20
Filer og indekser.

Filbegrebet.
Sekventielle og direkte filer.
Filer i Eiffel.
Eksternt dataområde for objekter.
Lageradminstration og postlængder.
Indeksbegrebet.
Konsekutiv, kædet og vilkårlig organisering.
Multisammenkædning af poster.
B+ træer.
Tries.

PS1 -- Filer og indekser

© Kurt Nørmark, Aalborg Universitetscenter, 1994.

Noter

Introduktion til filer.

En fil er en *persistent* datastruktur.

Sekventiel fil.

- En liste af poster med *sekventiel tilgang*.
- Sekventiel tilgang:
  - Læse næste post.
  - Skrive næste post.
  - Spole tilbage til begyndelsen.

Fil med direkte tilgang.

- En liste af poster med *direkte tilgang*.
- Direkte tilgang ala arrays.

Filer er et vigtigt begreb i operativsystemer.
I programmeringssprog er filer en abstrakt datatype, som udgør et interface til et bestemt filbegreb i et operativsystem.

© Kurt Nørmark, Aalborg Universitetscenter, 1994.

Noter

En *persistent* struktur er mere vedvarende og permanent eksisterende end indholdet i datamaskinens arbejdslager. Pladelagre og magnetbånd er vigtige eksempler på datamedier med information, som vi opfatter som persistent.

Sekventielle filer var "i gamle dage" lagrede på magnetbånd. De fysiske egenskaber ved dette medium afleder direkte det temmeligt restriktive repertoire af operationer på sekventielle filer. I mange sammenhænge har filer på diske overtaget den sekventielle natur, som vi er påtvungne, når vi arbejder med bånd.

Når vi arbejder med filer på diske, kan det dog lade sig gøre at få direkte tilgang til data.

Filbegreber.

- En *blok*.
  - Den mindste mængde information, som fysisk kan transporteres mellem en fil og arbejdslageret.
- En *post*.
  - Logisk helhed af information, som lagres på en fil.
  - En post inddeles i *felter*.
    - En *nøgle*: Et felt, hvis værdi entydigt indentificerer en post på en fil.
      - En *primær nøgle*: En nøgle som bestemmer den indbyrdes ordning af posterne på en fil.
  - Typisk flere poster pr. blok.
- En *tekstfil*.
  - En fil med information, der er struktureret som en liste af tegn (bytes).
  - Filer i Unix er tekstfiler.
  - En streng af sammenhængende tegn på en tekstfil kan opfattes som en post.

PS1 -- Filer og indekser

© Kurt Nørmark, Aalborg Universitetscenter, 1994.

Noter

Vi vælger betegnelsen 'en post' for en logisk sammenhængende mængde af information på en fil. Alternative betegnelser kunne være 'en record' eller 'et objekt', men både record- og objekt-betegnelsen reserverer vi til interne data i arbejdslageret. Anvendelsen af en særlig terminologi, når vi taler om data på filer, hænger sammen med, at vi som regel skal være meget bevidste om transporten af data til og fra filer, ligesom vi skal tage de meget lange køretider i betragtning, når vi manipulerer fildata.

Det er nogle sammenhænge attraktivt at tale om persistente objekter, og om objekt-orienterede databaser. Med dette abstraherer vi fil-begrebet bort, og vi lader det være en egenskab ved objekterne at de er lagret på et persistent medium. Der er ganske store udfordringer i at udvide en objekt-model til at omfatte en egentlig objekt-orienteret database.

Filer i Eiffel.

FILE er en abstrakt klasse, som beskriver generelle egenskaber ved sekventielle filer.

UNIX_FILE er en specialisering af FILE, som spejler UNIX's filbegreb ind i en Eiffel klasse.

Et objekt af typen UNIX_FILE er tillige en fil med direkte tilgang.

Oversigt over operationer i UNIX_FILE:

Åbning og lukning.

Som creation-procedurer, og som operationer på eksisterende fil-objekt.

Læsning og skrivning af integers, booleans, reals, doubles, characters og strings fra/til tekstuel repræsentation.

Tilgang til en række fil-egenskaber, eksempelvis beskyttelse og ejerskab.

Direkte-fil navigeringsoperationer.

Back, finish, start, forth, go(abs_pos), move(rel_pos), ...

