Laboration 7 – rekryteringsalgoritm

Denna laboration är en **parlaboration**. Du löser den i samarbete med en annan student (observera: inte större grupper än två och två). Ni ska båda kunna förklara lösningen och svara på handledarens frågor. Om du vill arbeta själv går det också bra.

Mål: Målet med laborationer är att lära sig mer om vektorer, både vektorer med primitiva typer och sådana med objekt. Dessutom tar den upp filläsning och parsning av filer, i samband med detta blir felsökning en viktig del.

Förberedelseuppgifter

- Läs avsnittet Bakgrund.

Bakgrund

Alla datorprogram innehåller algoritmer och många algoritmer är idag en integrerad del i våra liv. Men vi får inte glömma att alla algoritmer faktiskt konstrueras av människor. Eftersom Du kommer vara en viktig del av samhällsutvecklingen är det viktigt att du i framtiden tänker efter vid hantering av data, skapandet av algoritmer samt vad som krävs för att det ska vara möjligt att ersätta en människa med en dator.

Föreställ dig att du jobbar på IT-företaget Moogle som årligen får in massor av intresseanmälningar från studenter som söker arbete. Personalchefen på Moogle ger dig i uppdrag att skriva ett program med en algoritm som ska sålla ut de ansökningar som är värda att titta närmare på. Chefen hävdar att en sådan algoritm skulle spara pengar och minska belastningen på en ganska trött personalavdelning. Chefen hävdar också att genom att titta på snittbetyg sållar man effektivt fram de bästa kandidaterna. Håller du med honom?

Samtliga ansökningar kommer in via ett webgränssnitt som sparar data i textfiler. Den fil du ska bearbeta är filen som innehåller betyg för respektive person. Varje person har själv matat in sina betyg på, vad Moogle anser vara, de 5 viktigaste kurserna i datavetenskap. För enkelhets skull antar vi för denna uppgift att alla namn [3] är unika (men givetvis skulle användande av personnummer vara mer realistiskt). Filens format är som följer:

```
Förnamn Efternamn betyg1,betyg2,betyg3,betyg4,betyg5
```

Filerna har alltid en och endast en person per rad och betygsskalan är tänkt att vara som på LTH med stegen U, 3, 4, 5.

- betyg1 är betyget i Programmering grundkurs
- betyg2 är betyget i Programmering fördjupningskurs
- betyg3 är betyget i Databasteknik
- betyg4 är betyget i Objektorienterad Modellering och Design
- betyg5 är betyget i Realtidsprogrammering

Du kommer få i uppdrag att arbeta med olika filer längs uppgiftens gång men samtliga förväntas innehålla betyg och vara uppbyggda på samma sätt. Gå inte händelserna i förväg utan ta det steg för steg.

[3] Namnen i filerna är genererade och nästan ;-) slumpmässigt valda utifrån Sveriges 100 vanligaste efternamn samt manliga och kvinnliga förnamn. Källa: SCB, `http://www.scb.se/hitta-statistik/statistik-efter-amne/befolkning/amnesovergripande-statistik/namnstatistik/`, 2015. Inga verkliga studieresultat är en del av data i denna uppgift.

En del av uppgifterna innehåller frågor där du förväntas skriva ner ditt svar. Slarva inte med detta, svaren kan behövas till kommande uppgifter och den som rättar din lab kan mycket väl komma att kontrollera om du har svarat på frågorna.

Uppgifter

1. Öppna projektet *Lab07* och studera filen `Applicant.java` som är skriven för att kunna lagra data om en ansökande. Vad innehåller klassen för attribut? Vad ska skickas in till konstruktorn, när den anropas, och varför? Finns det något i klassen du måste implementera för att den ska bli komplett? (du ska inte implementera något nu, utan svara bara på frågorna)

   Svar: avgGrade måste implementeras

   Överkurs: `Applicant.java` innehåller något vi inte gått igenom i kursen, nämligen `implements Comparable<Applicant>`. Det är för att möjliggöra sortering med hjälp av javas inbyggda sorteringsmetoder men det behövs mer för att få sorteringen att fungera, mer om den senare.

2. Studera de andra två klasserna, `FindBestCandidates` och `FileReader`. Kan du se vad tanken med dessa klasser är? Kan du se att projektets tre klasser har en tydlig ”ansvarsfördelning”?

3. Öppna filen *applications_small.txt*, och titta på innehållet. Den innehåller några exempel på hur data kan se ut och det är denna filen vi ska arbeta med till att börja med.

