Indeksiranje

ISAM indeks:

IDEJA ISAM indeksa:
Imamo datoteko oseb, urejeno po polju "starost".
Uporabimo naslednjo možnost za pohitritev iskanja:

1. Izdelamo dodatno datoteko, s po enim zapisom za vsako stran v (osnovni) podatkovni datoteki in jo uredimo po polju "starost".

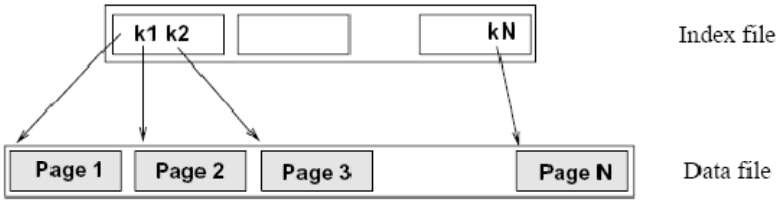


Figure 9.2 One-Level Index Structure

2. Vsak zapis v dodatni (indeksni) datoteki vsebuje par <ključ prvega zapisa na strani, kazalec na zapis>
3. Vsak ključ v indeksni datoteki predstavlja mejnik vsebine, na katero kažeta levi in desni kazalec → vsaka stran v indeksu vsebuje en kazalec več, kot je ključev.
4. Binarno iskanje se izvede nad indeksno datoteko, ki je manjša od osnovne datoteke => hitrejše iskanje.
5. **IDEJA!!! Zakaj ne bi ponovili koraka in zgradili še eno dodatno datoteko, ki bi imela za vsako indeksno stran en zapis, tako da bi velikost končnega indeksa znašala samo eno stran?**

Skica ideje **ISAM indeksa:**

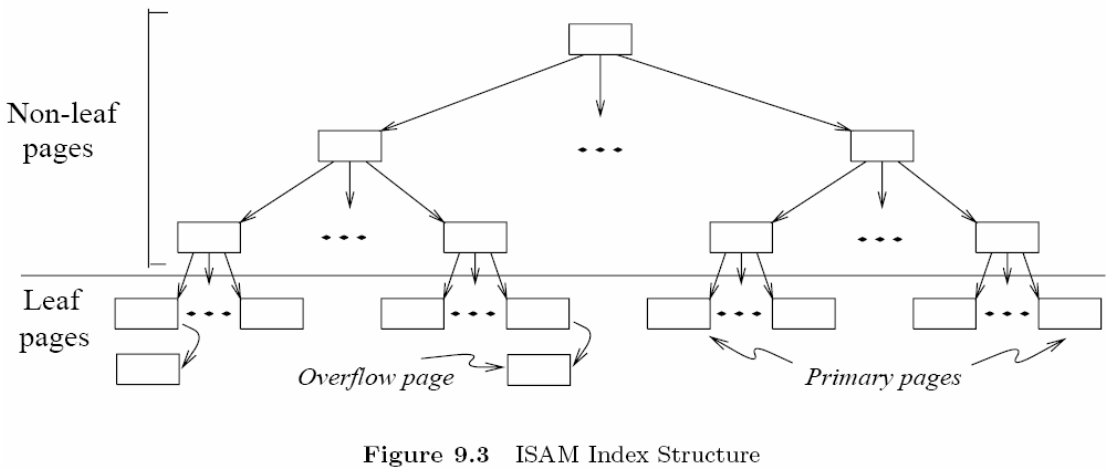


Figure 9.3 ISAM Index Structure

ZNAČILNOSTI ISAM INDEKSA:

- ISAM indeks je statičen (z izjemo prelivnih strani) → potrebno ga je reorganizirati.
- ISAM indeks je sestavljen iz indeksnih zapisov - vozlišč in listov (listnih strani).

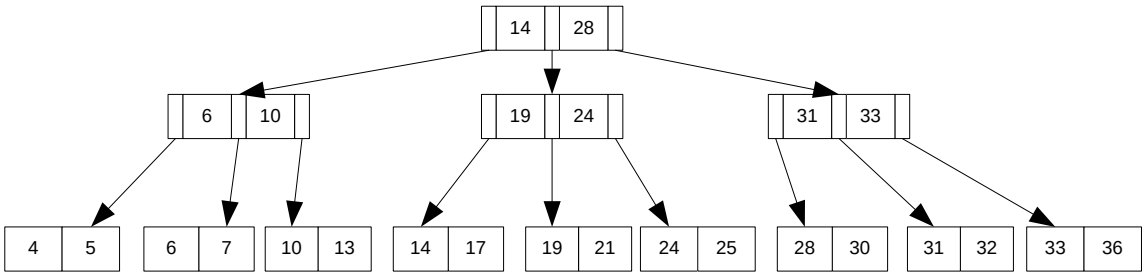
- Problem ISAM strukture se pokaže, ko naraste število prelivnih strani.
  - Podatki v prelivnih straneh so načeloma lahko urejene, običajno pa niso (zaradi učinkovitosti dodajanja zapisov)
  - Problem omilimo tako, da v začetku, ko izgradimo indeks, pustimo nekaj praznega prostora v straneh listov.