PS1 -- Filer og indekser

© Kurt Nørmark, Aalborg Universitetscenter, 1994.

Noter

I forbindelse med input-output har vi tidligere i disse noter set på Eiffel klassen STD_FILES og FILE (se kapitlet "Kommandoer, Udtryk og Input Output i Eiffel).

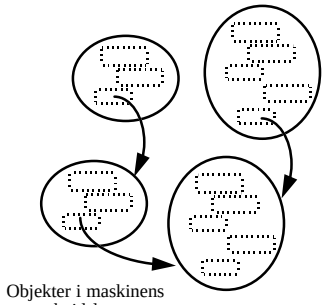
Som det ses ovenfor, er FILE i Eiffel3 en abstrakt klasse. Hvis man konkret skal bruge en fil, skal man lave en instans af UNIX_FILE. Der henvises til afsnit 23.12 i "Eiffel the Libraries" for en beskrivelse af klientinterfacet af UNIX_FILE. I on-line brug af Eiffelomgivelsen kan man naturligvis også lave en flat-short beskrivelse af UNIX_FILE.

Et objekt af typen UNIX_FILE er en filbeskrivelse, som kan knyttes til en navngiven fil ude i operativsystemet. Der er forskellige creation procedurer i UNIX_FILE, som lader os åbne en operativsystemfil på forskellige måder.

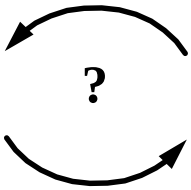
Det er værd at bemærke, at UNIX_FILE er mere end en sekventiel fil: Der findes operationer som gør en UNIX_FILE til en fil med direkte tilgang. Der henvises til operationerne i feature gruppen "cursor movement" i UNIX_FILE.

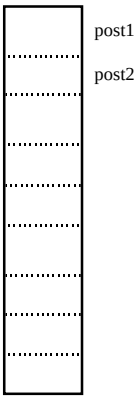
Abstraktion af direkte filer til “Eksternt Dataområde” (1).

Problemet:



Objekter i maskinens arbejdslager





En direkte fil som indeholder bitmønstre eller tegn

Hvordan gemmer man et eksisterende objekt som en post på en fil?

Hvilke felter? Hvordan håndteres referencer?

Hvordan etablerer man et objekt ud fra en post (af tegn eller bits) på en fil?

PS1 -- Filer og indekser

© Kurt Nørmark, Aalborg Universitetscenter, 1994.

Noter

På denne slide nøjes vi med at identificere problemet, der består i at slo bro mellem

> på den ene side objekter, som er instanser af klasser, og hvor imellem der kan være referencer

og

> på den anden side poster på en fil, hvor en post blot er repræsenteret som tegn (bytes) eller bits (afhængig af det valgte abstrationsniveau).

På næste side vil vi givet et bud på en løsning på problemet.

Abstraktion af direkte filer til “Eksternt Dataområde” (2).

Direkte filer abstraheres op til et *objekt-lager*.

Objekter af forskellige typer har operationer, som transformerer disse til og fra tekststrenge.

```
class OBJECT_STORE
      [T -> STORAGE_ELEMENT]
feature
  Create(F: FILE)
    -- opretter et objektlager på filen F.
  Put(x:T)
    -- Gemmer x i en udvalgt post i lageret.
  Last_put_index: Integer
    -- Returnerer positionen af sidste put.
  Get(i: integer): T
    -- Returnerer objektet, som er gemt på post i.
  Delete(i: integer)
    -- Sletter objektet fra lageret, som er gemt i den i-te post.
end
```

```
deferred class STORAGE_ELEMENT
feature
  Encode: STRING
    -- indkoder et nuværende objekt til en
    -- streng af længde Encode_count.
  Decode(s: STRING)
    -- afkod s og initialiser det nuværende
    -- objekt med resultatet af afkodningen.
  Encode_count: integer
    -- antallet af tegn i indkodningen af
    -- dette objekt (denne type).
end
```


© Kurt Nørmark, Aalborg Universitetscenter, 1994.

