Poglavje 7

Indeksi z uporabo drevesnih struktur

Povzeto po [1]

Indeksi – uvod..

- Včasih hočemo najti vse zapise, ki imajo določeno vrednost v določenem polju
- INDEKS je zunanja podatkovna struktura, ki pomaga najti zapise, ki ustrezajo določenemu pogoju
- INDEKS lahko bistveno pospeši iskanje

Indeksi – uvod..

- Vsak indeks vsebuje **iskalni ključ**, ki je sestavljen iz **enega ali več polj** zapisov, ki se nahajajo v datoteki. Iskalni ključ je lahko katerikoli podmnožica polj zapisov v datoteki
- **Primer**: Indeks nad tabelo ŠTUDENT ima lahko indekse nad naslednjimi polji oz. iskalnimi ključi:
  - Indeks1: <Vpisna stevilka>
  - Indeks2: <ime, priimek>
  - Indeks3: <kraj, ime, priimek>

Indeksi – uvod..

- Datoteki, ki jo indeksiramo, pravimo tudi **indeksirana datoteka**

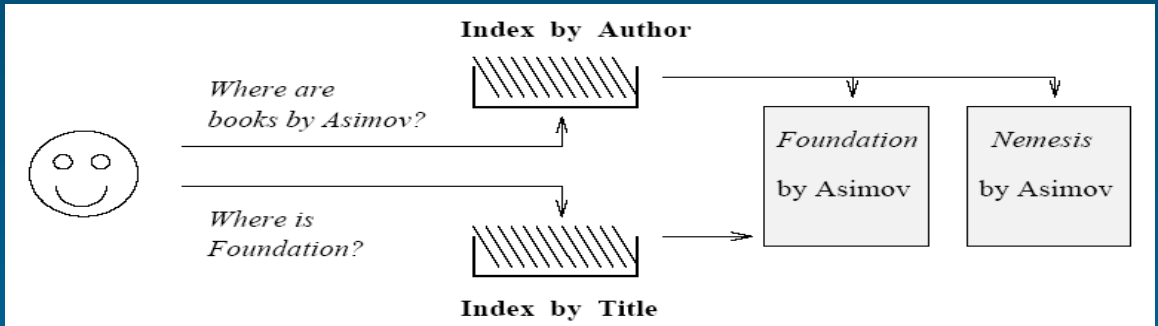


Figure 7.6 Indexes in a Library

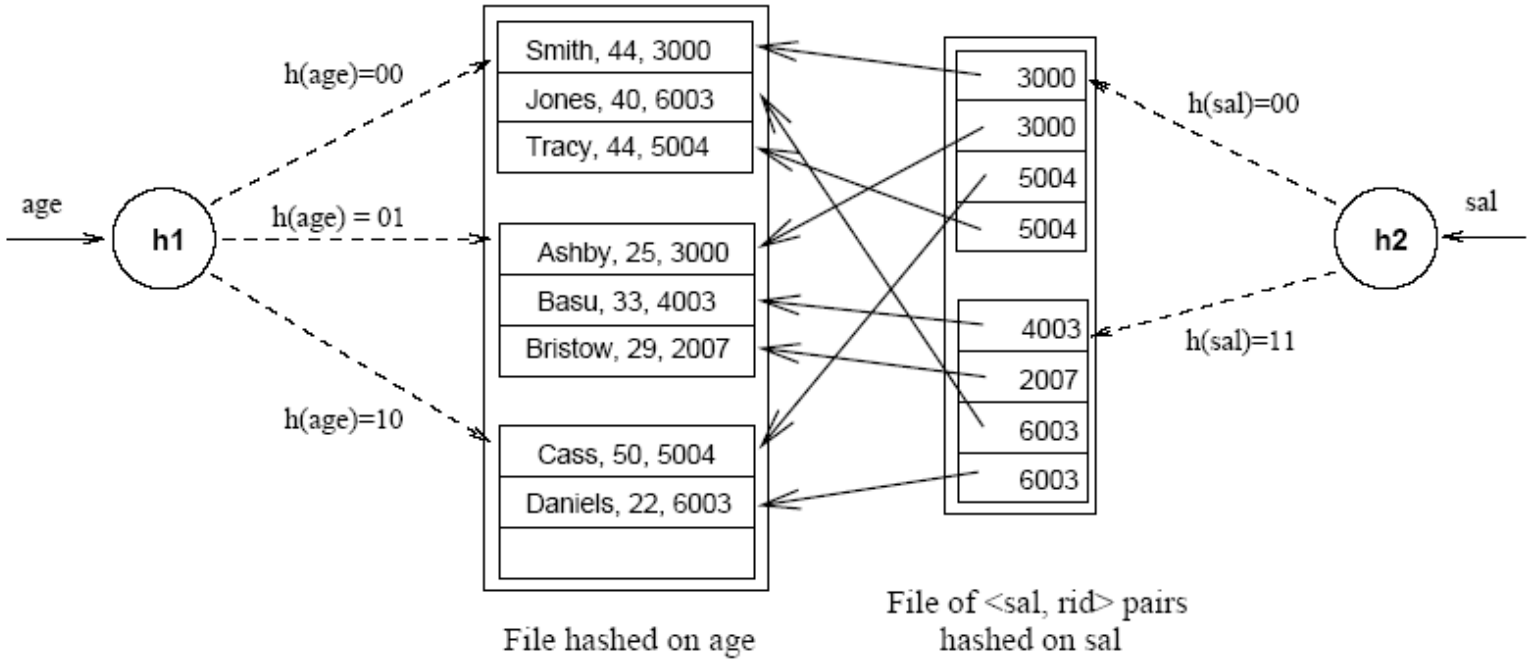
Indeksi – uvod..

- Indeks je datoteka, katere vsebina omogoča **hiter dostop** do zapisov v osnovni datoteki, ki je z indeksom indeksirana
- Pod hitrim dostopom pojmujemo dostop, ki je (občutneje) hitrejši, kot pa ga sicer omogoča organizacija osnovne datoteke
- Indeks si lahko predstavljamo **kot zbirko vpisov** *k**, kjer je *k* ključ indeksa, * pa kazalec(ci) na zapise, ki imajo polja v zapisu enaka ključu indeksa

Indeksi – uvod..

