Implementació de mètodes d'accés

Dolors Costal Costa

P03/05053/02051

UOC
Universitat Oberta de Catalunya

Índex

Introducció

En una base de dades hi ha dades emmagatzemades a les quals s'ha de poder accedir per tal de fer-hi consultes i actualitzacions. Per aquest motiu, una de les funcions dels SGBD és la de proporcionar l'accés a les dades que gestionen. La problemàtica de com oferir aquest accés és l'objecte d'estudi d'aquest mòdul didàctic.

En un altre mòdul expliquem les tècniques bàsiques que utilitzen els SGBD per a estructurar les seves dades en els suports no volàtils. Tot això ho complementem en aquest mòdul amb l'estudi de les implementacions que els SGBD fan servir per a accedir-hi.

Vegeu l'apartat 4 del mòdul "Components d'emmagatzematge d'una base de dades" d'aquesta assignatura.

Els diferents SGBD són força heterogenis en la manera d'implementar els mètodes d'accés (com ho són també en la manera d'estructurar l'emmagatzematge). Atès que seria impracticable intentar explicar exhaustivament i detalladament totes les implementacions que hi ha, estudiarem els principis d'algunes implementacions representatives. Aquests principis ens han d'ajudar a comprendre les diverses implementacions concretes que proporcionen els SGBD del mercat.

Objectius

Els materials didàctics associats a aquest mòdul pretenen facilitar a l'estudiant l'assoliment dels objectius següents:

1. Conèixer els diferents mètodes d'accés que són necessaris per a poder fer consultes i actualitzacions a les dades emmagatzemades a les BD.

2. Entendre la importància que té la reducció del nombre d'E/S en les implementacions dels mètodes d'accés.

3. Comprendre la utilitat dels índexs per a la implementació dels accessos per valor.

4. Conèixer l'estructura dels índexs arbres B^+.

5. Conèixer els índexs organitzats amb funcions de dispersió.

6. Saber quines són les característiques dels índexs agrupats.

7. Entendre els avantatges i inconvenients dels índexs arbres B^+, dels organitzats amb funcions de dispersió i dels índexs agrupats amb vista a la implementació dels accessos per valor.

8. Conèixer els índexs de valors compostos i entendre'n els avantatges i els inconvenients per a la implementació dels accessos per diversos valors.

1. Els mètodes d'accés a una BD

Una de les funcions dels SGBD és proporcionar l'accés a les dades que gestionen. L'accés a les dades ha de permetre consultar i actualitzar * les dades emmagatzemades. Sempre que es llegeix o s'enregistra alguna dada en una BD es fa mitjançant algun dels mètodes d'accés dels quals disposa l'SGBD.

*** Hem d'entendre actualitzacions de dades en sentit ampli; és a dir, la seva inserció, esborrat i modificació.**

1.1. Les dades

Les dades a què cal accedir són les que estan contingudes a les taules de la BD. Cal tenir en compte la seva estructura. En un altre mòdul didàctic expliquem que els SGBD, internament, estructuren les seves dades en EV que es componen de pàgines virtuals, i que aquestes pàgines virtuals s'emmagatzemen físicament als discos magnètics en pàgines reals, i també exposem com s'estableix la correspondència entre unes i altres.

Vegeu l'apartat 4 del mòdul "Components d'emmagatzematge d'una base de dades" d'aquesta assignatura.

En aquest mòdul, per tal de presentar els mètodes d'accés a les dades de manera entenedora, suposem sempre que les dades a les quals cal accedir es perceben segons la visió que proporciona el nivell virtual i no segons la del nivell físic. Així, doncs, gairebé sempre que en aquest mòdul s'empra la paraula *pàgina* hem d'interpretar que ens referim a pàgines virtuals; si ens referim a pàgines reals ho esmentarem explícitament.

A fi de simplificar, també suposem que tots els accessos es fan a dades que hi ha emmagatzemades en una única taula situada en un únic EV. Així, doncs, no considerem els casos en què cal combinar dades de taules (i espais) diferents per a obtenir les dades a les quals volem accedir. Cal fer notar, però, que el que estudiem és directament aplicable a accessos en els quals intervenen diferents taules si les tenim en un únic espai d'agrupació.

1.2. Els accessos per posició

En aquest subapartat veurem els tipus d'accés més simples que utilitzen els SGBD.

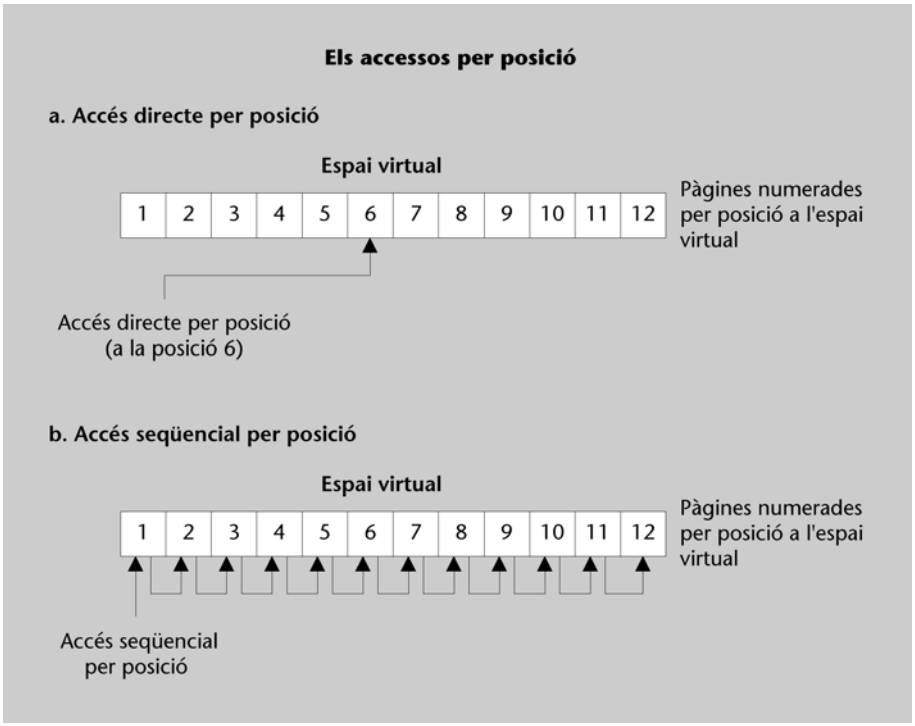
L'**accés directe per posició** consisteix a obtenir una pàgina que té un número de pàgina determinat dins un espai.

L'**accés seqüencial per posició** consisteix a obtenir les pàgines d'un espai seguint l'ordre definit pels seus números de pàgina.

Nota

Els accessos per posició són vells coneguts. L'accés seqüencial i l'accés directe per posició s'utilitzen també en la tecnologia dels fitxers. En concret, el fitxer seqüencial ofereix l'accés seqüencial per posició i el fitxer relatiu ofereix tant l'accés directe per posició com l'accés seqüencial per posició.

La figura següent mostra aquests dos tipus d'accés per posició:



Aquests tipus d'accés simples són suficients per a casos en què només calgui efectuar consultes i actualitzacions molt senzilles a la BD.

Exemples d'accessos per posició

Per a veure com funcionen els accessos per posició, en mostrem un exemple de cada un:

a) Exemple d'accés directe per posició

Suposem que disposem d'una BD que conté la taula EMPLEATS (*num_empl*, *nom_empl*, *num_despatx*, *sou*). Suposem també que aquesta taula s'emmagatzema en un espai determinat i que les files dels empleats se situen a l'espai esmentat per ordre d'inserció. Considerem ara l'actualització següent, descrita en SQL:

```
INSERT INTO empleats
VALUES (25, 'Joan Tarrago', 150, 2000);
```

Vegeu les sentències SQL al mòdul "Components lògics d'una base de dades" d'aquesta assignatura.

Aquesta actualització es pot efectuar fàcilment. L'SGBD té enregistrat el número de la primera pàgina de l'espai amb capacitat d'absorbir noves files. Aleshores, es fa servir l'accés directe per posició per a inserir l'empleat a la pàgina esmentada (afegint-hi la fila corresponent a l'empleat nou).

b) Exemple d'accés seqüencial per posició

A la base de dades anterior, se li podria fer una consulta per a recuperar les dades de tots els empleats, com ara la següent:

```
SELECT * FROM empleats;
```

Per a resoldre aquesta consulta, l'SGBD pot utilitzar l'accés seqüencial per posició, mitjançant el qual l'SGBD va obtenint totes les pàgines que contenen les files dels empleats. D'aquesta manera pot proporcionar els empleats en el mateix ordre amb què els obté. Observeu que la consulta no requereix cap ordenació particular dels empleats i, aleshores, l'SGBD els pot proporcionar en el mateix ordre en què estan emmagatzemats.

1.3. Els accessos per valor

En aquest subapartat expliquem alguns tipus d'accés més complexos que els accessos per posició que acabem de veure.

L'**accés directe per valor** consisteix a obtenir totes les files que contenen un valor determinat per un atribut.

L'**accés seqüencial per valor** consisteix a obtenir diverses files per l'ordre dels valors d'un atribut.

Nota

Ja coneixeu bé els accessos per valor. L'accés directe i l'accés seqüencial per valor s'empren també en la tecnologia dels fitxers.

Exemples d'accessos per valor

Tot seguit mostrem alguns exemples que requereixen l'accés per valor.

a) Exemples d'accés directe per valor

Considerem les sentències de manipulació següents:

- Sentència 1:

```
SELECT * FROM empleats WHERE num_despatx = 150;
```

- Sentència 2:

```
UPDATE empleats SET sou = 2500 WHERE num_despatx = 200;
```

- Sentència 3:

```
DELETE FROM empleats WHERE num_despatx = 150;
```

Observeu que en tots tres exemples és necessari buscar les files dels empleats que tenen un número de despatx determinat, ja sigui per a mostrar-los, modificar-los o esborrarlos, respectivament. És a dir, en tots tres casos cal fer cerques d'un valor per un atribut determinat.

D'aquests exemples es dedueix que l'accés directe per valor és necessari per a poder executar consultes i actualitzacions sobre files que contenen un determinat valor d'un atribut.

b) Exemples d'accés seqüencial per valor

Considerem les sentències de manipulació següents:

- Sentència 1:

```
SELECT * FROM empleats ORDER BY num_despatx;
```

- Sentència 2:

```
SELECT * FROM empleats
WHERE num_despatx >= 100 AND num_despatx <= 300;
```

- Sentència 3:

```
UPDATE empleats SET sou = 2500
WHERE num_despatx >= 100 AND num_despatx <= 300;
```

- Sentència 4:

```
DELETE FROM empleats
WHERE num_despatx >= 100 AND num_despatx <= 300;
```

En tots aquests exemples és necessari obtenir algunes files de la taula `empleats` en una seqüència ordenada per l'atribut `num_despatx`.

A la primera de les sentències cal obtenir els empleats ordenats pel número de despatx a causa de la clàusula `ORDER BY num_despatx`. Per a executar les tres sentències restants també s'ha de disposar de l'accés seqüencial per valor, encara que la causa és potser menys evident. La clàusula `WHERE num_despatx >= 100 AND num_despatx <= 300` fa que calgui localitzar tots els empleats que tenen un número de despatx entre el 100 i el 300, per a consultar-ne les dades, modificar-les o esborrar-les, respectivament. Una manera de localitzar tots els empleats és disposar d'algun mecanisme que accedeixi a cadascun d'ells per ordre de número de despatx a partir del valor 100 i fins al 300. Aquest mecanisme és l'accés seqüencial per valor.

Dels exemples anteriors es desprèn que l'accés seqüencial per valor es fa servir per a executar consultes en què el resultat ha d'estar ordenat per un atribut i per a consultes i/o actualitzacions que afectin un conjunt de files que contenen el valor d'un atribut que es troba dins un rang determinat de valors.

1.4. Els accessos per diversos valors

Hem considerat accessos pel valor d'un sol atribut. Ens manca considerar els casos en què es vol accedir segons els valors de diversos atributs: els **accessos per diversos valors**. Els accessos per diversos valors poden ser, com els accessos per un sol valor, directes o seqüencials.

Els accessos per diversos valors...

... no tenen el seu corresponent en la tecnologia dels fitxers, a diferència del que passava amb els altres accessos que hem explicat.

Exemples d'accés per diversos valors

Considerem els exemples d'accés per diversos valors següents:

- Sentència 1:

```
SELECT *
FROM empleats
WHERE num_despatx = 150 AND sou = 2000;
```

- Sentència 2:

```
SELECT *
FROM empleats
ORDER BY num_despatx, sou;
```

- Sentència 3:

```
DELETE *
FROM empleats
WHERE num_despatx >= 100 AND sou >= 1800;
```

En aquests exemples s'accedeix a les dades segons els valors de dos atributs: l'atribut `num_despatx` i l'atribut `sou`. La primera sentència correspon a un accés directe per diversos valors i les dues següents a un accés seqüencial per diversos valors.

També pot passar, però, que els accessos per diversos valors siguin mixtos.

Els **accesos mixtos per diversos valors** són accessos en què es combina l'accés directe pels valors d'alguns atributs i l'accés seqüencial pels valors d'altres atributs.

Exemples d'accessos mixtos per diversos valors

Considerem les sentències següents:

- Sentència 1:

```
SELECT *
FROM empleats
WHERE num_despatx = 150 AND sou >= 2000;
```

- Sentència 2:

```
SELECT *
FROM empleats
WHERE sou = 2000
ORDER BY num_despatx;
```

- Sentència 3:

```
DELETE *
FROM empleats
WHERE num_despatx >= 100 AND num_despatx <= 300 AND sou = 2000;
```

La primera de les sentències combina un accés directe pel valor de l'atribut `num_despatx` amb un accés seqüencial pel valor de l'atribut `sou`. Les dues següents combinen un accés seqüencial pel valor de l'atribut `num_despatx` amb un accés directe pel valor de l'atribut `sou`.

2. Implementació dels accessos per posició

Per a la implementació dels accessos per posició, els SGBD es basen gairebé completament en el SO.

Les rutines de gestió d'E/S de l'SO permeten obtenir una pàgina real donada la seva adreça i també permeten enregistrar en el disc una pàgina real concreta. Podríem dir que l'SO implementa l'accés per posició a pàgines reals.

D'acord amb el punt de partida que acabem de presentar, la implementació dels accessos per posició és molt simple.

En el cas de l'**accés directe per posició**, només cal que l'SGBD transformi els números de posició de les pàgines virtuals en adreces de pàgines reals emmagatzemades al disc.

El tipus de relació que hi ha entre pàgines virtuals i pàgines reals i la manera com l'SGBD és capaç de determinar l'adreça de la pàgina real que correspon a una pàgina virtual s'explica en un altre mòdul.

Vegeu el mòdul "Components d'emmagatzematge d'una base de dades" d'aquesta assignatura.

Si l'SGBD necessita fer un **accés seqüencial per posició**, el mateix procediment que hem explicat per a l'accés directe per posició li serveix per a anar accedint a totes les pàgines, des de la primera a la darrera.

3. Implementació dels accessos per valor

La implementació dels accessos per valor és més complexa que la dels accessos per posició. La dificultat prové del fet que si l'SGBD es basés únicament en l'SO per a implementar-los no sempre obtindria un bon rendiment. Serà necessari, doncs, que l'SGBD disposi de mecanismes propis especialitzats que els implementin de manera eficient.

En aquest apartat veurem que per a implementar d'una manera eficient l'accés directe i l'accés seqüencial per valor, els SGBD fan servir unes estructures de dades auxiliars que s'anomenen *índexs*. Per a la implementació dels accessos per valor amb índexs partirem del fet que ja disposem dels accessos per posició que hem explicat. Els SGBD fan servir els accessos per posició per a implementar els accessos per valor.

Vegeu la implementació dels accessos per posició a l'apartat 2 d'aquest mòdul didàctic.

3.1. Necessitat dels índexs

Començarem per explicar per què els índexs són necessaris si es desitja obtenir un bon rendiment quan es fan accessos per valor. Per a fer-ho presentem algunes implementacions que no els utilitzen i vegem els problemes que comporten.

Considerem la sentència següent, que requereix un accés directe per valor de l'atribut `num_despatx`:

```
SELECT * FROM empleats WHERE num_despatx = 150;
```

Cal tenir en compte que les files de la taula `empleats` són en un espai virtual. Aleshores, com que disposem de l'accés seqüencial per posició a les pàgines de l'espai virtual, el que pot fer l'SGBD és emprar aquest accés per a obtenir totes les pàgines una a una, comprovant per a cadascuna d'elles quines files contenen el valor buscat.

L'inconvenient de la solució anterior és que requerirà un gran nombre d'E/S, sobretot si hi ha molts empleats a la BD. Caldrà consultar totes les pàgines de la BD que tenen files d'empleats per tal d'aconseguir únicament els empleats del despatx 150, que poden ser molt pocs.

Ara analitzem un altre exemple que requereix un accés seqüencial per valor; vegeu la sentència següent:

```
SELECT * FROM empleats ORDER BY num_despatx;
```

Per a obtenir els empleats per ordre de número de despatx convindria que estiguessin emmagatzemats segons aquest mateix criteri. Llavors, utilitzant l'accés seqüencial per posició que ens facilita l'SO, s'aconseguirien els empleats en l'ordre desitjat.

El problema d'aquesta solució és que l'ordenació dels empleats per número de despatx aniria molt bé per a la consulta anterior, però molt malament si també es volgués executar, per exemple, la sentència següent:

```
SELECT * FROM empleats ORDER BY num_empl;
```

Com que en una BD han de conviure habitualment consultes i actualitzacions que requereixen ordenacions diferents d'unes mateixes dades, la solució que hem presentat no és satisfactòria per a totes les sentències que caldrà poder executar de manera eficient.

Les solucions que hem analitzat per a implementar els accessos per valor no ens permeten aconseguir un rendiment prou acceptable per a molts casos. Tot seguit analitzarem com els índexs ens permetran millorar aquesta situació.

3.2. Característiques generals dels índexs

Hi ha índexs de molts tipus diferents, però tots ells tenen algunes característiques generals comunes que fan que siguin implementacions adequades dels accesos per valor. Els índexs dels SGBD tenen una utilitat semblant a la dels índexs que contenen els llibres per a localitzar-ne ràpidament un apartat determinat.

L'índex d'un llibre

Aquest mòdul té un índex al principi. Si volem localitzar amb rapidesa el subapartat dedicat als arbres B^+, el que hem de fer és consultar l'índex. Allà trobarem fàcilment el número de pàgina on està situat el subapartat dels arbres B^+, perquè l'índex no és gaire llarg i no tardarem a llegir-lo. Un cop conegut el número de pàgina només ens cal localitzar la pàgina que té aquell número.

