppfyller villkoret, men det ska i så fall tydligt markeras.

Den första restriktionen är att shellsort inte examineras. Om ni i en bisats råkar nämna att shellsort är $O(n \log n)$ kommer ni inte att klandras för det, men inga frågor om implementationsdetaljer.

Den andra restriktionen är för att få in variation i frågepoolen. Samma tabell som första veckan används, fast den här gången är det den gruppmedlem som står *sist* vars användarnamn bestämmer vilken restriktionsgrupp ni tillhör. *Obs!*

1. Frågan måste jämföra minst två sorteringsalgoritmer.
2. Frågan måste behandla hur minst ett av följande kriterier påverkar valet av sorteringsalgoritm: storlek på det som ska sorteras, datatyp som ska sorteras, datastruktur som ska sorteras, om datat redan är sorterat eller inte, om dubletter förekommer eller inte.

3. Frågan måste behandla *mergesort*.
4. Frågan måste behandla *quicksort*.
5. Frågan måste behandla enkla sorteringsalgoritmer som *selection* eller *insertion sort*.

7.7 Vecka 7: grafer

Multigrafer

En multigraf är en graf där flera bågar är tillåtna mellan ett nodpar. Vilka av de nedanstående algoritmerna fungerar utan modifikationer för multigrafer? Vilka förändringar måste man göra för att de övriga ska fungera för multigrafer?

Motivera dina svar så att det framgår vad i algoritmen som gör att en den kan eller inte kan hantera multigrafer. Algoritmer

- Prims algoritm
- Kruskals algoritm
- Dijkstras algoritm
- Djupet först-sökning
- Bredden först-sökning
- Topologisk sortering

Grafer med negativa vikter

Bågar i grafer kan ha en negativ vikt, det vill säga ha ett negativt värde. Vilka av algoritmerna från uppgift 1 fungerar utan modifikation för grafer med negativa vikter? Vilka förändringar måste man göra i de övriga för att de ska fungera för grafer med negativa vikter? Observera att lösningen inte får förändra skillnaden mellan olika vikter. Om en båge har vikten -10 och annan båge vikten 10 är ska skillnaden även efter eventuella modifikationer vara 20.

Motivera dina svar så att det framgår vad i algoritmen som gör att den kan eller inte kan hantera negativa vikter.

Minimalt spännande träd

Boken tar upp två algoritmer för att få fram ett minimalt spännande träd ur en graf: Prims och Kruskals. Din uppgift är att skapa en implementation av en oriktad graf samt skriva en metod som använder någon av dessa algoritmer för att hitta ett minimalt spännande träd.

Du får själv välja hur du vill implementera din graf förutsatt att den följer interfacet `MiniGraph`. Värt att tänka på är att olika implementationer har olika fördelar, så fundera på vilken som gör det enklast att implementera den algoritm du valt algoritm.

Du ska också förklara hur du har implementerat algoritmen samt motivera varför du valt din grafimplementation.

Eftersom ni har full frihet när det gäller såväl grafimplementation som algoritmval kan vi inte lägga upp några speciellt effektiva testfall. De testfall som finns kommer därför bara att kontrollera ett par kända grafer, inget mer.

Förslag på pooluppgift

Även denna vecka vill vi ha variation och återanvänder tabell 7.1 på sidan 38. Denna gång är det gruppmedlemmen i mitten vars användarnamn bestämmer vilken restriktion som gäller. Om gruppen består av mindre än tre medlemmar så är det den som står först som används.

1. Frågan måste behandla minimalt spännande träd.
2. Frågan måste behandla hur grafer kan implementeras.
3. Frågan måste behandla *djupet-förstalgoritmer* för sökning i en graf.
4. Frågan måste behandla *bredden-förstalgoritmer* för sökning i en graf.
5. Frågan måste behandla viktade grafer.

7.8 Vecka 8: algoritmdesigntekniker

Denna vecka får ni själva välja vilka uppgifter ni vill göra på de två första uppgifterna. Den enda Restriktionen är att de olika uppgifterna måste behandla olika algoritmdesigntekniker. Om ni till exempel ska göra en poolfråga om giriga algoritmer så får ni inte välja detta ämne för någon av de två första uppgifterna.

Beskriv en algoritmdesignteknik

Denna uppgift är att skriva en kort beskrivning av en algoritmdesignteknik. Målgruppen är människor som har läst en grundläggande kurs i programmering, men som i övrigt inte har någon bakgrund eller förkunskaper inom algoritmer och datastrukturer. Du måste därför förklara så pass utförligt att ämnet blir begripligt för någon ur målgruppen.