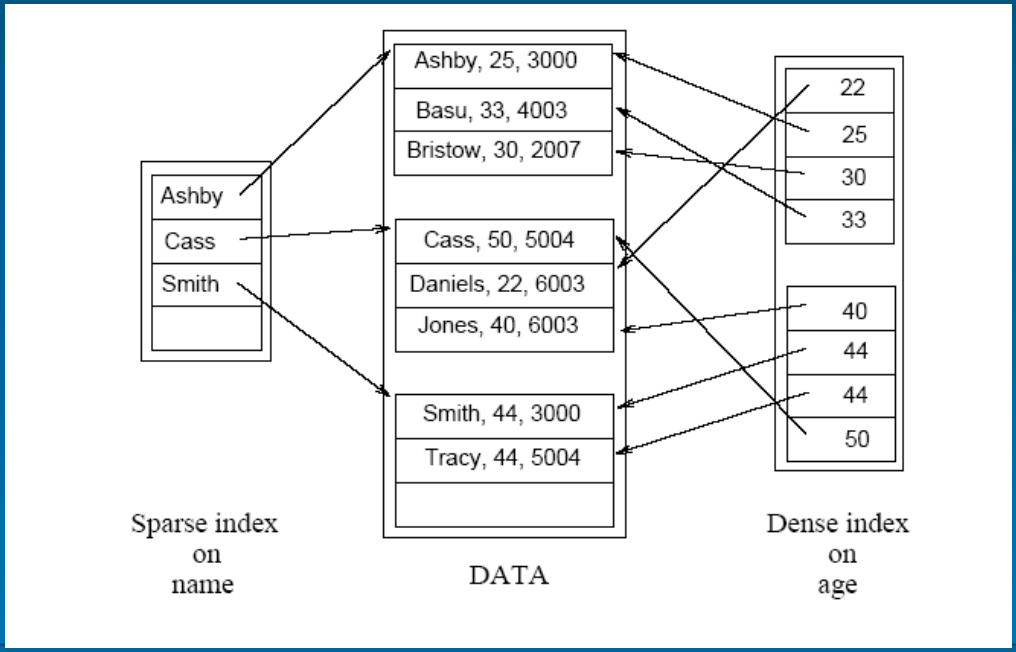
- Pojavita se 2 vprašanji:
  - Kako so podatkovni vpisi v indeksu organizirani, da bodo podpirali hitro in učinkovito iskanje zapisov?
    - Ena rešitev za organizacijo indeksa je uporaba razpršene datoteke (naslednja slika), ki omogoča hitro iskanje zapisov
    - Obstaja tudi več drugih indeksnih struktur, ki omogočajo hitro iskanje in bodo predstavljene v nadaljevanju
  - Kaj natančno je shranjeno v podatkovnem vpisu (Podatkovni vpis *k** nam omogoča dobiti enega ali več zapisov z vrednostjo ključa *k*)
    1. k* je dejanski zapis (z iskalnim ključem k),
    2. podatkovni vpis je par <k, rid>, kjer rid predstavlja *record id* zapisa z vrednostjo indeksnega ključa k,
    3. podatkovni vpis je par <k, rid-list>, kjer je rid-list seznam *record id-jev* tistih zapisov, ki imajo v poljih vrednost enako indeksnemu ključu.

Indeksi – uvod



Vrste indeksov..

- **Gosti indeks**: na vsak zapis osnovne datoteke, kaže kazalec iz indeksa
- **Redki indeks**: kazalci iz indeksa kažejo na skupine zapisov osnovne datoteke



Vrste indeksov..

- **Primarni indeks**: indeksiranje osnovne datoteke je izvedeno po njenem ključu (ključ indeksa in primarni ključ osnovne datoteke sta enaka)
- **Sekundarni indeks**: indeksiranje osnovne datoteke je izvedeno po podatkovnem elementu (polju), ki nastopa v zapisu osnoven datoteke, vendar pa ni ključ

Vrste indeksov..

- **Enonivojsko indeksiranje**: indeksirana je osnovna datoteka – v indeksu poiščemo kazalec na polje ali skupino polj v osnovni datoteki, kjer nato poiščemo iskani zapis
- **Večnivojsko indeksiranje**: indeksirana je osnovna datoteka, indeksiran je indeks na osnovno datoteko, indeksiran je indeks na indeks itd.

Vrste indeksov

- **Statično indeksiranje**: indeks ostaja nespremenjen, čeprav se osnovni datoteki zapisi dodajajo ali iz nje brišejo; ko postane iskanje neučinkovito, se izvede reorganizacija indeksa
- **Dinamično indeksiranje**: indeks se ob dodajanju in brisanju zapisov osnovne datoteke prilagaja vsebini, tako da so iskalne poti do vseh zapisov v osnovni datoteki enako dolge; indeksa ni potrebno nikoli reorganizirati

Primarni in sekundarni indeks

- Primarni indeks ne vsebuje duplikatov
- Sekundarni indeks lahko vsebuje duplikate
- Če vemo, da duplikati ne obstajajo, potem pravimo temu indeksu **unique indeks**
- Za dva podatkovna vpisa pravimo, da sta **duplikata**, če vsebujeta enaki vrednosti za iskalni ključ

Indeksi s kompozitnimi iskalnimi ključi

- Indeksni ključ lahko vsebuje več polj. Takemu ključu pravimo **kompozitni iskalni ključ**
- Slika prikazuje razliko med kompozitnima iskalnima ključema <age,sal> in <sal,age>
- Če imamo indeks z indeksnim ključem, lahko izvajamo poizvedbe, pri katerih iščemo zapise z vrednostmi polj, ki so enake ključu (to imenujemo **equality query**)
- Druga vrsta poizvedbe pa nam omogoča, da je samo ena vrednost v ključu konstantna, druge pa ne. Tako lahko pri poizvedbi dobimo več zapisov (**range query**)

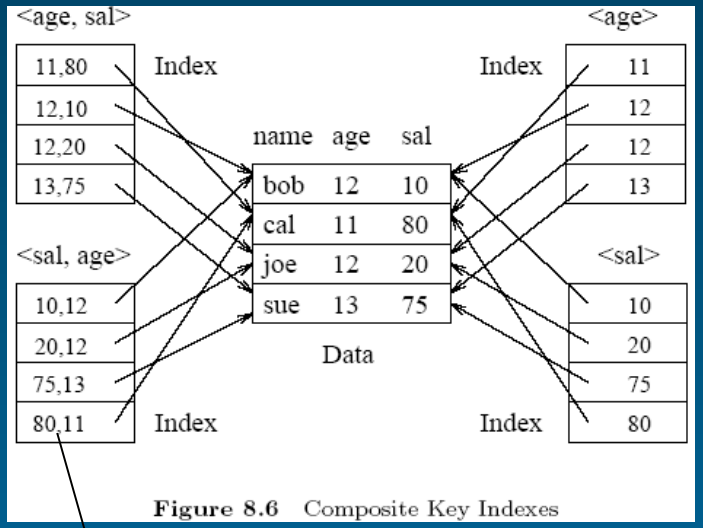


Figure 8.6 Composite Key Indexes

Specifikacija indeksa v SQL 92

- Standard SQL 92 ne opredeljuje nobenega stavka za kreiranje ali brisanje indeksov. Še več, standard sploh ne zahteva implementacije indeksov
- V praksi je drugače; vsi komercialni SUPB podpirajo eno več vrst indeksiranja
- Primer SQL stavka za kreiranje B+ drevesnega indexa:

```
CREATE INDEX IndAgeRating ON Students
       WITH  STRUCTURE = BTREE,
             KEY = (age, gpa)
```

Drevesne strukture za indekse

- ISAM (Indexed Sequential Access Method):
  - je statičen indeks,
  - učinkovit v primerih, ko osnovna datoteka ne spreminja pogosto,
  - ni učinkovit pri datotekah, ki se hitro povečujejo ali krčijo.
- B+ index:
  - je dinamična struktura, ki se zelo dobro prilagaja spremembam v osnovni datoteki podatkov,
  - B+ indeks je najbolj uporabljana vrsta indeksa, saj omogoča hitro iskanje elementov, iskanje intervalov in se učinkovito prilagaja spremembam v osnovni datoteki.