Noter

Objekter, som vi ønsker at gemme i et objekt-lager, skal være instanser af en klasse, der har STORAGE_ELEMENT som superklasse. Det betyder, at sådanne objekter kan ind- og afkodes ved brug af operationerne Encode og Decode. I stedet for at arbejde direkte på filer, gør vi vores applikation til klient af en eller flere OBJECT_STOREs. Et objekt-lager parametriseres med den type af objekter, der kan gemmes på filen (hvilket, som det fremgår, skal være en specialisering af STORAGE_ELEMENT).

Vi lægger i det ovenstående op til, at encode afleverer en tekststreng. Det kunne naturligvis også være en anden type, f.eks. en BITS type.

I Eiffel er det forbundet med vanskeligheder at instantiere et objekt af typen T i klassen OBJECT_STORAGE. Årsagen er, at T er en typeparameter til klassen. Man kan f.eks. arbejde sig uden om dette problem ved at ændre signaturen af get til følgende:

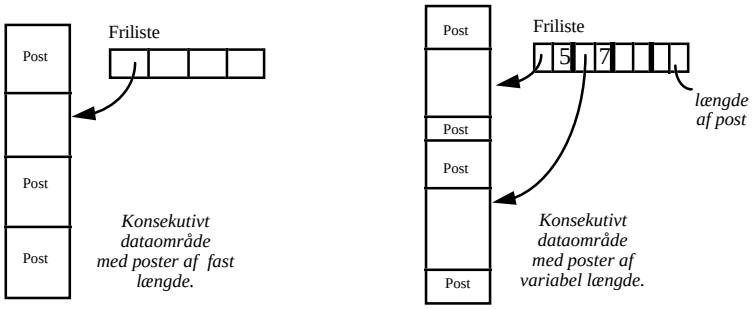
```
get(i: Integer;  res: T)
```

Når get kaldes overfører man et (gerne uinitialiseret) objekt af typen T som den anden parameter til get. Det er så get’s opgave at initialisere dette objekt (kopiere indhold over i objektet).

Bemærk, at det er encodes opgave at udvælge hvilke felter der skal lages i en post på en fil. Ultimativt er det kun de simple typer (tal, tegn, strenge af disse mv.) der overhovedet kan lagres i felter. I forbindelse med udvælgelsen kan encode vælge at følge referencer til andre objekter. Det giver aldrig mening at lagre referencer direkte på en fil.

Lageradminstration.

- **Fastlængde poster**.
  - Friliste med huller.
  - Vilkårlige data kan udfylde et hul.
  - Relativt simpelt.
- **Variabel længde poster**.
  - Risiko for fragmentering.
  - Sammenpresning kan blive nødvendig.
  - Kompliceret.



PS1 -- Filer og indekser

© Kurt Nørmark, Aalborg Universitetscenter, 1994.

Noter

På denne slide ser vi på en vigtig egenskab ved poster på en data-fil.

Et hul i datafilen kan fremkomme ved, at vi har slettet en eksisterende post "i midten af filen". Det er i dette tilfælde oplagt at holde styr på sådanne huller i en friliste. Næste gang vi har behov for at skrive en post på filen kan vi (afhængig af evt. krav til indbyrdes rækkefølge af posterne) vælge at udfylde hullet, som angivet af frilisten.

Hvis posterne er af variabel længde kan der opstå huller af varierende længde. Nabohuller kan naturligvis slåes sammen til større huller. Man kan også vælge en gang i mellem at "feje med den store kost", således at alle poster skubbes hen i forenden af filen; derved ændres alle post-adresser, hvilket betyder, at alle indekser skal modificeres (se senere om indekser til datafiler). Alt dette kan blive meget komplekst at administrere.

Indekser.

Et indeks er en funktion, som afbilder en værdi til en fil-adresse på en post, som indeholder værdien i en bestemt af postens felter.

Indeks: felt-værdi -> fil-adresse

- Formålet med at have et indeks er
  - at kunne tilgå data i en fil hurtigere end ved sekventiel søgning gennem filens poster.
  - at kunne tilgå data i en rækkefølge bestemt af indekset.
- Det kan være hensigtsmæssig at kunne tilgå data via forskellige datafelter:
  - Primær og sekundære nøgler: Identificerer en post på filen entydigt. Giver primær og sekundære indekser.
  - Andre datafelter: Mange poster med samme nøgle.
- Tilgang til data:
  - Simpel tilgang: data med netop én nøgle eller ét felt ønskes.
  - Interval tilgang: data mellem to felt-værdier ønskes.
- Omfang af indeks:
  - *Tæt*: Ét element i indekset for hver post.
  - *Tynd*: Flere poster for hvert element i indekset.
- Indekset kan enten være lagret internt i arbejdslager eller være lagret eksternt på disk.