De tekniker som du kan välja på är:

- giriga algoritmer
- divide and conquer
- backtracking
- dynamisk programmering
- slumpmässiga algoritmer

Tänk på att ni måste behandla olika ämnen i de olika uppgifterna och att ni inte får välja den sista uppgiften fritt.

Implementation

Denna uppgift går ut på att implementera någonting. Uppgifterna är hämtade ur kursboken, och de du kan välja på är följande:

- giriga algoritmer
- Uppgift 10.16: implementera *closest-pairalgoritmen* från boken. (divide and conquer)
- Uppgift 10.58b: implementera en backtrackingalgoritm för det så kallade åttadamersproblemet.
- Uppgift 10:52a: implementera en algoritm för att dela upp ord i rader med hjälp av dynamisk programmering.
- Uppgift 10.36: implementera insättning, borttag och sökning i en skip list. (slumpmässiga algoritmer)

Tänk på att ni måste behandla olika ämnen i de olika uppgifterna och att ni inte får välja den sista uppgiften fritt.

Förslag på pooluppgift

Även denna vecka vill vi ha variation och återanvänder tabell 7.1 på sidan 38. Denna gång är det gruppmedlemmen i mitten vars användarnamn bestämmer vilken restriktion som gäller. Om gruppen består av mindre än tre medlemmar så är det den som står sist som används.

1. Frågan måste behandla giriga algoritmer.
2. Frågan måste behandla divide and conquer.
3. Frågan måste behandla backtracking.
4. Frågan måste behandla dynamisk programmering.
5. Frågan måste behandla slumpmässiga algoritmer.

7.9 Vecka 9: avancerade datastrukturer

Två datastrukturer

Nedan finns en lista på några av de datastrukturer som listas i kapitel 12 i kursboken.

- Rödsvarta träd
- Splayträd
- Deterministiska skip lists
- AA-träd

- k-d-träd
- Treap
- Pairing heaps

Välj ut två av datastrukturerna. För varje datastruktur som du har valt: beskriv hur datastrukturen är strukturerad, vilka fördelar som datastrukturen har samt i vilka tillämpningar som datastrukturern är lämpliga att använda. Du ska också beskriva hur operationerna som man kan göra på datastrukturen (insättning, uttag och så vidare) går till.

Som vanligt är målgruppen människor som har läst en grundläggande kurs i programmering, men som i övrigt inte har någon bakgrund eller förkunskaper inom algoritmer och datastrukturer.

Förslag på pooluppgift

Även denna vecka vill vi ha variation och återanvänder tabell 7.1 på sidan 38. Denna gång är det gruppmedlemmen som står först vars användarnamn bestämmer vilken restriktion som gäller.

1. Frågan måste behandla rödsvarta träd
2. Frågan måste behandla splayträd
3. Frågan måste behandla k-d-träd
4. Frågan måste behandla någon av datastrukturerna som tas upp förutom de tre som listades explicit ovan.
5. Frågan måste behandla någon av datastrukturerna som tas upp förutom de tre som listades explicit ovan.

Del III

Programmeringsuppgifter

8 Programmeringsuppgifter

Här hittar du förslag på programmeringsuppgifter som du kan göra och redovisa på muntan istället för att ta någon av slumpfrågorna. Du får max göra två sådana byten, och tänk också på att om du redan är godkänd på ett tema så är det ingen större idé att bara göra godkäntdelen.

För vissa uppgifter finns det också JUnit-testfall (se sidan 13) att ladda ner i Moodle. Dessa testfall är främst till för att fånga upp vanliga fel som studenter gjort tidigare år. De är alltså inte avsedda att vara kompletta, och bara för att din kod går igenom dem betyder det inte att du kommer att bli godkänd.

Till skillnad från veckouppgifterna får du välja språk själv här. Kravet är att det ska vara ett objektorienterat språk och ett som muntaledaren har en rimlig chans att kunna läsa. C#, C++, Java, Python och Ruby faller alla under denna kategori. Vill du använda något annat språk bör du fråga först.

8.1 Fritt vald uppgift

Rubriken på uppgiften är inte ett skämt. Eftersom kursen täcker väldigt mycket så vill vi inte begränsa er i onödan. Dessutom vill vi att ni får syssla med uppgifter som ni själva tycker är intressanta. Har du ett förslag på en uppgift så den med kursledningen. Vi bedömer den och säger om den passar kursen och vilket betyg du eventuellt skulle kunna få. Om uppgiften är bra så lägger vi naturligtvis till den till vår uppgiftspool också.

