Indeksiranje
Baze podataka 2

dr Miloš CVETANOVIĆ

Osnovni koncepti

- Mehanizmi indeksiranja u upotrebi da se ubrza pristup željenim podacima
  - Npr. katalog autora u biblioteci, indeks termina na kraju udžbenika
- Ključ pretrage (**search key**) – atribut ili skup atributa koji se koristi za pretragu zapisa
- Indeksna datotetka sadrži zapise (ulazi indeksa)

search-key	pointer

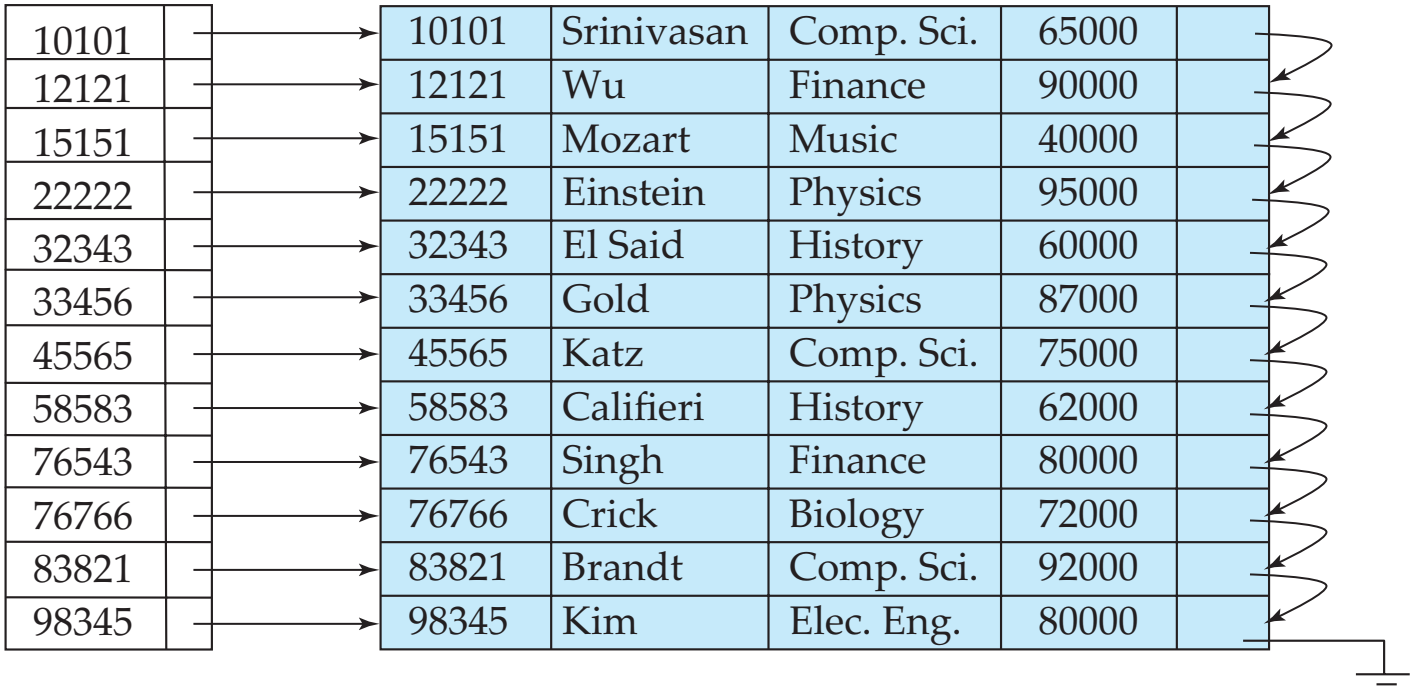
- Indeksna datoteka je tipično znatno manja od originalne datotetka
- Dve osnovne vrste indeksa:
  - Uređeni indeksi: ključevi pretrage su sačuvani u sortiranom redosledu
  - Heš indeksi: ključevi pretrage su distribuirani uniformno po “baketima” (**bucket**) na osnovu heš funkcije
- Efikasno podržani načini pristupa
  - Zapisi sa određenom (tačno specificiranom) vrednošću atributa
  - Zapisi kod kojih vrednost atributa pada u određeni (specificirani) opseg vrednosti
- Evaluacione metrike
  - Vreme: vreme pristupa (**access**), ubacivanja (**insert**), brisanja (**deletion**)
  - Prostor: utrošak prostora (**space overhead**)

Uređeni indeksi

- Uređeni indeksi – ulazi indeksa se čuvaju sortirani na osnovu vrednosti ključa pretrage
- Grupišući indeks (**clustering index**): ključevi određuju sekvencijalnu uređenost datotetka
  - Često se nazivaju primarnim indeksima (**primary index**)
  - Ključ pretrage je često, ali ne i neophodno, primarni ključ relacije
- Sekundarni indeks (**secondary index**): ključevi specificiraju redosled drugačiji od sekvencijalne uređenosti datoteke.
  - Često se nazivaju ne-grupišući indeksi (**nonclustering index**)
- Indeks-sekvencijalne datotetke (**index-sequential file**): sekvencijalna datoteka uređena po ključu pretrage sa grupišućim indeksom na osnovu ključa pretrage
- Gusti (**dense**) i retki (**sparse**) indeksi – na osnovu odnosa zapisa indeksa i ključeva pretrage

