Høgskolen i Telemark

EKSAMEN

5610 ALGORITMAR OG DATASTRUKTURAR

12.05.2015

Tid:	9-14 (5 timar)
Målform:	Bokmål og nynorsk
Sidetal:	9 (forside + 4 + 4)
Hjelpemiddel:	Alle trykte og skrivne
Vedlegg:	Ingen

Eksamensresultata blir offentliggjort på nettet via Studentweb

Råd og retningslinjer. Les oppgaveteksten godt før du går i gang med å løse oppgaven. Deloppgavene er uavhengige av hverandre i den forstand at om du ikke får til en oppgave, kan du likevel gjøre neste, som om den første var løst. Fordél tiden godt på alle oppgavene. Om du mener en oppgave er upresis, så skriv din egen presisering. Lag de hjelpemetoder du måtte ønske.

Oppgave 1 – binærtre (35%)

1a (5%)

Et binærtre (ikke nødvendigvis et binært søketre) har ti noder med verdiene 1, 2, 3, ... 10.
Postorden traversering av treet gir sekvensen 6, 9, 4, 3, 1, 2, 10, 8, 5, 7.
Tegn et tre som passer med denne beskrivelsen – det finnes mange løsninger. Du skal ikke levere tegningen, men derimot skrive ut nodene i preorden, inorden og nivåorden rekkefølge.

1b (5%)

Et binærtre (ikke nødvendigvis et binært søketre) har 8 noder med verdiene A, B, C, ..., H.
Preorden traversering av treet gir sekvensen H, E, A, F, D, B, C, G.
Inorden traversering av treet gir sekvensen F, A, D, E, H, C, B, G.
Tegn et tre som passer med denne beskrivelsen – det finnes bare én løsning. Du skal ikke levere tegningen, men derimot skrive ut nodene i postorden og nivåorden rekkefølge.

1c (15%)

Med utgangspunkt i lærebokas klasser for binærtre (class BinaryTree og class BinaryNode) skal du lage fire public metoder som skal ligge i BinaryTree. De skal traversere treet og skrive ut nodene i

- baklengs preorden rekkefølge
- baklengs inorden rekkefølge
- baklengs postorden rekkefølge
- baklengs nivåorden rekkefølge

Utskriftene skal altså liste opp nodene i motsatt rekkefølge av hva de fire vanlige traverseringsrekkefølgene gjør. Dersom du ønsker kan du lage (evt rekursive) hjelpemetoder i class BinaryTree og/eller i BinaryNode.

1d (10%)

Dersom hver node i et binærtre også har en referanse til foreldrenoden, kan man gjøre inorden traversering iterativt, uten rekursjon eller ekstra datastruktur. Den neste i inorden rekkefølge finnes slik:

- Dersom noden har høyrebarn, gå til noden lengst til venstre i høyre subtre.
- Ellers: Så lenge noden vi står i er høyre barn av forelder – gå til forelder. Gå så ett trinn til oppover.

Lag metoden

```
public void skrivInorden()
```

i class BinaryTree etter denne idéen, der vi antar at BinaryNode også har foreldrereferanse «parent». Rotnoden sin foreldrereferanse er null.

Oppgave 2 – Norsk Tipping (65%)

Du er leid inn for å lage nytt datasystem for Norsk Tipping sin tradisjonelle fotballtipping. En tippekupong består av 12 kamper, se figur 1.

Figur 1: Tippekupong

Man tipper ved å sette kryss i rutene H/U/B, som står for hjemmeseier/uavgjort/borteseier. Dersom man setter ett kryss ut for hver av de 12 kampene, har man tippet én rekke. Det er to måter å fylle ut tippekupongen på:

1. Man kan tippe én eller flere slike *enkeltrekker*. Figur 1 viser to rekker, men det kan være opp til 10.
2. Man kan tippe *system*, som betyr at man kan *gardere* – sette flere kryss i en eller flere kamper. Det effektive antall rekker (og dermed prisen) øker: For hver halvgardering (to kryss i samme kamp) dobles antall rekker, hver helgardering (tre kryss i samme kamp) gir tre ganger så mange rekker. 12 helgarderinger ville i prinsippet utgjøre $3^{12} = 531441$ rekker, men det er bare lov å ha inntil 486 rekker på én kupong.