3d-grafik

Det här är bara en idé på en uppgift som går utanför boken. Det är inte en uppgift man bör ta på sig om man inte är intresserad och har en hyfsad grund i matte. Den kräver att man läser in sig rejält på området.

Skriv ett bibliotek för 3d-grafik.

Det här är en uppgift som kan varieras i oändlighet både i omfång och fokus. Det går därför inte att sätta några klara gränser för betyg eller till vilket tema uppgiften ska räknas. Det beror på vad du faktiskt gör och måste därför diskuteras med kursledningen innan du börjar.

8.2 Linjära datastrukturer: lista, stack och kö

De linjära datastrukturer som tas upp på kursen är så pass enkla att uppgiftern i denna kategori inte kan ge ett högre betyg än godkänt.

Utskrift av del av lista

Detta är en variant på uppgift 3.1 från boken: du ges två stycken listor där den andra innehåller `Integer`s sorterade i storleksordning. Din uppgift är att

skriva en metod som returnerar en lista med de värden ur den första listan vars index finns i den andra. Du får bara använda dig av de publika delarna av Javas collection-api, dvs de metoder som finns i `List`.

Ett krav för att du ska bli godkänd är att din lösning ska gå i $O(N)$ för alla listtyper i Javas standardbibliotek. Den ska alltså fungera lika bra för `LinkedList` som för `ArrayList`. JUnit-testfallen som finns med kan inte kontrollera detta direkt, men gör sitt bästa.

Observera det här kravet!

Det är vidare *inte* okej att lösa uppgiften genom att kopiera innehållet i innehållslistan till en array eller motsvarande, utan den ska lösas med hjälp av de listor som skickas in.

OrderedCollection

Det här är en variant på uppgift 1.14 ur boken: att *designa, implementera och dokumentera* en samlingsklass som hanterar Comparables med hjälp av en array. Boken listar ett antal metoder som klassen ska ha. Observera att det inte finns med någon `get`-metod. Detta betyder antagligen att konceptet med index är ganska meningslöst, åtminstone utåt. Det är därför knappast troligt att `insert` ska ta ett index, vilket också betyder att man måste fundera på hur `delete` ska fungera.

Det finns inga JUnit-testfall till denna uppgift eftersom dessa skulle styra designen alltför mycket. Det är därför väldigt viktigt att du testar koden noga.

8.3 Algoritmanalys

Av, förhoppningsvis, uppenbara skäl finns det inga programmeringsuppgifter till detta tema. Här måste du klara inlämningsuppgifterna om du inte vill ta delen på muntan.

8.4 Träd

Huffmankodning

Huffmankodning är ett exempel på en packningsalgoritm. Uppgiften består i att skriva ett program som avkodar en huffmankodad fil. Eftersom instruktionerna för detta är ganska långa har vi lagt dem i en separat uppgiftsbeskrivning som finns i Moodle. Där finns också ett antal testfiler. Med testfiler menar vi i detta sammanhang filer att avkoda, inte JUnit-testfall.

Kompilator

Det här förslaget är bara en idé om ett område som kan vara intressant, speciellt för dem som läst AUTO och/eller PROP. Kontakta Henrik för mer information.

Kompilatorer är ett av de mest välstuderade ämnena inom datavetenskapen. Här finns det många möjligheter att göra intressanta nedslag. En lämplig uppgift skulle till exempel kunna vara att för hand skirva en lexer och en parser

för ett enkelt språk och generera ett explicit parseträd och sen visa hur man skulle kunna vandra igenom det för att generera kod.

8.5 Hashtabeller och hashning

Alla versioner av hashning implementeras inte i boken, så man kan tänka sig en implementation och jämförelse mellan olika versioner på hashtabeller. Något som också kan fungera är att titta på en lämplig hashfunktion för en viss typ av data. Det sistnämnda kan dock vara svårt att få upp till VG-nivå utan att man också är beredd att diskutera det teoretiskt.

8.6 Prioritetsköer

Inga direkta förslag på detta ämne. Har du några idéer om vad du vill göra så kontakta kursledningen.

Uppgift 6.37 i boken kan kanske vara en idé på en uppgift, dock bara på G-nivå.

8.7 Sortering

Inga direkta förslag på detta ämne. Har du några idéer om vad du vill göra så kontakta kursledningen.