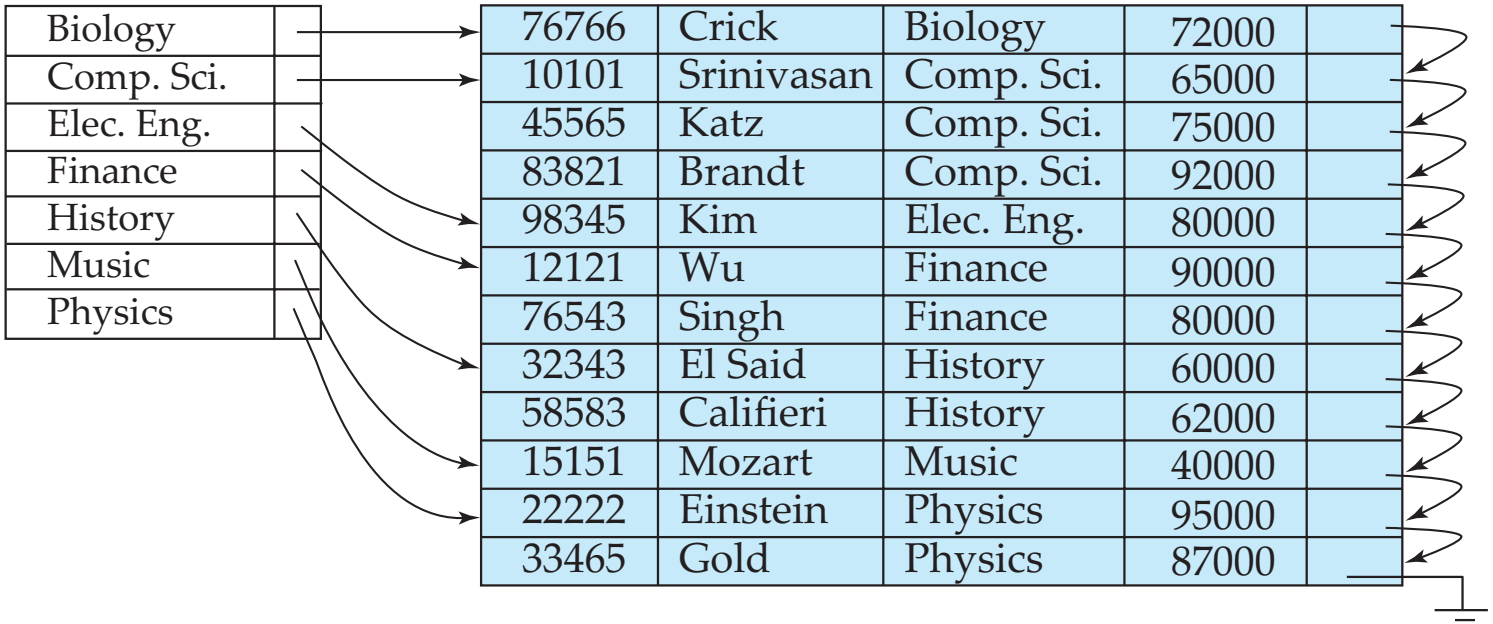
Gusti indeksi (dense index)

- Gusti indeks (**dense index**) – zapis indeksa postoji za svaku vrednost ključa pretrage u datoteci
- Npr. Gusti indeks po *ID* u relaciji *Instructor*