Els **índexs** que empren els SGBD són unes estructures de dades auxiliars que, com els índexs dels llibres, faciliten les cerques sobre unes determinades dades.

Un dels motius pels quals les cerques poden ser més ràpides si es fan mitjançant l'índex que si es fan directament sobre les dades és que habitualment els índexs ocupen menys espai que les dades.

Les files que contenen les dades generalment són voluminoses perquè emmagatzemen molts atributs. Els índexs, en canvi, normalment contenen només una col·lecció de parelles, anomenades **entrades**, formades per un valor i un RID (un RID es compon d'un número de pàgina i un número d'element del VAF).

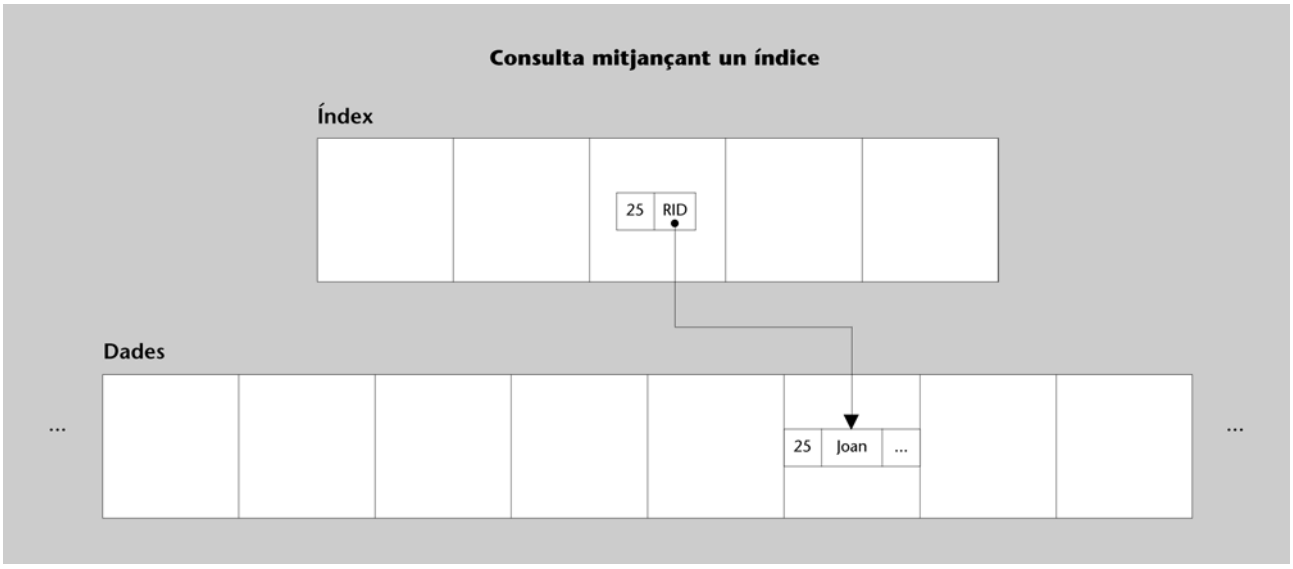
El concepte de RID s'explica detalladament al subapartat 4.3.1 del mòdul "Components d'emmagatzematge de bases de dades" d'aquesta assignatura.

Espai ocupat pels índexs i per les dades

Les files d'una taula d'empleats podrien enregistrar el número d'empleat, el nom, el DNI, l'adreça, el telèfon, el sou, el número de despatx, el departament on està assignat, etc. En canvi, si volem tenir un índex per accedir als empleats segons el seu número d'empleat, les entrades de l'índex hauran d'estar formades només per un número d'empleat i un RID.

Una cerca mitjançant un índex consisteix a localitzar primer els valors a l'índex per tal de conèixer-ne el RID. Un cop es disposa del RID, s'empra un accés directe per posició per a aconseguir la pàgina que conté la fila de les dades que es buscava.

La figura següent il·lustra la consulta de l'empleat número 25 amb un índex:



Dels paràgrafs anteriors podem deduir que els SGBD utilitzen l'accés per posició per tal d'implementar els accessos per valor.

Un índex ha d'estar organitzat d'alguna manera que faciliti les cerques dels valors que conté. Hi ha diverses maneres d'organitzar els índexs: els arbres B^+, les funcions de dispersió, etc. Alguns d'aquests tipus d'índex només faciliten l'accés directe per valor (per exemple, les funcions de dispersió). D'altres, serveixen tant per a l'accés directe com per a l'accés seqüencial per valor (per exemple, els arbres B^+).

Exemple

Sobre una taula que conté dades d'empleats podem tenir un índex que faciliti l'accés per número d'empleat, un altre que faciliti l'accés per nom, un altre que faciliti l'accés per número de despatx, etc.

Una característica molt útil de la majoria de tipus d'índex és que permeten la possibilitat de tenir diversos índexs sobre unes mateixes dades.

Tal com expliquem en un altre mòdul, les peculiaritats dels índexs fan aconsellable gestionar-los separadament de les dades de les taules. Per això, hi ha un tipus d'espai virtual per a contenir-los, anomenat **espai d'índexs**.

Vegeu l'espai d'índexs al subapartat 4.4.5 del mòdul "Components d'emmagatzematge d'una base de dades" d'aquesta assignatura.

Més endavant estudiem els tipus d'índexs que fan servir més sovint els SGBD per a implementar els accessos per valor: els estructurats com a arbres B^+ i els estructurats segons funcions de dispersió. Cada SGBD presenta les seves pròpies particularitats en la implementació d'un tipus d'índex determinat. Atès que seria impracticable intentar explicar exhaustivament i amb tots els detalls les implementacions que hi ha, només estudiem els principis comuns a la majoria d'implementacions d'índexs estructurats com arbres B^+ i els dels índexs estructurats mitjançant funcions de dispersió. Aquests principis ens faciliten la comprensió de les implementacions particulars que proporcionen els SGBD del mercat.

Observeu que...

... els índexs també es poden fer servir per a implementar els accessos que requereixen els fitxers per valor.

L'estimació del cost d'execució dels accessos a les dades mitjançant la utilització d'índexs és un aspecte que ens caldrà considerar d'ara endavant per tal d'avaluar la bondat dels diferents tipus d'índex. Per a aquesta estimació prenem les convencions següents:

1) Considerem només el cost de les E/S i no altres components del cost, com ara el que correspon als càlculs de la UCP. El motiu és que el cost de les E/S és normalment el component dominant en el cost de les operacions que es fan a les BD i ens proporciona una bona aproximació als costos reals.

UCP és la sigla del terme *unitat central de processament*.

2) Per a comptabilitzar el cost de les E/S adoptem la simplificació de comptar el nombre de pàgines que es llegeixen o que es graven al disc. Aquesta simplificació ens serveix perquè suposem que totes les E/S transporten únicament una pàgina* i que per tal d'accedir a una pàgina sempre cal fer una E/S, tot i que en alguns casos concrets podria no ser necessari (per exemple, si ja s'ha accedit a la pàgina amb anterioritat i encara se'n té una còpia a la memòria interna).

*** El transport d'una única pàgina és el cas més habitual.**

3.3. Arbres B^+

En els subapartats successius presentem un tipus d'índex que serveix per a facilitar els accessos directe i seqüencial pel valor d'un atribut: els arbres B^+.

Els arbres B^+ són un tipus particular d'arbre de cerca. L'objectiu primordial de l'estructura dels arbres B^+ és el d'intentar aconseguir que les cerques s'efectuïn amb un nombre petit d'E/S.

Recordeu el concepte d'*arbre* i el concepte d'*arbre binari de cerca* explicats en una altra assignatura. És important tenir en compte que els arbres binaris de cerca estan orientats a fer cerques en la memòria interna, mentre que els arbres B^+ que estudiem aquí estan orientats a fer cerques al disc.

Vegeu els arbres i els arbres binaris de cerca a l'assignatura *Estructura de la informació*.

L'estudi dels arbres B^+ és molt interessant perquè la majoria de sistemes comercials* l'implementen.

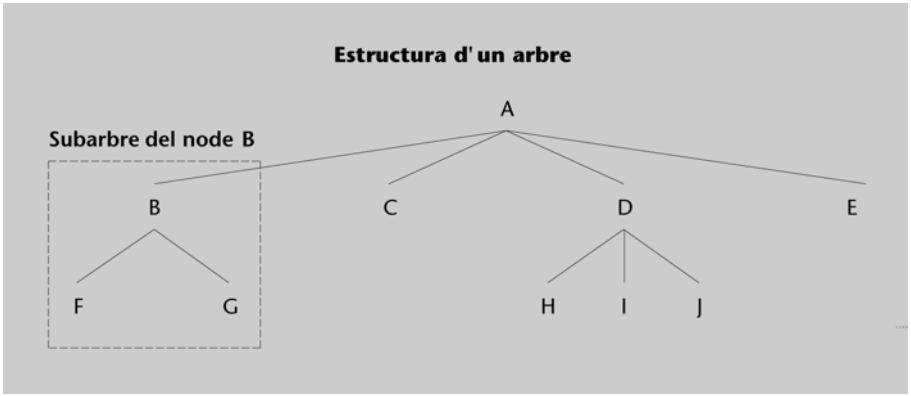
*** Com ara Informix, DB2, Oracle, Rdb, etc.**

3.3.1. Terminologia de les estructures de dades d'arbre

Comencem per presentar breument la terminologia que s'empra en parlar de les estructures de dades en forma d'arbre.

Un **arbre** es compon de nodes. Cada **node de l'arbre**, excepte un node especial anomenat *arrel*, té un node pare i diversos (zero o més) nodes fills. El **node arrel** no té pare. Els nodes que no tenen fills s'anomenen **nodes fulla** i els nodes que no són fulles s'anomenen **nodes interns**. El nivell d'un node és sempre el nivell del seu pare més un, i el nivell del node arrel és un. Un **subarbre d'un node** consisteix en el node i tots els seus nodes descendents: els seus nodes fills, els nodes fills dels seus fills, etc.

La figura següent il·lustra l'estructura de dades d'arbre que acabem de presentar:



Elements de l'arbre

- El node arrel de la figura és A.
- Els nodes fills de A són B, C, D i E.
- Els nodes fulla són F, G, C, H, I, J i E.
- El subarbre del node B és el marcat a la figura.

3.3.2. Estructura dels nodes

En primer lloc, cal considerar que tot arbre B^+ té un **número d'ordre *d*** que indica la capacitat dels seus nodes. Més concretament, si un arbre B^+ té ordre *d* aleshores els seus nodes contenen com a màxim $2d$ valors.

En un arbre B^+ els nodes interns i els nodes fulla tenen una estructura diferent. Les causes d'aquesta diferència estructural són tres:

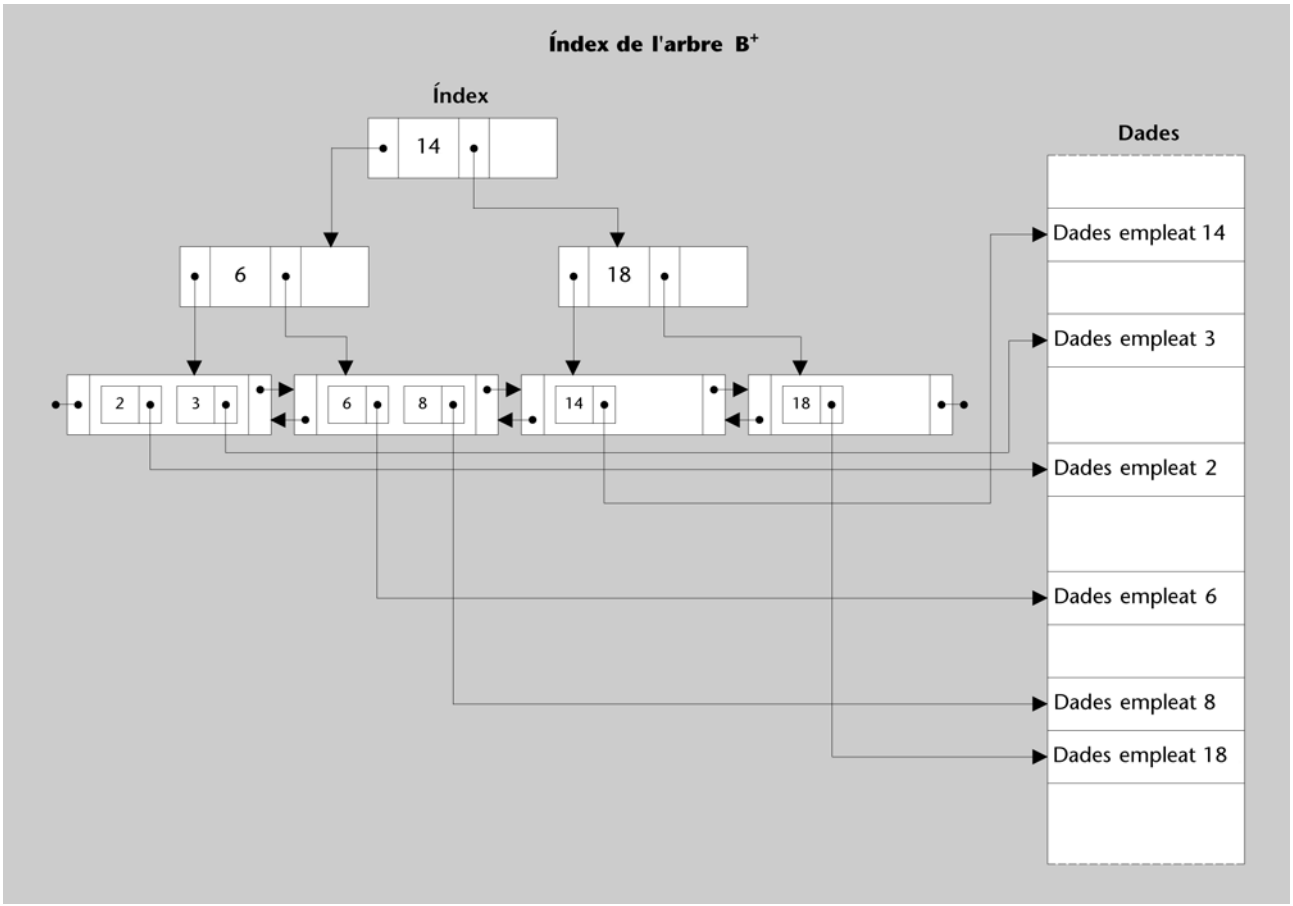
1) Els nodes fulla són els que contenen totes les entrades de l'índex (o sigui, les parelles de valor i RID).

2) Els nodes interns tenen com a objectiu dirigir la cerca de la fulla que té l'entrada corresponent a un valor determinat. L'accés directe per valor consistirà,

doncs, a fer un recorregut de l'arbre que començarà a l'arrel i anirà baixant pels nodes de l'arbre fins a arribar a la fulla adequada (la que conté, si existeix, l'entrada corresponent al valor buscat).

3) Els nodes fulla estan connectats per apuntadors, que serveixen per a facilitar l'accés seqüencial per valor (el recorregut de les fulles seguint els apuntadors proporciona les entrades ordenades per valor).

La figura següent il·lustra el que acabem d'explicar:



Estructura d'un arbre B^+

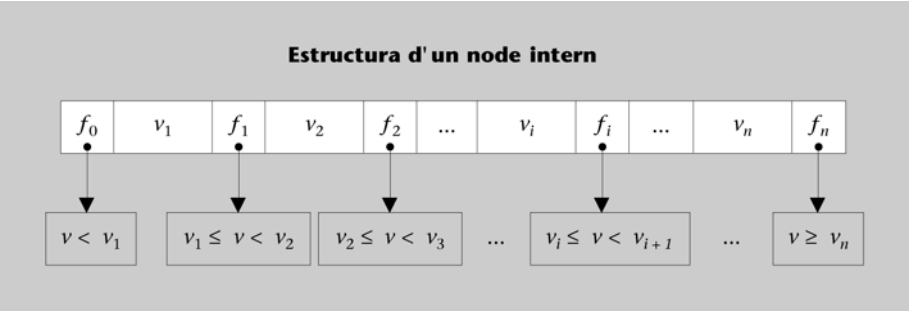
La figura mostra un índex arbre B^+ d'ordre $d = 1$ que serveix per a accedir a dades d'empleats segons el valor de l'atribut *número d'empleat*. Fixeu-vos que els RID que apunten a les dades són només als nodes fulla, que els nodes interns serveixen per a buscar els valors de les fulles i que el fet que les fulles estiguin connectades entre elles facilita l'accés seqüencial per valor.

A continuació descriurem l'estructura dels nodes interns i després l'estructura dels nodels fulla dels arbres B^+.

1) Estructura d'un node intern

Un **node intern** conté valors i apuntadors cap als seus nodes fills (una forma d'implementar un arbre és tenir en cada node apuntadors cap a tots els seus nodes fills).

L'estructura d'un node intern que conté n valors (on n és menor o igual que $2d$) és la que mostra la figura següent:



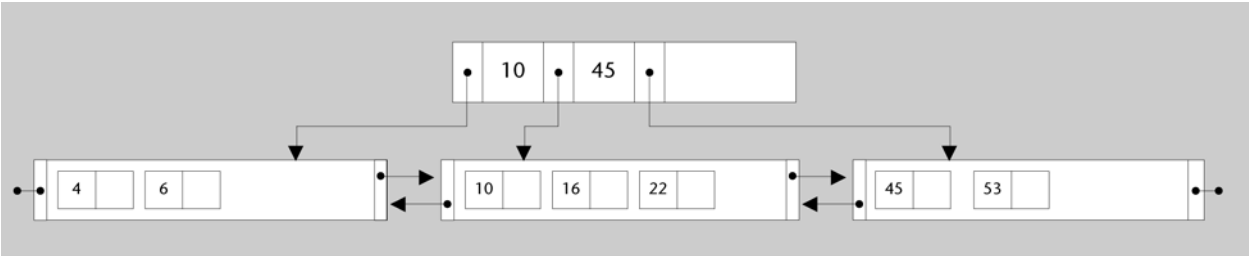
Cada v_i indica un valor i cada f_i indica un apuntador a un node fill de l'arbre. Si un node conté n valors, aleshores té $n + 1$ apuntadors a altres nodes de l'índex.

Tal com indica la figura, es compleix que el subarbre del node apuntat per f_i conté valors v tals que:

- $v_i \leq v < v_{i+1}$, si $i > 0$ i $i < n$.
- $v < v_1$, si $i = 0$.
- $v \geq v_n$, si $i = n$.

