Høgskolen i Telemark

EKSAMEN

5610 ALGORITMAR OG DATASTRUKTURAR

15.05.2013

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 - Sortering (28%)

Oppgave 1a (8%)

Du skal, i følgende ulike situasjoner, anbefale en sorteringsalgoritme, og forklare hvorfor du anbefaler denne.

Case 1) Flere tusen enkeltarrayer, hver med over over hundre tusen heltall, skal sorteres så raskt som mulig. Innholdet av hver enkelt array er "godt blandet". Det er god plass i RAM til ekstra datastruktur, men vi vet ikke noe om hvor store tallene kan være. Det er her den totale kjøretiden for alle sorteringene som er av størst betydning.

Case 2) Samme situasjon som i case 1), men hver array inneholder nå bare omkring 10 heltall.

Case 3) Et epostsystem skal kunne sortere epostene etter ulike kriterier som mottatt-dato, avsender, tittel osv.

Case 4) Store tabeller, hver med over over hundre tusen heltall, skal sorteres så raskt som mulig. Det er kjent at tabellene er nesten sortert på forhånd.

Oppgave 1b (8%)

Utfør shellSort på følgende tabell med gap-sekvens 7, 3, 1. Vis tabellen for hver gap-verdi.

2	9	4	5	7	3	1	8	3	5	6	9	0	2	4

Oppgave 1c (12%)

En tabell av lengde n skal sorteres med quickSort. De mange rekursive kallene i quickSort tar noe tid. Det blir foreslått å eliminere ett av de to rekursive kallene i metoden, ved å erstatte det med løkke.

- Forklar/skissér hvordan quickSort vil se ut om man følger denne idéen, gjerne med henvisning til den opprinnelige koden i læreboken.
- Ca. hvor mange rekursive instanser opprettes totalt i standard quickSort, dersom du er så heldig å splitte på midten hver gang, og gitt at man ikke bruker CUTOFF, dvs rekursjonen går helt «til bunns»? Forklar.
- Ca. hvor mange rekursive instanser blir opprettet av denne nye versjonen, igjen gitt at det splittes på midten hver gang, og at man ikke bruker CUTOFF? Forklar.
- Ca. hvor mange rekursive instanser blir opprettet i standard quickSort dersom man bruker CUTOFF = 8? Forklar.

Oppgave 2 – Grafanalyse (72%)

Vi antar at en rettet, uvektet graf er lagret på en tekstfil. Første linje inneholder ett tall: antall noder i grafen. Resten av filen består av linjer som hver inneholder to navn, henholdsvis *franode* og *tilnode*, og tolkes slik at det går en kant fra *franode* til *tilnode*. Gitt filen:

```
4
A B
A C
A D
B D
C D
D A
```

Oppgave 2a (8%)

Tegn denne grafen, og tegn grafen representert som naboskapsmatrise (adjacency matrix).

Videre skal vi programmere en klasse UvektetMatriseGraf for å analysere rettet, uvektet graf representert som naboskapsmatrise. Klassen skal bare ha to instansvariable: selve matrisen (2-dimensjonal tabell), og en HashMap som muliggjør kobling mellom nodenavn og tilhørende indeks i matrisen, slik at man effektivt kan oversette nodenavn til indeks. Matrisen skal inneholde heltall, 1 betyr at det *er* en kant, 0 betyr at det *ikke* er en kant.

Oppgave 2b (12%)

Lag klassens to instansvariable, og en konstruktørmetode. Konstruktørmetoden skal ha som innparameter navnet på en fil. Filen har format som forklart over, nodenavnene er separert med blank. Metoden skal åpne filen, lese antall noder, opprette matrise og HashMap, og lese inn grafen.

Oppgave 2c (10%)

Lag tilgangsmetodene

```
public int getIndeks(String node)
public void setKant(String fra, String til, boolean kant)
public int getKant(String fra, String til)
public boolean isKant(String fra, String til)
```

Metoden getIndeks skal returnere indkesen til node dersom den finnes, ellers kaste unntak. Dersom *fra* og *til* er lovlige (eksisterende) nodenavn, skal setKant sette inn 1 i aktuell posisjon dersom kant er true, 0 ellers. getKant skal returnere innholdet i aktuell posisjon som int, isKant skal returnere false dersom matrisa inneholder 0 i aktuell posisjon, true ellers.