Primer izračuna stroškov iskanja zapisa v ISAM strukturi:

- Število I/O operacij = $\log_F N$, kjer je N število primarnih strani listov, F število otrok vsake indeksne strani (koliko kazalcev gre iz vozlišča).
- Število I/O operacij pri binarnem iskanju po urejeni datoteki je $\log_2 N$. Pri iskanju po eno-nivojskem indeksu: $\log_2(N/F)$
- Primer: datoteka z 1.000.000 zapisi, 10 zapisov na stran v listih in 100 zapisov v indeksnih straneh:
  - Strošek branja cele datoteke: 100.000
  - Strošek binarnega iskanja po urejeni datoteki: 17
  - Strošek binarnega iskanja po eno-nivojskem indeksu: 10
  - Strošek binarnega iskanja po ISAM strukturi: 3 (brez dodatnih strani)

Naloga 1:

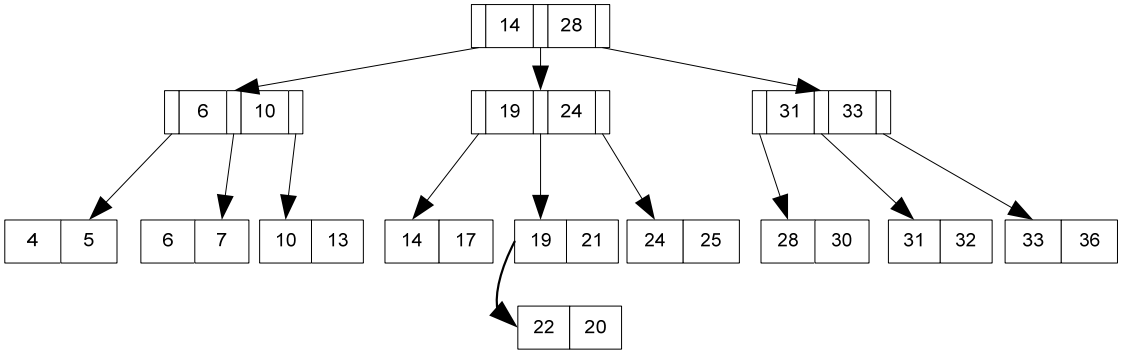
Zapisi osnovne datoteke so urejeni po polju teža. Vsaka stran v osnovni datoteki vsebuje po dva zapisa. Zapisi v osnovni datoteki imajo za polje teža naslednje vrednosti:
4,5,6,7,10,13,14,17,19,21,24,25,28,30,31,32,33,36
ISAM indeks:



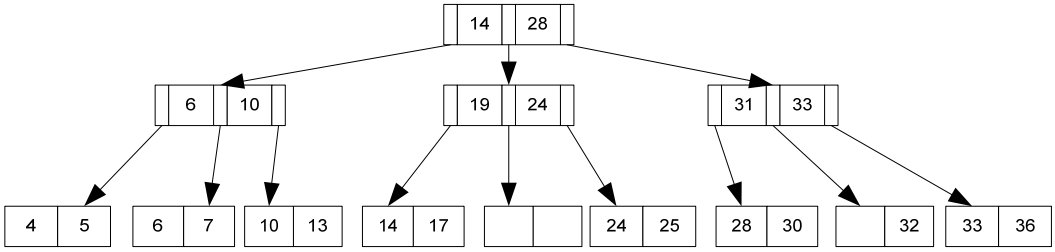
(a) V osnovno datoteko dodajte zapis s ključem 22 in 20.
(b) Iz osnovne datoteke zbrišite zapise z vrednostmi ključev: 22, 20, 21, 19, 31 (izhajajte iz slike, ki jo dobite pod postavko a.). Rešujte po korakih.
(c) Kolikšno je število I/O operacij za branje določenega zapisa?

Rešitev naloge 1:

a)



b)



Komentar:
Primarno stran v listu pustimo prazno za morebitna kasnejša vstavljanja. Prelivno stran pa zbrišemo, ker ta predstavlja dinamični del ISAM indeksa, ki se dodaja in krči po potrebi.

Ko je ISAM indeks skreiran, se vnašanje in brisanje dogaja samo v listih. To lahko povzroči kreiranje veliko prelivnih strani.

Indeksna vozlišča služijo izključno iskanju ustreznih strani v listih (usmerjanje iskanja). To je povsem normalno, saj je ISAM statičen.

Pri brisanju podatka s ključem 31 iz osnovne datoteke vidimo, da je ključ 31 v indeksnem vozlišču ostal. Šele nato, ko preiščemo ustrezno listno stran, lahko ugotovimo, da zapisa s ključem 31 v osnovni datoteki ni več.