Exemple d'estructura d'un node intern

La figura següent mostra diversos nodes d'un índex arbre B^+ d'ordre 2 per l'atribut *número d'empleat*:

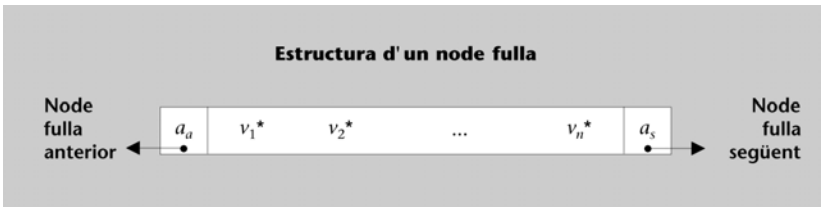


Considereu el node arrel. Té tres apuntadors que apunten als seus nodes fills. Fixeu-vos que el subarbre apuntat pel primer d'aquests apuntadors conté valors menors que 10. El subarbre apuntat pel segon conté valors més grans o iguals que 10 i més petits que 45. Finalment, el subarbre apuntat pel tercer conté valors més grans o iguals que 45.

2) Estructura d'un node fulla

Els **nodes fulla dels arbres B^+** contenen entrades formades pels valors als quals s'ha d'accedir i el RID que apunta a les dades, un apuntador al node fulla anterior, i un apuntador al node fulla següent.

L'estructura d'un node fulla que conté n valors (on n és menor o igual que $2d$) és la que mostra la figura següent:



A la figura...

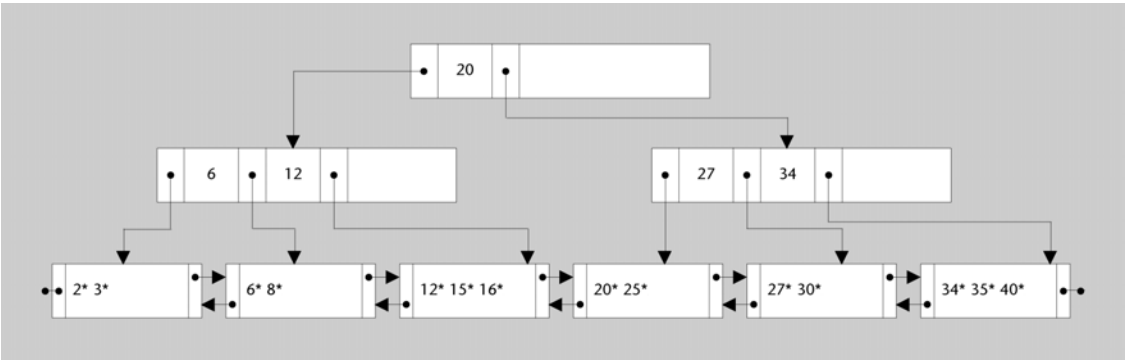
... cada v_i^* correspon a l'entrada del valor v_i (per tant, v_i i el seu RID), a_a indica l'apuntador al node fulla anterior i a_s indica l'apuntador al següent.

Addicionalment, cal que els nodes fulla compleixin les condicions següents:

- Tots els valors d'un node fulla són més petits que els valors del node fulla següent.
- Tots els valors d'algun node intern de l'arbre estan repetits en alguna fulla de l'arbre (així, les fulles contenen tots els valors de l'arbre).

Exemple d'estructura d'un node fulla

La figura següent mostra diversos nodes d'un índex arbre B^+ d'ordre 2 per l'atribut *número d'empleat*:



Considereu, per exemple, el node fulla que està situat més a l'esquerra. Conté les entrades de l'empleat número 2 i de l'empleat número 3. També té els apuntadors a les fulles anterior i següent de l'arbre*. Fixeu-vos que tots els valors del node considerat (els valors 2 i 3) són menors que els valors de la fulla següent (6 i 8). Finalment, observeu que el valor 6 de la segona fulla està repetit en un node intern. El motiu és que en un arbre B^+ les fulles han de contenir tots els valors de l'arbre.

*** En aquest cas concret, no existeix fulla anterior.**

3.3.3. Accés directe per valor

Per a fer un accés directe per valor amb un índex del tipus arbre B^+ cal, primer, localitzar la fulla que té l'entrada del valor buscat i, després, fer servir el RID de l'entrada per a trobar les dades a les quals es vol accedir.

Exemple de localització de la fulla que té l'entrada del valor buscat

Mostrarem com es pot fer la localització d'un valor v a l'arbre amb un exemple. Suposem que volem trobar el valor 8 a l'arbre de l'exemple anterior.

Vegeu l'"Exemple d'estructura d'un node fulla" al subapartat 3.3.2 d'aquest mòdul didàctic.

Accedim primer al node arrel. Després seguim el primer apuntador perquè $8 < 20$. A continuació seguim el segon apuntador, atès que $6 \leq 8 < 12$, i localitzem el node fulla que conté l'entrada del valor desitjat.

Suposem ara que busquem el valor 26 en el mateix arbre. Accedirem igualment al node arrel. Seguirem l'apuntador al fill de la dreta perquè $26 \geq 20$. Després seguirem el primer apuntador perquè $26 < 27$. El node fulla que obtenim aquesta vegada no conté l'entrada del valor 26, la qual cosa indica que el 26 no és a les nostres dades.

A continuació descriurem un algoritme que fa la cerca del node fulla que hauria de contenir una entrada determinada en un arbre B^+. L'algoritme sempre retorna un node fulla. Pot passar que el node fulla retornat contingui l'entrada buscada. Aleshores, només cal fer servir el RID de l'entrada per a localitzar les dades. També es pot donar el cas que el node fulla retornat no contingui l'entrada desitjada, la qual cosa ens indica que el valor al qual volem accedir no és a les dades. La descripció de l'algoritme l'hem extret, amb petites variacions, de l'obra especificada al marge.

Lectura complementària

Podeu trobar la descripció de l'algoritme de cerca presentat al text a l'obra següent:

R. Ramakrishnan (1998). *Database Management Systems*. Boston: McGrawHill.

El pseudocodi (a alt nivell i seguint un enfocament de programació estructurada) que esbossa l'algoritme de cerca es detalla a continuació:

Algoritme de cerca

```
funcio cercar(valor v) retorna apuntador_node
{donat un valor troba l'apuntador al node fulla que hauria de contenir la seva entrada}
retorna cerca_arbre(ap_arrel, v); {cerca des de l'arrel}
ffuncio

funcio cerca_arbre(apuntador_node apn, valor v) retorna apuntador_node
{cerca l'entrada dins de l'arbre}
si apn apunta a una fulla llavors retorna apn
   sino
      sigui N el node apuntat per apn
      si v1 al node N és menor que v
         llavors retorna cerca_arbre(fo,v)
                {fo és apuntador a node dins el node N}
         sino
            si vm al node N és menor o igual que v
               llavors retorna cerca_arbre(fm,v)
                  {fm és apuntador a node dins el node N}
                  {m és el número de valors del node N}
               sino
                  trobar i al node N tal que vi≤v<vi+1
                  retorna cerca_arbre(fi,v)
                     {fi és apuntador a node dins el node N}
            fsi
      fsi
fsi
ffuncio
```

3.3.4. Accés seqüencial per valor

Per tal de fer un accés seqüencial per valor amb un índex del tipus arbre B^+ cal, primer, localitzar ordenadament totes les entrades dels valors als quals es vol accedir i, per a cadascuna, fer servir el RID que apunta a les dades per a trobar el valor.

La localització ordenada de les entrades de l'arbre és senzilla. Convé recordar que les fulles contenen tots els valors de l'arbre, que cada fulla té un apuntador a la fulla següent i que els valors d'un node fulla són menors que els valors del node fulla següent. Aleshores, només cal accedir primer a la fulla de més a l'esquerra i, després, anar accedint a la fulla següent fins que s'acabi la cadena de fulles.

Observeu que...

... si seguim el procediment d'accés seqüencial per valor explicat en aquest subapartat amb l'arbre de la figura de l'"Exemple de l'estructura d'un node fulla" anirem obtenint totes les entrades de l'arbre de manera ordenada.

3.3.5. Propietats destinades a millorar el rendiment

Considerem un accés directe per valor que s'implementa amb un arbre B^+. Per tal de localitzar la fulla que conté l'entrada del valor, el nombre de nodes que cal recórrer és igual al del nivell de la fulla esmentada. En conseqüència, si reduïm el nivell de les fulles d'un arbre B^+, aconseguirem reduir el nombre de nodes que cal recórrer quan es fa un accés directe per valor.

Una propietat dels arbres B^+ destinada a reduir el nivell de les fulles és que tots els nodes de l'arbre (excepte l'arrel) han d'estar plens, com a mínim, fins a un 50% de la seva capacitat.

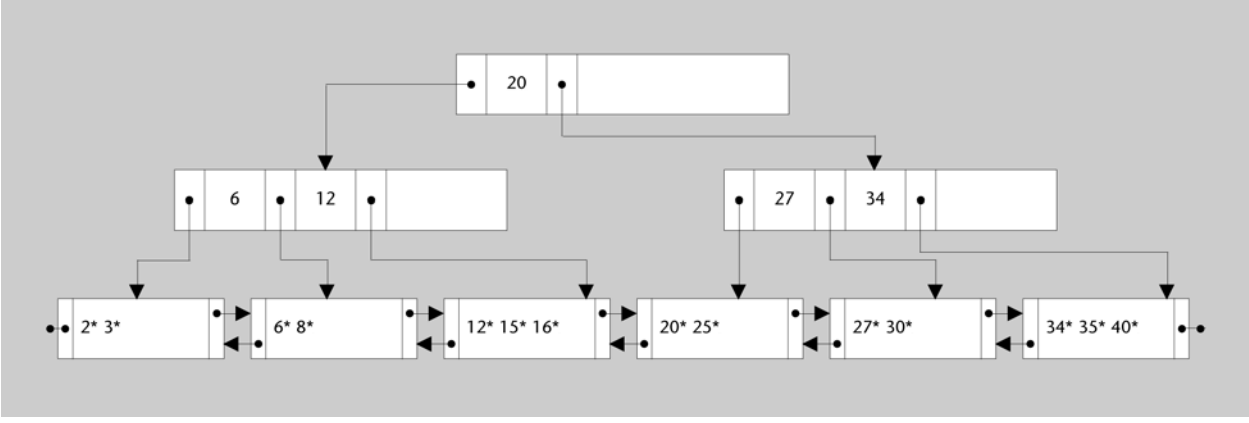
La norma que acabem d'enunciar equival a exigir que, en un arbre d'ordre *d*, tots els nodes excepte l'arrel continguin, almenys, *d* valors.

Un exemple amb tres nivells

A l'"Exemple d'estructura d'un node fulla", el valor 8 és en una fulla situada al nivell 3; per tant, cal passar per tres nodes per a trobar l'entrada corresponent al valor 8.

Exemple d'arbre amb nodes plens com a mínim fins al 50%

L'arbre d'ordre 2 de la figura següent compleix la propietat que acabem d'explicar en aquest subapartat:



Aquesta norma evita tenir arbres B^+ on molts dels nodes estan pràcticament buits. Si aconseguim que els nodes d'un arbre no estiguin gaire buits, l'arbre tindrà menys nodes i també menys nivells.

Amb vista al bon rendiment d'un índex, habitualment és desitjable que el nombre de nodes que cal recórrer sigui sempre més o menys el mateix, independentment de quin sigui el valor al qual es vol accedir.

Pel motiu anterior, els arbres B^+ compleixen una segona propietat: ser equilibrats. Un arbre és equilibrat si totes les fulles d'aquest estan al mateix nivell.

En un arbre equilibrat el nivell de les fulles és el que s'anomena **alçada de l'arbre**. La localització de qualsevol fulla sempre requerirà recórrer un nombre de nodes que coincidirà amb l'alçada de l'arbre.

Fixeu-vos que...

... l'arbre de la figura de l'exemple anterior és un arbre equilibrat i la seva alçada és 3.

3.3.6. Emmagatzematge de l'arbre i cost de localització d'una entrada

Tal com s'explica en un altre mòdul, les peculiaritats dels índexs fan que sigui aconsellable gestionar-los separadament de les dades de les taules. Per aquest motiu, hi ha un tipus d'espai virtual per a contenir els índexs anomenat **espai d'índexs**.

Vegeu l'espai d'índexs al subapartat 4.4.5 del mòdul "Components d'emmagatzematge d'una base de dades" d'aquesta assignatura.

Una qüestió important de l'emmagatzematge d'un arbre és l'elecció de la mida adequada per als seus nodes (que depèn, almenys, de l'ordre de l'arbre). Ja hem explicat que interessa que un arbre B^+ tingui una alçada peti ta. Per tal de facilitar-ho cal que els nodes de l'arbre siguin grans però sense sobrepassar la mida d'una pàgina, perquè altrament no es podrien consultar amb una única E/S.

En conseqüència, la mida habitual dels nodes d'un arbre B^+ coincideix amb la mida de les pàgines i cada node de l'arbre s'emmagatzema en una pàgina virtual diferent. Observeu que, segons aquesta forma d'emmagatzematge, en un arbre B^+ d'alçada h són necessàries h E/S per a localitzar una entrada de l'arbre*.

*** Per exemple, si $h = 3$ caldran tres E/S.**

De vegades, el node arrel d'un arbre B^+ es guarda en una memòria intermèdia i, llavors, el nombre d'E/S necessari per a localitzar una entrada es redueix en un*.

*** Per exemple, si $h = 3$ i l'accés al node arrel no requereix E/S, caldran només dues E/S.**

El fet que els nodes siguin de la mida d'una pàgina (que emmagatzema normalment 4kB) permet tenir habitualment arbres d'ordres entre 50 i 100.

Ordre habitual d'un arbre i volum de dades indexades

a) Suposem que disposem de pàgines de 4 kB i els valors que cal indexar ocupen 20 bytes. Considerem també que la mida del RID és de 4 bytes i els apuntadors a pàgines de l'arbre ocupen 3 bytes. Segons aquestes dades, en els nodes interns hi caben 176 valors i 177 apuntadors a altres nodes de l'arbre; en els nodes fulla hi caben 170 entrades (parelles de valor i RID) i els 2 apuntadors a les fulles anterior i següent. Com que la capacitat en valors ha de ser la mateixa a tots els nodes la restringirem a 170 (cas pitjor). L'ordre de l'arbre serà, doncs, la meitat de 170, és a dir, $d = 85$. Fixeu-vos que hem obtingut un ordre que es troba entre 50 i 100.

b) Considerem ara un arbre d'ordre $d = 50$, d'alçada $h = 3$ i on tots els nodes tenen una ocupació del 69%. Si $d = 50$, la capacitat màxima dels nodes és de 100 i una ocupació del 69% suposa tenir 69 valors a cada node. En aquest arbre tindrem el nombre de valors següent a cada nivell:

- Nivell 1: 1 node amb 69 valors i 70 apuntadors.
- Nivell 2: 70 nodes amb 69 valors cadascun i 70 apuntadors cadascun.
- Nivell 3: 4.900 nodes amb 69 entrades cadascun.

És a dir, 338.100 entrades en total. Segons això, l'arbre B^+ anterior, que té una alçada de només 3 nivells, ens permet indexar un volum de dades de 338.100 files.

D'aquest exemple es desprèn que amb ordres $d \geq 50$ i una alçada de $h = 3$ es pot indexar una quantitat de dades força considerable.

Exemple

Si pensem que l'arbre B^+ es construeix sonbre el número d'empleat, i suposant que aquest número és la clau primària de la relació d'empleats, la nostra relació podrà contenir fins a 338.100 empleats diferents.

3.3.7. Insercions i supressions

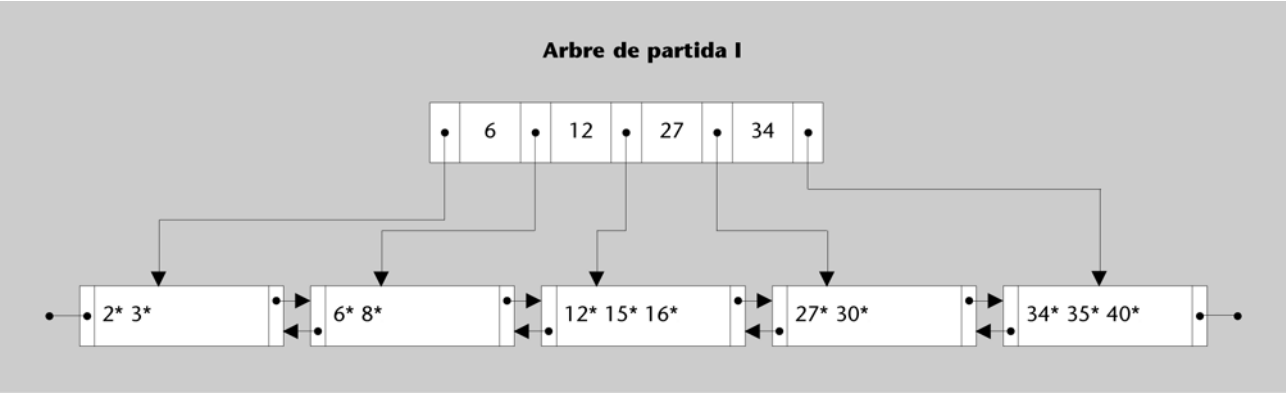
Les insercions i supressions s'han de fer de manera que l'arbre resultant satisfaci totes les propietats dels arbres B^+. Això suposa fer, en alguns casos, una reestructuració de l'arbre.

Insercions

Descrivim un algoritme que, donada una entrada, localitza el node fulla que li correspon i si aquest node té espai lliure, la hi insereix. En el cas que aquest node fulla no tingui espai lliure per a la nova entrada, reestructura l'arbre per a fer-li lloc.

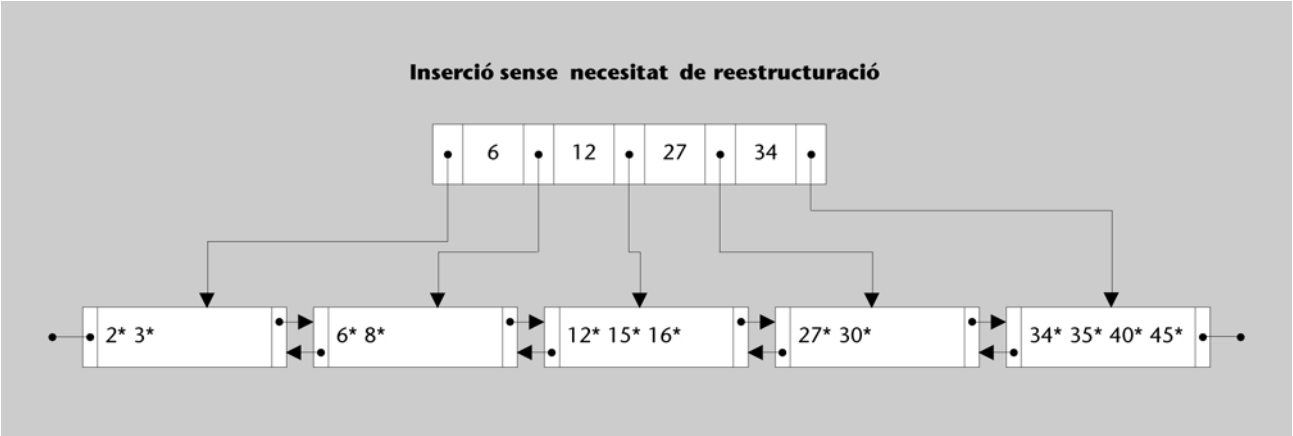
Comentarem alguns exemples que ens ajudaran a entendre amb més exactitud què ha de fer aquest algoritme d'inserció, la descripció detallada del qual la trobareu a l'annex d'aquest mòdul didàctic.