ISAM..

- Za primer vzemimo sortirano datoteko oseb, po atributu “starost”
- Če hočemo najti vse osebe starejše od 30 let, moramo najprej najti prvo (s pomočjo binarnega iskanja), ki je starejša od 30 in od te naprej prebrati preostale zapise datoteke
- Lahko bi izdelali dodatno datoteko, s po enim zapisom za vsako stran v osnovni datoteki (dodatna datoteka je tudi urejena po atributu starost)

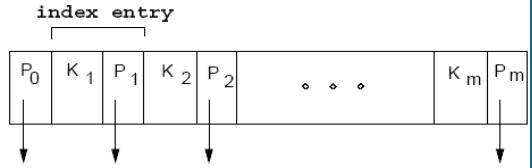


Figure 9.1 Format of an Index Page

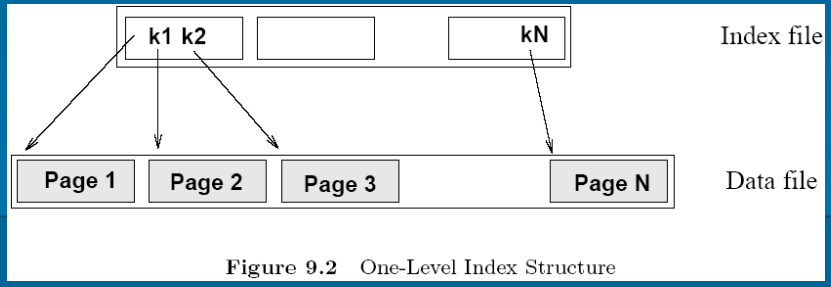


Figure 9.2 One-Level Index Structure

ISAM..

- Format strani v indeksni datoteki prikazuje leva slika. Vsaka stran v indeksu vsebuje en kazalec več, kot pa je ključev na strani. Ključ predstavlja separator vsebine, na katero kažeta levi in desni kazalec
- Binarno iskanje se sedaj izvede nad indeksno datoteko, ki je manjša od osnovne datoteke. Na ta način smo dosegli hitrejše iskanje

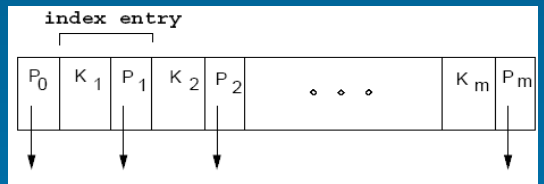


Figure 9.1 Format of an Index Page

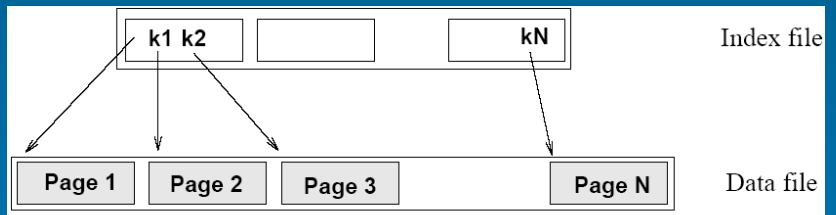
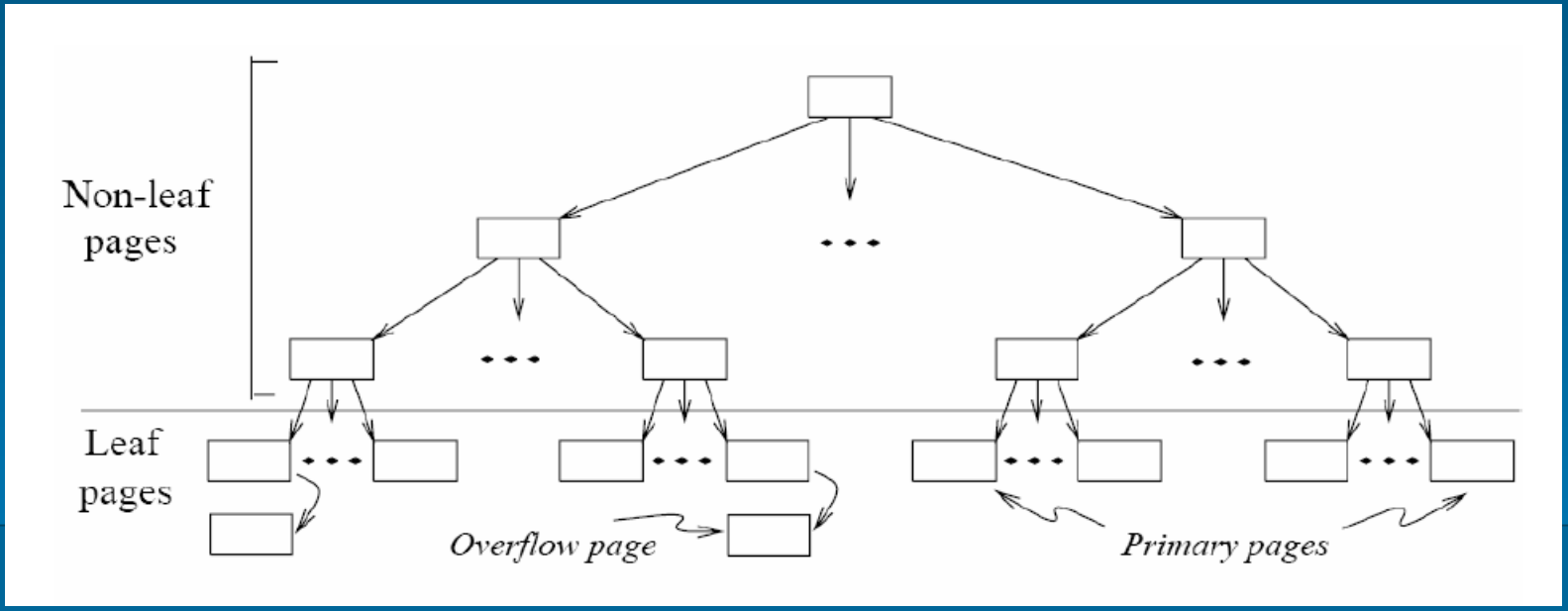