Det är knappast någon idé att be att få implementera en sorteringsalgoritm. De vanligaste finns ju direkt i boken, och veckouppgiften handlar om det, så vi känner inte att det ger något nytt. Man kan dock tänka sig att göra något med visualisering, liksom olika typer av mätning på hur stor skillnaden blir mellan olika implementationer med olika tradeoffs.

8.8 Grafer

Kevin Bacon-game

Implementera ett program som frågar efter namnet på en skådespelare och som skriver ut dennes Bacon-nummer. Se kursboken uppgift 9.53 för en en förklaring till Bacon-nummer.

Listor med skådespelare finns på `ftp://ftp.fu-berlin.de/pub/misc/movies/database/`. De relevanta filerna är *actors.list.gz* och *actresses.list.gz* som uppackade blir närmare en gigabyte. Skelettkod för att läsa in data från filerna finns i Moodle.

För betyget G ska du definiera, implementera och dokumentera en datastruktur och en sökalgoritm för problemet. Det ställs inga speciella krav på att sökningen ska vara effektiv, men den ska fungera med de givna filerna. En rekommendation är att du kapar ner filerna betydligt när du testar.

För betyget VG ska din lösning vara effektiv och du måste kunna motivera alla val ordentligt.

SL-light

En av de tjänster som finns på SL.se är en reseplanerare. Man skriver in varifrån man vill åka och var man vill åka till och sen räknar applikationen ut den bästa resvägen. Den här uppgiften går ut på att implementera en förenklad version av denna tjänst.

För betyget G ska du definiera, implementera och dokumentera en lämplig datastruktur för denna typ av tjänst. Naturligtvis är det frågan om en viktad graf, men det räcker inte med att ha med vikten mellan noderna (hållplatser och stationer) utan du måste ju hålla reda på när till exempel en buss går och vilken buss det är. Varje hållplats och station ska också ha kartkoordinater. *Vanligt fel!*

För betyget VG ska du också implementera en heuristisk sökalgoritm (i stil med A*) som givet starttiden, startpunkten och slutpunkten hittar den bästa färdvägen.

Vill du göra uppgiften mer realistisk så kan du dessutom testa att lägga in ytterligare funktionlitet som att man kan gå mellan hållplatser eller att man försöker minimera antalet byten om det inte påverkar restiden för mycket.

Non intelligent text generator

Ett enkelt sätt att generera text som åtminstone ytligt sett ser ok ut är att läsa in ett antal meningar och sen bygga upp en graf över hur orden hänger ihop. I sin enklaste form kollar man bara vilka ord som kan följa på vilka. Ett bra exempel[1] är `http://www.jibble.org/dynamicgraph.php`.

Om ni testar att skriva in några meningar i ovan nämnda implementation så ser ni efter ett tag att de inte längre blir så naturliga. Anledningen är att man bara tar hänsyn till ett ord i taget. Ett enkelt sätt att få det mer naturligt är att titta på flera ord. Mer information finns på:

- `http://en.wikipedia.org/wiki/Markov_chain`
- `http://en.wikipedia.org/wiki/Markov_chain#Markov_text_generators`
- `http://en.wikipedia.org/wiki/Dissociated_press`

Uppgiften går naturligtvis ut på att skriva ett program som läser in text från användaren och skriver ut ett svar på det. Dvs något i stil med det tidigare nämnda programmet på `http://www.jibble.org/dynamicgraph.php`

Appleten på `http://www.jibble.org/dynamicgraph.php` ritar upp grafen hela tiden. Det är praktiskt vid debugging och för att förstå idén, men är inte ett krav.

[1] Niall som nämns på sidan var ett av Henriks favoritprogram för många år sedan. Namnet står för Non Intelligent AMOS Language Learner. Vad tiden går...

Rita grafer

Beroende på vilken algoritm man använder sig av kan denna uppgift även falla under algoritmdesigntekniker eller avancerade datastrukturer. Vill du ha den där så kontrollera det först med kursledningen.

Det finns gott om algoritmer för att rita upp träd och grafer på ett snyggt sätt. En uppgift skulle kunna vara att göra ett program som ritar upp en viss graftyp enligt en viss algoritm. En teoretisk att motivera ett val. Beroende på hur pass långt man går kan det också ge ett VG, men det måste bedömas från fall till fall.