PS1 -- Filer og indekser

© Kurt Nørmark, Aalborg Universitetscenter, 1994.

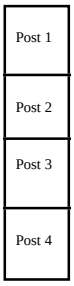
Noter

Vi ser, at et indeks opfattes som en funktion, der afbilder en eller anden feltværdi til den adresse i filen, hvor den relevante post er lagret. Vi vil ofte være udsat for, at denne funktion er repræsenteret som en tabel med par af formen (felt-værdi, fil-adresse).

I det realistiske tilfælde er det ikke blot filen, men også indekstabellen der skal lagres på filer. Det vil sige, at det ikke er tilstrækkeligt at have indekset som en tabel i arbejdslageret, som på en eller anden måde etableres ved åbning af datafilen. Efter hver ændring af data-filen, skal indekset (på en anden fil) opdateres.

Konsekutivt organiserede poster.

Konsekutiv datafil

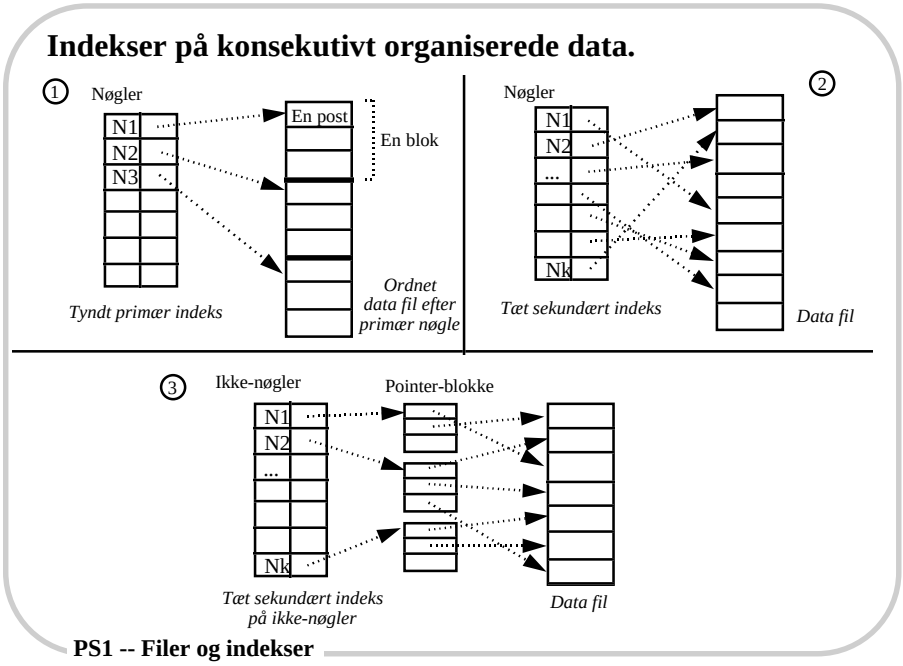


- Sekventielt ordnet og **konsekutiv** efter primær nøgle:
  - Indsættelse og sletning er kostbar idet poster typisk skal “skubbes”.
  - Lineært gennemløb efter primær nøgle kan foretages direkte på datafilen.

© Kurt Nørmark, Aalborg Universitetscenter, 1994.

Noter

Ovenstående er en af de mest oplagte måder at organisere poster på. Posterne er ordnet efter det vi kalder den primære nøgle. Det betyder at vi kan gennemløbe posterne i rækkefølge, uden videre. Det betyder desværre også, at det er meget kostbart at tilføje nye poster, eller at slette eksisterende, således at ordningen og konsekutiviteten opretholdes. Derfor er denne organisering mest brugbar, hvis vores data er forholdsvis uforanderlige (der sker sjældne tilføjelser eller sletninger af poster).



© Kurt Nørmark, Aalborg Universitetscenter, 1994.