Figure 9.2 One-Level Index Structure

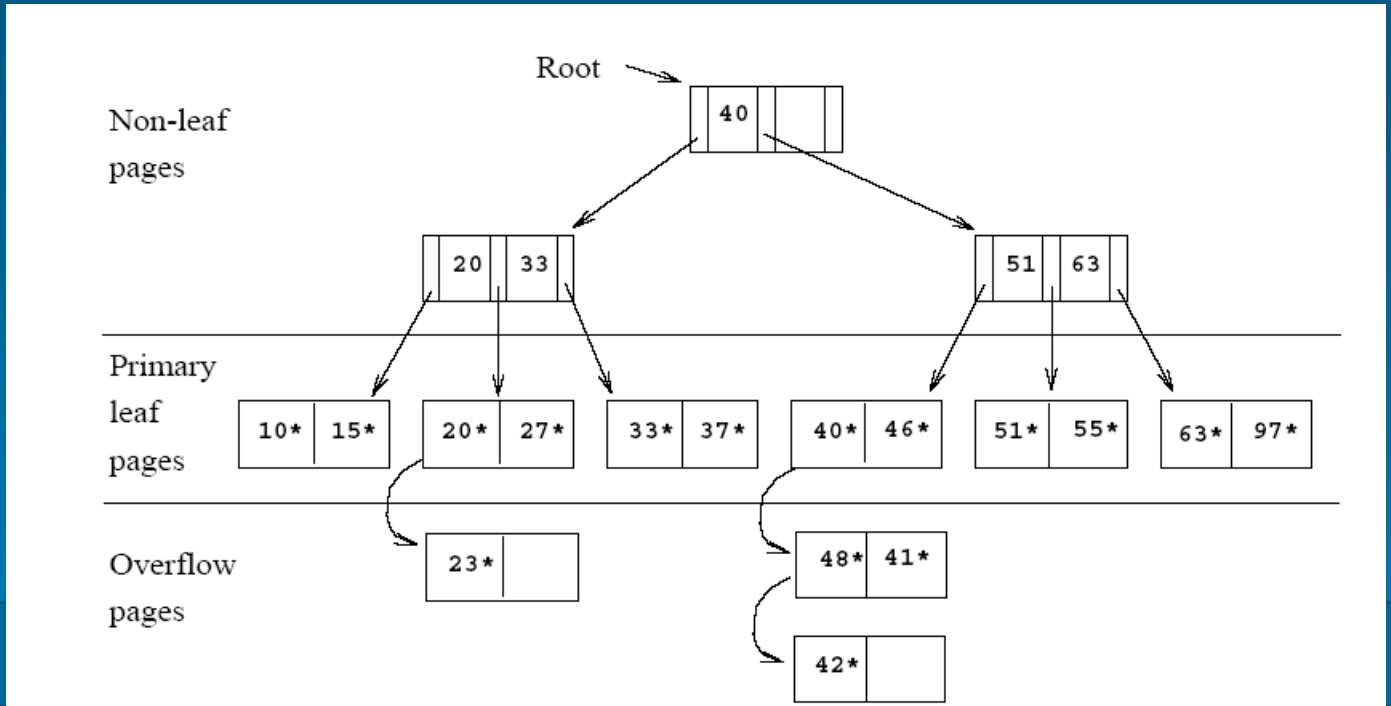
ISAM..

- Velikost indeksne datoteke poraja idejo o ISAM indeksu: **Zakaj ne bi pri gradnji indeksa uporabili REKURZIJO, tako da bi velikost vsakega posameznega indeksa znašala samo eno stran?**
- Gradnja indeksa nas pripelje do drevesne strukture



ISAM..

- ISAM indeks sestavljajo tri vrste strani, ki vsebujejo:
  - Vozlišča, ki niso listi (**non-leaf pages**)
  - Vozlišča, ki so primarni list in so alocirani sekvenčno (vozlišča, ki so bila listi takoj po kreiranju indeksa) (**primary leaf pages**)
  - Vozlišča, ki so dodana primarnim listom (in so tudi primarni listi) zaradi spreminjanja vsebine tabele in s tem indeksa (**overflow pages**)



ISAM..

- Struktura ISAM indeksa je od njegovega kreiranja dalje popolnoma statična (razen overflow strani)
- Vsako indeksno vozlišče je velikosti ene strani, vsi podatki (pari: podatek, kazalec na podatkovno stran) pa se nahajajo v listih drevesa
- Ko se indeksna datoteka kreira, so vse strani v listih urejene zaporedno in urejene po ključu

ISAM..

- Ko se podatkovno datoteko in posledično v listne strani dodaja podatke, je lahko potrebno v indeks dodati nove strani (overflow pages), kajti drevesna struktura indeksa je statična

ISAM..

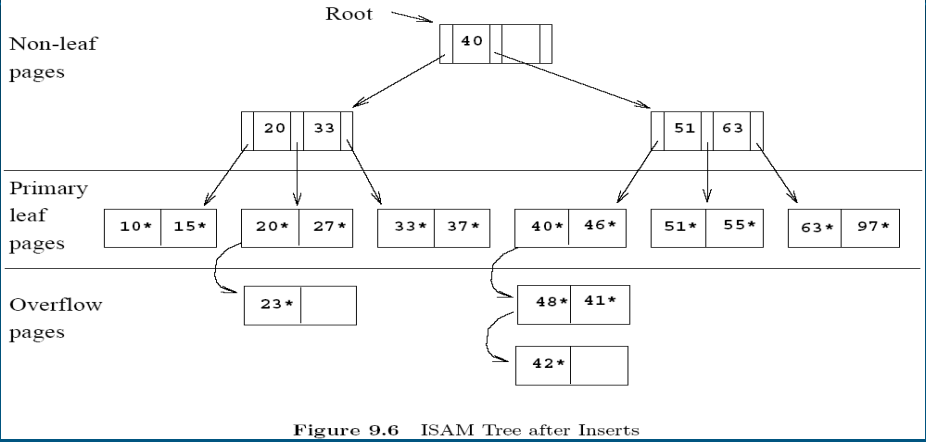


Figure 9.6 ISAM Tree after Inserts

- Ko je enkrat indeks kreiran, brisanja in vstavljanja vplivajo le na spremembe listov
- Dobilo lahko dolge verige overflow strani => počasno delovanje
- REŠITEV: ponovna gradnja indeksa (znebimo se overflow strani)
- ISAM je dober za datoteke, ki se jih malo ažurira. Začetni indeks se lahko zgradi tako, da so strani v listih 20% nezasedene (imamo nekaj maneverskega prostora za vstavljanje)

ISAM..