class TippeKupong skal utvikles for å representere en tippekupong. Følgende skisse er laget:

```
public class TippeKupong {
  final static int KAMPER = 12;      // 12 kamper på kupongen
  final static int REKKEPRIS = 1;    // Pris kr 1 per rekke
  public boolean system(){...}       // returnerer true dersom systemtipping
                                     // og false ellers
  public int antallRekker(){...}     // antall rekker på kupongen
  public int pris(){...}             // prisen for hele kupongen
  public int tiRette(String res){...}     // Antall rekker med ti rette
  public int elleveRette(String res){...} // Antall rekker med elleve rette
  public int tolvRette(String res){...}   // Antall rekker med tolv rette
}
```

Parameteren «res» (resultat) må inneholde tolv tegn bestående av H, U og B.
Prosjektgruppa diskuterer hvilken datastruktur som skal brukes til å representere enkeltrekke-kuponger og systemkuponger. Det er i alle fall klart at det trengs ulike strukturer, det skal derfor lages to subklasser, en for hver av de to kupongtypene.

2a (10%)

Lag klassen TippeKupong som en abstrakt klasse. Dersom noen av metodene kan implementeres så gjør det.

2b (20%)

Vi skal først se på representasjon av enkeltrekker. Hver enkelt rekke skal utgjøre ett objekt, men også disse objektene kan lages på mange måter. Her er det laget et grensesnitt:

```
interface EnkeltRekke{
  void set(int kamp, char tippet);      // sett tipping i bestemt kamp, der
                                        // kamp er 1..12.
  char get(int kamp);                   // hent ut tipping i bestemt kamp
  boolean riktig(int kamp, char resultat);  // return true dersom tipping
                                        // er lik resultat i bestemt kamp
  int antallRette(String resultat);     // sammenlign tipping og
                                        // resultat, returner antall rette
}
```

Lag klassen EnkeltRekke1 som implementerer grensesnittet EnkeltRekke ved å representere rekka som en char array med lengde 12. Den må implementere alle metodene i grensesnittet. Den skal videre ha parameterløs konstruktørmetode som setter alle kampene til hjemmeseier, og en konstruktørmetode som får inn hele rekka som en String. Parametre og returverdier av type char skal være en av {H, U, B}. Gjør grundig feilsjekking – kast unntak!

2c (20%)

Lag klassen EnkeltRekkeKupong som subklasse av TippeKupong, og som holder på inntil ti enkeltrekker i en LinkedList. EnkeltRekkeKupong skal ikke være abstrakt. Skriv klassen slik at det senere er enkelt å bytte ut EnkeltRekke1 med andre måter å representere enkeltrekkene. Klassen skal ha parameterløs konstruktørmetode, samt en metode

```
public void nyRekke(String rekke){...}
```

-som legger inn en ny enkeltrekke.

2d (15%)

Vi skal her se på hvordan vi kan representere systemkuponger. Man kunne tenke seg å bruke samme prinsipp som enkeltrekkene – bruke char. Men eksperimenter og regnestykker viser at det tar mye plass, det skal derfor brukes et mer kompakt format. 12 kamper kan ha kryss eller ikke kryss for H/U/B. Det utgjør 12x3 = 36 biter informasjon (egentlig litt mindre – ikke alle kombinasjoner er mulige). Disse 36 bitene skal lagres i en long-variabel, altså bruke 64 biter lagringsplass. class SystemKupong skal være subklasse av TippeKupong. Den skal ha to interne hjelpemetoder:

```
protected boolean kryss(int kamp, int rute){...}
protected void settKryss(int kamp, int rute){...}
```

Den første undersøker om kupongen har kryss i bestemt kamp og rute (H=0, U=1, B=2) ved å sjekke en bit i long-variabelen. Den andre registrerer et kryss i bestemt kamp og rute ved å sette en bit til 1. Du bestemmer selv hvilken bit av de 64 du vil bruke til hver kamp/rute. Hjelpemetodene kan komme til nytte i de andre metodene.

Lag class SystemKupong, inklusive de to hjelpemetodene, men du skal ikke lage metodene tiRette, elleveRette og tolvRette.