Noter

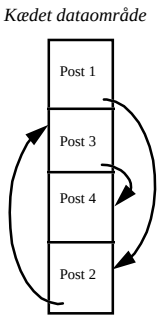
Ovenfor er vist forskellige indekser, som er tabellagt. I hver indgang i tabellen repræsenteres en nøgle (eller et felt) og denne nøgles filadresse. For at kunne søge effektivt vil vi antage, at der findes en total ordning på nøglerne, og at der er direkte tilgang til tabellens indgange. Dermed er forudsætningerne opfyldt for at kunne lave binær søgning efter en nøgle i indekset.

I situation 1 er indekset tyndt, i og med at der kun er én nøgle i indekset pr. blok. Når man skal finde frem til en post må man derfor søge efter posten i en blok. Dette er en pragmatisk fornuftig strategi, idet vi jo under alle omstændigheder skal hente en hel blok fra disken. Hvis ikke vi har styr på blokke gennem vores fil-begreb, er denne organisering af indekset nok mindre oplagt.

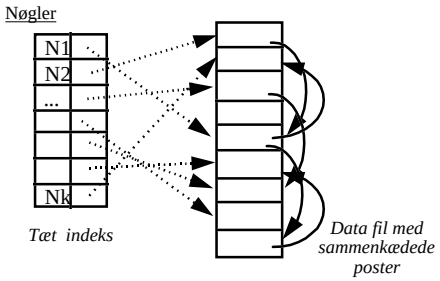
I situation 2 vises et tæt indeks, hvor der jo er netop én nøgle pr. post i datafilen. Bemærk at vi kan skippe ideen om konsekutiv organisering og ideen om primære nøgler til fordel for et tæt indeks ala 2, og en vilkårlig organisering af poster på datafilen. Mere om dette senere.

Situation 3 illustrerer et indeks på et felt, som ikke er en nøgle. Problemet er her, at der for en givet feltværdi kan være flere poster, som indeholder denne værdi. Derfor har vi introduceret en mellemniveau af såkaldte “pointer-blokke”, der refererer til de relevante poster. Bemærk, at en pointer-blok kan løbe fuld, og at vi måske derfor har behov for en eller anden sammenkædning også af pointer-blokkene. Dette kan let blive ganske kompliceret.

Sammenkædede poster.



- Sekventiel ordnet og **sammenkædet:**
  - Lineært gennemløb efter primær nøgle kan foretages direkte, ved at følge kæden.
  - Direkte tilgang kan kun ske gennem et evt. indeks.
  - Forholdsvis omstændeligt at sammenkæde data på en fil.



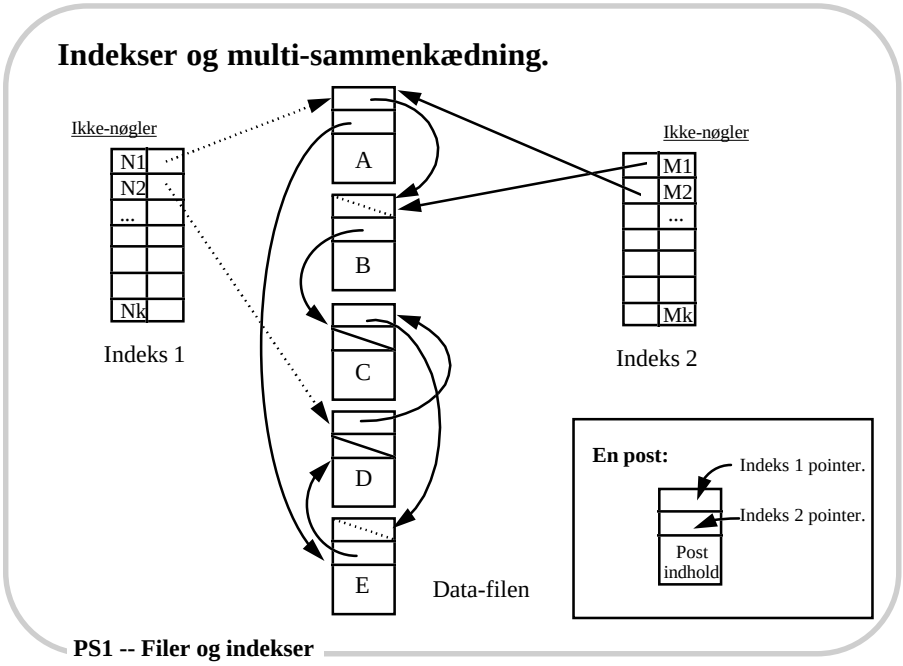
PS1 -- Filer og indekser

© Kurt Nørmark, Aalborg Universitetscenter, 1994.