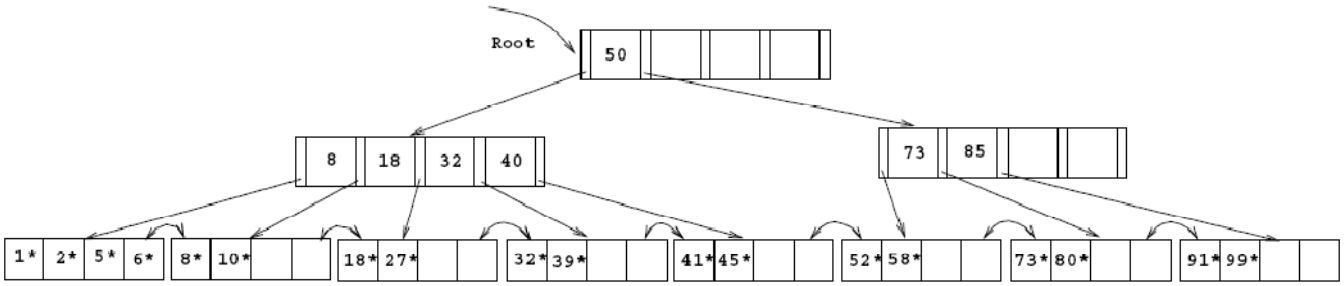
Drevesno strukturo ISAM indeksa običajno skreiramo tako, da 20% prostora na listih pustimo praznega.

c)
Število *I/O operacij* = $log_F N$, kjer je N število primarnih strani listov, F število otrok vsake indeksne strani. V našem primeru je število I/O operacij enako 2 v indeksnem delu ISAM drevesa. Temu je potrebno prišteti še branje listne strani (seveda, če predvidevamo, da se podatek nahaja v listu in ne v overflow strani), torej je število potrebnih branj za iskanje določene vrednosti enako 3.
Število I/O operacij je enako številu nivojev drevesa.

B+ Indeks:

Naloga 1:

Nad neurejeno datoteko je zgrajen B+ indeks, ki ga prikazuje spodnja slika:



1. Kako se spremeni B+ indeks, če se v osnovno datoteko vstavi zapis s ključem 9?

2. Kako se spremeni B+ indeks, če se v osnovno datoteko vstavi zapis s ključem 3?

3. Kako se spremeni B+ indeks, če se iz osnovne datoteke zbrišemo zapis s ključem 8, pri čemer predvidevamo, da je levo vozlišče (sibling) možno uporabiti za redistribucijo?

4. Kako se spremeni B+ indeks, če se iz osnovne datoteke zbrišemo zapis s ključem 8, pri čemer predvidevamo, da je desno vozlišče (sibling) možno uporabiti za redistribucijo?

5. Kako se spremeni B+ indeks, če v osnovno datoteko najprej vnesemo zapis s ključem 46, nato pa iz iste datoteke zbrišemo zapis s ključem 52? (2 rešitvi - odvisno od tega, koliko vozlišč vzamemo za premik v desno).

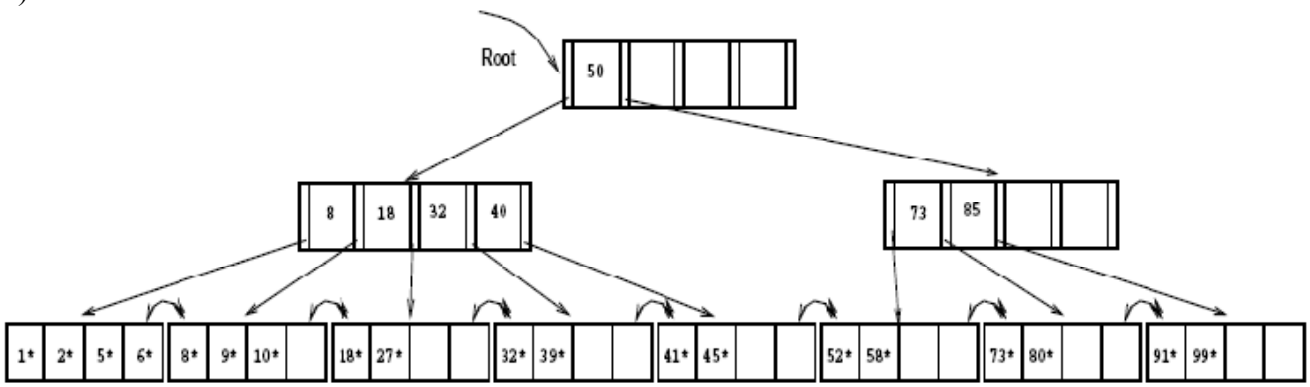
6. Kako se spremeni B+ indeks, če iz osnovne datoteke zbrišemo zapis s ključem 91 (izhajamo iz začetnega indeksa)?

7. Kako se spremeni B+ indeks, če v osnovno datoteko najprej vnesemo zapis s ključem 59, nato pa iz iste datoteke zbrišemo zapis s ključem 91? (enako kot 6., samo da se predhodno doda ključ 59).