4. Om den sökande har U på någon av kurserna kommer det att lagras som heltalsvärdet noll internt i vårt program. Kan du se att konstruktorn i klassen `Applicant` redan har kod som fixar detta? På vilka rader hittar du det som tolkar betygen och översätter dem till heltalsvärden? Vad heter attributet i vilket heltalsbetygen ligger lagrade?

   Svar: rad 26, grades

5. Titta vidare i klassen `Applicant`. Klassen har en metod som heter `getAvgGrade()` och som ska returnera medelbetyget för den sökande. Implementera metoden.

6. Nu ska vi gå vidare genom att skriva koden som läser in data från fil och skapar motsvarande `Applicant`-objekt för varje rad. Metoden `readFromFile` är förberedd i klassen `FileReader`. Metoden tar ett filnamn för att den behöver veta vilken fil som ska läsas samt ett heltal `nbrOfRows`. `nbrOfRows` anger hur många rader som max ska läsas ur filen.

```
/* Läser max nbrOfRows rader ur filen fileName.
   Om nbrOfRows är större än antalet rader läses alla rader.
   Resultatet returneras i som Applicant-objekt (ett per rad i filen)
   lagrade i en vektor med längden nbrOfRows.*/
static Applicant[] readFromFile(String fileName, int nbrOfRows);
```

   Implementera hela `readFromFile`-metoden. Tips:

   - Använd en while-loop som hela tiden kontrollerar om det finns fler rader. Se till att den kör max `nbrOfRows` gånger. Om `nbrOfRows` är större än antalet rader i filen ska helt enkelt alla rader som finns i filen läsas.
   - Det finns en färdig metod i klassen String som heter split. Den kan dela upp en sträng i flera vektorelement. Exempel:

```
String string = "Bo-Ali";
String[] parts = string.split("-");
String part1 = parts[0]; // Bo
String part2 = parts[1]; // Ali
```

7. Testa din `readFromFile`-metod genom att anropa den från `main`. Metoden är statisk (`static`) vilket betyder att du inte behöver skapa något objekt av klassen `FileReader` för att kunna anropa den. Du ska testa med filen *applications_small.txt* och den har 7 rader. Vill vi läsa in alla rader blir anropet därför:
`FileReader.readFromFile(''applications_small.txt'', 7);`
men glöm inte att också spara resultatet i en vektor för att kunna göra nästa uppgift.

8. Skriv, med `System.out`, ut alla sökande du har hittat i filen (dvs. skriv ut hela vektorn som du fick som resultat av anropet till `readFromFile`, ännu har vi ju inte implementerat filtreringen). Hur blir utskriften? Om den ser konstig ut, vad beror det på? När du skrivit ner ditt svar så gå till nästa uppgift.

   Svar: Den skriver ut referenserna till objekten

9. Gå till klassen `Applicant` och implementera metoden `toString` (ta bort kommentaren runt metoden). Metoden ska returnera en sträng på följande form:
`Rut Andreasson[5, 5, 5, 4, 3](avg: 4.4)`

10. Kör nu din main-metod igen. Ser utskriften bättre ut? Förbättra din `toString`-metod tills du är nöjd med utskriften.

11. Nu börjar det bli dags att göra själva urvalet. Till att börja med skriver vi inte en särskilt avancerad algoritm utan vi gör helt enkelt ett urval på de som har högst snittbetyg. Implementera metoden `findBestCandidates(Applicant[] applicants)` enligt beskrivningen i kodskelettet. I klassen finns konstanten `MIN_AVG_GRADE` som nu är satt till 4.0 och är tänkt att utgöra det snittbetyg man måste ha för att bli utvald. Det är ett absolut krav att du använder konstanten `MIN_AVG_GRADE` för att kontrollera vilka snittbetyg som är tillräckligt höga. Varför är det en bra idé att deklarera viktiga värden på ett ställe i koden och sen bara använda sig av konstant/variabelnamnet i resten av koden?

Svar: För då kan enkelt ändra värdet utan att ändra koden

12. Efter att du implementerat din `findBestCandidates`-metod ska du nu testa den. Bygg ut din mainmetod så att den anropar `findBestCandidates` och skriver ut resultatet av urvalet istället för att, som tidigare, skriva ut alla sökande som fanns i filen. Testa återigen på filen *applications_small.txt*, testa med olika värden på `MIN_AVG_GRADE` och jämför med värdena i filen. Verkar resultatet rimligt? Ändra och testa igen om det visar sig att du har buggar i din kod. Ändrar sig urvalet av sökande när du ändrar på `MIN_AVG_GRADE`? Vilket värde på konstanten hade du valt om du vore chef?