Noter

Vil illustrerer her et alternativ til den konsekutive ordning. Vi har stadig en bestemt ordning af posterne på datafilen, men dette er repræsenteret anderledes, nemlig ved en sammenkædning af poster (med pointere, som er adresser på poster).

Nederst til højre illustrerer vi et tæt indeks på en sammenkædet datafil. Som observeret ovenfor er der brug for indekset for at at skaffe sig effektiv adgang til data på filen. Indekset kan være et primært indeks, i hvilket tilfælde den kædede orden på datafilen skal være identisk med den konsekutive orden på nøglerne i indekset. Ret beset er det nok lidt underligt at repræsentere denne orden begge steder, og vi kunne uden tab af muligheder droppe sammenkædningen af posterne. Indekset kan naturligvis også være et sekundært indeks.



© Kurt Nørmark, Aalborg Universitetscenter, 1994.

Noter

Vi studerer her indekser på felter, som ikke er nøgler. Vi har behov for at kunne associere flere poster til én feltværdi i indekset. I stedet for at have et indirekte niveau af pointer-blokke (som tidligere diskuteret) klarerer vi os med at sammenkæde poster, som har samme feltværdi i et indeks. Når vi har behov for to eller flere sådanne indekser (på ikke-nøgler) skal vi administrere multiple sammenkædninger af poster, som vist ovenfor.

Fra indeks 1 udgår stiplede pointere. Feltværdi N1 findes i post A og B, hvorfor disse er kædet sammen. Feltværdi N2 findes i D, C og E.

Fra indeks 2 udgår fuldt optrukne pointere. Feltværdi M2 findes i post A, E og D. Feltværdi M1 findes i post B og C.

Læsere henvises til afsnit 10.3.3 af Horiwitz og Sahni's bog *Fundamentals og Data Structures* for flere detaljer om multikædning. En tilsvarende ide for interne datastrukturer er iøvrigt diskuteret i Weiss i afsnit 3.2.7 (side 57).

Som det klart fremgår af illustrationen, foregår kædningen blandt selve posterne i datafilen. Man kan naturligvis også forestille sig, at denne kædning håndteres i de forskellige indekser (indekstabeller). I Horowitz og Sahni diskuteres dette kort under betegnelsen *inverterede filer*. Der henvises til afsnit 10.3.4 i denne bog.

Indicering på virkårligt organiserede poster.

- Vilkårligt organiserede poster på en datafil:
  - Indsættelse og sletning er let.
  - Lineært gennemløb skal ske gennem indeks.
  - Intervalforespørgsler vanskeliggøres.

- Indiceringsteknikker på virkårligt organiserede poster.
  - Direkte adressering.
    - Der er et felt, hvis værdi anvendes som postnummer på den direkte fil.
    - Risiko for dårlig udnyttelse af posterne i datafilen.
  - Hashing.
    - Postnummeret på filen bestemmes af en hashfunktion.
    - Kollisionshåndtering skal være effektiv i forhold til lagermediet.
  - Brug af eksplicitte indekser som ved konsekutiv og kædet organisering.

© Kurt Nørmark, Aalborg Universitetscenter, 1994.

Noter

Når vi taler om vilkårligt organiserede poster på en datafil tillægger vi ikke den indbyrdes ordning af posterne nogen som helst betydning. Dette er stærk modsætning til den konsekutive ordning (efter primær nøgle). Det er ikke muligt fra én post at komme til "den næste" relativ til en ordning af disse. Dermed er den vilkårlige organisering også anderledes end en sammenkædet organisering af posterne.