Considerem l'arbre B^+ d'ordre 2 de la figura següent:

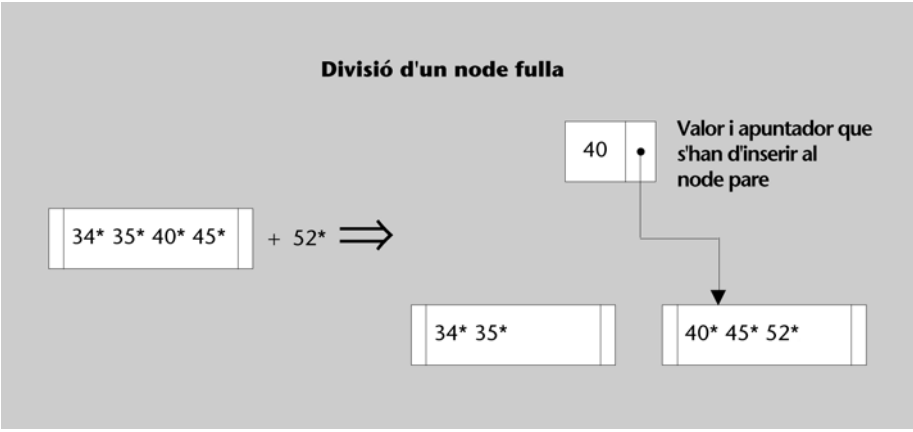


A partir d'aquest arbre, i considerarem els casos d'inserció que exposem tot seguit:

a) Suposem que es vol inserir en aquest arbre l'entrada 45*. El 45 ha de pertànyer a la fulla de més a la dreta. Atès que aquesta fulla té espai lliure, l'entrada 45* es pot col·locar a la fulla que li correspon i no cal fer cap reestructuració de l'arbre. S'obté l'arbre següent:



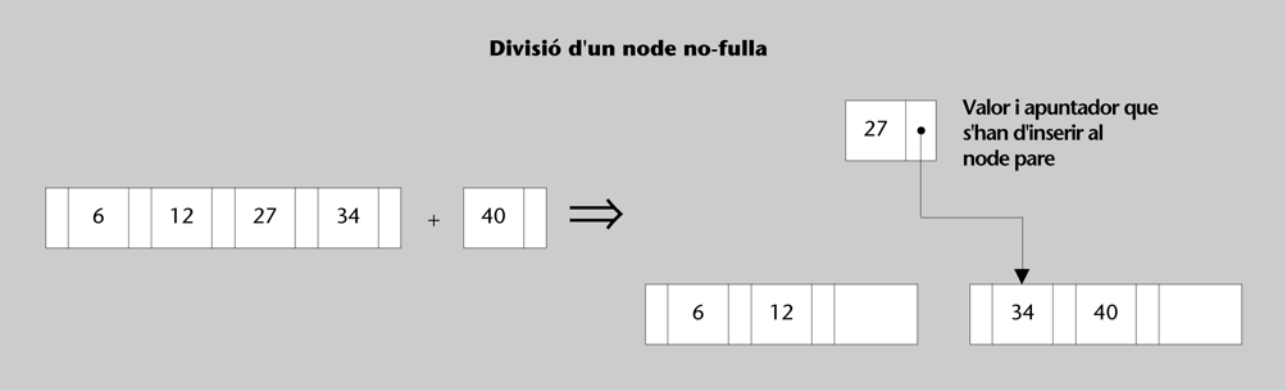
b) Ara considerem que es vol inserir l'entrada 52* a l'arbre que acabem d'obtenir. Li correspon la fulla de més a la dreta, però no té espai lliure. Per a poder inserir la nova entrada cal dividir la fulla en dues. Les dues fulles resultants de la divisió es mostren a la figura següent:



Quan es divideix un node, cal inserir en el seu node pare un valor i un apuntador al nou fill. En el nostre exemple, aquest valor és el 40 perquè és el valor més petit del segon node fulla que ha resultat de la divisió. Cal, doncs, inserir el 40 i l'apuntador al node pare. El node pare no té espai i caldrà dividir-lo.

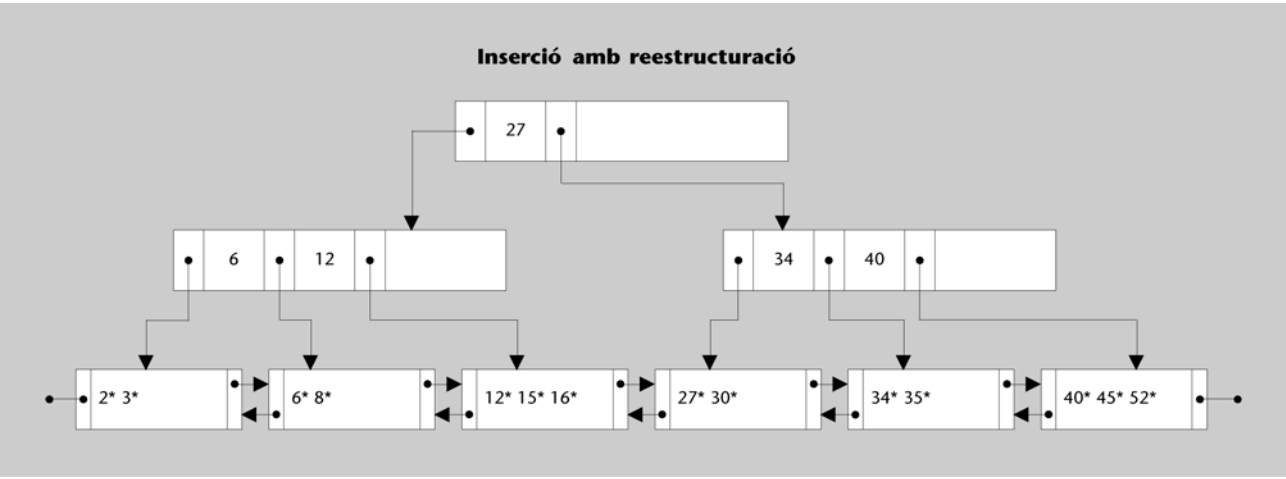
Quan es divideix un node no-fulla, els seus primers d valors es deixen al node (el 6 i el 12), els d valors més grans es col·loquen en un nou node (el 34 i el 40) i el valor del mig s'insereix al node pare (el 27) juntament amb un apuntador

cap al nou node. La figura següent ens mostra aquesta divisió d'un node no-fulla:



Observeu les diferències que hi ha entre aquesta divisió i la divisió d'un node fulla.

Com que acabem de dividir el node arrel de l'arbre, cal crear un nou node arrel, en el qual es col·loquen el valor 27, l'apuntador al nou node i també un apuntador a l'antiga arrel. L'arbre que s'obté finalment és el que podeu veure a la figura següent:



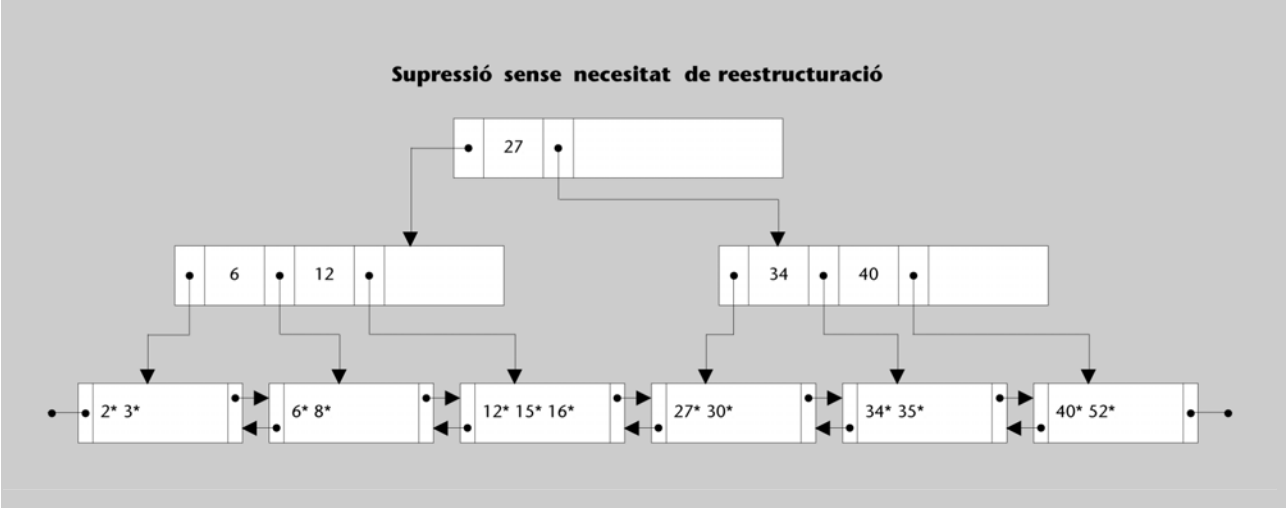
Supressions

Descriurem un algoritme que, partint d'una entrada, localitza el node fulla que li correspon, l'esborra i, si és necessari, reestructura l'arbre perquè tots els nodes de l'arbre (excepte l'arrel) tinguin, almenys, *d* valors.

Comentarem alguns exemples per a entendre de manera més precisa com funciona l'algoritme, la descripció detallada del qual la trobareu a l'annex d'aquest mòdul didàctic.

Considerem l'arbre de la figura anterior i suposem els casos de supressió següents:

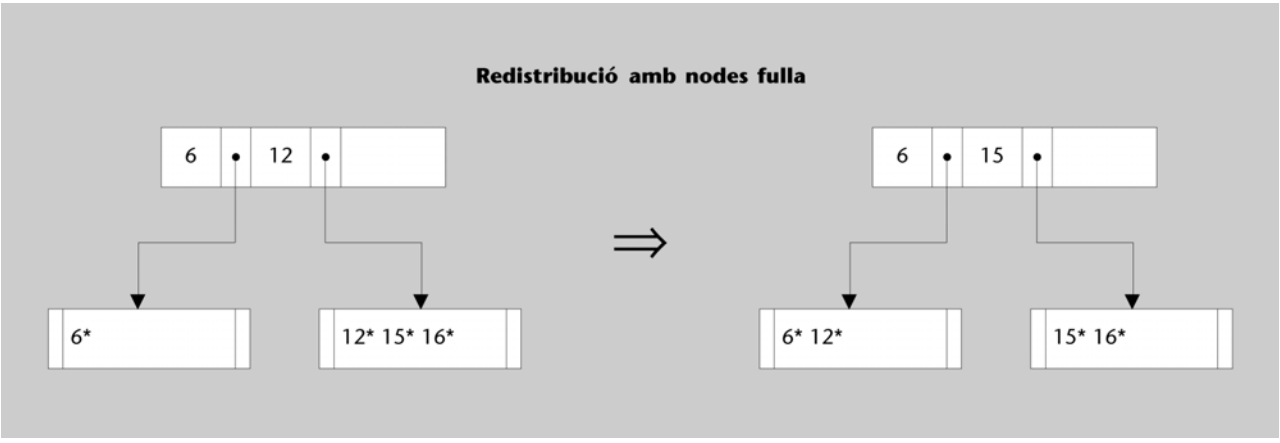
a) Suposem que volem esborrar el valor 45 de l'arbre. La fulla que conté l'entrada del valor 45 és la que està situada més a la dreta. Després d'esborrar 45* la fulla encara conté dos valors i, per tant, no cal fer cap reestructuració. L'arbre obtingut és el següent:



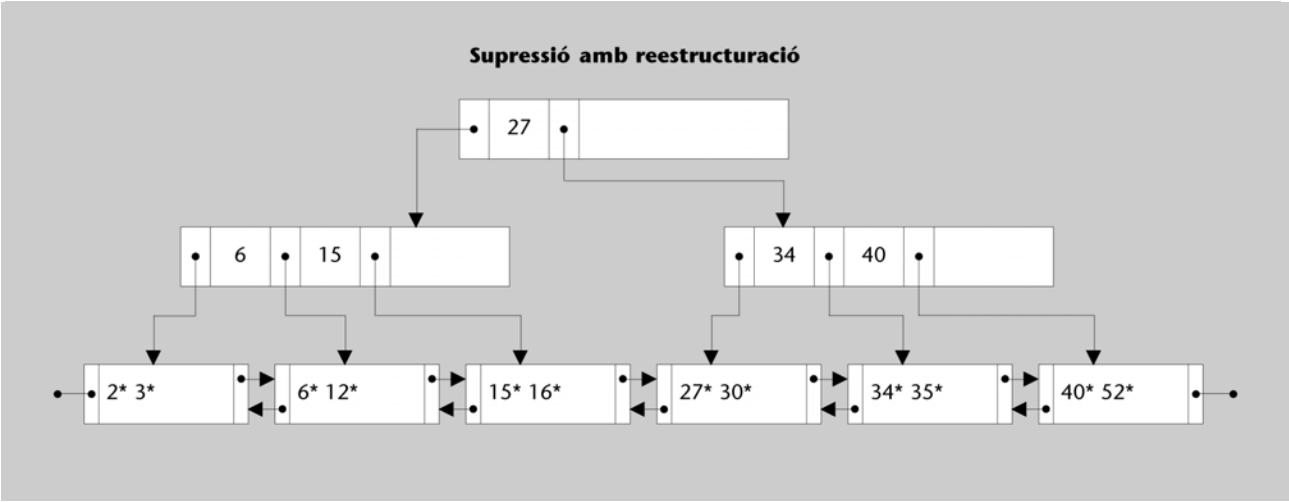
b) Ara suposarem que volem esborrar el valor 8 de l'arbre que acabem d'obtenir. La seva entrada està situada a la segona fulla de l'arbre. Després d'esborrar-la, la fulla es queda amb menys de dues entrades, cosa que cal corregir. Per a garantir que l'ocupació d'un node sigui correcta, és a dir, que l'estructura resultant després de la supressió segueixi essent un arbre B^+, sempre utilitzem el seu node germà* de la dreta; només si no té germà a la dreta farem servir el de l'esquerra.

*** Dos nodes són germans si comparteixen el mateix node pare.**

En el nostre exemple, com que el node germà de la dreta té tres entrades, aleshores es fa una redistribució d'entrades entre els dos nodes fulla. La figura següent il·lustra aquesta redistribució:

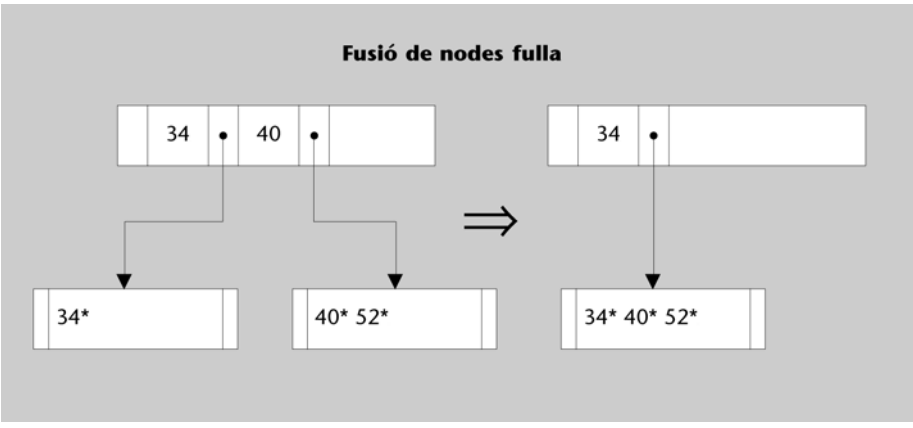


Observeu que la redistribució afecta el contingut del node pare dels dos nodes que hi participen. El valor 12 del node pare es canvia pel 15. L'arbre que s'obté finalment és el següent:



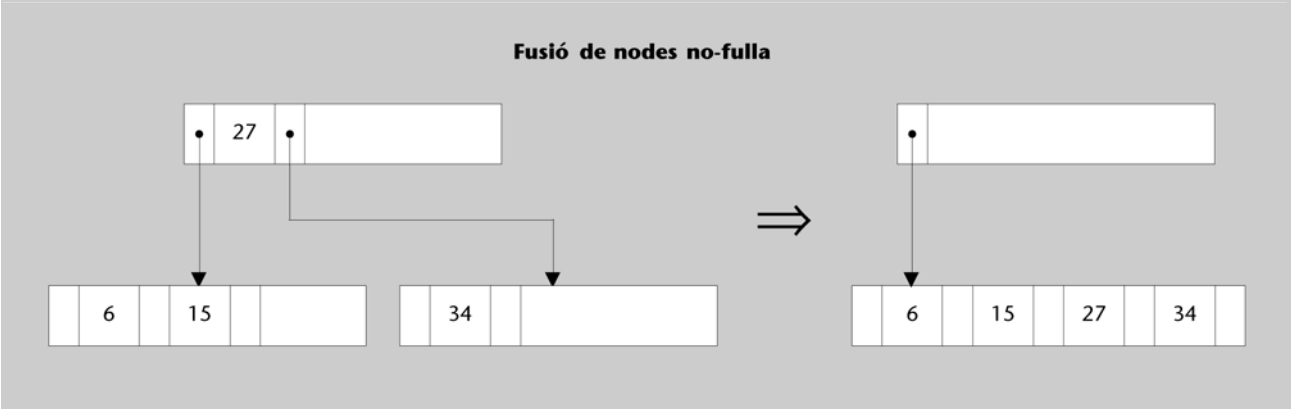
c) Considerem ara el cas de l'esborrament del 35 de l'arbre que acabem d'obtenir. L'entrada del 35 està situada a la penúltima fulla. Després d'esborrar-la, la fulla es queda amb menys de dues entrades, de manera que cal fer una reestructuració. En aquest cas, el seu node germà de la dreta té només dues entrades i, per tant, no es farà una redistribució com en el cas anterior (el node germà no té entrades de sobres per a redistribuir), sinó que es farà una fusió dels dos nodes fulla en un únic node.