Definér multiplikasjon mellom kvadratiske (høyde og bredde er lik) matriser slik: Resultatet skal bli en matrise av samme størrelse. Innholdet i posisjon [fra][til] regnes ut slik: Gå langs rad *fra* i første matrise, og langs kolonne *til* i andre matrise. For hver posisjon multipliseres de to verdiene. Summér alle produktene. $u = m1 * m2$ skal altså bety, uttrykt som formel:

$$u[f][t]=\sum_{i=0}^{n-1} m1[f][i]*m2[i][t]$$

-der *u* er utmatrise, *m1* og *m2* er matrisene vi skal multiplisere, og n er matrisestørrelsen.
Et eksempel: Vi skal multiplisere to 3x3 matriser, indeksert 0..2. For å f.eks. beregne utverdi i posisjon *fra=1*, *til=0*, ser vi på rad 1 (=*fra*) i m1 og kolonne 0 (=*til*) i m2. Første element multipliseres med første element, andre med andre, og tredje med tredje. De tre produktene summeres. Totalt: 2*1 + 0*2 + 1*1 = 2 + 0 + 1 = 3, som legges i [1][0] i utmatrisa til venstre.

$$\begin{bmatrix} x & x & x \\ 3 & x & x \\ x & x & x \end{bmatrix} = \begin{bmatrix} x & x & x \\ 2 & 0 & 1 \\ x & x & x \end{bmatrix} * \begin{bmatrix} 1 & x & x \\ 2 & x & x \\ 1 & x & x \end{bmatrix}$$

For å beregne hele utmatrisa, må dette gjentas for alle *f* (*fra*) og *t* (*til*).

Oppgave 2d (8%)

Lag metoden
`public static UvektaMatriseGraf mult(UvektaMatriseGraf m1, UvektaMatriseGraf m2)`
i class UvektaMatriseGraf. Den skal altså få inn to graf-matriser og legge produktet av de i en ny, og returnere den. Det nye objektet sin hashMap kan kopieres fra m1.
Hvilken orden vil denne algoritma ha?

Så stiller man seg (kanskje) spørsmålet: Hvilke noder kan man nå i presis to steg? Det viser seg at det kan leses ut av naboskapsmatrisa multiplisert med seg selv, m^2. Dersom man spør seg hvilke noder man kan nå på (presis) *p* steg, vil det kunne leses fra naboskapsmatrisa multiplisert med seg selv *p-1* ganger, m^p. Dersom m^p[fra][til] inneholder 0, er det ikke mulig å bevege seg fra *fra* til *til* på presis *p* steg. Dersom m^p[fra][til] > 0 er det mulig, og verdien m^p[fra][til] forteller hvor mange ulike stier som finnes!

Oppgave 2e (8%)

Lag metoden
`public UvektaMatriseGraf opphøyd(int p)`
i class UvektaMatriseGraf. Den skal opprette et nytt grafobjekt og sette det lik «dette»^{p}, dette matriseobjektet opphøyd i p. Den må gi rett svar for p>=1, og kaste unntak ellers.

Oppgave 2f (8%)

Er det mulig å gå fra *fra* til *til* på presis p steg? Lag metoden
`public boolean mulig(String fra, String til, int p)`
i class UvektaMatriseGraf. Den skal returnere true dersom det er mulig, false ellers.

Oppgave 2g (10%)

Implementér algoritma «korteste, uvekta sti» slik at den kan analysere en naboskapsmatrise. Du skal altså lage metoden

```
public Node[] kortesteSti(String startNode)
```

-der *Node* er:

```
public class Node{
  public Node(int l, int fn){
      lengde = l;
      forrigeNode = fn;
  }
  public int lengde;
  public int forrigeNode;
}
```

Metoden skal analysere grafen. Retur-tabellen skal kunne indekseres med node-indeks, og inneholde Node-objekter som forteller om nodenes lengde (avstand) fra startnoden, og hvilken node (indeks) som er forrige node i korteste sti.

Oppgave 2h (8%)

Lag metoden

```
public static String rapport(Node[] resultat, String sluttNode)
```

-som rapporterer hele sekvensen av node-*indekser* og avstander fra startNode til sluttNode. Den skal være rekursiv, og rapportere i rekkefølge fra startNode til sluttNode. Innparametre er resultatet fra oppgave 2g, samt ønsket sluttNode, returverdi er rapporten.

