Algoritmer og Datastrukturer Høst 2018

Oblig 1

Frist: Fredag 14. september 2018 klokken 18:00

Oppdatert mandag 3. september 2018

Det skal leveres en fil (navn: **`Oblig1.java`**) som inneholder klassen **`public class Oblig1`** og i den skal de metodene det spørres etter i oppgavene, settes inn. De mer teoriaktige spørsmålene besvares i en kommentarsetning foran den tilhørende metoden. Pass på at metodene dine er testet og at de behandler alle spesialtilfellene korrekt (se testprogrammet på Canvas). Klassen skal være selvforsynt med metoder. Hvis du bruker en metode fra undervisningen, må den kopieres inn i klassen **`Oblig1`**.

Det er ok at det under arbeidet med å lage metodene legges inn utskrifts-setninger for testing i metodene. Men de SKAL FJERNES når obligen leveres inn. Klassen Oblig1 skal heller ikke ha noen main-metode når den leveres inn.

En gruppe på til og med 5 studenter kan levere en felles løsning. Filen **`Oblig1.java`** og resultatet av kjøring av testprogrammet (som en tekstfil) leveres inn på Canvas i en zip-fil. **De skal «pakkes ned» som en zip-fil**. Skriv navn og studentnummer i en fil som heter «lesmeg.txt». Hvis det er en gruppe som leverer, må det stå navn og studentnummer på alle i gruppen.

Det er et krav at gruppen bruker github for å lagre koden og dokumentere sitt arbeid. **Utskrift av git log / skjermbilde fra github skal vedlegges innleveringen for å vise at alle studenter har bidratt.**

Huskeliste før du sender inn løsningen:

- Står navn/studentnummer på alle i gruppen øverst på filen `Oblig1.java`?
- Har du laget filen «lesmeg.txt» med navn på alle på gruppen?
- Har du lagt ved resultatet av kjøring av testprogrammet (som en tekstfil)?
- Unngå å bruke norske tegn (æ, ø, å) i kildekode og kommentarer
- Er alle utskrifts-setninger inne i metodene fjernet?
- Er en eventuell main-metode inne i `class Oblig1` fjernet?
- Gir din IDE (NetBeans, Eclipse, IntelliJ eller hva annet du måtte bruke) noen advarsler (warnings)? Det må du unngå! Gjør de nødvendige endringene/tilleggene i koden din!
- Er alle metodene testet? De må passere testprogrammet før løsningen sendes inn.
- Har du tatt skjermbilde av git log på github og lagt ved?

OBS: Hvis det som leveres inn som 1. oblig ikke blir godkjent, vil en få det i retur med krav om at det må forbedres. Med andre ord får en flere sjanser.

Oppgave 1

I undervisningen har vi sett på ulike teknikker for å finne posisjonen til den største verdien i en tabell. I denne oppgaven skal det brukes en annen teknikk. Men siden denne teknikken endrer tabellen, er det selve verdien som skal returneres (og ikke posisjonen). Lag metoden `public static int maks(int[] a)`. Den skal finne og returnere den største **verdien** (ikke posisjonen) i parametertabellen *a*. Du skal bruke følgende teknikk:

1. Sammenlign først `a[0]` og `a[1]`. Hvis `a[0] > a[1]`, bytter de to verdiene plass (en ombytting).
2. Sammenlign så `a[1]` og `a[2]`. Hvis `a[1] > a[2]`, bytter de to plass.
3. Fortsett med `a[2]` og `a[3]` etc. helt til slutten av tabellen

Når denne prosessen er ferdig, vil tabellens største verdi ligge bakerst (sjekk at det stemmer). Dermed kan den verdien returneres. Det skal kastes en `NoSuchElementException` (med en passende tekst) hvis tabellen *a* er tom (har lengde 0). En tom tabell har ingen verdier og dermed ingen største verdi.

I en metode av typen *maks* er det først og fremst antall sammenligninger av tabellverdier vi er opptatt av. Hvor mange (som funksjon av *n*) blir det for en tabell med *n* verdier.