La figura que presentem a continuació ens mostra aquesta fusió dels nodes fulla:



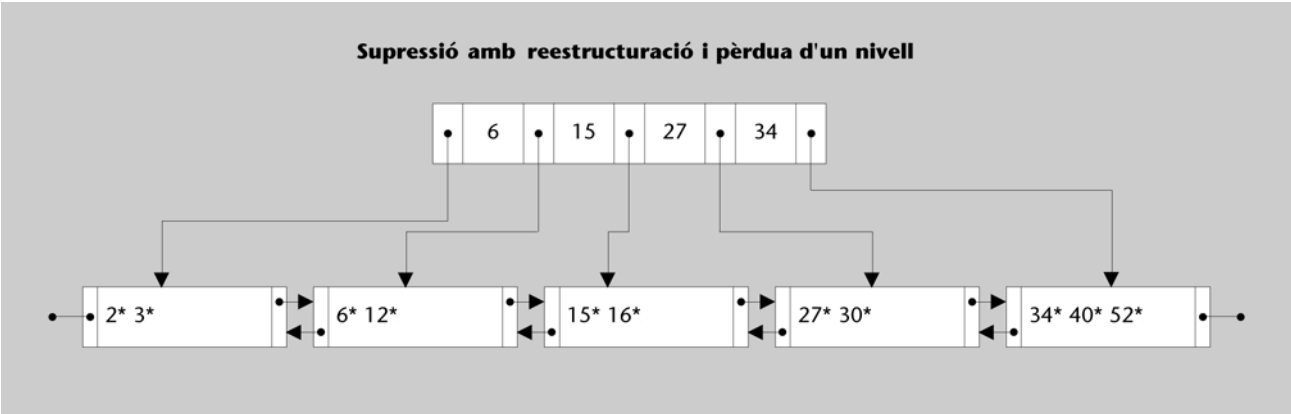
Fixeu-vos que quan es fusionen dos nodes el seu node pare perd un valor. A causa d'aquesta pèrdua, el node pare del nostre exemple ens ha quedat amb menys de dos valors. En aquest cas, caldrà fer encara una altra reestructuració de l'arbre per tal de corregir-ho. El node pare no té cap germà a la dreta; aleshores farem servir el de l'esquerra per a reestructurar l'arbre.

El germà de l'esquerra té només dos valors i, en conseqüència, es farà una fusió dels dos nodes. La figura següent ens il·lustra aquesta fusió de dos nodes no-fulla:

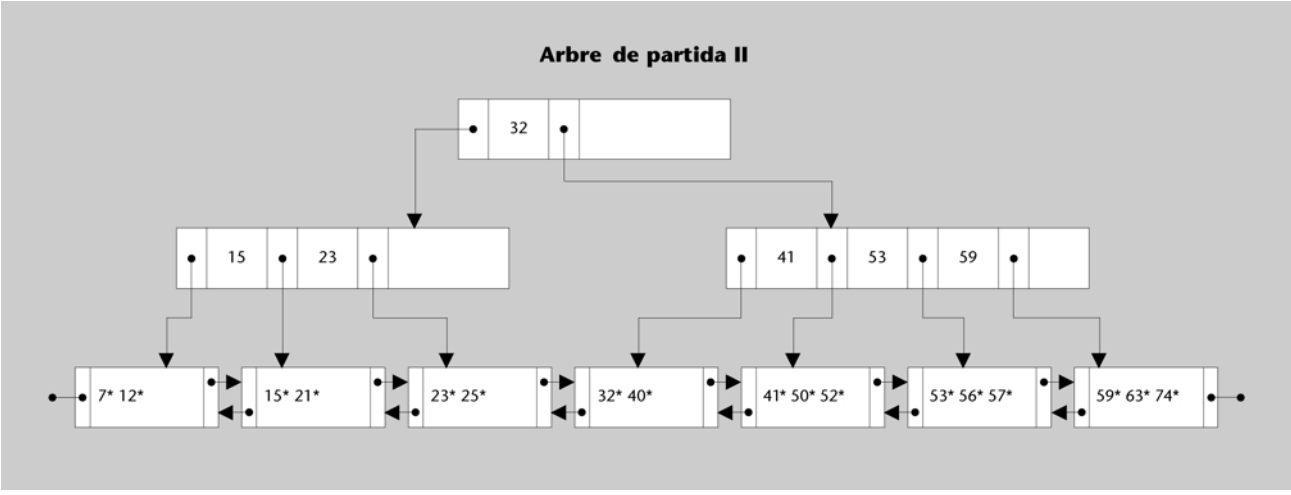


Convé observar les diferències respecte de la fusió anterior de dues fulles.

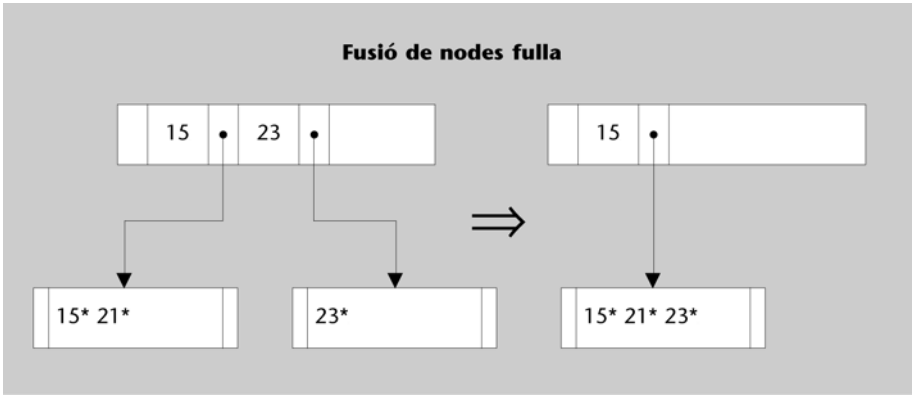
Després d'aquesta darrera fusió ens ha quedat un arbre amb l'arrel buida. En aquests casos cal descartar l'arrel antiga i fer que l'arrel nova sigui el node resultant de la fusió. Finalment, s'obté l'arbre següent:



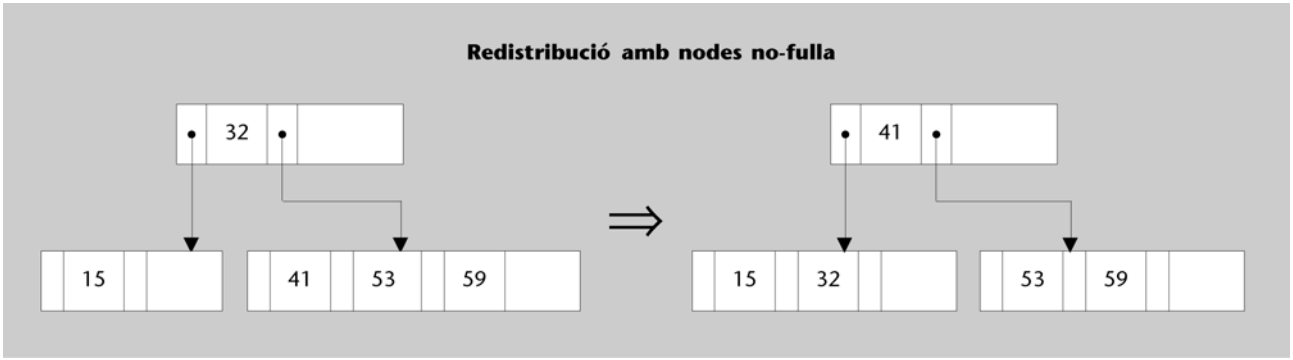
d) Per al darrer exemple de supressió escollim l'arbre B^+ de partida següent:



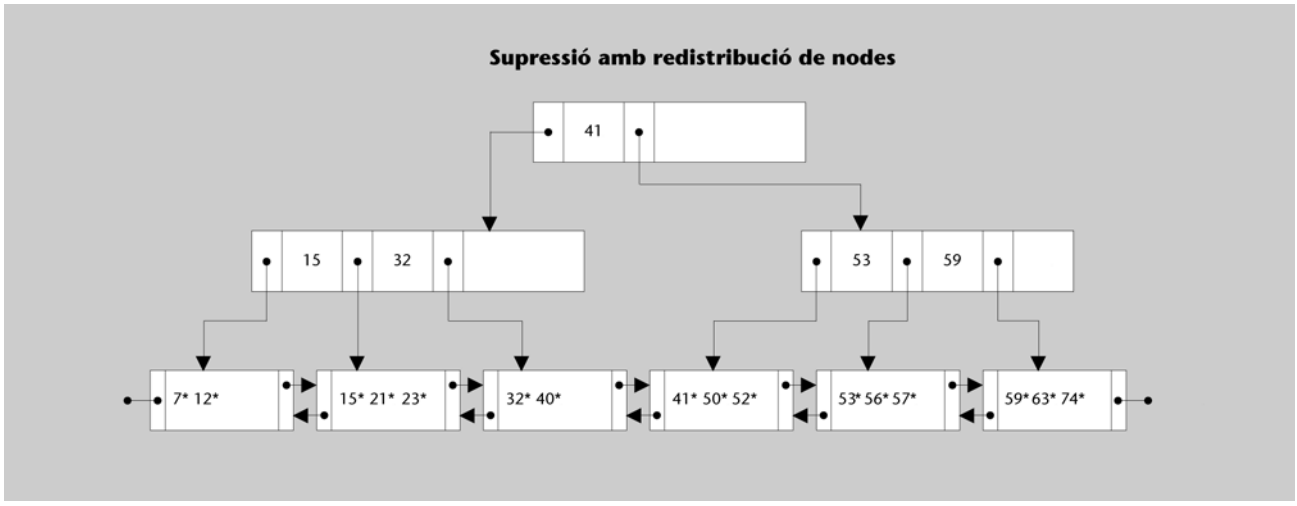
Considerem l'esborrament del valor 25 de l'arbre. L'entrada del 25 és a la tercera fulla. Després d'esborrar-lo, la fulla es queda amb menys de dues entrades. Aquesta fulla no té cap fulla germana a la dreta i cal fusionar-la amb la de l'esquerra, com mostra la figura següent:



Després de la fusió el node pare s'ha quedat amb menys de dos valors. Caldrà fer una redistribució amb el seu node germà de la dreta (que té més de dos valors). A continuació, es mostra aquesta redistribució de valors entre nodes no-fulla:



Aleshores, l'arbre B^+ que s'obté finalment és el següent:

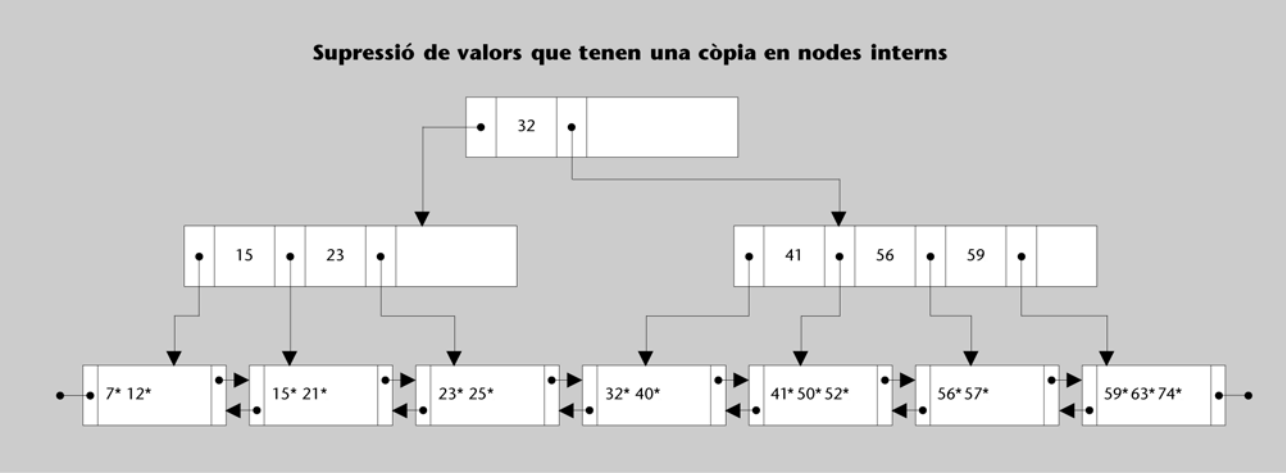


Supressions de valors que tenen una còpia en nodes interns

Hem analitzat un procediment d'esborrat dels valors d'un arbre B^+ que només apareixen en un node fulla de l'arbre. Com ja sabem, un arbre B^+ pot tenir valors que figuren en dos llocs: en un node fulla i en un node no-fulla (node intern). Per a esborrar els valors que tenen una còpia en nodes interns, podem utilitzar el mateix procediment si fem prèviament una substitució del valor copiat en el node intern pel valor de l'arbre que el segueix segons l'ordre dels valors. Un cop fet això podrem emprar el procediment de supressió anterior per a eliminar el valor de la fulla.

Vegeu els nodes fulla i els nodes interns al subapartat 3.3.2 d'aquest mòdul didàctic.

Esborrem el valor 53 de l' *Arbre de partida II*, que té l'entrada a la penúltima fulla de l'arbre i una còpia en el node pare de l'esmentada fulla. Substituïm la còpia del 53 pel valor que el segueix, que és el 56. Un cop fet això esborrem el 53 de la fulla. L'arbre que ens queda es mostra a la figura que presentem tot seguit:



Alguns sistemes*, però, quan han d'esborrar un valor que té una còpia en un node intern, l'esborren de la fulla però no l'esborren del node intern.

*** Com ara el sistema DB2.**

3.3.8. Valors repetits

En les cerques, insercions i supressions dels arbres B^+ que acabem d'explicar no hem considerat la possibilitat que hi pogués haver valors repetits a les dades; hem considerat que indexàvem els valors d'atributs identificadors.

Hi ha diverses solucions per al tractament dels valors repetits en els arbres B^+; en comentarem dues breument:

1) Una possibilitat és permetre l'existència d'entrades diferents amb el mateix valor a l'arbre. Aleshores, fulles diferents poden tenir entrades corresponents a un mateix valor. L'algoritme d'accés directe per valor haurà de localitzar pri-

mer l'entrada de més a l'esquerra del valor i després totes les possibles entrades addicionals del mateix valor. Aquestes entrades addicionals es poden trobar en altres fulles i es localitzen seguint els apuntadors entre fulles de l'arbre. Les insercions i supressions també s'han d'adaptar.

2) Una altra possibilitat és tenir una sola entrada que contingui diversos RID (un per a cada aparició del valor). Aquesta possibilitat fa que la mida de les entrades sigui variable i que la gestió d'aquestes sigui més complexa. Per contra, com que el valor en entrades diferents no es repeteix, s'estalvia espai, i, conseqüentment, operacions d'E/S.

El sistema DB2...

... fa servir la segona alternativa que presentem al text per al tractament dels valors repetits en la seva implementació dels índexs arbre B^+.

3.4. Dispersió

Hi ha un altre tipus d'índexs que serveixen per a facilitar l'accés directe per valor, però no l'accés seqüencial per valor: els índexs basats en la dispersió*.

*** En anglès, *hashing*.**

La dispersió per a fer cerques en la memòria interna ja s'ha estudiat en una altra assignatura. Aquí analitzarem la utilització de la dispersió per tal de fer cerques al disc, on és fonamental la minimització del nombre d'E/S.

Vegeu la dispersió per a fer cerques en la memòria interna a l'assignatura *Estructura de la informació*.

Els **índexs basats en la dispersió** aconsegueixen normalment un rendiment una mica millor que els índexs arbre B^+ quan es tracta d'implementar els accessos directes per valor. Per contra, no permeten implementar els accessos seqüencials per valor, mentre que els arbres B^+ sí. En conseqüència, alguns sistemes comercials proporcionen només implementacions basades en els arbres B^+.

Alguns sistemes comercials,...

... com ara Informix i DB2, proporcionen només implementacions basades en els arbres B^+. D'altres, per exemple, el sistema Rdb, proporcionen tots dos tipus d'índexs.

3.4.1. Introducció a la dispersió

La característica fonamental dels índexs basats en la dispersió és que les entrades es col·loquen a les pàgines segons una funció de dispersió *h*.

La **funció de dispersió *h*** és una funció que a partir d'un valor *v* ens retorna un número de pàgina *p*, és a dir, $p = h(v)$. El **número de pàgina *p*** serà un enter de l'interval [1, ..., N]. Aleshores l'entrada corresponent al valor *v*, que representem per *v**, se situarà a la pàgina de número *p* de l'interval.

Quan es vulgui accedir al valor*v*, podrem accedir a la seva entrada*v** de la manera següent:

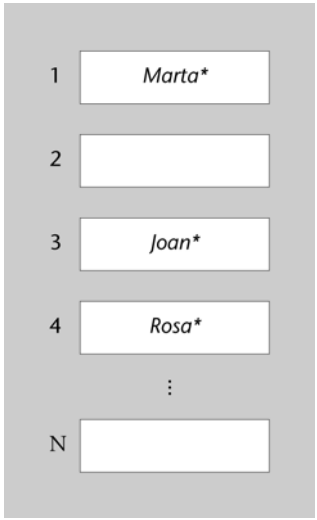
- Es calcula el número de pàgina *p* tal que $p = h(v)$.
- S'accedeix a la pàgina *p* per a localitzar l'entrada del valor *v*.

Podeu trobar la descripció detallada d'algunes funcions h de dispersió que transformen un valor numèric o alfanumèric en un número enter de l'inter val [0, ..., N - 1] en una altra assignatura. Podrem fer servir qualsevol d'aquestes funcions sempre que sumem 1 al resultat que ens donin, perquè ens interessa obtenir números de pàgina que estiguin a l'interval [1, ..., N].

Vegeu amb detall algunes funcions *h* a l'assignatura *Estructura de la informació.*

Localització d'una entrada amb un índex basat en la dispersió

Volem indexar unes dades pel valor d'un atribut que representa noms de pila. Suposem que *h*(Joan) = 3, *h*(Marta) = 1 i *h*(Rosa) = 4. Aleshores, les entrades d'aquests valors quedaran col·locades a les pàgines de l'índex de la manera següent:



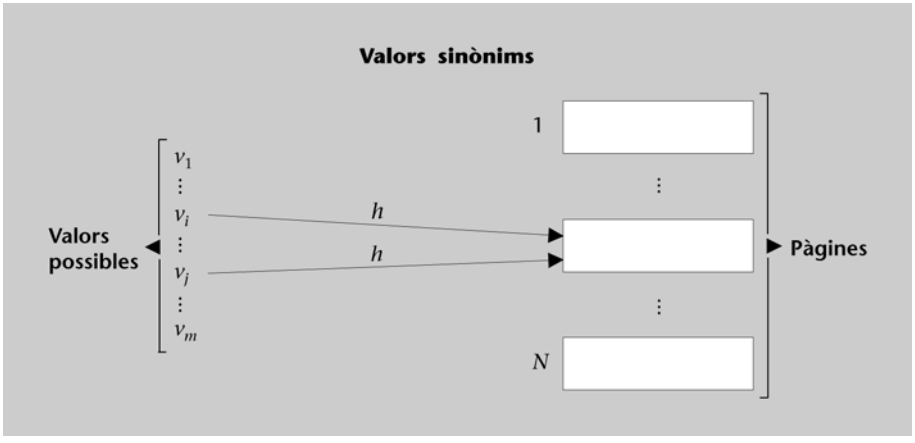
Per tal de localitzar l'entrada del valor *Rosa* a l'índex anterior calcularem *h*(Rosa). Aquest càlcul ens donarà el número de pàgina 4. Aleshores accedirem a la pàgina 4, on trobarem l'entrada corresponent a *Rosa*.