- Operaciji iskanja in vstavljanja

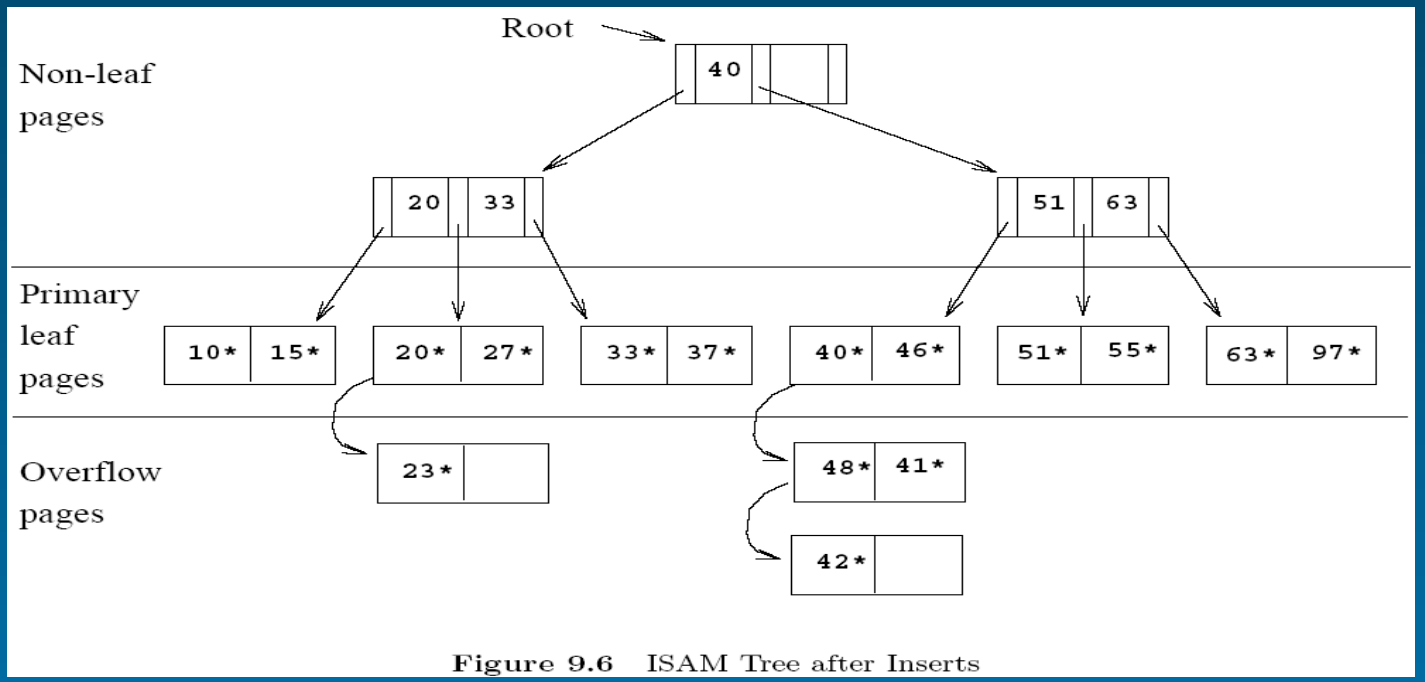


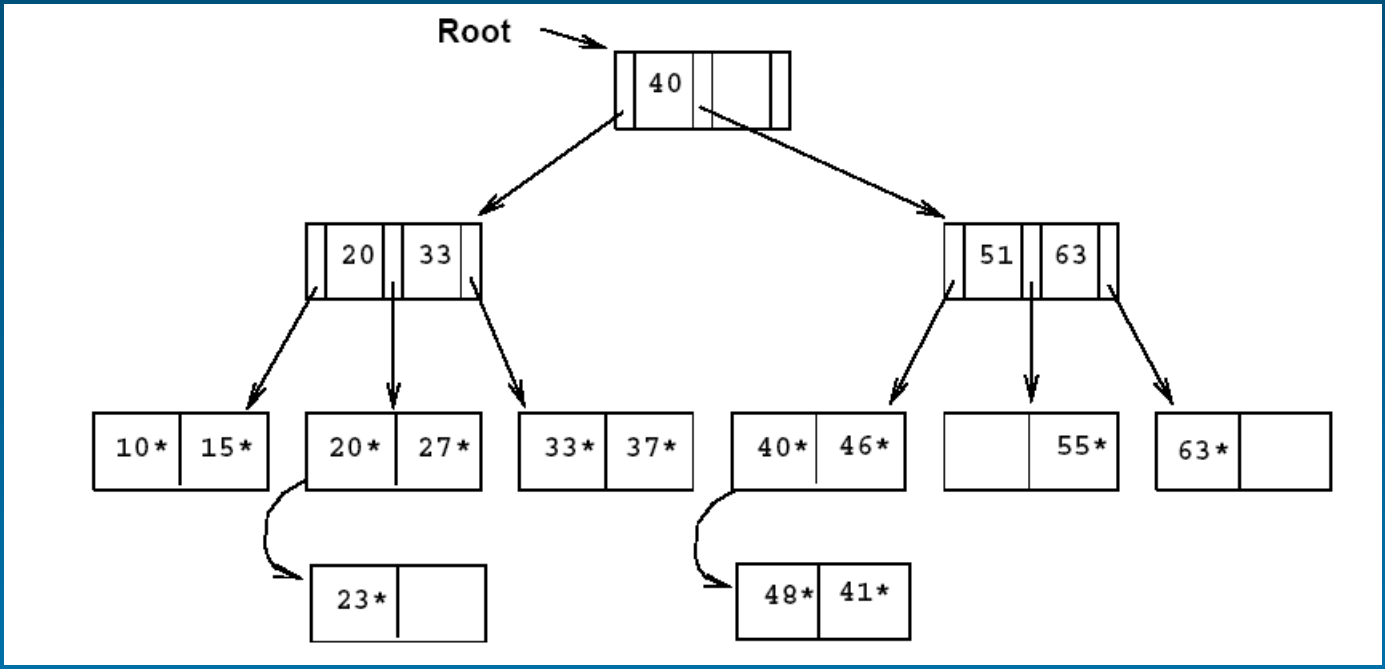
Figure 9.6 ISAM Tree after Inserts

ISAM..

- Operacija brisanja:
  - Če ostane vozlišče na overflow strani prazno, sta vozlišče (in stran) odstranjena
  - Če zbrišemo zapis, ki ima vrednost ključa enako podatku v primarnem listu, potem pustimo stran (kjer je primarni list) nespremenjeno (lahko ostane tudi prazna). Namenjena je bodočim vstavljanjem
  - Če zbrišemo zapis, ki ima ključ enak vrednosti v vozlišču, potem vozlišča, ki ni list, ne spreminjamo. Vozlišča, ki niso listi, so usmerjevalnega značaja za iskanje in ostale operacije (glej primer na naslednji strani)

ISAM..