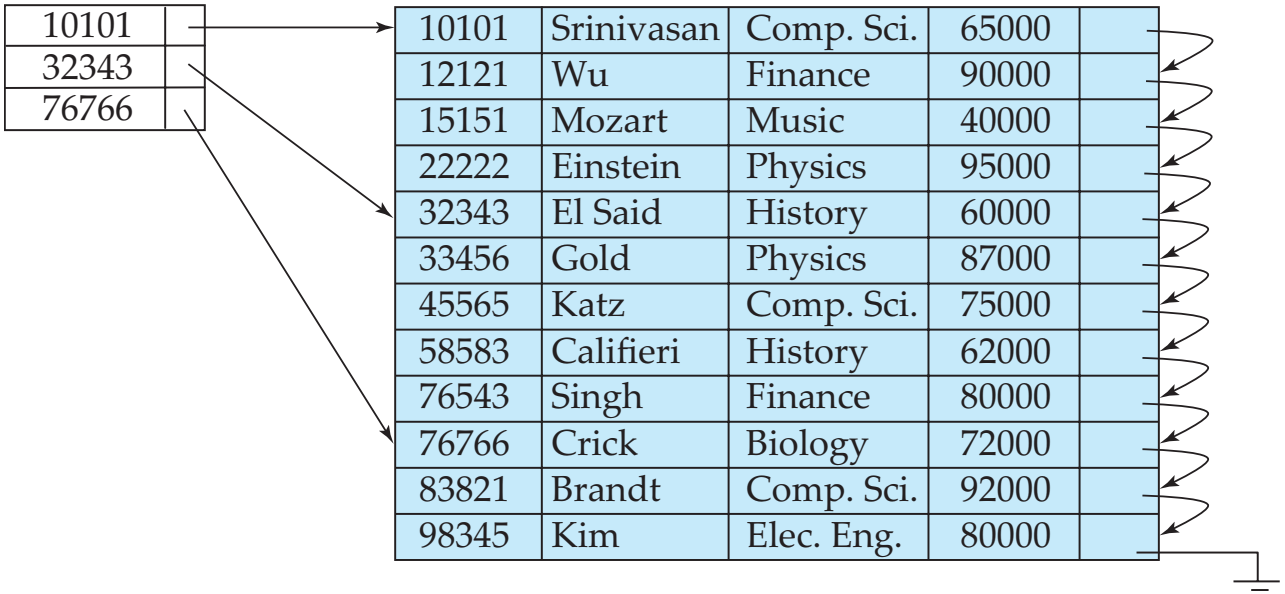
Gusti indeksi (dense index)

- Npr. Gusti indeks po *dept_name* u relaciji *Instructor* sortiranoj po tom atributu



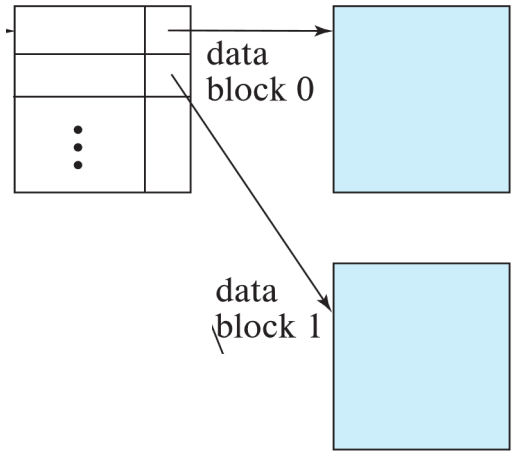
Retki indeksi (sparse index)

- Retki indeks (**sparse index**) – zapisi indeksa postoje samo za neke vrednosti ključa pretrage u datoteci
  - Primenljivo kada su zapisi datoteke sekvencijalno uređeni po ključu pretrage
- Da bi se locirao zapis za vrednost ključa pretrage K, potrebno je:
  - Naći zapis indeksa sa najvećom vrednošću ključa pretrage takvom da je < K
  - Sekvencijalno pretražiti ostatak datoteke, polazeći od pronađenog zapisa indeksa



Retki indeksi (sparse index)

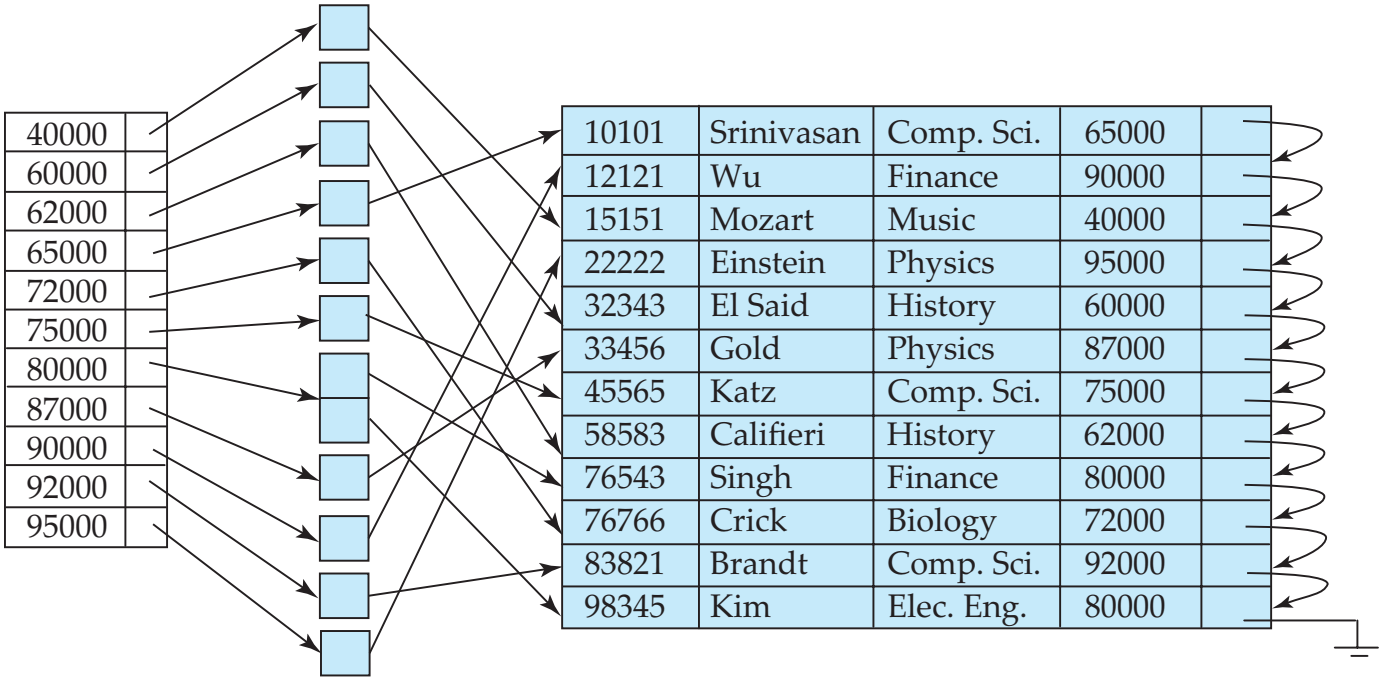
- U poređenju sa gustim indeksima:
  - Zauzimaju manje prostora i zahtevaju manje održavanja pri dodavanju i brisanju zapisa
  - Generalno su sporiji od gustih indeksa kada za potrebe pretrage zapisa
- **Dobar balans:**
  - Za grupišući indeks (clustering): retki indeks (sparse) sa zapisima indeksa, za svaki blok u datoteci, koji odgovara najmanjem ključu pretrage u tom bloku