Ett bra ställe att börja leta information är biblioteket. Gör en sökning i deras databas efter *graph drawing* och du kommer att hitta en stor mängd elektroniska resurser om ämnet. De flesta är samlingar av rapporter från vetenskapliga konferenser. Dessa kan vara väldigt intressanta, men många av rapporterna är för avancerade, det ska man vara på det klara med. Av mer intresse är att det finns en bok tillgänglig elektroniskt: *Planar Graph Drawing* av Nishizeki och Rahman [9]. Har inte kollat igenom den, men innehållsmässigt sett verkar den helt ok. Även *Drawing Graphs: Methods and Models* av Michael Kaufmann och Dorothea Wagner [4] är bra, men finns inte elektroniskt.

Minimering av ändliga automater

Ändliga automater, finite state automata på engelska, är en viktig klass av abstrakta [datorlika] maskiner med många användningsområden, bland annat vid språkigenkänning, kompilatorer och AI.

Hopcrofts $O(NlogN)$-algoritm för minimering av ändliga automater [2] är den bästa kända. Tyvärr är den relativt krånglig att implementera så att den verkligen kör i $O(NlogN)$-tid och inte degenererar. Uppgiften i det här fallet består av två delar:

För godkänt: läs in dig på ändliga automater och deras praktiska användningsområden. En bra översikt för dem som inte läst AUTO ges i kapitel 2–4 i *Introduction to Automata Theory, Languages, and Computation* [3]. Implementera sedan en klass som representerar en ändlig automat. Förutom de metoder som krävs för att bygga upp automaten så ska den ha en enda metod:

```
public boolean accept(String text);
```

För väl godkänt: läs in dig på och implementera Hopcrofts minimeringsalgoritm. En bra källa är *A Taxonomy of Finite Automata Minimization Algorithms* [10]. Henriks magisteruppsats [1] innehåller också en sammanfattning. Testa sedan om din implementation kör i $O(N \log N)$ eller inte. Vid redovisningen ska du kunna förklara varför, eller varför inte, som din implementation kör i $O(N \log N)$.

8.9 Algoritmdesigntekniker

Skiplists

Kursbokens kapitel 10.4.2 och 12.3 diskuterar skip lists i olika varianter. Det är en typ av sorterade listor där insättning och sökning går i $O(logN)$. Uppgiften

är enkel: gör en egen implementation av en skip list. (Valfri version.)

För godkänt krävs att klassen fungerar, är vältestad och att den är ordentligt dokumenterad. Denna uppgift går tyvärr inte att bygga ut till VG.

Ni får om ni vill implementera något av interfacen i Javas collection-api, men det är inget krav.

MinMax

Kapitel 10.5.2 i kursboken tar upp minmax-strategier för spel.

För ett godkänt betyg så ska du implementera ett spel med en datorspelare som använder sig av minmax-strategi. Fem-i-rad, eller reversi/Othello är att rekomendera, men såväl schack som poker skulle nog också fungera. (Du får inte ta tre-i-rad/tic-tac-toe, det måste bli någon sökning...)

Ditt spel behöver inte ha något grafiskt användargränssnitt om du inte vill. Hur som helst påverkar det inte betyget i någon riktning.

För ett VG ska du också implementera alpha-beta-pruning, eller motsvarande.

8.10 Avancerade datastrukturer

Inga direkta förslag på detta ämne. Har du några idéer om vad du vill göra så kontakta kursledningen.

Flera av uppgifteran i boken skulle kunna fungera, liksom att empiriskt jämföra olika datastrukturer.

Del IV

Frågepool till muntan

9 Frågepool till muntan

Denna del kommer att växa till sig under kursens gång. Samtliga frågor på muntan kommer att väljas ur denna frågepool. Observera dock att frågorna kan förändras när som helst, alltså även efter att muntorna börjat ges.

9.1 Linjära datastrukturer: lista, stack och kö

Status: en första genomgång har gjorts. Behöver fortfarande uppdateras.

Detta första tema är så pass grundläggande att vi inte lagt ner något arbete på att ta fram kriterier för att visa på den djupare förståelse som krävs för de högre betygen. Vi vill inte helt utesluta möjligheten, men det är antagligen enklare att lägga ner sin tid på uppgifter från senare teman om man villha ett högt betg.

Implementation av listor

Kontrastera skillnaden mellan en arraybaserad lista och en länkad lista. Hur implementeras de två varianterna? Vad skiljer dem åt?

Användning av listor

Vid vilka tillfällen kan det vara fördelaktigt att använda en arraybaserad respektive en länkad lista? Varför är det så? Förklara på ett sätt så att en programmerare kan i framtiden välja lämplig typ av lista för sina program.