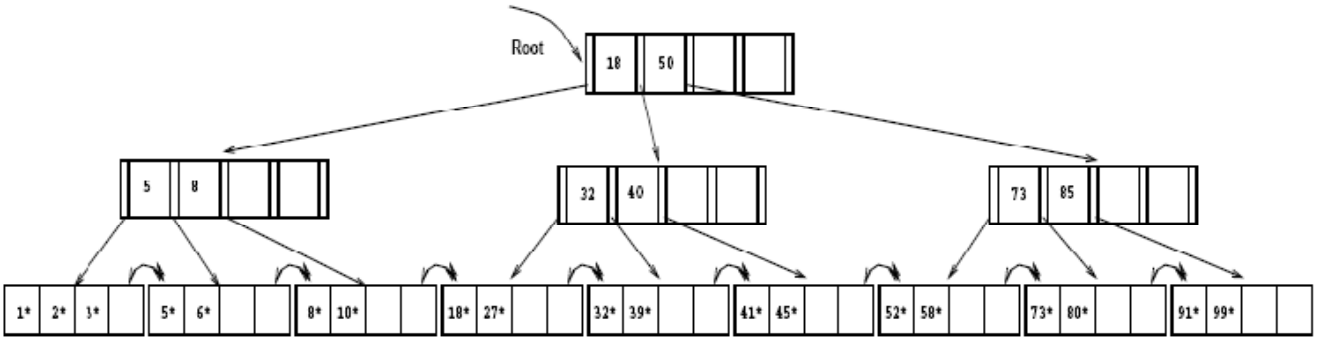
8. Kako se spremeni B+ indeks, če bi iz osnovne datoteke postopama brisali zapise s ključi 32, 39, 41, 45 in 73? (rešitev je ena, ker se pri zlivanju gleda desni brat).

Rešitev naloge 1:

1.)

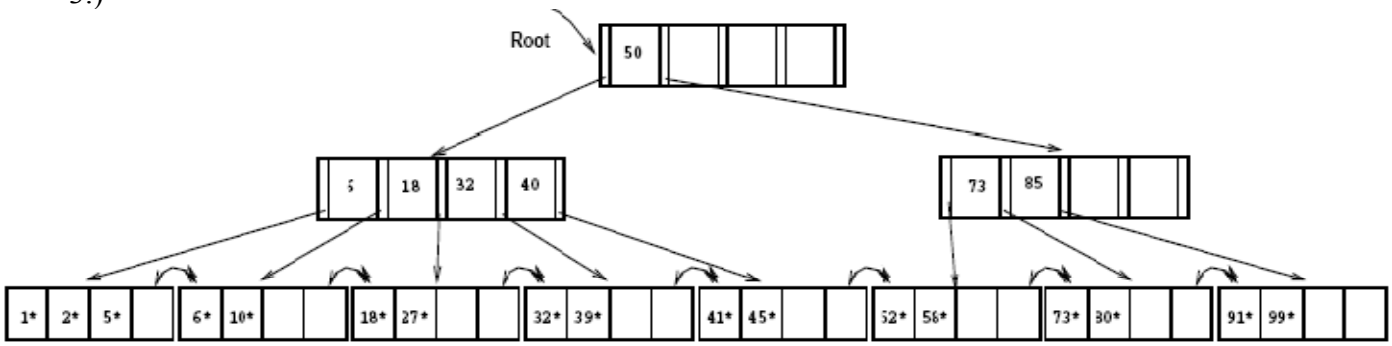


2.)

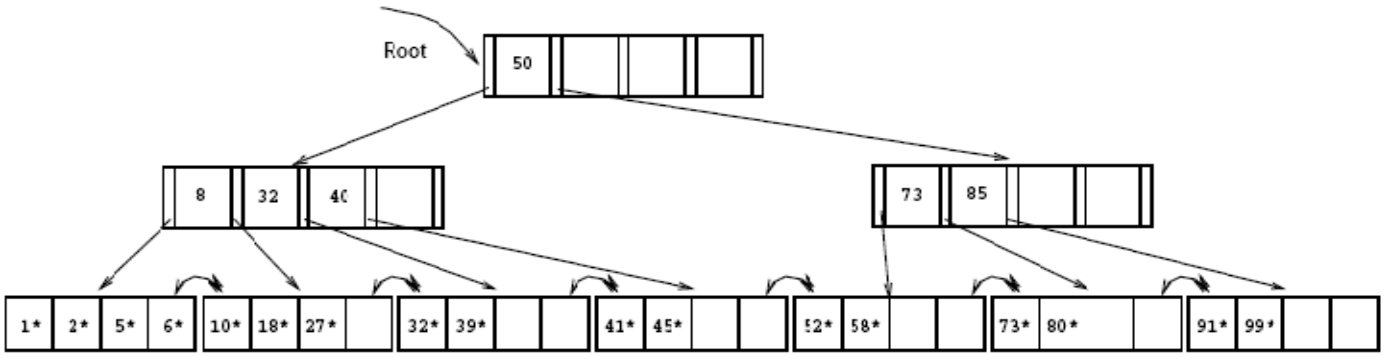


Tukaj se indeksi zapis 3 sicer postavi v levi blok, ki ga dobimo pri razcepu originalnega lista. Vendar bi po algoritmu bilo pravilno, da gre 2 v desni list. Algoritem namreč pravi, da gre d zapisov v levi list, preostanek pa v desni. 1. zapis v novonastalem desnem listu pa postane tudi delitveni ključ v indeksnem vozlišču.

