## **Teoriuppgifter till lab 1**

**1.Vilka är rollerna vid parprogrammering och vilka uppgifter har varje roll?**

* Vid parprogrammering finns det två roller och de är föraren och navigatören. Föraren är den som faktiskt skriver koden medan navigatören ser till att undersöka, planera och komma upp med sätt för att komma fram till lösningar till problemen. Så navigatören blir helt enkelt hjärnan bakom arbetet och föraren implementerar det som navigatören har kommit på.

**2. Indexinformationen för ett ord (det vill säga i vilka teckenpositioner ordet förekommer i den stora texten) kan bli mycket stor. Hur bör positionerna lagras för att det ska bli effektivast, som text eller binärt (data streams i Java)? Bör indexinformationen lagras tillsammans med själva ordet eller på ett separat ställe?**

* Positionerna i index-filen är egentligen pekare som pekar till ett visst byte-nummer i den stora filen och bör därför lagras **binärt** för att det ska vara effektivt. Att lagra ett tal som text tar för mycket plats.
* Man kan nog göra detta på olika sätt. För att lagra index-informationen (positionerna för sökordet) effektivt så kan man lagra alla sökord i en fil tillsammans med en pekare som pekar på alla positioner för sökordet i en annan fil. Då kan man göra binärsökning för att hitta rätt ord i första filen och sedan läsa in alla positioner i den andra filen med hjälp av pekaren för positionerna (se exempel 1 nedan).

### *Exempel 1*

***Fil 1******Fil2***

*<sökord\_1> <positionpekare\_sökord\_1> <position\_1\_sökord\_1, …, position\_x\_sökord\_1>*

*<sökord\_2> <positionpekare\_sökord\_2> <position\_1\_sökord\_2, …, position\_y\_sökord\_2>*

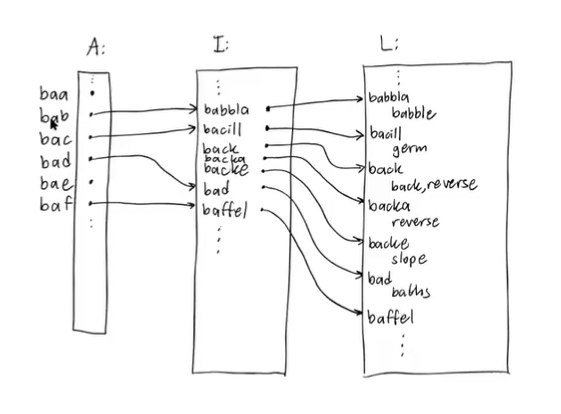
* Man kan även lagra positionerna tillsammans med sökordet. Om man väljer att lagra positionerna tillsammans med sökordet där varje rad innehåller sökordet och samtliga positioner till sökordet så kan det dock ta lite längre tid (se exempel 2 nedan). För varje rad vi läser in nu så läser vi även in alla positioner för sökordet men dessa behöver vi inte under binärsökningen när vi bara ska jämföra två ord. Ju mer data vi behöver läsa in desto längre tid tar det (detta behövde vi inte göra i exempel 1).

### *Exempel 2*

***Fil1***

*<sökord 1> <position\_1\_sökord\_1, …, position\_x\_sökord\_1>*

*<sökord 2> <position\_1\_sökord\_2, …, position\_y\_sökord\_2>*



L är en fil med alla orden i lexikon. I är en fil med alla sökord sorterade i bokstavsordning tillsammans med pekare (minnesadress) för respektive ord i L. A är en fil som innehåller de första tre hashade bokstäver av alla sökord.

**3. I denna labb ska datastrukturen för konkordansen huvudsakligen ligga på fil, vilket betyder att sökningar görs i filen istället för som vanligt i internminnet. Det påverkar till exempel hur man representerar pekare (lämpligen som bytenummer i filen). Diskutera för- och nackdelar med olika implementationer av konkordansen med avseende på följande egenskaper:**

* **snabbhet (antal filläsningar (Read) och filpositioneringar (Seek) per sökning),**
* **minneskomplexitet för fillagringen (bara konstant mycket internminne ska användas vid sökningen)**
* **enkelhet att konstruera och lagra på fil.**