Men det har også interesse å finne antall ombyttinger. En ombytting er en relativt kostbar operasjon. Anta at tabellen *a* inneholder en tilfeldig permutasjon av tallene fra 1 til *n*.

- Når blir det flest ombyttinger?
- Når blir det færrest?
- Hvor mange blir det i gjennomsnitt?

For å finne svaret på det siste spørsmålet skal du lage en metode som bruker samme teknikk som maks-metoden, men den skal isteden telle opp ombyttingene. Antallet ombyttinger skal returneres. La den hete `public static int ombyttinger(int[] a)`. Da trenger du en hjelpevariabel som øker med 1 for hver ombytting og det er verdien til den som skal returneres. Lag tilfeldige permutasjoner av tallene fra 1 til *n* og bruk så metoden. På den måten kan du få en indikasjon på hvor mange det blir i gjennomsnitt (det finnes en formel for gjennomsnittet). Kan du på grunnlag av dette si om metoden *maks* er bedre (eller dårligere) enn de maks-metodene vi har sett på tidligere?

Oppgave 2

Lag metoden `public static int antallUlikeSortert(int[] a)`. Hvis *a* ikke er sortert stigende, skal det kastes en `IllegalStateException` (med en passende tekst). Tabellen *a* kan ha like verdier. Metoden skal returnere antallet forskjellige verdier i *a*. Hvis f.eks. *a* inneholder 3, 3, 4, 5, 5, 6, 7, 7, 7 og 8, skal metoden returnere 6 siden det er 6 forskjellige verdier. Metoden skal ***ikke*** endre noe på tabellens innhold. Pass på at hvis tabellen er tom (har lengde 0), skal metoden returnere 0 siden det er 0 forskjellige verdier i en tom tabell. Med andre ord er ikke en tom tabell en feilsituasjon.

Oppgave 3

Lag metoden `public static int antallUlikeUsortert(int[] a)`. Tabellen *a* kan nå være en hvilken som helst heltallstabell, dvs. den behøver ikke være sortert. Den kan også ha flere like verdier. Metoden skal finne og returnere antallet forskjellige verdier i *a*. Metoden skal ***ikke*** endre noe på tabellens innhold. Hvis *a* f.eks. inneholder tallene 5, 3, 7, 4, 3, 5, 7, 8, 6 og 7, skal metoden returnere 6 siden det er 6 forskjellige verdier. Pass på at hvis tabellen

er tom (har lengde 0), skal metoden returnere 0 siden det er 0 forskjellige verdier i en tom tabell. Metoden skal ***ikke bruke hjelpetabeller***. At arbeidet skal foregå innenfor tabellen *a*. Du kan selvfølgelig bruke en eller flere hjelpevariabler.

Oppgave 4

Lag metoden `public static void delsortering(int[] a)`. Den skal dele parametertabellen *a* i to sorterte deler. Venstre del skal inneholde oddetallene sortert og høyre del partallene sortert. Flg. eksempel viser hvordan den skal virke:

```
int[] a = {6,10,9,4,1,3,8,5,2,7};

delsortering(a);

System.out.println(Arrays.toString(a));

// Utskrift: [1, 3, 5, 7, 9, 2, 4, 6, 8, 10]
```

Tabellen *a* kan være tom (det er ingen feilsituasjon), inneholde både negative og positive tall, kun oddetall eller kun partall. ***Metoden skal ikke bruke hjelpetabeller***. Men en eller flere hjelpevariabler kan inngå. Lag den så effektiv som mulig.

Oppgave 5

Det kan være aktuelt å «rotere» elementene i en tabell. En rotasjon på én enhet gjøres ved at det siste elementet blir det første og alle de andre forskyves én enhet mot høyre.