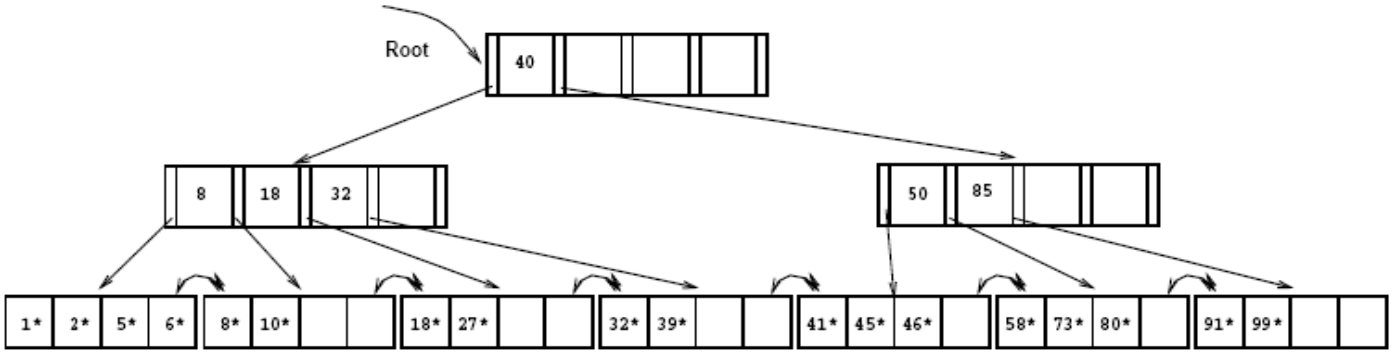
3.)



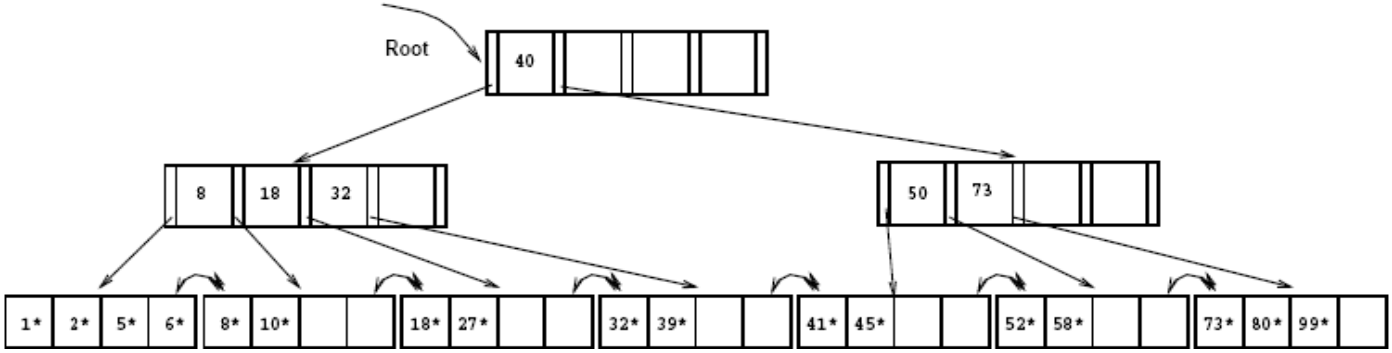
4.)



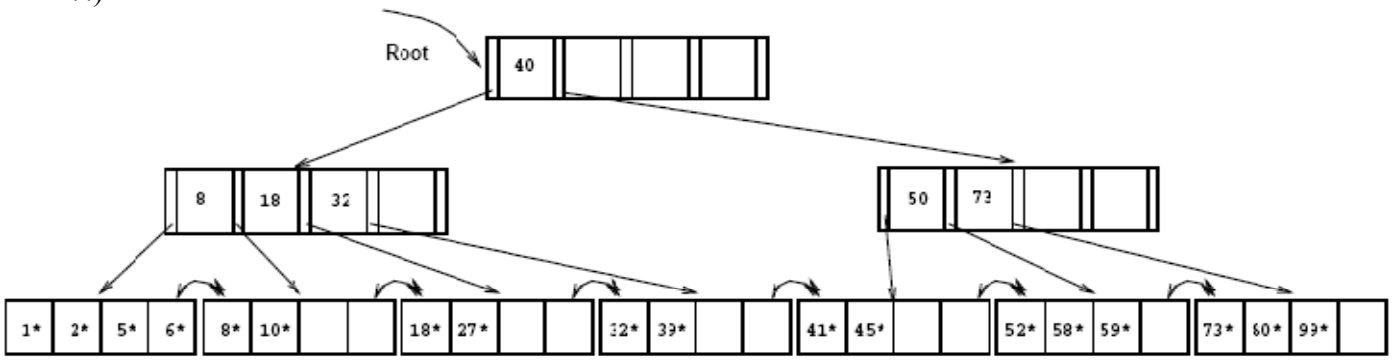
5.)