Svar: Fyra

13. Metoden `readFromFile` har ju en parameter som anger antalet rader som ska läsas. Prova att ändra ditt anrop till `FileReader.readFromFile(''applications_small.txt'', 20);`. Vad händer? Rätta till ditt program om detta inte fungerar. (det är inte nödvändigtvis så att `readFromFile` måste ändras, det kan också vara ok att göra ändringen där vektorn används, dvs. i klassen `FindBestCandidates`. Du väljer själv.) Vad ändrade du? Varför?

Svar: Null check i loopen för findBestcandidates

14. När ditt program nu fungerar för en liten mängd sökande ska vi se hur det reagerar på en annan mängd. Ändra din mainmetod så att den istället läser från filen *applications_x.txt*. Vad händer? Varför?

Svar: Den crashar pga av fel i datatypshantering. får in en char när den expectar siffra

15. Studera *applications_x.txt*. Varför är viss indata så konstig? Jo nu visar det sig att webgränssnittet i vilket data matats in är ganska undermåligt. Det ligger utanför din kontroll att ändra detta. Det har inte varit tydligt hur betygen ska anges och det har varit möjligt att helt enkelt mata in lite allt möjligt i fälten avsedda för betyg. Anteckna nedan: vilka problem finns det med indata och vilka är dina förslag på hur det skulle kunna lösas?

Svar: Bokstavsbetyg istället för siffor, -1 istället för U, färre än fem betyg, slår samma tangent två ggr

Skriva kod som tar hänsyn till detta

16. Ändra ditt program så att det slutar krascha för den data som finns i *applications_x.txt* (obs: du ska alltså absolut inte ändra i själva textfilen utan i koden som tolkar data). I din lösning är det ok om du tolkar alla felaktiga indata som betyget noll (du måste alltså inte implementera alla förslag du skrev ner ovan) och det är också ok att felaktiga siffror (tex 44) går in som riktiga betyg. Gör en mer ambitiös lösning om du vill men enda kravet för denna uppgiften är att ditt program inte kraschar när du försöker välja ut kandidater.

17. Provkör nu ditt program på ett lite större exempel. Använd istället filen *applications_all.txt*. Testa för lite olika värden på `MIN_AVG_GRADE` och se hur urvalet ändrar sig. När det är lite mer data ser vi plötsligt att det hade underlättat om resultaten skrevs ut i ordning. Lite senare i kursen ska ni öva på att skriva egna sorteringsalgoritmer men nu kan vi låta Java sortera vektorn åt oss. Om ditt urval ligger i en vektor som heter bestPersons kan du sortera denna vektor genom att anropa `Arrays.sort(bestPersons);` innan du skriver ut den. *Notera: Javas inbyggda sortering fungerar eftersom det finns en fungerande `compareTo`-metod i klassen `Applicant` samt att klassen implementerar `Comparable` (som tidigare nämnts). Detta gör det möjligt för sort-metoden att jämföra objekt med varandra men är överkurs för denna kursen.*

Har du några kommentarer eller reflektioner efter att ha sett det sorterade resultatet? Kvarstår några problem med konstig indata eller får alla en rättvis bedömning?

Svar: Personerna är sorterade enligt namn vilket innebär om vi bara tar in de tjugo första får v i bara in namn med bokstäver från början av alfabetet

Kommentar: Idag är det inte ovanligt att program måste hantera riktigt mycket data. Om textfilen vore riktigt stor hade vi fått problem med att lagra hela i en vektor i arbetsminnet, men hur vi skulle lösa det ligger utanför denna kursen.

18. Som vi redan sett innebåller *applications_x.txt* en del svåra fall. Kanske till och med svårare än vad det ser ut vid första anblicken. Öppna filen *applications_comments.txt* och läs kommentarerna om de svåra fallen. Efter att ha läst kommentarerna, har du några förslag på förbättringar till din urvalsalgoritm? Du behöver inte implementera alla, huvudsaken att du reflekterar.

    Svar: Betyg är ingen bra beskrivning av faktisk förståelse och för snabb urplockning av data kan ta bort rimliga kandidater. Låt inte algoritmer ta bort kandidater bara pga mänskliga faktorer. Titta alltid på datan.

19. **Frivillig uppgift:** Förbättra din kod på minst ett sätt, baserat på något av dina förslag i föregående uppgift. Testa och se om du tycker att det blir mer rättvist.

20. Har du tagit dig hit är det dags att redovisa. Se till att du minns vad du gjort och att du har svaren på frågorna redo. Många av er kanske ilsket tänker att det helt enkelt är utvecklaren av inmatningsgränssnittet som borde skärpa sig, men faktum är att detta speglar en ganska vanlig situation i arbetslivet: att indata inte är särskilt tillförltlig, förutsägbar eller helt enkelt bara inte riktigt som man trott.