A	B	C	D	E	F	G	H	I	J

Tabell 2 : Bokstavene fra A til I

J	A	B	C	D	E	F	G	H	I

Tabell 3 : Elementene i Tabell 2 forskjøvet én enhet

På figuren over har elementene i den første tabellen blitt «rotert» én enhet. Lag metoden `public static void rotasjon(char[] a)`. Den skal «rotere» innholdet i tabellen *a* én enhet. En rotasjon i en tom tabell eller i en tabell med nøyaktig ett element er ingen feilsituasjon. Men rotasjonen vil da ikke endre noe.

Oppgave 6

Her skal vi gå videre fra *Oppgave* 5. Hvis vi tenker oss at tabellen er «bøyd til en sirkel», er det mer naturlig å se på dette som en rotasjon. Dermed kan vi «rotere» et valgfritt antall enheter. Lag metoden `public static void rotasjon(char[] a, int k)` der *k* er et vilkårlig heltall. Hvis *k* = 1, skal metoden ha samme effekt som metoden i *Oppgave* 5. Hvis *k* er negativ, skal rotasjonen gå motsatt vei. En rotasjon i en tom tabell eller i en tabell med nøyaktig ett element er ingen feilsituasjon. Men rotasjonen vil da ikke endre noe. Det er ingen grense på størrelsen til *k*. Målet er å gjøre metoden så effektiv som mulig. Følgende programbit viser hvordan metoden skal virke:

```
char[] a = {'A','B','C','D','E','F','G','H','I','J'};

System.out.println(Arrays.toString(a));


rotasjon(a,3); System.out.println(Arrays.toString(a));

rotasjon(a,-2); System.out.println(Arrays.toString(a));


// Utskrift:

[A, B, C, D, E, F, G, H, I, J]   // originaltabellen

[H, I, J, A, B, C, D, E, F, G]   // en rotasjon på tre enheter mot høyre

[J, A, B, C, D, E, F, G, H, I]   // en rotasjon to enheter mot venstre
```

Oppgave 7

a) Lag metoden `public static String flett(String s, String t)`. Den skal «flette» sammen tegnstrengene *s* og *t* slik at resultatet blir en tegnstreng der annethvert tegn kommer fra *s* og annethvert fra *t*. Hvis *s* og *t* har ulik lengde, skal det som er «til overs» legges inn bakerst. Resultatet skal returneres. Flg. eksempel viser hvordan det skal virke:

```
String a = flett("ABC","DEFGH");

String b = flett("IJKLMN","OPQ");

String b = flett("","AB");

System.out.println(a + " " + b + " " + c);


// Utskrift: ADBECFGH IOJPKQLMN AB
```

b) Lag metoden `public static String flett(String... s)`. Den skal «flette» sammen tegnstrengene i *s*. Husk at *s* nå er en tabell av tegnstrenger. I koden vil derfor `s[0]` være første streng i tabellen *s*, osv. Flettingen skal være slik: Først hentes fortløpende det første tegnet fra hver tegnstreng, deretter fortløpende det andre tegnet, osv. De tegnstrengene som er «brukt opp», dvs. vi er ferdige med alle tegnene der, hoppes over. Resultatet skal returneres. Flg. eksempel viser hvordan den skal virke:

```
String a = flett("AM ","L","GEDS","ORATKRR","","R TRTE","IO","TGAUU");

System.out.println(a);
```

```
// Utskrift: ALGORITMER OG DATASTRUKTURER
```

Oppgave 8

Lag metoden **public static int[] *indekssortering*(int[] a)**. Den skal returnere en tabell med indekser til verdiene i tabellen *a* der *a* ***ikke*** skal endres. Se på. flg. eksempel:

```
int[] a = {6,10,16,11,7,12,3,9,8,5};

int[] indeks = indekssortering(a);

System.out.println(Arrays.toString(a));        // skriver ut a

System.out.println(Arrays.toString(indeks));  // skriver ut indeks

// Utskrift: [6, 10, 16, 11, 7, 12, 3, 9, 8, 5] a er ikke endret

// Utskrift: [6, 9, 0, 4, 8, 7, 1, 3, 5, 2]
```