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 - Sortering (28%)

Oppgåve 1a (8%)

Du skal, i fylgjande ulike situasjonar, anbefale ei sorteringsalgoritme, og forklare kvifor du anbefaler denne.

Case 1) Fleire tusen enkeltarrayar, kvar med over over hundre tusen heiltal, skal sorterast så raskt som mogeleg. Innhaldet av kvar enkelt array er "godt blanda". Det er god plass i RAM til ekstra datastruktur, men vi veit ikkje noko om kor store tala kan vera. Det er her den totale køyretida for alle sorteringane som er viktigast.

Case 2) Same situasjon som i case 1), men kvar array inneheld nå berre omkring 10 heiltal.

Case 3) Eit epostsystem skal kunne sortere epostane etter ulike kriteriar som mottatt-dato, avsendar, tittel osv.

Case 4) Store tabellar, kvar med over over hundre tusen heiltal, skal sorterast så raskt som mogeleg. Det er kjent at tabellane er nesten sortert på førehand.

Oppgåve 1b (8%)

Utfør shellSort på fylgjande tabell med gap-sekvens 7, 3, 1. Vis tabellen for kvar gap-verdi.

2	9	4	5	7	3	1	8	3	5	6	9	0	2	4

Oppgåve 1c (12%)

Ein tabell av lengde n skal sorterast med quickSort. Alle dei rekursive kalla i quickSort tar noko tid. Det vert foreslått å eliminere eitt av dei to rekursive kalla i metoden, ved å erstatte det med løkke.

- Forklar/skissér korleis quickSort vil sjå ut om ein fylgjer denne idéen, gjerne med å vise til den opphavlege koden i læreboka.
- Ca. kor mange rekursive instansar opprettast totalt i standard quickSort, dersom du er så heldig å splitte på midten kvar gong, og gitt at ein ikkje bruker CUTOFF, dvs rekursjonen går heilt «til botnars»? Forklar.
- Ca. kor mange rekursive instansar vert oppretta av denne nye versjonen, igjen gitt at det splittast på midten kvar gong, og at ein ikkje bruker CUTOFF? Forklar.
- Ca. kor mange rekursive instansar vert oppretta i standard quickSort dersom ein bruker CUTOFF = 8? Forklar.

Oppgåve 2 – Grafanalyse (72%)

Vi antek at ein retta, uvekta graf er lagra på ei tekstfil. Første linje inneheld eitt tal: antal noder i grafen. Resten av fila består av liner som kvar inneheld to namn, *franode* og *tilnode*, og tolkast slik at det går ein kant fra *franode* til *tilnode*. Gitt fila:

```
4
A B
A C
A D
B D
C D
D A
```

Oppgåve 2a (8%)

Teikn denne grafen, og teikn grafen representert som naboskapsmatrise (adjacency matrix).

Vidare skal vi programmere ei klasse UvektaMatriseGraf for å analysere retta, uvekta graf representert som naboskapsmatrise. Klassa skal berre ha to instansvariable: sjølve matrisa (2-dimensjonal tabell), og ein HashMap som gjer det mogeleg å kople nodenamn og tilhøyrande indeks i matrisa, slik at ein effektivt kan oversette nodenamn til indeks. Matrisa skal innehalde heiltal, 1 betyr at det *er* ein kant, 0 betyr at det *ikkje* er ein kant.

Oppgåve 2b (12%)

Lag klassa si to instansvariable, og ein konstruktørmetode. Konstruktørmetoden skal ha som innparameter namnet på ei fil. Fila har format som forklart over, nodenamna er separert med blank. Metoden skal opne fila, lese antal noder, opprette matrise og HashMap, og lese inn grafen.

Oppgåve 2c (10%)

Lag tilgangsmetodane

```
public int getIndeks(String node)
public void setKant(String fra, String til, boolean kant)
public int getKant(String fra, String til)
public boolean isKant(String fra, String til)
```

Metoden getIndeks skal returnere indeksen til node dersom den finst, ellers kaste unntak. Dersom *fra* og *til* er lovlege (eksisterande) nodenamn, skal setKant sette inn 1 i aktuell posisjon dersom kant er true, 0 ellers. getKant skal returnere innhaldet i aktuell posisjon som int, isKant skal returnere false dersom matrisa inneheld 0 i aktuell posisjon, true ellers.