Root
40
8
18
32
50
85
1*
2*
5*
6*
8*
10*
18*
27*
32*
39*
41*
45*
46*
58*
73*
80*
91*
99*

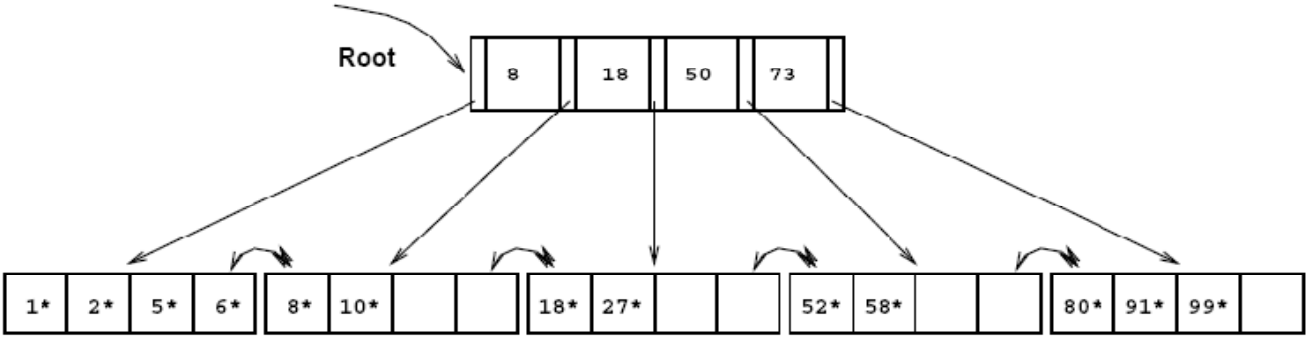
6.)


Root
40
8
18
32
50
73
1*
2*
5*
6*
8*
10*
18*
27*
32*
39*
41*
45*
52*
58*
73*
80*
99*

7.)


Root
40
8
18
50
1*
2*
5*
6*
8*
10*
18*
27*
52*
58*
59*
73*
99*

8.)


Root
8
18
50
73
1*
2*
5*
6*
8*
10*
18*
27*
52*
58*
80*
91*
99*

Naloga 2:

Predpostavimo, da PB podpira oba tipa indeksov, statičnega ISAM in dinamičnega B+. Ali bi se sploh kdaj odločili za uporabo statičnega indeksa namesto dinamičnega? Svojo odločitev utemeljite.

Rešitev naloge 2:

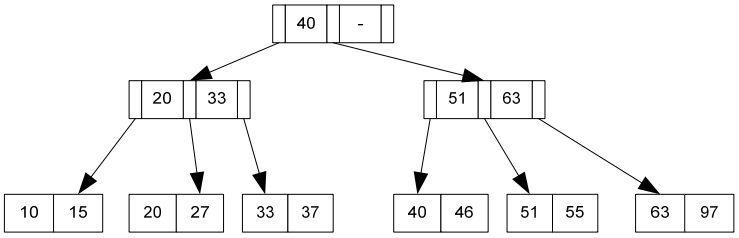
Statični indeks ISAM brez overflow strani je hitrejši kot B+ indeks pri operacijah update in delete, in sicer zato, ker se indeksna vozlišča pri teh dveh operacijah v primeru statičnega indeksa samo bere.

Če je množica ključev, ki se bo v prihodnosti vstavila v indeks znana v naprej, potem lahko statični indeks zgradimo tako, da predvidimo zadosti praznega prostora za bodoča vstavljanja. Če se sistem periodično ugaša, se v tem času lahko statični indeks ponovno zgradi. Redka in vnaprej znana posodabljanja so indikator, ko je potrebno razmisliti o uporabi statičnega indeksa.

Naloga 3:

Nad datoteko zgradimo dva drevesna indeksa (ISAM in B+), ki sta na začetku identična (slika). Kako se spremenita indeksa, če v datoteko najprej dodamo zapis s ključem 41, nato pa še zapis s ključem 23. Za oba indeksa narišite sliko za obe ažuriranji.
Obkrožite pravilen odgovor:

a.	ISAM in B+ indeks sta dinamična indeksa.	DA	NE	NE VEM
b.	ISAM indeks predstavlja uravnoteženo drevo.	DA	NE	NE VEM
c.	Pri branju podatka je ISAM indeks s ključem 41 hitrejši kot B+ indeks.	DA	NE	NE VEM



Velja:

Max število kazalcev na h-tem nivoju je funkcija razvejanosti (F) in višine (h) in torej znaša: F^h, kjer $h \in \{1,2,3,\ldots\}$ in $F=2d+1$, kjer je d red drevesa

Min število kazalcev na h-tem nivoju je potem: $2(d+1)^{h-1}$, d+1 predstavlja število kazalcev za polovično zasedeno indeksno vozlišče. Ker imamo v root vozlišču samo dva kazalca, je v formuli 2. V bistvu imamo dva drevesa, ki imata za 1 manjšo višino in zanju potem izračunamo minimalno število kazalcev.

Naloga 4:

Izračunajte, koliko zapisov v osnovni datoteki lahko indeksiramo z B+ indeksom reda 50, ki ima 3 nivoje (h) indeksnih blokov, bloki v gostem indeksu (listi) pa so veliki 10 indeksnih zapisov.

Število kazalcev na h=3. nivoju: $F^h=(2d+1)^h=(2*50+1)^3=1030301$
Vsak indeksni kazalec na 3. nivoju kaže na indeksni blok gostega indeksa, ki ima še 10 indeksnih zapisov. Tako lahko indeksiramo datoteko, ki ima 1030301*10=**10303010** zapisov.