Implementation av en länkad lista

Beskriv kortfattat hur de grundläggande operationerna add(index), remove(index) och get(index) implementeras i en länkad lista. Vilka skillnader blir det i implementationen beroende på om listan är enkel eller dubbellänkad? Om listan använder eller inte använder så kallade sentinel-noder.

Implementation av stackar

Vilka för och nackdelar finns med att implementera en stack med hjälp av en länkad lista jämfört med att implementera den med hjälp av en array?

Implementation av köer

Vilka för och nackdelar finns med att implementera en kö med hjälp av en länkad lista jämfört med att implementera den med hjälp av en array?

Implementation för effektiv get

I en implementation där det behövs en lista och det är viktigt att get(int) går snabbt, däremot är det inga krav på ordningen av elementen. Element kommer

att tas bort i början och mitten av listan och läggas till på godtycklig plats. Vad för slags lista bör användas och vad för tekniker kan utnyttjas för att få bättre prestanda och vad får dessa tekniker för effekt på ordo-notationen av add(E), remove(int), get(int).

Allmänt

- Ge exempel på olika fall där man föredrar att använda en lista, en stack eller kö och varför?
- Varför vill vill man använda en stack när liknande funktionalitet kan uppnås genom att hämta ut det sista elementet i en lista?

Användning

Boken ger flera exempel på praktiska användningsområden för listor, stackar och köer. Ge en kortfattad sammanfattning av några av dessa. För godkänt måste du förhålla dig självständig i förhållande till boken.

Arrayer kontra listor

- Fråga 1: Vad är lämpligast att använda av linked list och array list när du ofta kommer behöva lägga till och ta bort element och varför?
- Fråga 2: I vilka listor kan det vara lämpligt att använda lazy deletion? När är det lämpligt?
- Fråga 3: När du tar bort eller lägger till element i en array, vilken är det bästa respektive sämsta fallet som kan hända uttryckt i ordo? Varför är det bara det värsta fallet som är relevant?
- Fråga 4: Varför vill du hellre göra en array-lista relativt liten för att sedan göra den större vid behöv istället för att ha en stor array från början?
- Fråga 5: Varför brukar det gå snabbare att göra sökningar i arrayer än länkade listor?

9.2 Algoritmanalys

Detta tema fungerar lite annorlunda än de andra. Här finns det nämligen bara en enda möjlig fråga egentligen. Du kommer att ges ett stycke kod, eller en algoritmbeskrivning, och ska diskutera tids och minneskomplexiteten hos densamma.

9.3 Träd

Rekursion

Why do many programmers prefer to use recursion instead of iteration when dealing with operations on trees? To obtain G, the following must be noted: Recursion usually results in simpler code, both to write and read. To obtain

VG, the followin must be noted: Recursion is usually slower, but not slow enough that the tradeoff between readability and performance is unacceptable.

Binära sökträd

Vilka förutsättningar ett binär träd måste ha för att vara ett binär sökträd och när är det effektivt att använda ett binär sökträd?

Skillnad mellan träd

Förklara de generella skillnaderna mellan de följande typerna av träd, binärt träd, binärt sökträd, AVL-träd. AVL-träd använder både enkla och dubbla rotationer, förklara när vilken av dessa är önskvärd att utföra i förhållande till vilket tillstånd trädet har. Finns det någon övergripande skillnad hur rotationer utförs för insättning av element i förhållande till bortagning av element?

Lazy deletion i ett AVL-träd (I)

Vid implementation av ett binärt sökträd kan man bland annat välja mellan att radera objekt direkt vid borttagning eller använda Lazy Deletion. Vilka argument finns det för respektive alternativ? Kan ett av dem ses som generellt effektivare än det andra? Om inte, i vilka situationer skulle de respektive alternativen vara starka? Är lazy deletion lika bra alternativ när man ska ta bort något i toppen av ett träd jämfört med när det är nära löven, eller är det någon skillnad och i så fall hur? Bortse från minnesåtgång och fokusera på tidsaspekten.

Lazy deletion i ett AVL-träd (II)

Varför är det en prestandavinst i att använda lazy deletion i Avl-träd, jämnfört med en vanlig delete-implementation, om man ska använda med många add och delete-anrop?

Varför kan man inte bara gå rakt ned i den vänstra grenen i ett Avl-träd (som implementerar lazy deletion) för att hitta det minsta värdet? Finns det någon gräns på hur många noder den kan behöva itterera igenom innan den måste hitta en node som är minst?