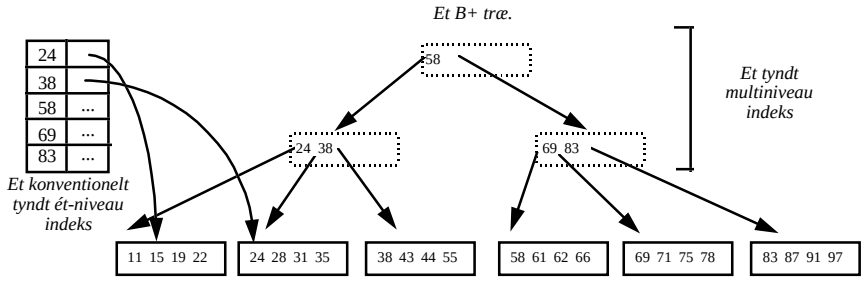
Ved direkte adressering findes der et felt, hvis værdier vi bruger direkte som en post-adresse på filen. Dette er meget simpelt, men det kræver omhyggeligt valgte feltværdier (uden for stor spredning). Hvis man vælger direkte adressering skal man være opmærksom på, at man har introduceret en stærk binding mellem en feltværdi og den fysiske placering af poster på filen. Som oftest ønsker vi ikke en sådan begrænsning. Vi foretrækker at have frit spil ved valg af datafelter i vore objekter og poster (dog således at nogle felter er udnævnt til nøgler). Den fysiske placering overlades således til f.eks. en hashfunktion eller til et eksplit håndteret indeks.

Indicering med B+ træer.

De indre knuder i et B+ træ er et *multi-niveau indeks*.

På bekostning af ekstra plads i knuderne undgår man at skulle skubbe store datamængder i indekstabeller.

Bladene i et B+ træ repræsenterer en "side-opdelt" datafil (nøgler og referencer til et dataområde).



PS1 -- Filer og indekser

© Kurt Nørmark, Aalborg Universitetscenter, 1994.

Noter

De indekser, vi hidtil har studeret, har kun haft ét niveau. Med andre ord, givet en nøgle kan vi via indekset få hånd på en fil-adresse på en post. I et multi-niveau indeks kan en nøgle afbildes i et nyt indeks (et underindeks). B+ træer er en særlig afart af et multi-niveau indeks, som igen kan betragtes som en variation af B-træer. Vi har allerede studeret B-træer og B+ træer i et tidligere kapitel i disse noter.

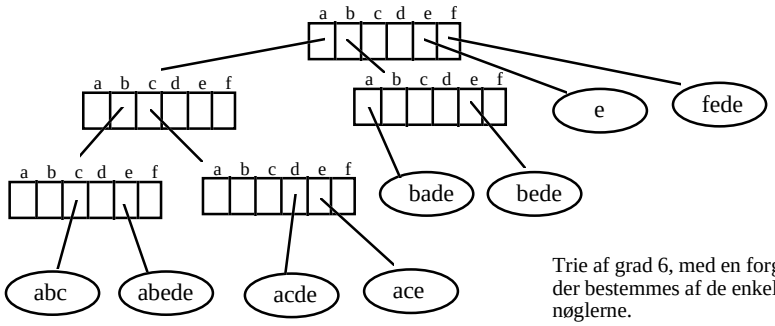
Ovenfor viser vi, for sammenligningens skyld, to tynde indekser til en datamængde, som findes i træets blade. Eksemplet er identisk med det B+ træ, vi studerede i kapitlet "Multivejstræer og B-træer".

Tries.

Et trie er et multivejstræ med en forgrening, der afhænger af nøglenes bestanddele.

Indre knuder kaldes *forgreningsknuder*. Blade kaldes *informationsknuder*.

Tries er velegnede til håndtering af nøgler af indbyrdes forskellig længde.



Trie af grad 6, med en forgrening der bestemmes af de enkelte tegn i nøglerne.

Alfabetetet er reduceret til 6 tegn.

© Kurt Nørmark, Aalborg Universitetscenter, 1994.

Noter

En sti fra roden til et blad angiver på en entydig måde netop en informationsknude. Dette mere end antyder en søgealgoritme på tries. I det valgte eksempel repræsenterer en sti et prefix af nøglen, men det kunne også være et suffix eller et helt andet "fix" af nøglen.

Indsættelse og sletning overlades til den interesserede læser. Man kan konsultere afsnit 10.2.4 i Horowitz og Sahni *Fundamentals og Data Structures* for detaljer. Weiss diskuterer også tries (i afsnit 10.1.2, dog kun binære tries).

Det ses, at en forgreningsknude altid har referencer til to eller flere informationsknuder (naturligvis bortset fra helt trivielle tilfælde).