- Po brisanju vrednosti 51 je ta vrednost ostala v vozlišču, ki ni list. Če bi naknadno iskali vrednost 51, bi šele v primarnem listu ugotovili, da je ni



ISAM

- Kompleksnost iskanja je $\log_F N$, kjer je N število strani s primarnimi listi, F pa število naslednikov indeksnega vozlišča (strani, ki vsebuje vmesna vozlišča, ki niso listi – nonleaf pages), kar je dosti hitreje kot binarno iskanje
- Dobra stran ISAM je, da zaradi statičnosti niso potrebna zaklepanja strani, ki niso listi, ker se vsebina le teh nikoli ne spreminja

B+ indeks..

- B+ indeks je dinamičen. Njegova struktura se dinamično prilagaja spremembam v osnovni datoteki
- B+ drevo predstavlja iskalno strukturo. To je uravnoteženo drevo, katerega vozlišča usmerjajo iskanje, listi pa vsebujejo podatke (ključe)
- Listi v B+ drevesu so povezani v dvosmerni urejen seznam strani

B+ indeks..

- Lastnosti B+ drevesa:
  - Operacije (insert, delete) ohranjajo drevo uravnoteženo,
  - Vozlišča (razen root-a) morajo biti vsaj 50% zasedena,
  - Iskanje določene vrednosti zahteva le pot od root vozlišča do ustreznega listnega vozlišča. Poti do vseh vozlišč so zaradi uravnoteženosti enake in določajo višino drevesa.

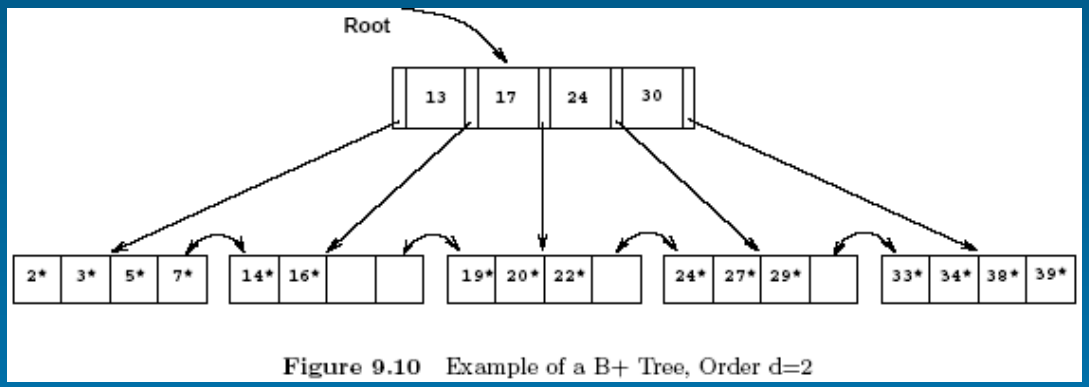


Figure 9.10 Example of a B+ Tree, Order d=2

B+ indeks

- Vsako vozlišče B+ drevesa vsebuje m vpisov; d=<m=<2d
- d je parameter B+ drevesa (red drevesa) in predstavlja kapaciteto vozlišča. Edina izjema je korensko vozlišče, za katerega velja 1=<m=<3d
- Lastnosti B+ dreves si poglejmo skozi uporabo operacij Iskanja, Vstavljanja in Brisanja. Predpostavili bomo, da v osnovni datoteki ni duplikatov. Primer našega B+ indeksa prikazuje slika (d=2):

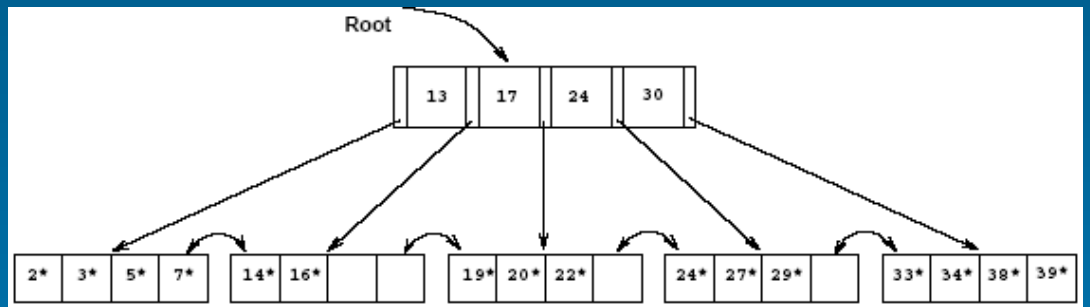


Figure 9.10 Example of a B+ Tree, Order d=2

B+ indeks - Iskanje

- Algoritem išče list, v katerem se nahaja iskalni ključ
- Če iščemo vrednost 5, sledimo levemu kazalcu, ker je 5 < 13
- Pri iskanju 14 ali 15 sledimo 2. kazalcu. Vrednosti 15 v listu ne najde, zato iskanje ne uspe

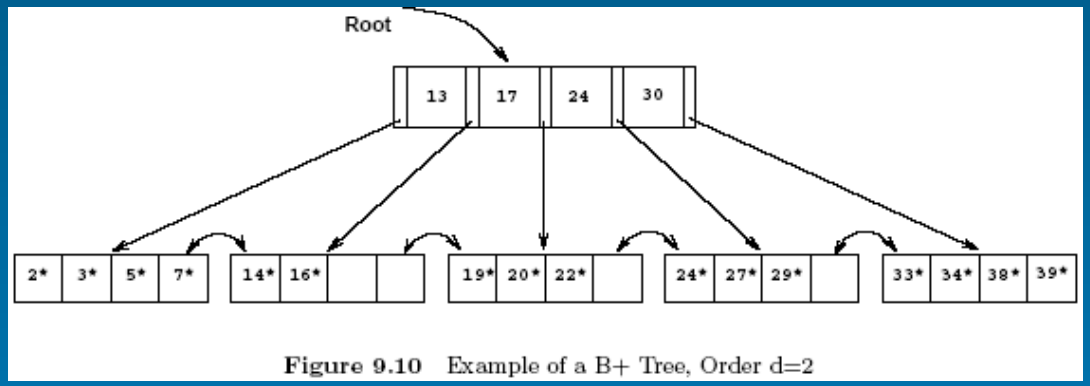


Figure 9.10 Example of a B+ Tree, Order d=2

B+ indeks – Vstavljanje..

- Algoritem za vstavljanje poišče list, v katerega spada nova vrednost in jo vanj vstavi, če to dopušča zasedenost lista
- Lahko je potrebno opraviti distribucijo ali razcepljanje strani

B+ indeks – Vstavljanje..