Lykke til!

Råd og retningsliner. Les oppgåveteksten godt før du går i gang med å løyse oppgåva. Deloppgåvene er uavhengige av kvarandre i den meining at om du ikkje får til ei oppgåve, kan du likevel gjere neste, som om den fyrste var løyst. Fordél tida godt på alle oppgåvene. Om du meiner ei oppgåve er upresis, så skriv din eigen presisering. Lag dei hjelpemetodar du måtte ynskje.

Oppgåve 1 – binærtre (35%)

1a (5%)

Eit binærtre (ikkje nødvendigvis eit binært søketre) har ti nodar med verdiane 1, 2, 3, ... 10.
Postorden traversering av treet gjev sekvensen 6, 9, 4, 3, 1, 2, 10, 8, 5, 7.
Teikn eit tre som passar med denne omtala – det finst mange løysingar. Du skal ikkje levere teikninga, men derimot skrive ut nodane i preorden, inorden og nivåorden rekkefylgje.

1b (5%)

Eit binærtre (ikkje nødvendigvis eit binært søketre) har 8 noder med verdiane A, B, C, ..., H.
Preorden traversering av treet gjev sekvensen H, E, A, F, D, B, C, G.
Inorden traversering av treet gjev sekvensen F, A, D, E, H, C, B, G.
Teikn eit tre som passer med denne omtala – det finst berre ei løysing. Du skal ikkje levere teikninga, men derimot skrive ut nodane i postorden og nivåorden rekkefylgje.

1c (15%)

Med utgangspunkt i læreboka sine klassar for binærtre (class BinaryTree og class BinaryNode) skal du lage fire public metodar som skal ligge i BinaryTree. Dei skal traversere treet og skrive ut nodane i

- baklengs preorden rekkefylgje
- baklengs inorden rekkefylgje
- baklengs postorden rekkefylgje
- baklengs nivåorden rekkefylgje

Utskriftene skal altså liste opp nodane i motsett rekkefylgje av kva dei fire vanlige traverseringsrekkefylgjene gjer. Dersom du ynskjer kan du lage (evt rekursive) hjelpemetodar i class BinaryTree og/eller i BinaryNode.

1d (10%)

Dersom kvar node i eit binærtre også har ein referanse til foreldrenoden, kan ein gjera inorden traversering iterativt, utan rekursjon eller ekstra datastruktur. Den neste i inorden rekkefylgje finnast slik:

- Dersom noden har høgrebarn, gå til noden lengst til venstre i høgre subtre.
- Ellers: Så lenge noden vi står i er høgre barn av forelder – gå til forelder. Gå så eitt trinn til oppover.

Lag metoden

```
public void skrivInorden()
```

i class BinaryTree etter denne idéen, der vi går ut frå at BinaryNode også har foreldrereferanse «parent». Rotnoden sin foreldrereferanse er null.

Oppgåve 2 – Norsk Tipping (65%)

Du er leigd inn for å lage nytt datasystem for Norsk Tipping sin tradisjonelle fotballtipping. Ein tippekupong består av 12 kampar, sjå figur 2.

Figur 2: Tippekupong

Ein tippar ved å sette kryss i rutene H/U/B, som står for heimesiger/uavgjort/bortesiger. Dersom ein set eitt kryss ut for kvar av dei 12 kampane, har ein tippa ei rekke. Det er to måtar å fylle ut tippekupongen på:

1. Ein kan tippe ei eller fleire slike *enkeltrekker*. Figur 2 viser to rekker, men det kan vera opp til 10.
2. Ein kan tippe *system*, som tyder at ein kan *gardere* – sette fleire kryss i ein eller fleire kampar. Det effektive talet på rekker (og dermed prisen) aukar: For kvar halvgardering (to kryss i same kamp) doblast talet på rekker, kvar heilgardering (tre kryss i same kamp) gjev tre gonger så mange rekker. 12 heilgarderingar ville i prinsippet utgjere 3^{12} = 531441 rekker, men det er berre lov å ha inntil 486 rekker på ein kupong.