Definér multiplikasjon mellom kvadratiske (høgde og bredde er lik) matriser slik: Resultatet skal bli ei matrise av same storleik. Innhaldet i posisjon [fra][til] reknast ut slik: Gå langs rad *fra* i fyrste matrise, og langs kolonne *til* i andre matrise. For kvar posisjon multipliserast dei to verdiane. Summér alle produkta. $u = m1 * m2$ skal altså bety, uttrykt som formel:

$$u[f][t]=\sum_{i=0}^{n-1} m1[f][i]*m2[i][t]$$

-der *u* er utmatrise, *m1* og *m2* er matrisene vi skal multiplisere, og n er matrisestorleik.
Eit døme: Vi skal multiplisere to 3x3 matriser, indeksert 0..2. For å til dømes berekne utverdi i posisjon *fra=1*, *til=0*, ser vi på rad 1 (=*fra*) i m1 og kolonne 0 (=*til*) i m2. Fyrste element multipliserast med fyrste element, andre med andre, og tredje med tredje. De tre produkta summerast. Totalt: 2*1 + 0*2 + 1*1 = 2 + 0 + 1 = 3, som leggast i [1][0] i utmatrisa til venstre.

$$\begin{bmatrix} x & x & x \\ 3 & x & x \\ x & x & x \end{bmatrix} = \begin{bmatrix} x & x & x \\ 2 & 0 & 1 \\ x & x & x \end{bmatrix} * \begin{bmatrix} 1 & x & x \\ 2 & x & x \\ 1 & x & x \end{bmatrix}$$

For å berekne heile utmatrisa, må dette gjentakast for alle *f* (*fra*) og *t* (*til*).

Oppgåve 2d (8%)

Lag metoden
`public static UvektaMatriseGraf mult(UvektaMatriseGraf m1, UvektaMatriseGraf m2)`
i class UvektaMatriseGraf. Den skal altså få inn to graf-matriser og legge produktet av dei i ei ny, og returnere den. Det nye objektet si HashMap kan kopierast frå m1.
Kva for orden vil denne algoritma ha?

Så stiller ein seg (kanskje) spørsmålet: Kva for nokre noder kan ein nå i presis to steg? Det viser seg at det kan lesast ut av naboskapsmatrisa multiplisert med seg sjøl, m^2. Dersom ein spør seg kva for nokre noder man ein nå på (presis) *p* steg, vil det kunne lesast frå naboskapsmatrisa multiplisert med seg sjøl *p-1* gonger, m^p. Dersom m^p[fra][til] inneheld 0, er det ikkje mogeleg å flytte seg fra *fra* til *til* på presis *p* steg. Dersom m^p[fra][til] > 0 er det mogeleg, og verdien m^p[fra][til] fortel kor mange ulike stiar som finst!

Oppgåve 2e (8%)

Lag metoden
`public UvektaMatriseGraf opphøgd(int p)`
i class UvektaMatriseGraf. Den skal opprette eit nytt grafobjekt og sette det lik «dette»^{p}, dette matriseobjektet opphøgd i p. Den må gje rett svar for p>=1, og kaste unntak ellers.

Oppgåve 2f (8%)

Er det mogeleg å gå fra *fra* til *til* på presis p steg? Lag metoden
`public boolean mogeleg(String fra, String til, int p)`
i class UvektaMatriseGraf. Den skal returnere true dersom det er mogeleg, false ellers.

Oppgåve 2g (10%)

Implementér algoritma «kortaste, uvekta sti» slik at den kan analysere ei naboskapsmatrise. Du skal altså lage metoden

```
public Node[] kortasteSti(String startNode)
```

-der *Node* er:

```
public class Node{
  public Node(int l, int fn){
      lengde = l;
      forrigeNode = fn;
  }
  public int lengde;
  public int forrigeNode;
}
```

Metoden skal analysere grafen. Retur-tabellen skal kunne indekserast med node-indeks, og innehalde Node-objekt som fortel om noden sin lengde (avstand) frå startnoden, og kva for node (indeks) som er forrige node i kortaste sti.

Oppgåve 2h (8%)

Lag metoden

```
public String rapport(Node[] resultat, String sluttNode)
```

-som rapporterer heile sekvensen av node-*indeksar* og avstandar frå startNode til sluttNode. Den skal vera rekursiv, og rapportere i rekkefylgje fra startNode til sluttNode. Innparametre er resultatet frå oppgåve 2g, og den sluttNode ein ynskjer, returverdi er rapporten.

Lykke Til!