Rotationer i AVL-träd

Förklara enkelt och dubbelrotationer i AVL-träd.

alt

I AVL-träd pratas det ofta om fyra cases som beskriver när ett träd behöver balanseras. Beskriv dessa cases och ge exempel på hur vi löser dessa i ett AVL-träd. Beskriv i pseudokod/vanlig kod eller beskriv tydligt med hjälp av bilder hur stegen ska genomföras.

Operationer på binära träd

Fråga om binära träd, och lite rekursion: Hur skulle man gå tillväga för att, med en enkel rekursiv metod, spegelvända pekarna till samtliga noder i ett

binärt träd? Se den något simplistiska modellen nedan: 5 Fig1 4 / 2 5 / 1 3 Fig1 ska alltså bli Fig2: Fig2 4 / 5 2 / 3 1

Du har fått i uppgift att implementera en metod för ett binärt sökträd som kontrollerar om det innehåller ett specifikt element. Vad är vikitgt att ta hänsyn till för att hitta ett element i trädet på ett effektivt sätt? Skulle du skriva koden med rekursion eller iteration? Motivera ditt val.

Definiera termerna preorder traversal och postorder traversal. Ge praktiska exempel på när dessa används.

Beskriv hur man kan räkna ut storleken på ett binärt träd rekursivt. (Exempelbild med några få noder...) Beskriv sedan steg för steg hur detta görs.

Skriv en funktion som tar bort alla löven från ett träd. (Alltså noderna som inte har några barn). En effektivare metod ger högre betyg.

AVL och B-träd

Klargör vad som kännetecknar ett AVL respektive B-träd. Vad skiljer dessa åt, vad är dess för och nackdelar? Motivera och resonera kring ditt svar.

Begrepp

Följande begrepp tillhör binärträd, för godkänd räcker det med 5 korrekta svar, för betyg högre än godkänd ska minst 8 av 9 begrepp vara korrekta. Riktad kant Rodnoden Förälder Barn Löv Djupet Höjden Syskon Delträd/subträd

Degenererade träd

Förklara vad som händer när ett träd degenererar. Vad för påverkan har det på tiden (ordo) för operationerna på trädet? Beskriv det värsta scenariot för ett degenererat träd. Ge exempel på teknik för att motverka degenerering och förklara hur det fungerar.

Rotationer

I AVL-träd utförs rotationer för att bibehålla rätt balans i trädet. Rotera dessa två träd (genom att rita upp träden efter rotationen). Förklara vad du gör.

Alt genom att sätta in ett antal värden i ett existerande träd.

Implementation av träd

Ge exempel på två olika sätt att implementera träd, valfria sorter. Beskriv de viktigaste implementationsm ässiga skillnaderna mellan dem.

9.4 Hashtabeller och hashning

Kollisionshantering

Nämn 3st olika kollissionshanteringsvarianter for hashfunktioner och beskriv kortfattat vad dem gör. för vg, ska svaret även innehålla en mer omfattande beksrivning hur dem fungerar.

Hur fungerar linear probing, quadratic probing och double hashing, och när bör man/bör man inte använda respektive metod för kollisionshantering?

Hashfunktioner

Vad bör man som programmerare ta i beaktning när man gör en hashningsalgoritm?

Hashing: Vad finns det för för- respektive nackdelar med open addressing och separate chaining och hur relaterar dem till varandra?

För objekt av typen person, med attributen str:namn, str:adress, int:ålder och int:personnummer, beskriv en hashfunktion för att lagra dem i en hashtabell. Argumentera för varför detta kommer ge minimalt antal kollisioner.

Förklara varför längden på hashtabeller generellt bör vara ett primtal. För högre betyg ska du även ange en typ av kollisionshantering där det är kritiskt att tabellängden är ett primtal, visa varför det är så och vad som kan hända ifall tabellens längd inte är ett primtal.

Load factor

What is meant with the term load factoränd how does it affect the performance of hash tables under linear and quadratic probing? At (roughly) what load factor must a linear or quadratic probed hash table be expanded?

Sätt in och plocka bort ett antal element i en hashtabell med en viss kollisionshanteringsstrategi

Ge ett exempel på omständigheter som kan få separate chaining att bli ett mer tidseffektivt val av kollisionshanteringsmetod än linear probing. Förklara sedan varför.

Förklara getHashCode, equals och compareTos roller vid insättning av objekt i en hashtabell i Java.

För separate chaining är det bra att hålla en load factor på ca 1. Då det gäller linear probing bör en load factor på ca 0.5 hållas. Varför skiljer de sig åt? För VG - förklara begreppet primary clustering.