Naloga 5:

Koliko največ listov imamo lahko v B+ drevesu, če je red drevesa 20 in imamo 3 nivoje v B delu drevesa?

d=20
F=(2d+1)=41
$F^h=(2*20+1)^3=(2*20+1)^3=68921$

Sočasni dostop do podatkov

Opis protokolov za preprečevanje mrtve zanke (Mohorič):

Vsaki izmed transakcij pripiše SUPB ob njenem pričetku časovno oznako - njen startni čas. Na ta način je možno za poljubni dve transakciji ugotoviti katera je "starejša" in katera "mlajša". Starejša transakcija je tista, ki se že dlje časa izvaja in je zato njen startni čas manjši.

Ko transakcija TA zahteva zaseženje podatka, ki ga je že zasegla transakcija TB, in se njeni zahtevi zaradi nekompatibilnosti zaseženj ne da pri priči ugoditi, se zgodi naslednje:

- po protokolu **Čakaj ali izdihni (Wait-Die)**:

  če je transakcija TA starejša od TB, preide TA v stanje čakanja na odobritev, če je mlajša, pa se njeno izvajanje prekine, transakcija se razveljavi in posreduje transakcijskemu programu v ponovno izvajanje;

- po protokolu **Rani ali čakaj (Wound-Wait)**:

  če je transakcija TA starejša od TB, se prekine, razveljavi in vrne v ponovno izvajanje transakcija TB (po njeni razveljavitvi se transakciji TA odobri zaseženje podatka), če je TA mlajša, pa preide v stanje čakanja.

Protokol Čakaj ali izdihni (Wait-Die) zagotavlja, da v primeru nekompatibilnosti zahtev po zaseženju podatkov starejše transakcije čakajo na zaključek mlajših in s tem na sprostitev po njih zaseženih podatkov. Tako se ne more nikoli pripetiti, da bi T0 čakala na T1, T1 na T2, ... in Tn na T0.

Po protokolu Rani ali čakaj (Wound-Wait) je čakanje na sprostitev podatkov tudi urejeno po starosti, vendar tako, da mlajše transakcije čakajo na zaključek starejših.

Po obeh protokolih so vedno mlajše transakcije tiste, katerih izvajanje se prekine, če obstaja potencialna nevarnost za nastop mrtve zanke. Da bi se ne dogajalo, da bi bila vedno ena in ista transakcija prekinjena in vrnjena v ponovno izvajanje, je potrebno poskrbeti, da vsaka transakcija prej ali slej postane starejša. To se lahko doseže tako, da se pri ponovnem izvajanju transakcije ohrani njena prejšnja časovna oznaka, kar pa pomeni, da mora pri dodeljevanju časovne oznake SUPB ločevati med transakcijami, ki se izvajajo prvič, in transakcijami, ki so bile razveljavljene in vrnjene v ponovno izvajanje zaradi pravil protokola.

Konstrukcija čakalnega grafa:

- WFG je usmerjen graf G = (N, E), kjer N vozlišča, E povezave.
- Postopek risanja WFG:
  - Kreiraj vozlišče za vsako transakcijo
  - Kreiraj direktno povezavo Ti → Tj, če Ti čaka na zaklepanje podatkovne enote, ki je zaklenjena s strani Tj.
- Pojav mrtve zanke označuje cikel v grafu.
- SUPB periodično gradi graf in preverja obstoj mrtve zanke.
- Ko je mrtva zanka detektirana, je potrebno eno ali več transakcij prekiniti.
- Pomembno:
  - Izbira transakcije za prekinitev: možni kriteriji: 'starost' transakcije, število sprememb, ki jih je transakcija naredila, število sprememb, ki jih transakcija še mora opraviti.
  - Koliko transakcije preklicati: namesto preklica cele transakcije včasih mrtvo zanko moč rešiti s preklicom le dela transakcije.
  - Izogibanje stalno istim žrtvam: potrebno preprečiti, da ni vedno izbrana ista transakcija. Podobno živi zanki (live lock)

Naloga 1:

Imamo račune R1=200 €, R2=300 € in R3=400 €. Ukazi transakcij T1 in T2 se izvedejo po naslednjem razporedu:

T1	T2
PoiščiPreberi (R1, a) a:=a-10 Ažuriraj (R1, a)	
	PoiščiPreberi (R2, c) c:=c-20 Ažuriraj (R2, c)
PoiščiPreberi (R2, b) b:=b+10 Ažuriraj (R2, b) Pomni	
	poiščiPreberi (R3, d) Pozabi

Vsota na računih R1, R2 in R3 mora po izvedbi transakcij biti enaka kot pred izvedbo T1 in T2. Če ni, opiši kaj je narobe in napiši urnik izvajanja, z uporabo protokola PXC.

Rešitev naloge 1:

- Če se T1 in T2 izvedeta v takem urniku kot je podan, bo vsota na računih na koncu naslednja:
  R1=190
  R2=300
  R3=400
  SUM= 890

  Pride do branja nepotrjenega podatka (branje neobstoječega podatka).