Observeu que el nombre de valors possibles pot ser molt gran, mentre que el nombre de pàgines que pot ocupar l'índex al disc sol ser molt més limitat. Això fa que el nombre de valors *v* possibles sigui molt més gran que el nombre N de pàgines possibles. La funció de dispersió*h* transforma valors d'un interval molt gran en posicions d'un interval més petit (l'interval [1, ..., N]).

Per exemple,...

... penseu en valors d'un atribut que representa un DNI. Els valors possibles de l'atribut es troben entre 0 i 99.999.999. En canvi, el nombre de pàgines que pot ocupar l'índex al disc difícilment arribarà als cent milions.

Per aquest motiu, és possible que per a dos valors diferents, v_i i v_j, passi que $h(v_i) = h(v_j)$, és a dir, que la funció ens retorni la mateixa pàgina per als dos valors, com es veu a la figura següent:



En aquest cas, els valors v_i i v_j s'anomenen **sinònims**. Les entrades corresponents als valors sinònims han d'estar a la mateixa pàgina. Les pàgines, per tant, poden haver d'emmagatzemar diverses entrades. Anomenem L el nombre d'entrades que caben en una pàgina.

Una diferència important...

... entre la dispersió a la memòria interna i la dispersió amb dades que hi ha emmagatzemades al disc és que les posicions p, en el segon cas, són pàgines amb capacitat per a emmagatzemar diversos sinònims. El propòsit d'això és fer que un nombre gran de sinònims siguin accessibles amb una sola E/S.

Exemple de valors sinònims

Si a l'índex per l'atribut*nom de pila* anterior hi afegim el valor*Jordi* i h(Jordi) = 4, aleshores tindrem la situació que es mostra a l'esquema següent:

De vegades, una pàgina pot ser insuficient per a contenir tots els sinònims que s'hi haurien de col·locar. Això passarà quan a una pàgina li corresponguin més de L sinònims.

Quan un sinònim no té espai a la pàgina que li correspon s'anomena **excedent** i es col·loca en alguna altra pàgina, que s'escull segons una determinada política de gestió d'excedents. Hi ha moltes maneres alternatives de gestionar els excedents. A continuació n'explicarem una.

3.4.2. Gestió d'excedents

La política de gestió d'excedents que estudiem es basa en el fet que l'índex el formen dos tipus disjunts de pàgines:

- **Pàgines primàries**: pàgines on col·locarem totes les entrades que no són excedents (n'hi haurà N).
- **Pàgines d'excedents**: pàgines destinades a guardar les entrades excedents.

Quan una pàgina primària p s'omple i s'hi insereix un nou valor que segons la funció de dispersió hauria d'anar a la pàgina p, l'entrada del valor es col·loca

en una pàgina d'excedents *e* i a la pàgina primària *p* se li afegeix un apuntador cap a la pàgina *e*.

Aquesta pàgina d'excedents *e* s'utilitzarà només per a encabir-hi els possibles excedents de la pàgina primària *p* que s'insereixin posteriorment. Si la pàgina *e* també s'omple, se li encadenarà una nova pàgina d'excedents *e'* on es podran col·locar més excedents de la pàgina primària *p*, i així successivament.

D'aquesta manera, per cada pàgina primària que té excedents es construeix una cadena de pàgines d'excedents. El primer apuntador de la cadena està situat a la pàgina primària que l'ha originat.

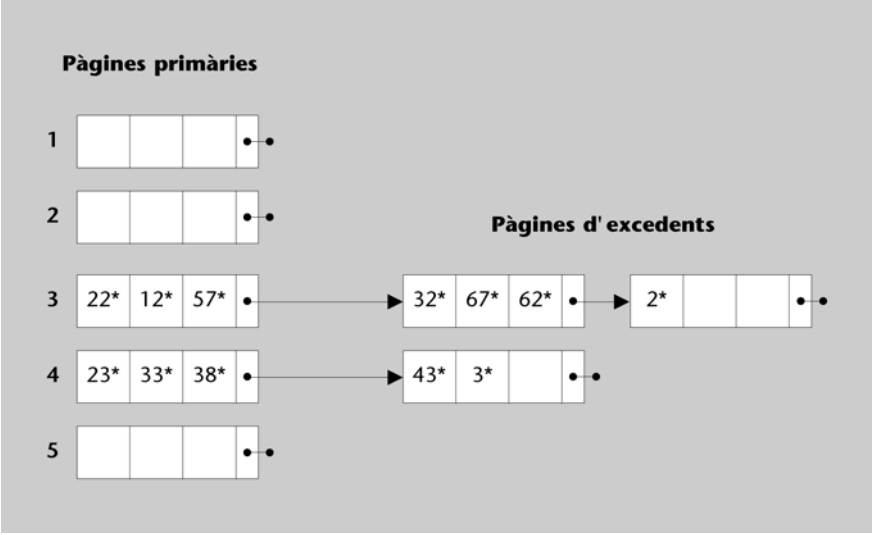
Com que hi ha una cadena separada de pàgines d'excedents per cada pàgina primària, aquesta política de gestió d'excedents de vegades s'anomena **encadenament separat***.

*** En anglès, *separate chaining*.**

Les supressions dels valors de l'índex es poden fer de diverses maneres. Una possibilitat és marcar el valor amb un senyal especial que indiqui que està esborrat, sense reorganitzar res. Una altra possibilitat diferent és aprofitar aquestes supressions de valors per a reduir la longitud de les cadenes d'excedents.

Exemple de gestió d'excedents amb encadenament separat

Suposem que tenim un índex amb 5 pàgines primàries ($N = 5$), que totes les pàgines tenen una capacitat de 3 entrades ($L = 3$) i que la funció de dispersió és $h(x) = (x \bmod 5) + 1$. Si inserim els valors: 22, 23, 12, 57, 33, 38, 32, 67, 62, 43, 2, 3, l'índex quedarà com mostra la figura següent:



3.4.3. Emmagatzematge i cost de localització d'una entrada de l'índex

Els índexs basats en la dispersió s'emmagatzemen en un tipus d'espai virtual anomenat **espai d'índexs** (igual que els índexs arbre B^+).

Vegeu els espais virtuals d'índexs en el subapartat 4.4.5 del mòdul "Components d'emmagatzematge d'una base de dades" d'aquesta assignatura.

És fàcil adonar-se que el cost de localitzar una entrada a l'índex emmagatzemat a les pàgines d'un espai d'índexs varia força segons si és excedent o no:

- Si una entrada no és excedent i, per tant, es troba a la seva pàgina primària, només caldrà fer una E/S per tal d'obtenir-la.

- Per a les entrades excedents, en canvi, sempre caldrà fer més d'una E/S.

En conseqüència, per a aconseguir un bon rendiment de l'índex interessa que hi hagi poques entrades excedents. Una manera de reduir el nombre d'entrades excedents és tenir més espai del que es necessita a les pàgines primàries; és a dir, aconseguir que les pàgines primàries estiguin poc carregades d'entrades.

S'anomena **factor de càrrega** el nombre d'entrades que s'espera tenir dividit pel nombre d'entrades que caben a les pàgines primàries:

$$C = M / (N \times L)$$

on M representa el nombre d'entrades que esperem tenir i N x L és el nombre d'entrades que caben a les pàgines primàries.

Com més baix sigui el factor de càrrega menys excedents hi haurà i menys E/S seran necessàries. En contrapartida, si el factor de càrrega és molt baix es gasta més espai.

Lectura complementària

Podeu trobar les taules que apareixen en aquest subapartat a l'obra següent:

D. Knuth (1973). "Sorting and Searching". *The Art of Computer Programming* (vol. 3). Reading (Massachusetts): Addison-Wesley.

Les taules següents, confeccionades per Knuth, ens mostren les E/S necessàries per a un determinat factor de càrrega i per a un determinat nombre$L$ d'entrades per pàgina:

a) Cas d'entrades que no són a l'índex:

	Mitjana d'E/S per a entrades que no són a l'índex									
L \ C	**0,1**	**0,2**	**0,3**	**0,4**	**0,5**	**0,6**	**0,7**	**0,8**	**0,9**	**0,95**
1	1,0048	1,0187	1,0408	1,0703	1,1065	1,1488	1,197	1,249	1,307	1,3
2	1,0012	1,0088	1,0269	1,0571	1,1036	1,1638	1,238	1,327	1,428	1,5
3	1,0003	1,0038	1,0162	1,0433	1,0898	1,1588	1,252	1,369	1,509	1,6
4	1,0001	1,0016	1,0095	1,0314	1,0751	1,1476	1,253	1,394	1,571	1,7
5	1,0000	1,0007	1,0056	1,0225	1,0619	1,1346	1,249	1,410	1,620	1,7
10	1,0000	1,0000	1,0004	1,0041	1,0222	1,0773	1,201	1,426	1,773	2,0
20	1,0000	1,0000	1,0000	1,0001	1,0028	1,0234	1,113	1,367	1,898	2,3
50	1,0000	1,0000	1,0000	1,0000	1,0000	1,0007	1,018	1,182	1,920	2,7

b) Cas d'entrades que són a l'índex:

	Mitjana d'E/S per a entrades que són a l'índex									
L \ C	0,1	0,2	0,3	0,4	0,5	0,6	0,7	0,8	0,9	0,95
1	1,0500	1,1000	1,1500	1,2000	1,2500	1,3000	1,350	1,400	1,450	1,5
2	1,0063	1,0242	1,0520	1,0883	1,1321	1,1823	1,238	1,299	1,364	1,4
3	1,0010	1,0071	1,0216	1,0458	1,0806	1,1259	1,181	1,246	1,310	1,4
4	1,0002	1,0023	1,0097	1,0257	1,0527	1,0922	1,145	1,211	1,290	1,3
5	1,0000	1,0008	1,0046	1,0151	1,0358	1,0699	1,119	1,186	1,286	1,3
10	1,0000	1,0000	1,0002	1,0015	1,0070	1,0226	1,056	1,115	1,206	1,3
20	1,0000	1,0000	1,0000	1,0000	1,0005	1,0038	1,018	1,059	1,150	1,2
50	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,001	1,015	1,083	1,2

Podem comprovar a les taules que els factors de càrrega més baixos requereixen menys E/S.

Els càlculs...

... presentats en aquestes taules s'han fet per a una gestió d'excedents amb encadenament separat.

Cal considerar, però, que si per exemple el factor de càrrega és 0,5 el rendiment en temps serà excel·lent, però la meitat de l'espai es malbaratarà.

Per a aconseguir un factor de càrrega C determinat, suposant que tenim una previsió de M i un L determinat, caldrà escollir un valor de N adequat.

Per a un mateix factor de càrrega, el fet que el nombre d'entrades per pàgina (L) sigui gran també redueix el nombre d'excedents, tal com es pot comprovar a les taules anteriors. L depèn de la mida de les pàgines i de la mida de les entrades.

Exemple de càlcul de *L*

Si disposem de pàgines de 4 kB, els valors indexats ocupen 40 bytes, els RID 4 i els apuntadors a pàgines 3, aleshores el nombre L d'entrades serà de 93, atès que s'ha de complir que 4 kB $\geq (40 + 4)L + 3$ (la pàgina ha de contenir fins a L entrades formades pel valor i un RID cadascuna, i també l'apuntador a la primera pàgina d'excedents).

En els casos en què la mida de les entrades sigui molt gran i això ens provoqui que L sigui excessivament petit, pot ser interessant aplicar alguna tècnica de compressió de dades a les entrades per a aconseguir que ocupin menys espai i així augmentar L.

3.4.4. Introducció a la dispersió dinàmica

La dispersió que acabem d'explicar és una dispersió estàtica, en el sentit que el nombre de pàgines primàries N és fix. Un problema que té aquest tipus de dispersió és que, si el nombre de dades indexades creix més del previst, pot passar que el factor de càrrega C augmenti excessivament i el rendiment de l'índex empitjori.

Una solució és crear un nou índex amb un N' de pàgines primàries més gran que N, canviar de funció de dispersió per tal que retorni valors de [0, ..., N'-1] i no valors de [0, ..., N-1] i reinserir totes les entrades al nou índex emprant la nova funció de dispersió. Però aquesta solució és molt costosa.

La **dispersió dinàmica** té l'objectiu d'admetre l'increment dinàmic del nombre de pàgines primàries sense que calgui la reinserció de totes les entrades.

Algunes tècniques de dispersió dinàmica permeten modificar la funció de dispersió dinàmicament per a acomodar-se a l'augment o disminució de les dades indexades. En aquest mòdul, però, no descrivim les tècniques de dispersió dinàmica, dues de les quals, la dispersió extensible i la dispersió lineal, les podeu trobar explicades amb tots els detalls a l'obra de Ramakrishnan.

Lectura complementària

Trobareu l'explicació detallada de la dispersió extensible i la lineal a l'obra següent:

R. Ramakrishnan (1998). *Database Management Systems*. Boston: McGrawHill.

3.5. Índexs agrupats

Els índexs que permeten implementar els accessos seqüencials per valor (com ara els arbres B^+) poden ser **índexs agrupats*** o **índexs no agrupats****. Alguns sistemes, com ara Informix, permeten al dissenyador escollir una d'aquestes dues possibilitats quan crea un índex.

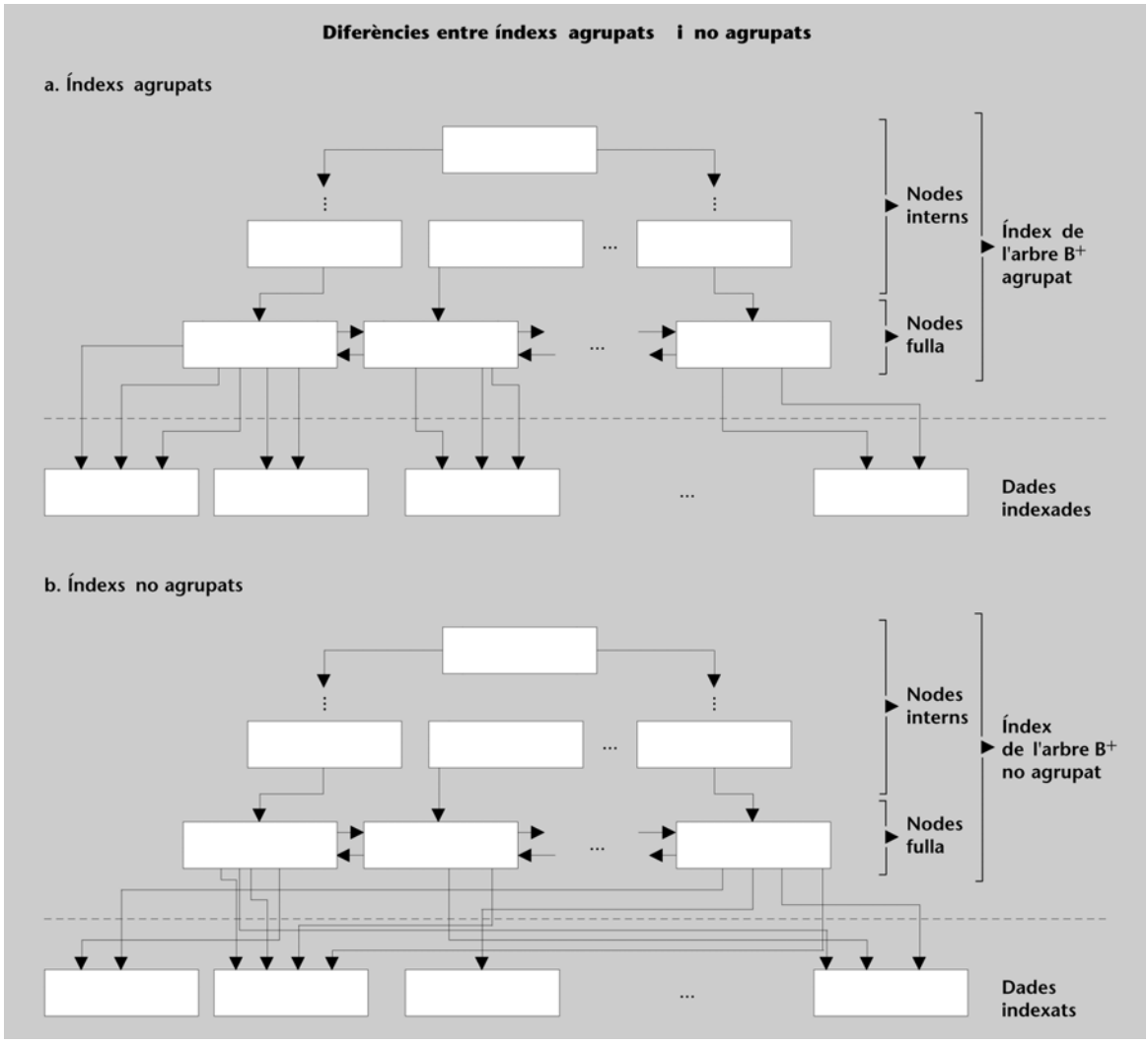
* En anglès, *clustered*.
** En anglès, *unclustered*.

Un **índex agrupat** és aquell en què les dades que indexa estan ordenades físicament segons l'accés seqüencial per valor que proporciona l'índex. En canvi, un **índex no agrupat** és un índex en què les dades indexades estan col·locades de manera aleatòria.

Nota

El concepte d'*índex agrupat* és diferent del d'*espai d'agrupació* que s'explica al mòdul "Components d'emmagatzematge d'una base de dades" d'aquesta assignatura, malgrat la similitud dels noms.

La figura següent il·lustra les diferències entre els índexs agrupats i els no agrupats:



El **cost d'un accés seqüencial per valor** varia molt segons si l'índex que ens proporciona l'accés és agrupat o no:

- Si l'índex és agrupat, els RID que s'obtinguin consecutivament apuntaran a files contigües. Llavors, caldrà fer molts accessos seguits a files de la mateixa pàgina (i es podran portar conjuntament amb una única E/S).

- En índexs no agrupats pot passar que la majoria de RID consecutius apuntin a files de pàgines diferents. Aquesta situació pot arribar a comportar la realització de tantes E/S com files a les quals calgui accedir.