  - Za ne-grupišući indeks (nonclustered): retki indeks (sparse) kreiran nad gustim indeksom (dense) da bi se formirao indeks sa više nivoa (**multilevel index**)

Sekundarni indeksi (secondary index)

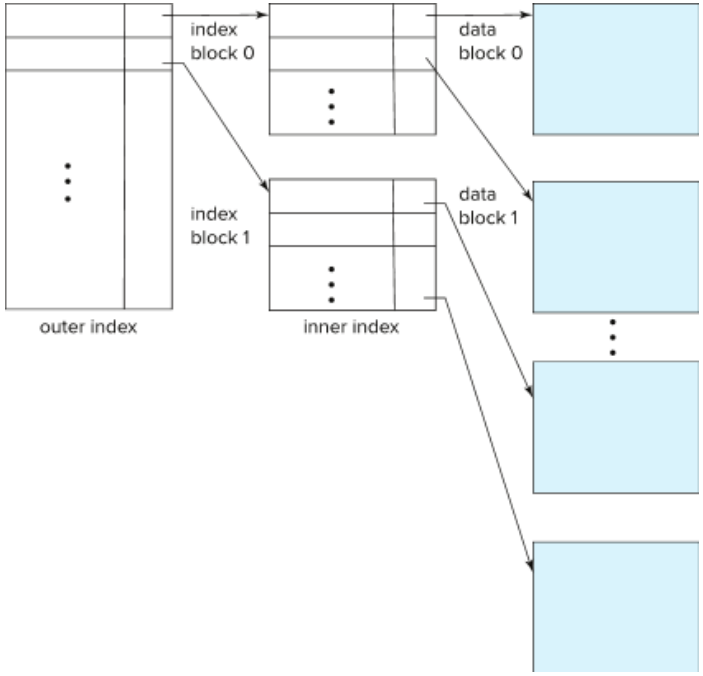
- Npr. Sekundarni indeks po atributu *salary* u relaciji *Instructor*



- Zapisi indeksa pokazuju na baket (**bucket**) koji sadrži pokazivače na sve zapise koji imaju određenu vrednost ključa pretrage
- Sekundarni indeks mora biti gust (dense)
- Sekvencijalna pretraga koristeći sekundarni indeks je “skupa” operacija

Indeksi sa više nivoa (multilevel index)

- Ukoliko indeks ne može da stane u memoriju, pristup postaje skup
- Rešenje: tretirati indeks koji se čuva na disku kao sekvencijalnu datoteku i konstruisati retki indeks (sparse) nad njim
  - Spolji indeks – retki indeks nad datotekom osnovnog indeksa
  - Unutrašnji indeks – datoteka osnovnog indeksa
- Ako je i spoljni indeks suviše veliki da bi stao u memoriju, onda se formirao novi nivo
- Indeksi na svim nivoima moraju biti ažurirani prilikom dodavanja i brisanja zapisa



