1. Vilka är rollerna vid parprogrammering och vilka uppgifter har varje roll?  
   Förare och navigatör. Föraren är den som skriver koden medan navigatören är den som sitter på sidan av och tittar och kommer med anmärkningar/råd till förbättringar för vad den andra personen gör. Föraren och navigatören bör med fördel växla roller hyfsat frekvent.
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?  
   Indexinformationen lagras antagligen helst binärt eftersom man slipper det extra utrymme som det innebär med strängar. Indexinformationen kan med fördel lagras separat från själva ordet – det vill säga att man har ett index med ord och ett med positioner. Detta gör det mer effektivt när man ska söka efter ord i indexet, eftersom det är mindre information att gå igenom.
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. Diskutera för- och nackdelar med olika implementationer av konkordansen med avseende på följande egenskaper:
   * snabbhet (antal filläsningar och filpositioneringar per sökning),
   * minneskomplexitet för fillagringen (bara konstant mycket internminne ska användas)
   * enkelhet att konstruera och lagra på fil.

|  |  |  |
| --- | --- | --- |
|  | Fördelar | Nackdelar |
| * + binärt sökträd | Sökning = O(lg n) med ett balanserat träd, O(n) med ett obalanserat. Samma gäller för insättning. | Svårt att lagra och läsa från fil. |
| sorterad array | Sökning = O(lg n) med hjälp av binärsökning. Enkelt att både sortera och lagra, eftersom man i stort sett bara kan lagra informationen efter varandra med en avskiljare mellan. | Inte dynamisk. Minneskomplexitet O(n). |
| hashtabell | Sökning i en hashtabell är i vanliga fall O(1), om varje värde har en unik nyckel. Insättning har samma tidskomplexitet som sökning. | Kan bli kollisioner, vilket kan öka komplexiteten för sökning och insättning till O(n). |
| trie | Sparar minne genom att kunna lagra ord som börjar likadant tillsammans, som ”glad” och ”gladast. Sökning är i värsta fall O(m), där m är längden på ordet man söker. | Det finns inte ett självklart sätt att vare sig lagra eller läsa in en trie från en fil. De behöver också lagras char för char. |
| * + latmanshashning | Sökningen är effektiv, eftersom man kommer till ett reducerat index och kan använda sig av binärsökning, först i det stora indexet och sedan det mindre. | Fördelen här är att man kan lagra så lite information som möjligt på själva disken. Flera abstraktionsnivåer bidrar till effektivare lagring. |

Ta åtminstone upp följande datastrukturer:

* + binärt sökträd,
  + sorterad array,
  + hashtabell,
  + trie (träd där varje nivå motsvarar en bokstav i ordet),
  + latmanshashning

Redovisa för- och nackdelarna i en tabell.

Arbeta gärna ihop med labbteoriuppgifterna, men var och en ska av administrativa skäl ta med en egen skriftlig lösning med namn på som lämnas in vid redovisningen. Skriv gärna för hand!