Un aspecte que cal tenir en compte dels índexs agrupats és que l'ordenació física de les seves dades és difícil de mantenir quan es produeixen insercions i supressions. En la pràctica, aquesta dificultat es resol de la manera següent:

1) Inicialment, es deixa espai lliure a les pàgines que contenen les dades per tal de poder absorbir insercions futures.

2) Si l'espai lliure d'una pàgina s'esgota, les insercions futures a la pàgina es fan en pàgines d'excedents que s'hi encadenen.

3) Si hi ha moltes pàgines d'excedents, les dades es reorganitzen per a millorar el rendiment.

Tot i que podem tenir...

... diversos índexs sobre unes dades, només un d'ells pot ser agrupat, perquè les dades poden tenir una única ordenació física.

4. Implementació dels accessos per diversos valors

La implementació dels accessos per diversos valors és, en alguns casos, molt similar a la implementació dels accessos per un sol valor que ja hem estudiat a l'apartat anterior. En altres casos, però, presenten algunes dificultats addicionals. Explicarem en primer lloc la implementació dels accessos directes per diversos valors i a continuació analitzarem els accessos seqüencials i mixtos per diversos valors.

4.1. Implementació dels accessos directes

Considereu la relació EMPLEATS(*num_empl*, *nom_empl*, *num_despatx*, *sou*), que té un índex arbre B^+ definit per tal de facilitar els accessos per valor de l'atribut `num_despatx`, i un altre índex arbre B^+ per a facilitar els accessos per valor de l'atribut `sou`.

Suposem que es vol executar la sentència següent, que requereix un accés directe per diversos valors, concretament pels valors de `sou` i de `num_despatx`:

```
SELECT *
FROM empleats
WHERE num_despatx = 150 AND sou = 1200;
```

Una bona estratègia per a processar la consulta anterior és la que presentem tot seguit:

1) Fer servir l'índex de l'atribut `num_despatx` per a obtenir tots els RID deles files d'empleats que tenen el despatx 150.

2) Fer servir l'índex de l'atribut `sou` per tal de trobar totsels RID de files d'empleats que tenen el sou 1.200.

3) Fer la intersecció entre els dos conjunts de RID. Els RID de la intersecció corresponen a empleats que tenen alhora el despatx 150 i un sou de 1.200.

Aquesta és una bona estratègia perquè aprofita el fet de tenir dos índexs per a les dades dels empleats. Tot i això, en alguns casos concrets pot tenir un mal rendiment.

Per exemple, si hi ha molts empleats al despatx 150, molts empleats amb un sou de 1.200 i molt pocs empleats que compleixin les dues condicions alhora, caldrà examinar molts RID per a localitzar pocs empleats. Una solució més eficient per a aquest cas és definir un únic índex, en lloc de dos com en el cas anterior, que utilitzi valors compostos dels atributs `num_despatx` i `sou`.

Un **índex de valors compostos pels atributs** $[A_1, ..., A_i, ..., A_n]$ té la mateixa estructura que els altres índexs. L'única diferència és que els valors de l'índex seran, de fet, llistes d'elements $[v_1, ..., v_i, ..., v_n]$ on v_i és un valor de l'atribut A_i.

La gestió de l'índex fa necessari poder establir una relació d'ordre entre els valors compostos de l'índex. S'ordenen segons el primer element de la llista i si el primer element coincideix es fa d'acord amb el segon. Si aquest també coincideix, s'ordena segons el tercer, etc.

Exemple d'índex de valors compostos

L'índex de valors compostos pels atributs [`num_despatx`, `sou`] té valors com ara [150, 1.200], [150, 1.500], etc.

Per a establir una relació d'ordre entre els valors compostos de l'índex cal poder deduir per als dos valors anteriors [150, 1.200], [150, 1.500] quin és el més gran. Aquí, com que el primer element de la llista coincideix, s'utilitza el segon per a ordenar. S'obté que [150, 1.200] < [150, 1.500].

En general, l'ordre que hi ha entre dos valors compostos $v = [v_1, ..., v_i, ..., v_n]$ i $w = [w_1, ..., w_i, ..., w_n]$ s'estableix de la manera següent:

- Si $v_1 > w_1$, aleshores $v > w$; i si $v_1 < w_1$, aleshores $v < w$.
- Si $v_1 = w_1$ i $v_2 > w_2$, aleshores $v > w$; i si $v_1 = w_1$ i $v_2 < w_2$, aleshores $v < w$.
- Si $v_1 = w_1, ..., v_{i21} = w_{i21}$ i $v_i > w_i$, aleshores $v > w$; i si $v_1 = w_1, ..., v_{i21} = w_{i21}$ i $v_i < w_i$, aleshores $v < w$.
- Si $v_1 = w_1, ..., v_i = w_i, ..., v_n = w_n$, aleshores $v = w$.

4.2. Implementació dels accessos seqüencials i mixtos

Hem vist que els índexs de valors compostos pels atributs $[A_1, ..., A_i, ..., A_n]$ són una bona implementació dels accessos directes per diversos valors. En aquest subapartat comentarem la seva aplicació per tal d'implementar accessos seqüencials i mixtos per diversos valors i veurem que presenta algunes deficiències.

Considereu una altra vegada la relació EMPLEATS(*num_empl*, *nom_empl*, *num_despatx*, *sou*), i recordeu que té un índex arbre B^+ de valors compostos pels atributs [num_despatx, sou].

Suposem que es volen executar les sentències següents:

- Sentència 1:

```
SELECT *
FROM empleats
ORDER BY num_despatx, sou;
```

  Aquesta sentència correspon a un accés seqüencial per diversos valors. L'ordre de presentació dels empleats que requereix aquesta sentència coincideix amb l'ordre dels valors compostos de l'índex per [num_despatx, sou]. Així, doncs, l'índex ens permetrà processar la sentència de manera eficient.

- Sentència 2:

```
SELECT *
FROM empleats
WHERE num_despatx = 100 AND sou > 960;
```

  Aquesta sentència correspon a un accés mixt per diversos valors. Observeu que l'ordre dels valors compostos de l'índex per [num_despatx, sou] concorda amb l'ordre en què ens interessa obtenir els empleats per a processar la consulta. Així, l'índex ens permet també processar la consulta de manera eficient.

Malauradament, no passarà el mateix amb altres exemples d'accessos seqüencials o mixtos per diversos valors, perquè no hi haurà concordança entre l'ordre de l'índex per [num_despatx, sou] i l'ordre que requereixi la consulta. Aquest és el cas de les sentències següents:

- Sentència 3:

```
SELECT *
FROM empleats
ORDER BY sou, num_despatx;
```

- Sentència 4:

```
SELECT *
FROM empleats
WHERE sou = 1200 AND num_despatx > 100;
```

Per a les sentències 3 i 4 necessitaríem un índex [sou, num_despatx] i no pas l'índex [num_despatx, sou].

Utilitat d'un índex multidimensional

Amb un sol índex multidimensional pels atributs sou i num_despatx podríem processar totes les sentències de consulta 1, 2, 3 i 4 presentades en aquest subapartat.

Hi ha altres tipus d'índexs, anomenats **índexs multidimensionals per diversos atributs** [A_1, ..., A_i, ..., A_n], que no imposen un ordre lineal en el conjunt de valors indexats. El que fan és organitzar les dades segons una cert lligam espacial. Cada valor es considera un punt d'un espai *n*-dimensional on *n* és el nombre d'atributs de l'índex.

S'han proposat diversos tipus d'índexs multidimensionals: els arbres R, els arxius en retícula, etc. Es fan servir sobretot en SGBD destinats a emmagatzemar informació geogràfica, però pocs sistemes relacionals els incorporen.

Lectura complementària

Podeu trobar la descripció dels arbres R i dels arxius en retícula a l'obra següent:

A. Silberschatz; H.F. Korth; S. Sudarshan (1998). *Fundamentos de bases de datos* (3a. ed.). Madrid: McGraw-Hill.

5. Índexs del sistema Informix

El sistema Informix posa a l'abast del dissenyador alguns dels índexs que hem estudiat en aquest mòdul.

5.1. Accessos per valor

Per a implementar accessos per valor, directes o seqüencials, el sistema Informix proporciona índexs arbre B^+ per un atribut que poden ser agrupats o no.

Per exemple, si tenim una relació EMPLEATS(*num_empl*, *nom_empl*, *num_despatx*, *sou*), podem declarar un índex per a accedir per valor de l'atribut anomenat `num_despatx` així:

```
CREATE INDEX index_num_despatx ON empleats(num_despatx);
```

Si es desitja que l'accés seqüencial sigui per ordre descendent cal declarar-ho explícitament (perquè per defecte l'ordre es considera ascendent):

```
CREATE INDEX index_num_despatx ON empleats(num_despatx DESC);
```

Podem aconseguir que l'índex sigui agrupat si declarem:

```
CREATE INDEX CLUSTER index_num_despatx ON empleats (num_despatx);
```

Finalment, si volem que l'índex no admeti valors repetits, la sentència serà:

```
CREATE DISTINCT INDEX index_num_despatx ON empleats (num_despatx);
```

5.2. Accessos per diversos valors

Per a implementar accessos per diversos valors, el sistema Informix proporciona índexs arbre B^+ de valors compostos que també poden ser agrupats o no.

Per exemple, si a la relació EMPLEATS(*num_empl*, *nom_empl*, *num_despatx*, *sou*) volem declarar un índex de valors compostos pels atributs [num_despatx, sou], farem:

```
CREATE INDEX index_desp_sou ON empleats(num_despatx, sou);
```

Si volem que l'accés seqüencial per algun dels atributs sigui descendent, cal declarar-ho explícitament. Per exemple, si volem que l'ordenació dels sous sigui descendent:

```
CREATE INDEX index_desp_sou ON empleats(num_despatx, sou DESC);
```

També podem aconseguir que l'índex sigui agrupat:

```
CREATE INDEX CLUSTER index_desp_sou ON empleats (num_despatx, sou);
```

Finalment, si es desitja que l'índex no admeti valors repetits, caldrà declarar:

```
CREATEDISTINCTINDEXindex_desp_souONempleats(num_despatx,sou);
```

Resum

En aquest mòdul didàctic hem identificat mètodes d'accés que són necessaris per a poder fer diferents tipus de consultes i actualitzacions a les BD. Aquests mètodes d'accés són els accessos directes i seqüencials per posició, els accessos directes i seqüencials per valor i, finalment, els accessos per diversos valors, que poden ser directes, seqüencials o mixtos.

Hem explicat que la implementació dels accessos per posició es fonamenta gairebé completament en els serveis proporcionats per l'SO i que, en canvi, la implementació eficient dels accessos per un o diversos valors és més complexa i requereix que l'SGBD disposi d'estructures pròpies especialitzades.

Hem descrit algunes de les estructures que els SGBD fan servir per a implementar els accessos per un o diversos valors: índexs del tipus arbre B^+, índexs estructurats segons funcions de dispersió, índexs agrupats i índexs de valors compostos. Hem mostrat l'impacte que poden tenir aquestes estructures en el nombre d'E/S necessari per a implementar els accessos a les dades.

Finalment, hem comentat quines de les estructures anteriors proporcionen un sistema concret: el sistema Informix i la seva declaració en el llenguatge SQL del sistema.

Per tal de simplificar, hem considerat només els accessos que es fan a dades que es troben emmagatzemades en una única taula situada en un únic espai virtual. De totes maneres, és fàcil adonar-se que les implementacions que hem estudiat són directament aplicables a accessos en els quals intervenen diverses taules si aquestes es troben situades en un únic espai d'agrupació.

Activitats

1. Consulteu a Ramakrishnan (1998) la descripció de la dispersió extensible.

2. Consulteu a Ramakrishnan (1998) la descripció de la dispersió lineal.

Exercicis d'autoavaluació

1. Donada la taula ESTUDIANTS(*num_matr*, *nom*, *telefon*, *any_inici_estudis*, *ciutat*), digueu quins mètodes d'accés impliquen les sentències d'SQL següents:
a) Sentència 1:

```
SELECT *
FROM estudiants
WHERE any_inici_estudis >= 1994;
```

b) Sentència 2:

```
SELECT num_matr, nom
FROM estudiants;
```

c) Sentència 3:

```
DELETE
FROM estudiants;
WHERE any_inici_estudis = 1950;
```

d) Sentència 4:

```
SELECT *
FROM estudiants
WHERE any_inici_estudis = 1998 AND ciutat = 'Girona';
```

2. Mostreu l'índex arbre B^+ d'ordre 2 que s'obté després d'inserir en un arbre buit els valors 10, 22, 64, 35, 71, 33, 12, 4, 85, 56, 63, 41, 90, 8, 52, 98.

3. Prenent sempre com a punt de partida l'arbre obtingut a l'exercici d'autoavaluació 2, mostreu l'índex arbre B^+ que s'obté després de suprimir-ne els valors següents:
a) El valor 41.
b) El valor 90 i a continuació el valor 98.
c) El valor 85 (cal suprimir totes les còpies del 85).

4. Considereu un arbre B^+ d'ordre 100 i de 3 nivells ($h = 3$):
a) Quants valors indexaria aquest arbre si tots els nodes estiguessin ocupats al 100%?
b) Quants valors indexaria si tots els nodes tinguessin una ocupació del 50% excepte l'arrel, que contindria un sol valor?

5. Suposeu que teniu un índex basat en la dispersió amb 7 pàgines primàries ($N = 7$). Cada pàgina té una capacitat de 3 entrades ($L = 3$); la funció de dispersió que s'utilitza és $h(x) = (x \bmod 7) + 1$ i la gestió d'excedents és l'encadenament separat. Mostreu el que s'obté si inserim en aquest índex inicialment buit els valors 7, 14, 38, 40, 49, 31, 21, 56, 52, 28, 70 i 17.

6. Suposeu que tenim un índex basat en la dispersió amb gestió d'excedents d'encadenament separat i amb pàgines de 4 kB. Suposeu també que els valors indexats ocupen 25 bytes, els RID ocupen 4 bytes i els apuntadors a pàgines, 3 bytes.
a) Indiqueu el nombre màxim d'entrades (L) que pot contenir cada pàgina de l'índex.
b) Si L és l'obtinguda al punt anterior, N (nombre de pàgines primàries) és 130 i M (nombre d'entrades) és 11.000, digueu quin serà el factor de càrrega C de l'índex.

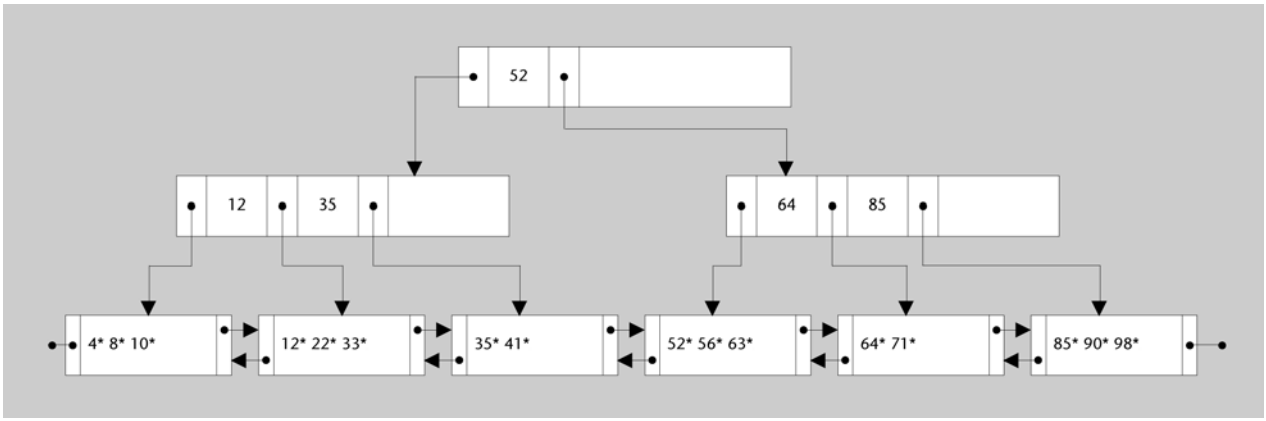
Solucionari

Exercicis d'autoavaluació

1. Cada sentència presentada a l'enunciat implica el mètode d'accés següent:

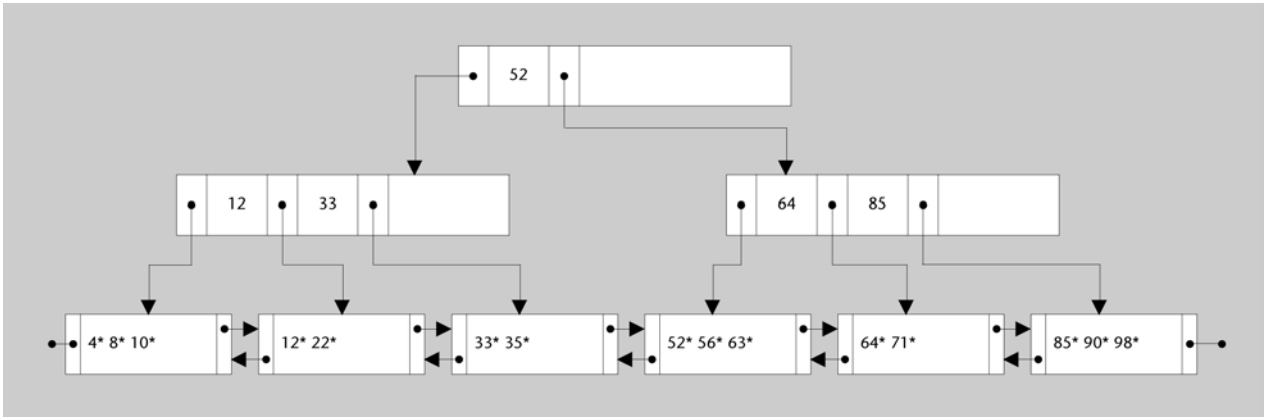
a) Accés seqüencial per valor.
b) Accés seqüencial per posició.
c) Accés directe per valor.
d) Accés directe per diversos valors.