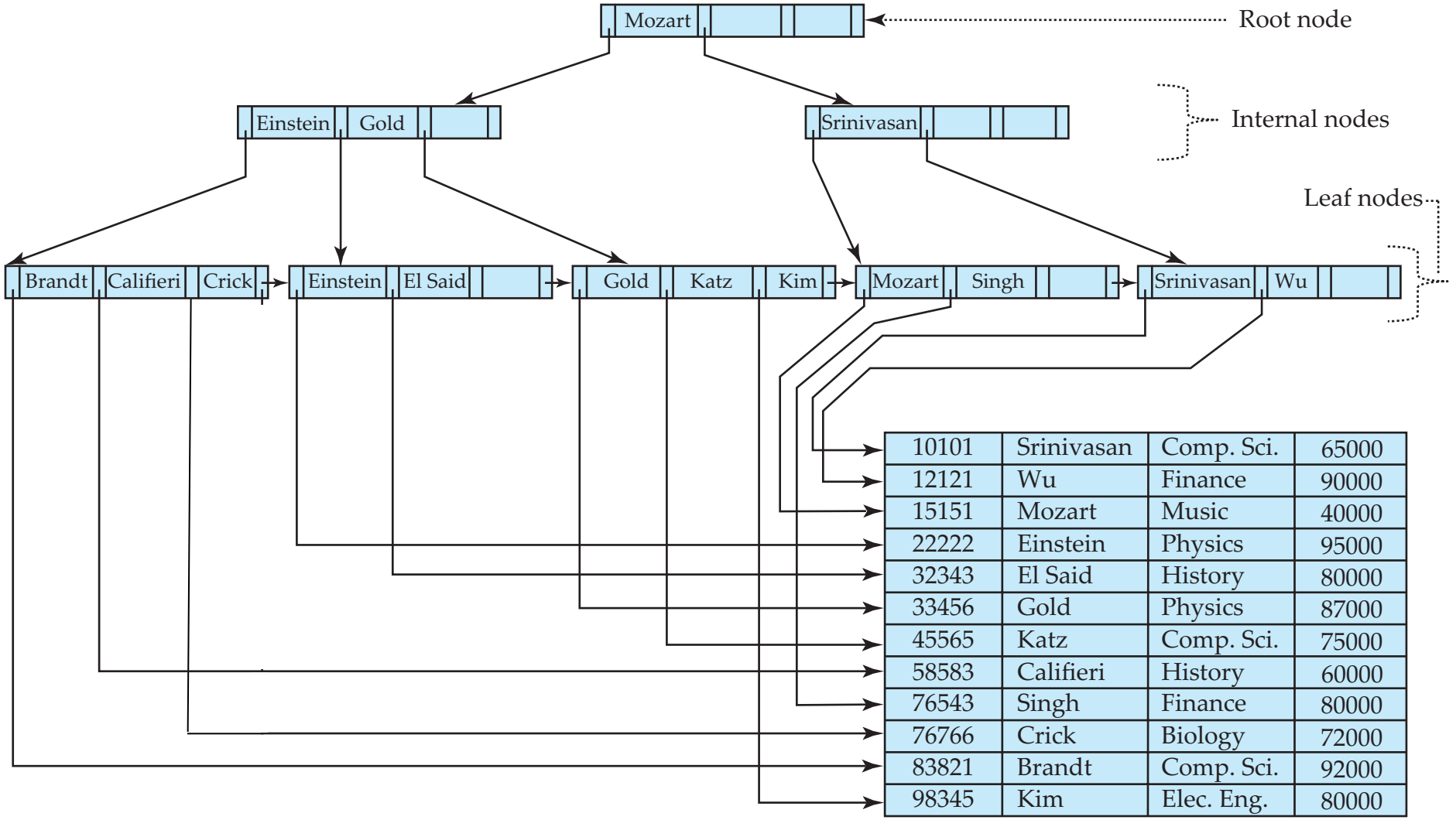
Indeksi sa više ključeva

- Kompozitni ključ pretrage – ključ pretrage se sastoji od više atributa
  - Npr. Indeks nad relacijom *Instructor* po paru atributa (*name*, *ID*)
  - Vrednosti ključeva se sortiraju leksikografski
    Npr. (John, 12121) < (John, 13514) kao i (John, 13514) < (Peter, 11223)
- Pretraga može biti samo po *name* ili po (*name*, *ID*)

Indeksi B^+-stabla

- Nedostaci indeks sekvencijalnih datoteka
  - Performanse degradiraju sa porastom datoteke, jer se kreiraju prelivni blokovi (**overflow block**)
  - Periodična reaogranizacija celikupnih datoteka je potrebna
- Prednosti indeksa B^+-stabla
  - Automatska reorganizacija sa malim, lokalnim, promenama usled dodavanja i brisanja zapisa
  - Reorganizacija cele datoteke nije potrebna da bi se očuvale performanse
- (Sitni) nedostaci indeksa B^+-stabla
  - Dodatni trošak vremena prilikom dodavanja i brisanja, trošak prostora
- Prednosti nadmašuju nedostatke, te se B^+-stabla intenzivno koriste

Indeksi B⁺-stabla - primer



10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	80000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	60000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