Første verdi i indeks-tabellen, dvs. `indeks[0]`, skal være indeksen til den minste verdien i *a*. Vi ser av utskriften at det er 6 og det passer siden den minste verdien i *a* (dvs. 3) har indeks 6. Neste verdi i indeks-tabellen, dvs. `indeks[1]` skal være indeksen til den nest minste verdien i *a*. Utskriften gir at `indeks[1]` = 9 og den nest minste verdien i *a* (dvs. 5) har indeks 9. Osv. til den siste. Det er 2 som er indeksen til den største verdien (dvs. 16) i *a*.

På denne måten kan vi få ut verdiene i *a* i sortert rekkefølge ved å gå veien om indeks-tabellen. Dvs. slik

```
int[] a = {6,10,16,11,7,12,3,9,8,5};

int[] indeks = indekssortering(a);

for (int i = 0; i < a.length; i++) System.out.print(a[indeks[i]] + " ");

// Utskrift: 3 5 6 7 8 9 10 11 12 16
```

Dette kan løses på flere måter. Her blir det ikke lagt vekt på effektivitet, men kun på at det virker. Du kan også bruke hjelpetabeller - en eller flere. Opprett i hvert fall en tabell `indeks` av typen `int[]` med samme lengde som *a*. Det er den som til slutt skal returneres. Tabellen *a*kan inneholde like verdier! I så fall blir det ingen entydig løsning. Hvis f.eks. den minste verdien forkommer to ganger, spiller det ingen rolle hvem av dem `indeks[0]` refererer til. Men da må `indeks[1]` referere til den andre av de to. Hvis *a* er tom, skal også returtabellen være tom.

Oppgave 9

Lag metoden **public static int[] *tredjeMin*(int[] a)**. Den skal finne ***indeksene*** til de tre minste verdiene i tabellen *a*. Den skal returnere en tabell med tre verdier der første verdi skal være

indeksen til den minste verdien i *a*, andre verdi indeksen til den nest minste i *a* og tredje verdi indeksen til den tredje minste verdien i *a*. Bruk samme type idé som i *Programkode 1.2.5 a)*. Bruk tre hjelpevariabler for verdier og tre hjelpevariabler for indekser. Gi dem korrekte startverdier ved hjelp av metoden `indekssortering()` fra Oppgave 8. Den kaller du på en tabell som KUN består av de tre første verdiene i *a*. Du skal IKKE bruke `indekssortering()` på hele *a*. Det blir svært ineffektivt siden du da gjør en full sortering. I den siste versjonen av testprogrammet ligger det en tidstest på Oppgave 9. Hvis tabellen *a*har færre enn tre elementer, skal det kastes en `NoSuchElementException` sammen med en passende tekst. Metoden skal ikke endre noe på innholdet i *a*.

Oppgave 10

Vi sier at et ord er inneholdt i et annet ord hvis hver bokstav i det første ordet forekommer minst like mange ganger i det andre ordet som i det første, men ikke nødvendigvis i samme rekkefølge. F.eks. er ABBA inneholdt i både ABBABBA, BARAB, BARBARER og RABARBRA. ABBA har to A-er og to B-er og minst så mange av de to bokstavene har også de fire «ordene». Men ABBA er hverken inneholdt i BARBERER eller i AKROBAT. BARBERER har to B-er, men kun én A og AKROBAT har to A-er, men kun én B. Lag metoden `public static boolean inneholdt(String a, String b)` der *a* og *b* er «ord». Du kan ta som gitt at tegnstrengene *a* og *b* kun har store bokstaver (A − Å). Metoden skal returnere `true` hvis *a* er inneholdt i *b* og `false` ellers. Vi tenker oss her at et «ord» rett og slett er en oppramsing av bokstaver. Et «tomt» ord (en tom tegnstreng) er innholdt i alle andre ord (tegnstrenger). Det er ingen grense på hvor lange ordene kan være. Lag metoden så effektiv som mulig.