Förklara kortfattat hur hashing fungerar, förklara även vilka fördelar det har gentemot sökträd? Fördjupande fråga Förklara hur operationerna insättning, borttag och sökning kan ske i genomsnittligt konstant tid med hashing.

För att ett typ av objekt skall kunna användas som nyckel i ett hashset eller nyckel i en hashmap så måste två stycken metoder från javas grundklass Object att överskuggas. a) Vilka är dessa metoder, och varför behövs dem inom hashning? b) Förklara mer specifikt hur dessa används vid en implementation som använder sig utan separate chaining.

9.5 Prioritetsköer

Vilka två operationer måste en prioritetskö ha? för vg, förklara hur varje operation går till och hur dem fungerar.

A) Sätt in 8, 55, 19, 7, 20 och 13 i en binär heap. Beskriv prossesen steg för steg. B) Ta bort tre tal och återigen beskriv prossesen.

I vilka situationer skulle man vilja implementera en prioritetskö i en array och när vill man istället implementera kön i ett binärt sökträd? Vad är fördelarna med respektive metod?

Prioritetsköer: Varför är det så vanligt med en trädimplementation av heaps och hur påverkas operationer som t.ex. insättning och borttag jämfört med om man t.ex. hade använt en sorterad lista?

Beskriv de strukturella krav som en prioritetskö måste uppfylla och de operationer den måste ha. Ge exempel på en situation då en prioritetskö är att föredra över en vanlig kö.

Ge exempel på två enkla sätt att implementera en prioritetskö med en länkad lista. För högre betyg ska du argumentera för vilken du tycker är bäst och varför.

Förklara hur en binary heap fungerar. För ett högre betyg, beskriv hur man implementerar en binary heap inuti en array.

Bredd kontra djup

Redogör för vilka potentiella vinster/förluster man gör med ett brett träd (en heap med fler än 2 barn på varje nod) jämnfört med ett djupt träd (med endast 2 barn på varje nod). För högre betyg ge förslag på när ett sådant brett träd är användbart.

Vilken grundläggande operation för en d-heap blir snabbare ju fler barn man har, och vilken blir långsammare? Ge ett exempel på en situation där det är fördelaktigt med en d-heap där elementen har många barn.

Vad är fördelarna med att använda sig av en heap gentemot av en linked list eller ett binärt sökträd då man ska implementera en prioritetskö?

a) Förklara konceptet bakom percolate up och percolate down när man använder heaps. Vad används de till och hur fungerar de? b) Visa hur du skulle genomföra en percolate up i heap A, och en percolate down i heap B som

nns på figuren nedan (se figur 1).

9.6 Sortering

9.7 Grafer

9.8 Algoritmdesigntekniker

9.9 Avancerade datastrukturer

Litteraturförteckning

[1] Henrik Bergström. *Applications, Minimisation, and Visualisation of Finite State Machines*. magisteruppsats, Institutionen för data- och systemvetenskap, Stockholms universitet, 1998.

[2] John E. Hopcroft. An n log n algorithm for minimizing the states in a finite automaton. In Zvi Kohavi and Azaria Paz, editors, *Theory of machines and computations : proceedings of an international symposium on ...*, pages 189–196, 1971.

[3] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison Wesley, 2007.

[4] Michael Kaufmann and Dorothea Wagner. *Drawing Graphs: Methods and Models*. Lecture Notes in Computer Science 2025. Springer, 2001.

[5] Donald E. Knuth. *The Art of Computer Programming: Volume 1 – Fundamental Algorithms*. Addison Wesley, third edition, 1997.

[6] Donald E. Knuth. *The Art of Computer Programming: Volume 2 – Seminumerical Algorithms*. Addison Wesley, third edition, 1998.

[7] Donald E. Knuth. *The Art of Computer Programming: Volume 3 – Sorting and Searching*. Addison Wesley, second edition, 1998.

[8] SUN Microsystems. How to write doc comments for the javadoc tool. `http://www.oracle.com/technetwork/java/javase/documentation/index-137868.html`, 2004.

[9] Takao Nishizeki and Md Saidur Rahman. *Planar Graph Drawing*. Lecture Notes Series on Computing. World Scientific Publishing Co., 2004.

[10] Bruce W. Watson. A taxonomy of finite automata minimization algorithms. `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.55.8900`, 1994.

[11] Mark Allen Weiss. *Data Structures and Algorithm Analysis in Java*. Addison Wesley, 2007.