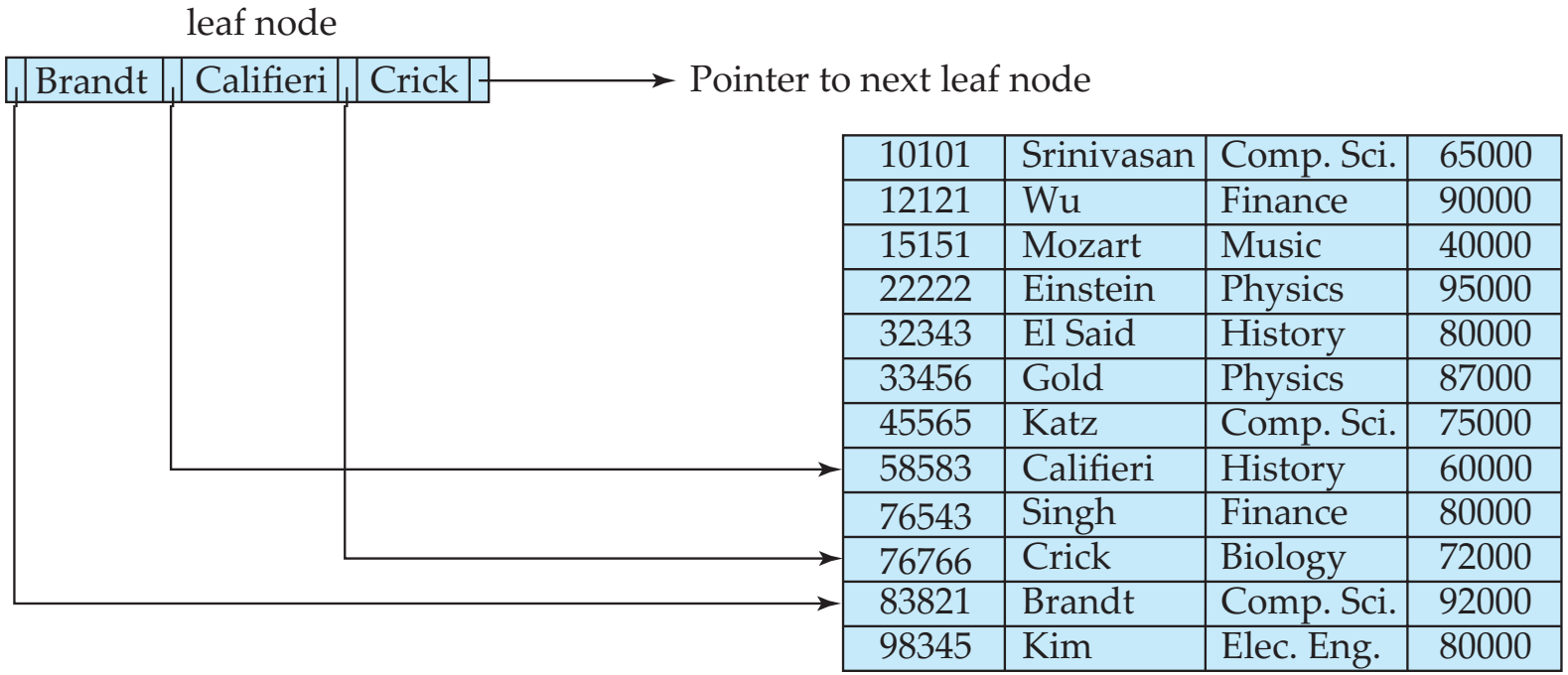
Indeksi B$^+$-stabla

- B$^+$-stabla su stabla koja zadovoljavaju sledeće osobine:
  - Sve putanje od korena do listova su iste dužine
  - Svaki čvor koji nije koren ili list ima između $\lceil n/2 \rceil$ i n dece
  - Svaki list ima između $\lceil (n-1)/2 \rceil$ i n-1 vrednosti
  - Specijalni slučajevi:
    Ako koren nije list, on ima najmanje 2 deteta
    Ako je koren ujedno list (tj. jedini čvorova u stablu) on ima između 0 i (n-1) vrednosti
- Tipičan čvor

  - K_i su ključevi pretrage
  - P_i su pokazivači na decu (u slučaju unutrašnjih čvorova)
    ili pokazivači na zapise ili grupe zapisa (u slučaju listova)
- Ključevi pretrage u čvoru su uređeni tj. sortirani $K_1 < K_2 < K_3 < \ldots < K_{n-1}$
- Pretpostavka je da nema duplikata ključeva pretrage

Indeksi B⁺-stabla – listovi

- Osobine listova u B⁺-stablu:
  - Za *i= 1, 2, …, n-1* pokazivači P_i pokazuju na zapise u datoteci sa ključem pretrage Ki
  - Ako su L_i, L_j listovi pri čemu $i < j$, onda je ključ pretrage lista L_i manji ili jednak ključu pretrage lista L_j
  - P_n pokazuje na sledeći list u redosledu sortiranosti ključeva pretrage



10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	80000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	60000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