- Vstavljanje vrednosti 8 z uporabo razcepljanja

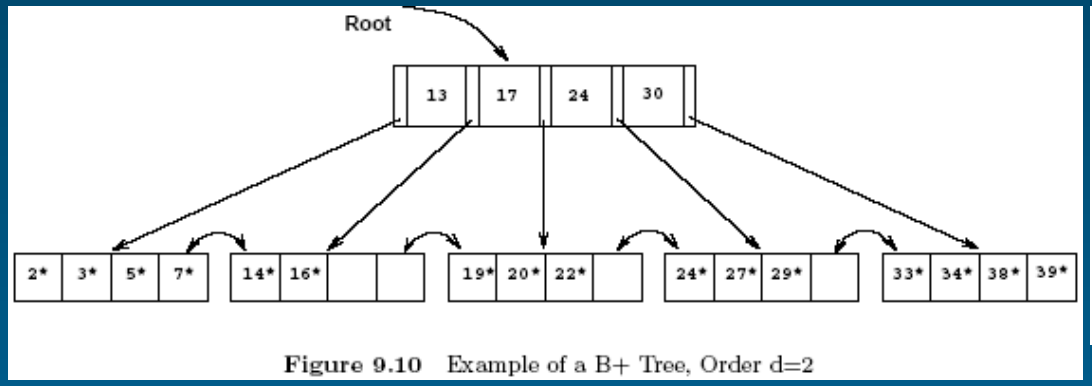
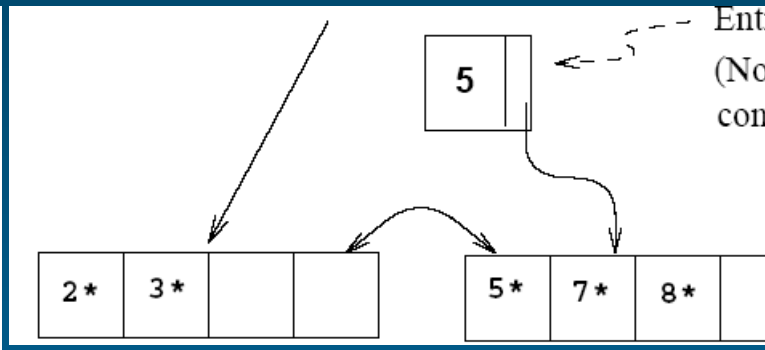
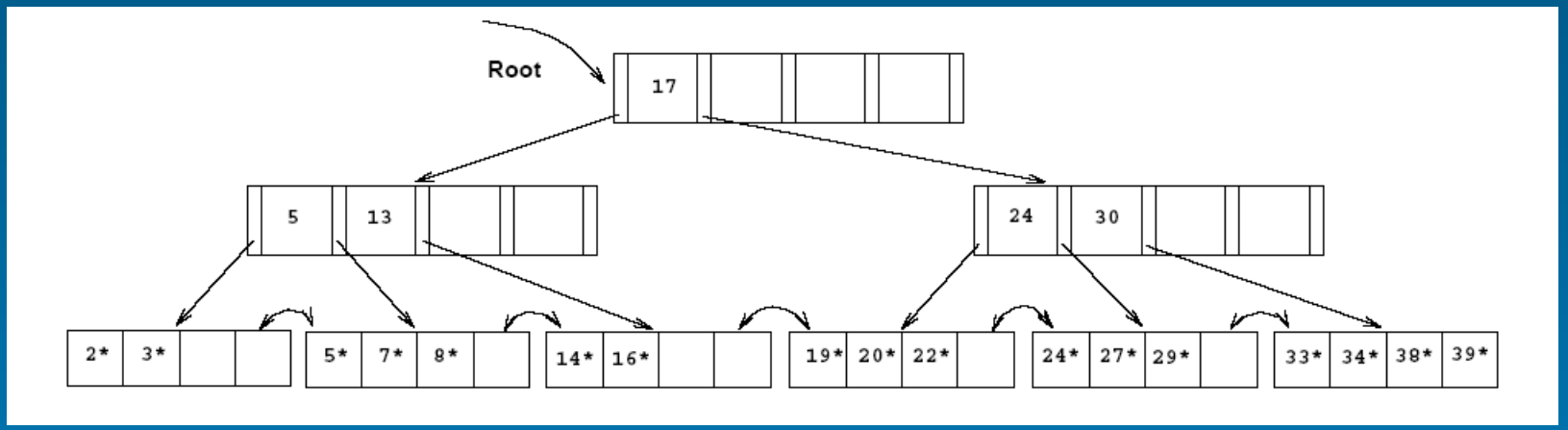


Figure 9.10 Example of a B+ Tree, Order d=2





B+ indeks - Vstavljanje

- Vstavljanje vrednosti 8 z uporabo redistribucije

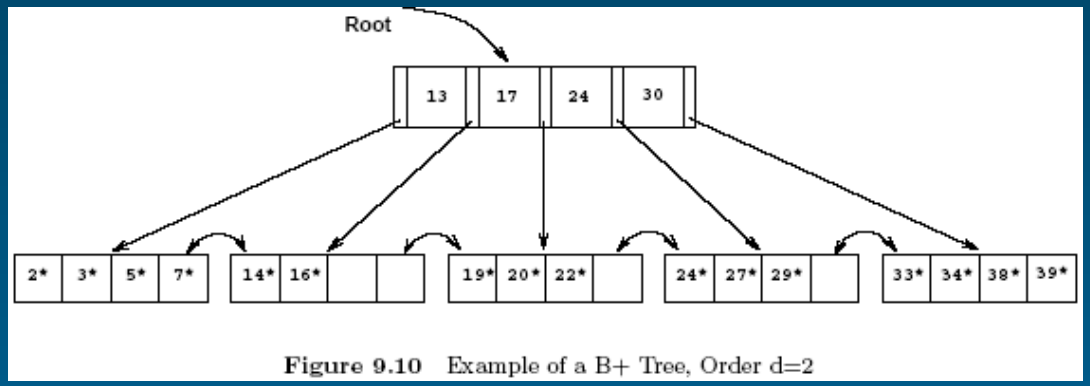
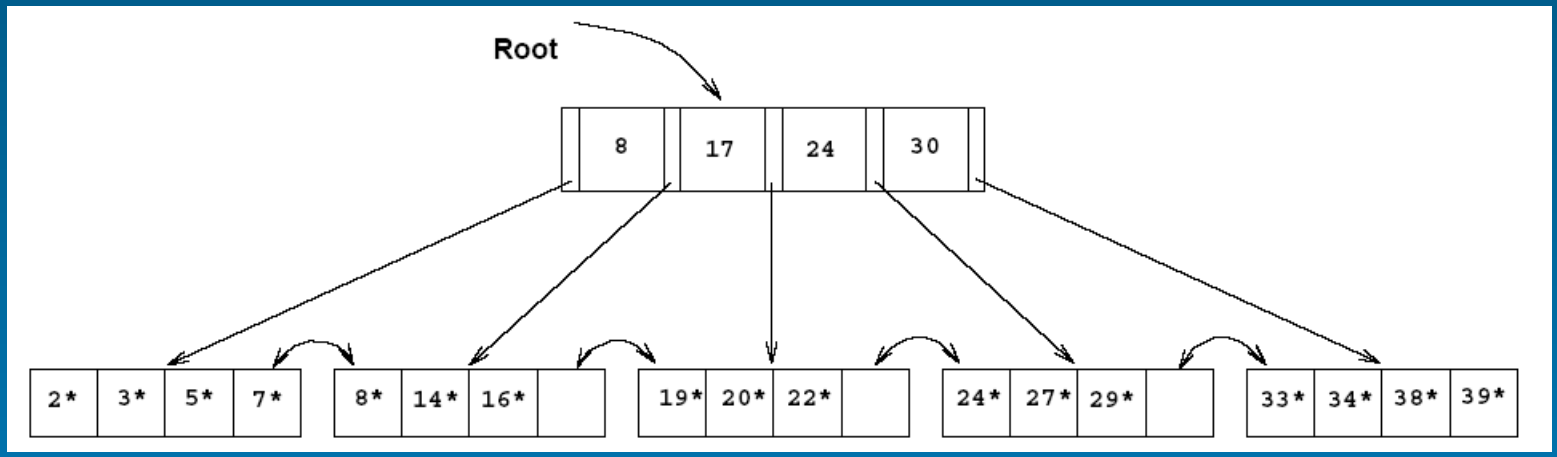