2. Després d'inserir tots els valors en un arbre buit, s'obté l'índex arbre B^+ següent:

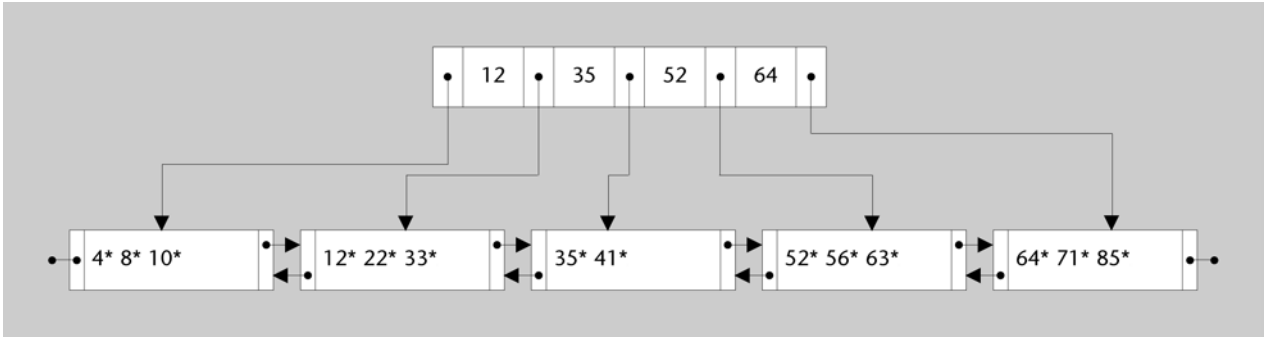


3. Partint de l'arbre obtingut a l'exercici 2:

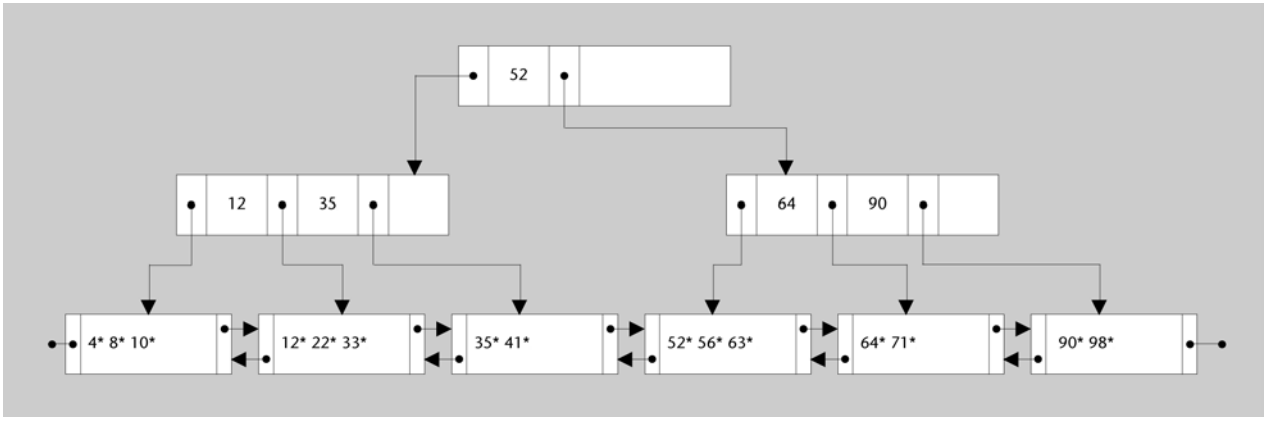
a) Si se suprimeix el valor 41 s'obté l'índex arbre B^+ següent:



b) Si se suprimeixen els valors 90 i 98, per aquest ordre, l'índex arbre B^+ que s'obté és el següent:



c) Si se suprimeixen totes les còpies del 85 s'obté l'índex arbre B^+ següent:

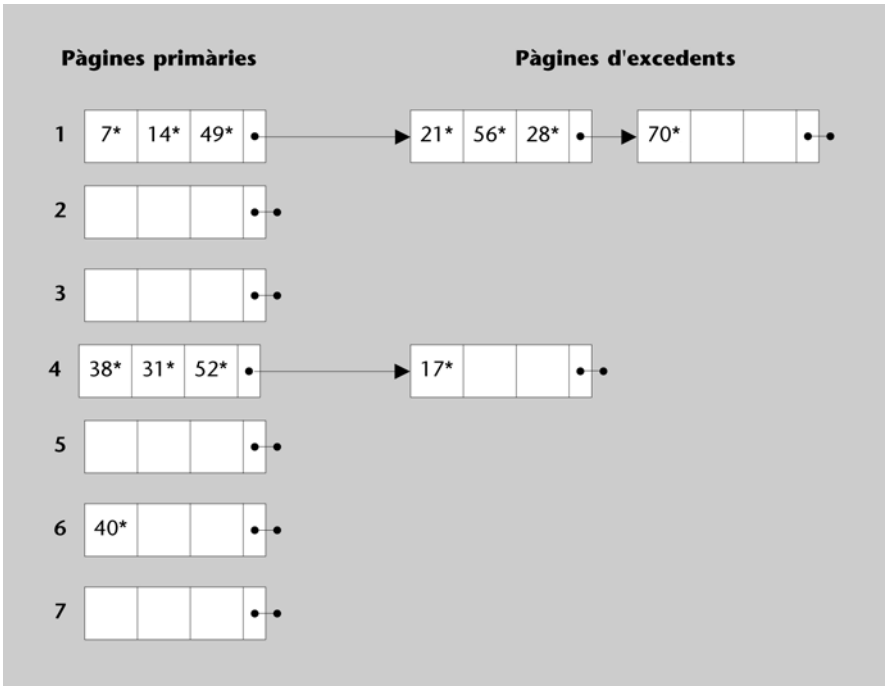


4. A partir de l'arbre considerat, es tenen els resultats següents:
a) L'arbre indexaria 8.080.200 valors. Concretament, tindria:
- Nivell 1 → 1 node amb 200 valors i 201 apuntadors.
- Nivell 2 → 201 nodes amb 200 valors cadascun i 201 apuntadors cadascun.
- Nivell 3 → 40.401 nodes amb 200 entrades cadascun.

En total, doncs, es tenen 8.080.200 entrades.
b) L'arbre indexaria 20.200 valors. Concretament, tindria:
- Nivell 1 → 1 node amb 1 valor i 2 apuntadors.
- Nivell 2 → 2 nodes amb 100 valors cadascun i 101 apuntadors cadascun.
- Nivell 3→ 202 nodes amb 100 entrades cadascun.

Per tant, 20.200 entrades en total.

5. Després d'inserir-hi tots els valors, s'obtindria l'índex següent:



6. En les circumstàncies exposades tenim els resultats següents:
a) *L* és 141.
b) *C* és 0,6.

Glossari

accés directe per posició *m* Mètode d'accés que consisteix a obtenir una pàgina que té un número de pàgina determinat dins un espai.

accés directe per valor *m* Mètode d'accés que consisteix a obtenir totes les files que contenen un determinat valor per un atribut.

accés per diversos valors *m* Mètode d'accés que consisteix a obtenir diverses files segons els valors de diversos atributs. Pot ser directe, seqüencial o mixt.

accés seqüencial per posició *m* Mètode d'accés que consisteix a anar obtenint les pàgines d'un espai seguint l'ordre definit pels seus números de pàgina.

accés seqüencial per valor *m* Mètode d'accés que consisteix a obtenir diverses files per ordre dels valors d'un atribut.

arbre B^+ *m* Estructura de dades que s'empra per a organitzar índexs que permeten implementar l'accés directe i l'accés seqüencial per valor.

dispersió *f* Manera d'organitzar valors que es pot fer servir en índexs que implementen l'accés directe per valor.

entrada *f* Element d'un índex que consisteix en una parella formada per valor i un RID.

E/S *f* Abreviació d'*entrada/sortida*.

espai virtual *m* Seqüència de pàgines virtuals.
sigla: EV

índex *m* Estructura de dades auxiliar que els SGBD fan servir per facilitar les cerques necessàries per a implementar els accessos per un o diversos valors.

índex agrupat *m* Índex que proporciona l'accés seqüencial per valor (i també l'accés directe per valor) i que indexa dades que estan ordenades físicament segons l'ordre de l'accés seqüencial per valor proporcionat.

índex de valors compostos *m* Índex de valors compostos pels atributs $[A_1, ..., A_i, ..., A_n]$. Té la mateixa estructura que els altres índexs, però amb la diferència que els valors de l'índex són, de fet, llistes de valors $[v_1, ..., v_i, ..., v_n]$ en les quals cada v_i és un valor de l'atribut A_i.

mètode d'accés *m* Tipus d'accés a les dades emmagatzemades per un SGBD.

pàgina real *f* Unitat de transferència de dades entre la memòria interna de l'ordinador i els fitxers d'una base de dades emmagatzemats a la memòria permanent.

pàgina virtual (o pàgina) *f* Imatge de la pàgina real.

RID *m* Identificador de fila.

SGBD *m* Sistema de gestió de bases de dades.

SO *m* Sistema operatiu.

VAF *m* Vector d'adreces de fila.

Bibliografia

Elmasri, R.; Navathe, S.B. (2000). *Sistemas de bases de datos. Conceptos fundamentales* (3a. ed.). Madrid: Addison-Wesley Iberoamericana.

Ramakrishnan, R. (1998). *Database Management Systems*. Boston: McGraw-Hill.

Silberschatz, A.; Korth, H.F.; Sudarshan, S. (1998). *Fundamentos de bases de datos* (3a. ed.). Madrid: McGraw-Hill.

Annexos

Annex 1

Lectura complementària

Podeu trobar l'algoritme d'inserció que presentem aquí a l'obra següent:

R. Ramakrishnan (1998). *Database Management Systems*. Boston: McGrawHill.

A continuació descrivim un **algoritme d'inserció** (a alt nivell i seguint un enfocament de programació estructurada) que serveix per a fer insercions en els arbres B^+. La descripció de l'algoritme l'hem extreta, amb petites variacions, de l'obra que esmentem al marge.

El propòsit de l'algoritme és inserir l'entrada mitjançant la invocació recursiva de l'algoritme d'inserció en el subarbre del node fill apropiat. Habitualment, això suposa descendir fins al node fulla corresponent a l'entrada, inserir-la i després tornar enrere cap a l'arrel.

Pot ser que el node fulla estigui ple i s'hagi de dividir. Quan un node es divideix, cal inserir en el seu pare un valor i un apuntador cap al nou node fill. En el pseudocodi representen aquest valor, juntament amb l'apuntador, per nnf (*nou_node_fill*). Aquest fet pot provocar divisions successives de nodes. Si es divideix el node arrel, es crea una arrel nova i l'alçada de l'arbre s'incrementa en 1.

Nota

Cal remarcar que la divisió d'un node fulla i d'un node no-fulla és diferent.

Algoritme d'inserció

```
accio inserir (apuntador_node apn, entrada e, nou_node_fill nnf)
{insereix l'entrada al subarbre amb arrel apuntada per apn, d és l'ordre de l'arbre, nnf és nul
inicialment i nul a l'acabament, llevat del cas que algun node hagi estat dividit}
si apn apunta a un node intern
   llavors
      sigui N el node intern apuntat per apn
      trobar i a N tal que v_i ≤ valor v d'e<v_{i+1}
      {escollir subarbre adient per l'entrada e}
      inserir(f_i, e, nnf)
      {f_i és apuntador a node dins N}
      si nnf no és nul
         llavors
            {el fill ha estat dividit i cal inserir nnf a N}
            si a N tenim espai lliure
               llavors
                  afegir nnf a N
                  nnf es posa a nul
               sino
                  {cal dividir N, observeu que la divisió dels nodes interns
                  és diferent a la divisió dels nodes fulla}
```

```
                    dividir N:
                        els d primers valors i d+1 primers apuntadors es queden a N
                        els d darrers valors i d+1 darrers apuntadors marxen a un nou node N2
                    nnf:=<valor d+1 del node N abans de dividir-lo, apuntador a N2>
                    si el node N era l'arrel llavors
                        {l'arrel s'acaba de dividir}
                        crear un nou node arrel amb <apuntador a N, nnf>
                        fer que l'apuntador a l'arrel apunti al nou node arrel
                    fsi
                fsi
            fsi
        fsi

        si apn apunta a un node fulla
            llavors
                sigui F el node fulla apuntat per apn
                si F té espai lliure
                    llavors
                        {si hi ha espai lliure a F no cal dividir el node fulla}
                        afegir l'entrada e a F
                        nnf es posa a nul
                    sino
                        {cal dividir F}
                        dividir F:
                            les d primeres entrades es queden a F
                            les d+1 entrades que resten marxen a un nou node F2
                        nnf:=<valor més petit de F2, apuntador a F2>
                        actualitzar els apuntadors aa i as de F i F2
                        si F era l'arrel
                            llavors
                                {l'arrel s'acaba de dividir}
                                crear un nou node arrel amb <apuntador a F, nnf>
                                fer que l'apuntador a l'arrel apunti al nou node arrel
                        fsi
            fsi
        fsi
        faccio
```

Annex 2

A continuació descrivim un **algoritme de supressió** (a alt nivell i seguint un enfocament de programació estructurada) que serveix per a eliminar valors dels arbres B^+. La descripció de l'algoritme l'hem extreta, amb petites variacions, de l'obra que esmentem al marge.

Lectura complementària

Podeu trobar la descripció de l'algoritme de supressió presentat al text a l'obra següent:

R. Ramakrishnan (1998). *Database Management Systems*. Boston: McGrawHill.

El propòsit de l'algoritme és esborrar l'entrada mitjançant la invocació recursiva de l'algoritme de supressió en el subarbre del node fill apropiat. Habitualment, això suposa descendir fins al node fulla corresponent a l'entrada, esborrar-la i després tornar enrere cap a l'arrel.

De vegades, el node fulla té, després de l'esborrament, menys ocupació que la mínima permesa a l'arbre B^+ (menys de d valors) i aleshores cal redistribuir entrades amb un node germà o fusionar el node amb un node germà.

Si es redistribueixen entrades cal modificar el node pare per a reflectir-ho: el valor que precedeix l'apuntador al segon node s'ha de canviar perquè sigui més petit o igual que el valor més petit del segon node. Quan es fusionen dos nodes, el seu node pare també s'ha de modificar perquè quedi reflectit: cal esborrar del node pare el valor intermedi als dos nodes fusionats i l'apuntador al segon node. En el pseudocodi representen aquest valor, juntament amb l'apuntador, per anf (*antic_node_fill*). Aquest fet pot provocar fusions successives de nodes i/o redistribucions. Si s'esborra del node arrel el darrer valor que contenia, l'alçada de l'arbre disminueix en 1.

Algoritme de supressió

```
accio suprimir (apuntador_node apnp, apuntador_node apn, entrada e, antic_node_fill anf)
{esborra l'entrada al subarbre amb arrel apuntada per apn, d és l'ordre de l'arbre, anf és null inicialment i nul a
l'acabament, llevat del cas que dos nodes s'hagin fusionat, apnp és l'apuntador al node pare del node apuntat per apn}
si apn apunta a un node intern
    llavors
        sigui N el node intern apuntat per apn
        trobar i a N tal que vi ≤ valor v d'e<vi+1
        {escollir subarbre adient per l'entrada e}
        suprimir(apn, fi, e, anf)
        {fi és apuntador a node dins N}
        si anf no és nul
            llavors
                {dos nodes s'han fusionat, cal eliminar anf de N}
                eliminar anf de N
                {implica eliminar un valor de N i l'apuntador al node que ha estat descartat}
                si N és el node arrel
                    llavors
                        si N es queda amb 0 valors
                            llavors
                                fer que l'apuntador a l'arrel de l'arbre B+ apunti a la nova arrel
                                {la nova arrel serà l'apuntador f0 de N, fixeu-vos que l'alçada de l'arbre haurà
                                disminuït en una unitat}
                        fsi
```

```
sino
    {N és un node intermedi}
    si N es queda amb d o més valors
        llavors
            anf es posa a nul
        sino
            {cal corregir l'ocupació de N, observeu les diferències respecte del cas dels nodes fulla}
            sigui G el node germà de N
            {G és el node germà dret si existeix, sinó s'obté el germà esquerre}
            si G té exactament d valors
                llavors
                    {Cal fusionar N i G}
                    fusionar N i G:
                        {també cal considerar el node pare de N i G, el qual està apuntat per
                        apnp}
                        sigui P el node apuntat per apnp
                        si G és germà dret
                            llavors
                                anf:=<valor v_i de P que discriminava entre els valors existents
                                        a N i G, apuntador f_i de P que apunta a G>
                            sino {G és germà esquerre}
                                anf:=<valor vi de P que discriminava entre els valors existents
                                        a N i G, apuntador f_i-1 de P que apunta a G>
                        fsi
                        moure tots els valors de G a N
                        afegir el valor v_i de P al node N
                        descartar el node G que ha quedat buit
                sino
                    {redistribució per corregir l'ocupació del node N, també és necessari
                    tenir en compte el node pare de N i G, el qual està apuntat per apnp}
                    sigui P el node apuntat per apnp
                    si G és germà dret
                        llavors
                            redistribuir valors entre N i G:
                                el valor de P que discriminava entre N i G es mou a N,
                                desapareixent de P
                                el valor mínim de G es mou a P, desapareixent de G
                            anf es posa a nul
                        sino {G és germà esquerre}
                            redistribuir valors entre N i G:
                                el valor P que discriminava entre N i G es mou a N,
                                desapareixent de P
                                el valor màxim de G es mou a P, desapareixent de G
                                    anf es posa a nul
                    fsi
```

```
                        fsi
                    fsi
                fsi
            fsi
        fsi

si apn apunta a un node fulla
    llavors
        sigui F el node fulla apuntat per apn
        esborrar l'entrada e de F
        si F es queda amb d o més entrades
            llavors
                {F té ocupació suficient}
                anf es posa a nul
            sino
                {F tenia exactament d entrades i després de la supressió s'ha quedat amb una ocupació insuficient}
                sigui G el node germà de F
                {G és el node germà dret si existeix, sinó s'obté el germà esquerre}
                si G té exactament d entrades
                    llavors
                        {cal fusionar F i G}
                        fusionar F i G:
                            {també cal considerar el node pare de G i F, el qual està apuntat per apnp}
                            sigui P el node apuntat per apnp
                            si G és germà dret
                                llavors
                                    anf:=<valor v_i de P que discriminava entre els valors existents a F i G,
                                            apuntador f_i de P que apunta a G>
                                sino {G és germà esquerre}
                                    anf:=<valor v_i de P que discriminava entre els valors a F i G,
                                            apuntador f_i-1 de P que apunta a G>
                            fsi
                            moure totes les entrades de G a F
                            actualitzar els apuntadors a_a i a_s
                            descartar el node G que ha quedat buit
                    sino {redistribució per corregir l'ocupació del node F}
                        si G és germà dret
                            llavors
                                redistribuir valors entre F i G:
                                    entrada mínima de G es mou a F
                                    {el valor associat a aquesta entrada també està al node pare de G i F,
                                    el qual està apuntat per apnp}
                                    sigui P el node apuntat per apnp
                                    copiar en P el nou valor mínim de G, substituïnt a l'antic mínim
                            anf es posa a nul
```

```
                    sino {G és germà esquerre}
                        redistribuir valors entre F i G:
                            entrada màxima de G es mou a F
                            sigui P el node pare de G i F que està apuntat per apnp
                            copiar en P el nou valor mínim de F, sustituïnt a l'antic mínim
                        anf es posa a nul
                fsi
            fsi
        fsi
    fsi
    faccio
```