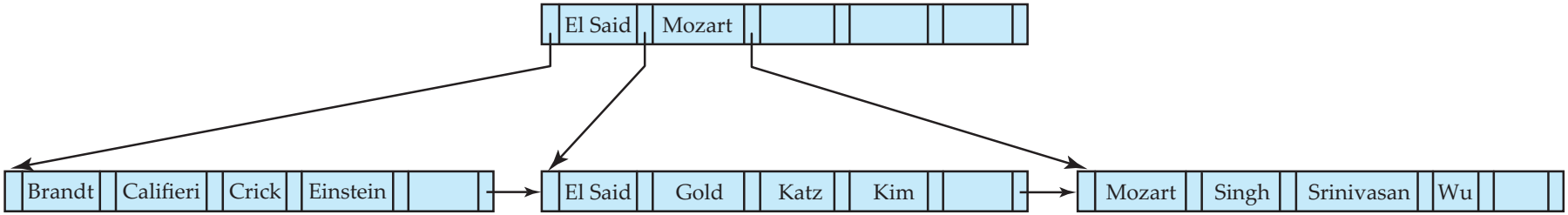
Indeksi B$^+$-stabla – unutrašnji čvorovi

- Unutrašnji čvorovi formiraju strukturu nalik retkim indeksima sa više nivoa nad listovima
  Za unutrašnji čvor sa *m* pokazivača:
  - Svi ključevi pretrage u podstablu na koje P1 pokazuje su manji od K1
  - Za $2 \leq i \leq n-1$ svi ključevi pretrage u podstablu na koje P_i pokazuje imaju vrednosti veće ili jednake K_{i-1} i manje od K_i
  - Svi ključevi pretrage u podstablu na koje P_n pokazuje imaju vrednosti veće ili jednake K_{n-1}

P_1	K_1	P_2	…	P_{n-1}	K_{n-1}	P_n

Indeksi B⁺-stabla – primer

- B⁺-stablo za Instructor relaciju (n = 6)



- Listovi moraju imati između 3 i 5 vrednosti ($\lceil (n-1)/2 \rceil$ i n-1, pri n = 6)
- Unutrašnji čvorovi koji nisu koren moraju imati između 3 i 6 dece ($\lceil n/2 \rceil$ i n, pri n = 6)
- Koren mora imati bar 2 dece

Indeksi B^+-stabla – zapažanja

- S obzirom da su unutrašnji čvorovi povezani pokazivačima
  onda blokovi koji su „logički“ blizu, mogu biti „fizički“ udaljeni
- Nivoi unutrašnjih čvorova fomiraju hijerarhiju retkih indeksa
- B^+-stabla imaju relativno mali broj nivoa

  Nivo ispod čvora ima najmanje $2 \bullet \lceil n/2 \rceil$ vrednosti

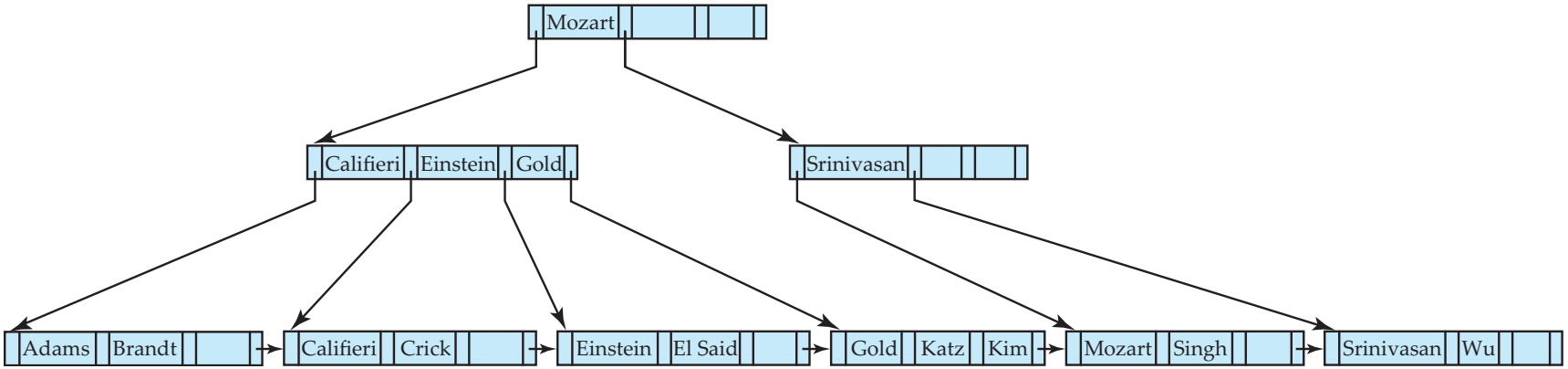
  Sledeći nivo ima najmanje $2 \bullet \lceil n/2 \rceil \bullet \lceil n/2 \rceil$ vrednosti

  Itd…
  - Ako u stablu ima K ključeva pretrage, onda je visina stabla ne veća od $\lceil \log_{\lceil n/2 \rceil} (K) \rceil$
  - To znači da se pretrage mogu obaviti efikasno
- Dodavanja i brisanja u glavnoj datoteci se mogu obaviti efikasno,
  jer se indeks (tj. stablo) restrukturira po logaritmaskoj složenosti