Figure 9.10 Example of a B+ Tree, Order d=2

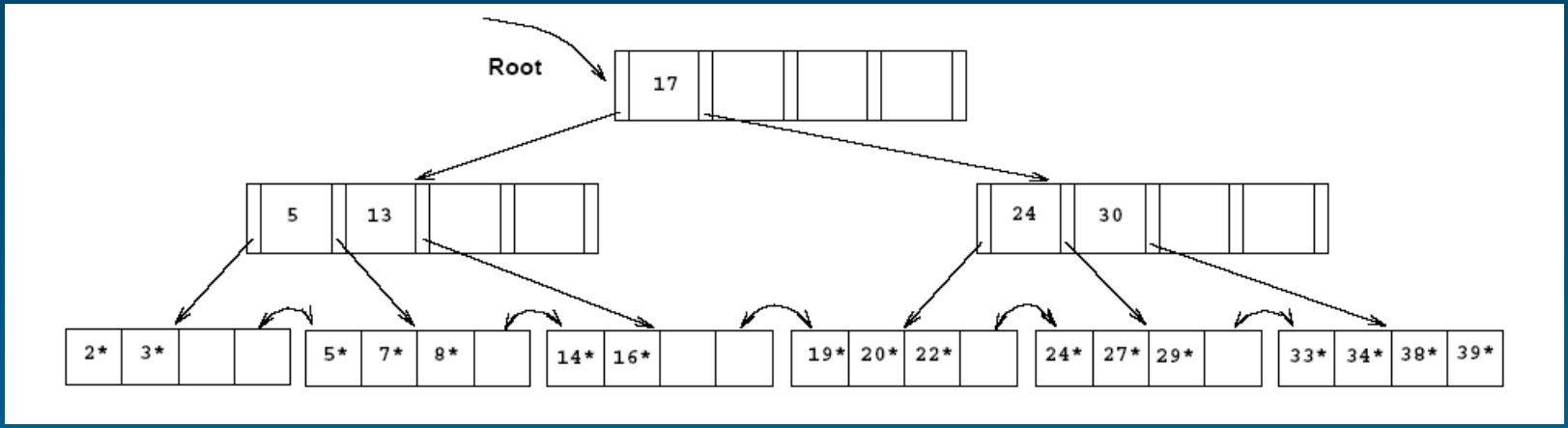


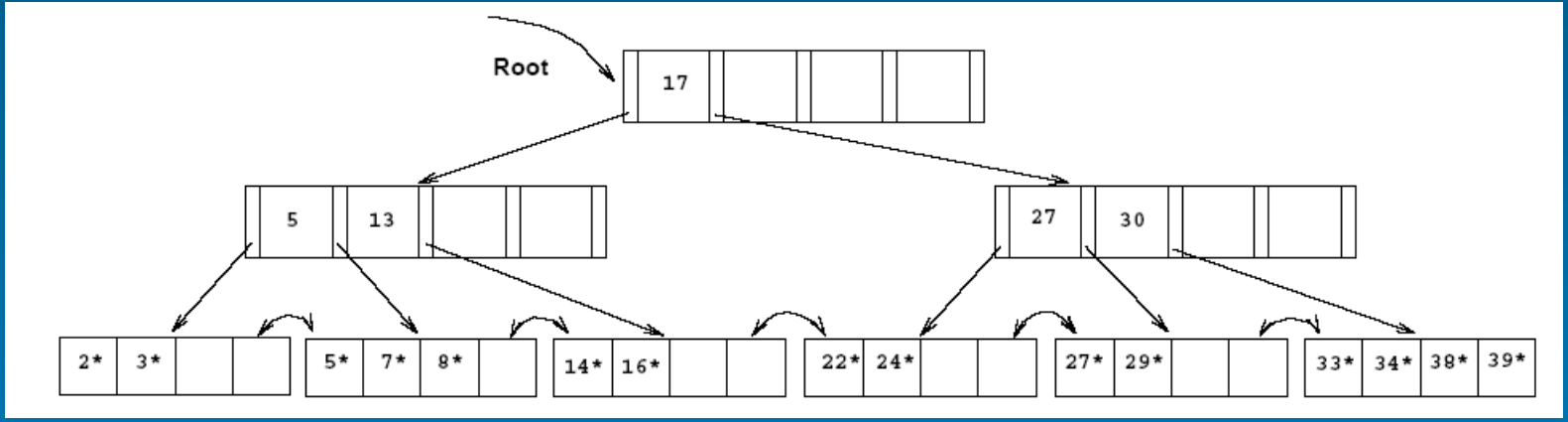
B+ indeks – Brisanje..

- Algoritem za brisanje poišče list, kjer je podatek za brisanje in ga odstrani iz drevesa
- Lahko se zgodi, da se zasedenost vozlišča po brisanju zniža pod dovoljeno mejo
- V tem primeru je potrebno uporabiti redistribucijo ali pa zlivanje dveh vozlišč v eno (v tem primeru posodobiti tudi prednike)

B+ indeks - Brisanje

- Brisanje ključev 19 in 20 z uporabo redistribucije





B+ indeks - Duplikati

- Lahko se poslužujemo overflow strani, kot v ISAM
- Lahko jih vstavljamo “normalno”, potrebno pa je spremeniti algoritem iskanja tako, da bo vedno poiskal “najbolj levi element” z iskano vrednostjo ključa
- Uporaba podatkovni vpis je par <k, rid-list> je najbolj naravna v tem primeru