class TippeKupong skal utviklast for å representere ein tippekupong. Fylgjande skisse er laga:

```
public class TippeKupong {
  final static int KAMPAR = 12;
  final static int REKKEPRIS = 1;   // Pris kr 1 per rekke
  public boolean system(){...}      // returnerer true dersom systemtipping
                                    // og false ellers
  public int talPåRekker(){...}     // tal på rekker på kupongen
  public int pris(){...}            // prisen for heile kupongen
  public int tiRette(String res){...}     // tal på rekker med ti rette
  public int elleveRette(String res){...} // tal på rekker med elleve rette
  public int tolvRette(String res){...}   // tal på rekker med tolv rette
}
```

Parameteren «res» (resultat) må innehalde tolv teikn av H, U og B.
Prosjektgruppa diskuterer kva for ein datastruktur som skal nyttast til å representere enkeltrekke-kupongar og systemkupongar. Det er i alle høve klart at det trengs ulike strukturar, det skal difor lagast to subklasser, ein for kvar av dei to kupongtypane.

2a (10%)

Lag klassa TippeKupong som ei abstrakt klasse. Dersom nokre av metodane kan implementerast så gjer det.

2b (20%)

Vi skal fyrst sjå på representasjon av enkeltrekker. Kvar enkelt rekke skal utgjere eitt objekt, men også desse objekta kan lagast på mange måtar. Her er det laga eit grensesnitt:

```
interface EnkeltRekke{
  void set(int kamp, char tippa);       // set tipping i bestemt kamp, der
                                        // kamp er 1..12.
  char get(int kamp);                   // hent ut tipping i bestemt kamp
  boolean rett(int kamp, char resultat); // return true dersom tipping
                                        // er lik resultat i bestemt kamp
  int talPåRette(String resultat);      // samanlikn tipping og
                                        // resultat, returner tal på rette
}
```

Lag klassa EnkeltRekke1 som implementerer grensesnittet EnkeltRekke ved å representere rekka som ein char array med lengde 12. Den må implementere alle metodane i grensesnittet. Den skal vidare ha parameterlaus konstruktørmetode som set alle kampane til heimesiger, og ein konstruktørmetode som får inn heile rekka som ein String. Parametre og returverdiar av type char skal vera ein av {H, U, B}. Gjer grundig feilsjekking – kast unntak!

2c (20%)

Lag klassa EnkeltRekkeKupong som subklasse av TippeKupong, og som held på inntil ti enkeltrekker i ein LinkedList. EnkeltRekkeKupong skal ikkje vera abstrakt. Skriv klassa slik at det seinare er enkelt å bytte ut EnkeltRekke1 med andre måtar å representere enkeltrekkene. Klassa skal ha parameterlaus konstruktørmetode, og vidare ein metode
`public void nyRekke(String rekke){...}`

-som legg inn ei ny enkeltrekke.

2d (15%)

Vi skal her sjå på korleis vi kan representere systemkupongar. Ein kunne tenkje seg å nytte same prinsipp som enkeltrekkene – nytte char. Men eksperiment og reknestykke viser at det tar mykje plass, det skal difor brukast eit meir kompakt format. 12 kamper kan ha kryss eller ikkje kryss for H/U/B. Det utgjer 12x3 = 36 bitar informasjon (eigentleg litt mindre – ikkje alle kombinasjonar er mogelege). Desse 36 bitane skal lagrast i ein long-variabel, altså nytte 64 bitar lagringsplass. class SystemKupong skal vera subklasse av TippeKupong. Den skal ha to interne hjelpemetodar:

```
protected boolean kryss(int kamp, int rute){...}
protected void settKryss(int kamp, int rute){...}
```

Den fyrste undersøker om kupongen har kryss i bestemt kamp og rute (H=0, U=1, B=2) ved å sjekke ein bit i long-variabelen. Den andre registrerer eit kryss i bestemt kamp og rute ved å sette ein bit til 1. Du bestemmer sjøl kva for ein bit av dei 64 du vil nytte til kvar kamp/rute. Hjelpemetodane kan kome til nytte i dei andre metodane.

Lag class SystemKupong, inklusive dei to hjelpemetodane, men du skal ikkje lage metodane tiRette, elleveRette og tolvRette.

Lykke til!