**Ta åtminstone upp följande datastrukturer:**

* + **binärt sökträd:**

text…

* + **sorterad array:**

text…

* + **hashtabell:**

text…

* + **trie (träd där varje nivå motsvarar en bokstav i ordet):**

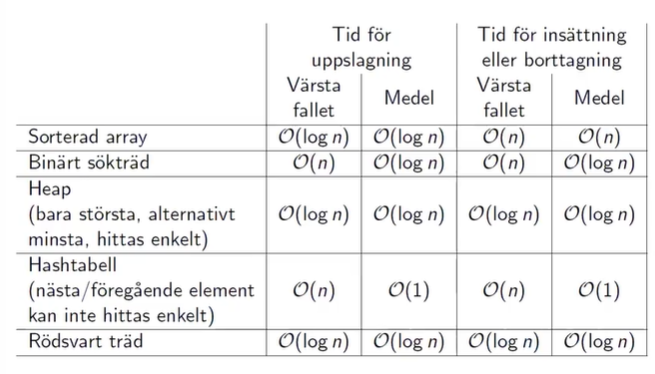
text…

* + **latmanshashning:**

text…

**Redovisa för- och nackdelarna i en tabell.**

| **Algoritm** | **Fördelar** | **Nackdelar** |
| --- | --- | --- |
| Binärt sökträd |  |  |
| Sorterad array |  |  |
| Hashtabell |  |  |
| Trie |  |  |
| Latmanshashning |  |  |

****

**4. Ge exempel på minst 7 indata (dvs ord) som är lämpliga testfall i labb 1 och motivera varför.**

**enbokstavsord, första eller sista ordet i korpusen**

1. “Amager” (Första ordet)
2. “mult” (Sista ordet)
3. “ö” (enbokstavsord)
4. “Vederslövssjön” (kollar ifall det spelar någon roll ifall ordet har stora och små bokstäver)
5. “vederslövssjön” (kollar ifall det spelar någon roll ifall ordet har stora och små bokstäver)
6. “den” (kommer att ge många träffar)
7. “Dragørs” (ord med specialtecken)
8. **Konstruera en hashfunktion för latmanshashning och motivera varför den fungerar. Hashfunktionen ska ta ett ord som indata och returnera ett unikt hashvärde (heltal) baserat på ordets tre första bokstäver.**

* Vi kan använda en hashfunktion som ska beskriva bokstäver i bas 30 och detta medför att varje bokstav ska få ett unikt värde. (29 bokstäver och bas 30 ger oss då olika värden) (Vi vill ha en hashfunktion som ger olika värden för olika ordningar)
* f[w[0]]\*30^2 + f[w[1]]\*30^1 + f[w[2]]\*30^0

## Uppgift 2

Positionerna i index-filen är egentligen pekare som pekar till ett visst byte-nummer i den stora filen och bör därför lagras **binärt** för att det ska vara effektivt. Att lagra ett tal som text tar för mycket plats.

Man kan nog göra detta på olika sätt. För att lagra index-informationen (positionerna för sökordet) effektivt så kan man lagra alla sökord i en fil tillsammans med en pekare som pekar på alla positioner för sökordet i en annan fil. Då kan man göra binärsökning för att hitta rätt ord i första filen och sedan läsa in alla positioner i den andra filen med hjälp av pekaren för positionerna (se exempel 1 nedan).

### *Exempel 1*

***Fil 1******Fil2***

*<sökord\_1> <positionpekare\_sökord\_1> <position\_1\_sökord\_1, …, position\_x\_sökord\_1>*

*<sökord\_2> <positionpekare\_sökord\_2> <position\_1\_sökord\_2, …, position\_y\_sökord\_2>*

Man kan även lagra positionerna tillsammans med sökordet. Om man väljer att lagra positionerna tillsammans med sökordet där varje rad innehåller sökordet och samtliga positioner till sökordet så kan det dock ta lite längre tid (se exempel 2 nedan). För varje rad vi läser in nu så läser vi även in alla positioner för sökordet men dessa behöver vi inte under binärsökningen när vi bara ska jämföra två ord. Ju mer data vi behöver läsa in desto längre tid tar det (detta behövde vi inte göra i exempel 1).

### *Exempel 2*

***Fil1***

*<sökord 1> <position\_1\_sökord\_1, …, position\_x\_sökord\_1>*

*<sökord 2> <position\_1\_sökord\_2, …, position\_y\_sökord\_2>*

Hej Dawood!

Jag vet inte vad Viggo har sagt på föreläsningen,

men jag tror att en del studenter kan försöka

lagra tal som strängar, tex 123 som strängen

"123" istället för som en int och det tar alldeles

för mycket plats.

Det är ett lätt misstag att göra om man inte har

programmerat och är van att tänka på data som

det man läser på papper.

Det Viggo menar är att du bör använda en

datatyp som tex int för att lagra varje index.

Då består filen med index av \*binärt\* data som

du inte kan titta på i en editor, men du kan läsa

den i ditt program om du använder rätt bibliotek.

Jag hoppas att det är svar på frågan.

Vänliga hälsningar,