- Uporaba protokola PXC (ekskluzivno zaklepanje podatkov):

T1	T2
E-PoiščiPreberi (R1, a) a:=a-10 Ažuriraj (R1, a)	
	E-PoiščiPreberi (R2, c) c:=c-20 Ažuriraj (R2, c)
E-PoiščiPreberi (R2, b) *(čakanje na odobritev)*	
	E-PoiščiPreberi (R3, d) *Pozabi*
(zaseženje odobreno, ukaz se izvede) b:=b+10 Ažuriraj (R2, b) *Pomni*	

Po izvedbi transakcij T1 in T2 je vsota na računih naslednja:
R1=190
R2=310
R3=400
R1 + R2 + R3 = 900 → OK

Naloga 2:

V podatkovni bazi imamo podatke o naslednjih računih: R1=100, R2=200 in R3=300 €. Transakciji T0 prenese vrednost 10 € iz računa R1 na R2, sočasno pa se izvaja transakcija T1, ki iz R2 na R1 prenese vrednost 20€.

1. Kakšna bi bila vsota na računih R1, R2 in R3, če se transakciji izvedeta zaporedno, najprej T0 in nato T1?
2. Kakšna je vrednost na računih, če se operacije transakcij izvedejo v podanem vrstnem redu?
3. Kako se izvedejo podane operacije z uporabo deljenega zaklepanja - PSC? Komentirajte, narišite čakalni graf!
4. Kako bi se navedeni transakciji izvedli z uporabo protokola "Čakaj ali izdihni", pri čemer uporabljamo ekskluzivno zaklepanje podatkov?

T0	T1
PoiščiPreberi (R1, a) a:=a-10 Ažuriraj (R1, a) PoiščiPreberi (R2, b) b:=b+10	
	PoiščiPreberi (R2, c) c:=c-20 Ažuriraj (R2, c)
Ažuriraj (R2, b) *Pomni*	
	PoiščiPreberi (R3, d) d:=d+20 Ažuriraj (R3, d) *Pomni*

Rešitev naloge:

1. Vsota na računih bi znašala 600 €. Prikaži potek.
2. Vsota po podanem vrstnem redu bi znašala 620 €. Prikaži potek.
3. Uporaba deljenega zaklepanja, nariši čakalni graf.

T0	T1
D-PoiščiPreberi (R1, a) a:=a-10 E-Ažuriraj (R1, a) D-PoiščiPreberi (R2, b) b:=b+10	
	D-PoiščiPreberi (R2, c) c:=c-20 E-Ažuriraj (R2, c) *(čakanje na odobritev)*
E-Ažuriraj (R2, b) *(čakanje na odobritev)* ***MRTVA ZANKA!!!*** *Pomni*	
	PoiščiPreberi (R3, d) d:=d+20 Ažuriraj (R3, d) *Pomni*

4. Potek operacij z uporabo protokola "Čakaj ali izdihni":

Ko transakcija TA zahteva zaseženje podatka, ki ga je že zasegla transakcija TB, in se njeni zahtevi zaradi nekompatibilnosti zaseženj ne da pri priči ugoditi, se zgodi naslednje:

* po protokolu **Čakaj ali izdihni (Wait-Die)**:
če je transakcija TA starejša od TB, preide TA v stanje čakanja na odobritev, če je mlajša, pa se njeno izvajanje prekine, transakcija se razveljavi in posreduje transakcijskemu programu v ponovno izvajanje;

T0	T1
E-PoiščiPreberi (R1, a) a:=a-10 Ažuriraj (R1, a) E-PoiščiPreberi (R2, b) b:=b+10	
	E-PoiščiPreberi (R2, c) *(prekinitev izvajanja)*

Ažuriraj (R2, b) *Pomni*	
	PoiščiPreberi (R2, c) c:=c-20 Ažuriraj (R2, c) PoiščiPreberi (R3, d) d:=d+20 Ažuriraj (R3, d) *Pomni*

Obnavljanje podatkov po nesrečah

Naloga 1:

Za zaščito PB se uporablja *obnavljanje s senčnimi stranmi*. Stanje tekoče tabele strani pred pričetkom transakcije je prikazano na sliki. V okviru transakcije se izvedejo po vrsti naslednji ukazi:
read(A,a),
read(D,d),
write(G,g),
read(C,c),
write(E,e),
write(I,i).

Kakšna je vsebina tekoče tabele strani, če se transakcija zaključi s Pozabi, in kakšna, če se zaključi s Pomni? V kakšnem zaporedju se strani čitajo z diska oziroma se izpisujejo nanj, če je v datotečnem vmesniku prostora le za dve strani hkrati, in se pri zasedenem vmesniku prepiše vedno najdalj časa neuporabljena stran?

Tekoča tabela strani

s1
s2
s3

Podatkovna baza

s1	A	B	C
s2	D	E	F
s3	G	H	I
s4			
s5			
s6			

Rešitev naloge 1:

s1	A	B	C
s2	D	E	F
s3	G	H	I
s4	G	H	I
s5	D	E	F
s6	G	H	I

Tekoča tabela strani:

s1
~~s2~~, s5
~~s3~~, ~~s4~~, s6

Datotečni vmesnik:

DV1	~~s1~~, ~~s3~~, s2
DV2	~~s2~~, ~~s1~~, s4