Indeksi B$^+$-stabla – pretraga

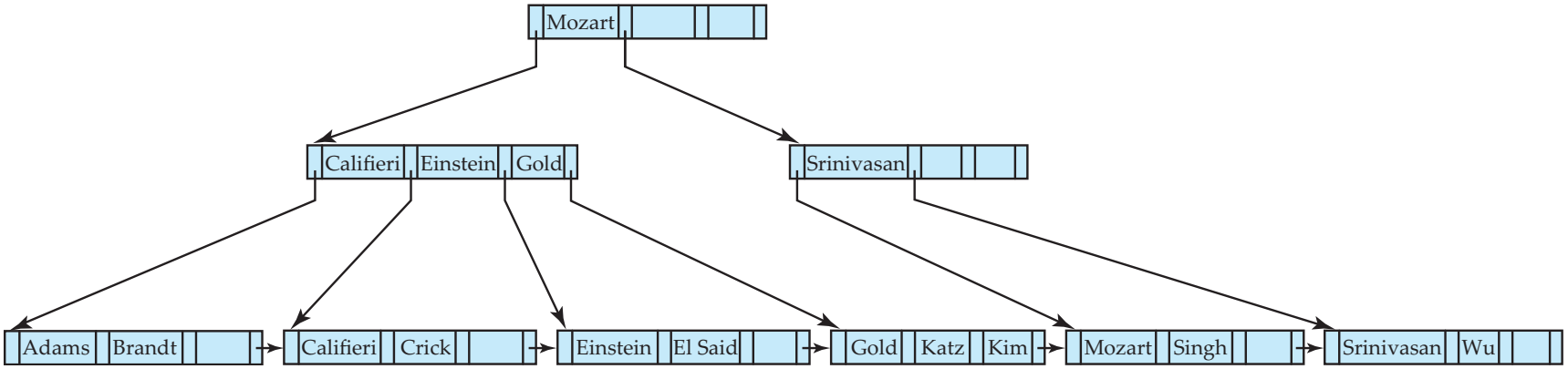
function *find(v)*

1. C=*root*
2. **while** (C is not a leaf node)
   1. Let *i* be least number s.t. $v \leq K_i$
   2. **if** there is no such number i **then**
   3. Set C = *last non-null pointer in C*
   4. **else if** ($v = C.K_i$) **then** Set C = P_{i+1}
   5. **else** set C = $C.P_i$
3. **if** for some i, $K_i = v$ **then** return $C.P_i$
4. **else** return null /* no record with search-key value *v* exists. */



Indeksi B$^+$-stabla – pretraga

- Pretraga za svim zapisima čiji je ključ pretrage u određenom opsegu (**range queries**)
  - Algoritam koji dohvati sve u opsegu **function** *findRange(lb, ub)*
  - Realna implementacija preko iteratora koji pozivom *next()* dohvata sledeći



Indeksi B$^+$-stabla – pretraga

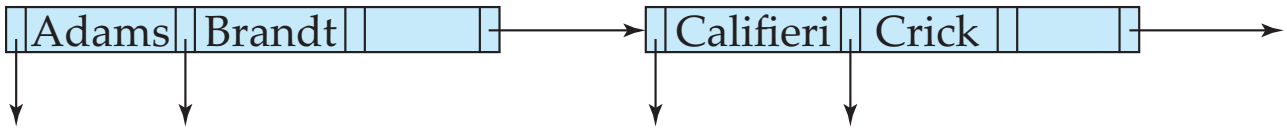
- Ako ima K ključeva pretrage u datoteci, visina stabla je ne veća od $\lceil \log_{\lceil n/2 \rceil} (K) \rceil$
- Čvor je generalno iste veličine kao i disk blok (tipično 4KB)
  - Tipično je n oko 100 (što znači da je 40B po zapisu indeksa)
- Za 1 milion ključeva pretrage i n = 100
  - Maksimalno se pristupa $\log_{50} (1{,}000{,}000) = 4$ čvora da se od korena stigne do lista
- Ukoliko se uporedi sa balansiranim binarnim stablom sa 1 milion ključeva pretrage kod koga je potrebno pristupiti do oko 20 čvorova kako bi se obavila pretraga
  - Razlika u broju potrebnih pristupa je značajna jer je za pristup jednom čvoru potrebna IO operacija koja je na magnetnom disku oko 10-20 ms

Indeksi B^+-stabla – dodavanja

- Pretpostavimo da je zapis već dodat u datoteku
  - Neka je *pr* pokazivač na zapis
  - Neka je *v* vrednost ključa pretrage za taj zapis
- Pronaći list u kome bi ključ pretrage treba da se pojavi
  1. Ukoliko ima mesto u listu, onda dodati par *(v, pr)* u taj list
  2. U suprotnom, podeliti list (uključujući i novi *(v, pr)* ulaz) i propagirati promenu naviše ka unutrašnjim čvorovima

Indeksi B$^+$-stabla – dodavanja

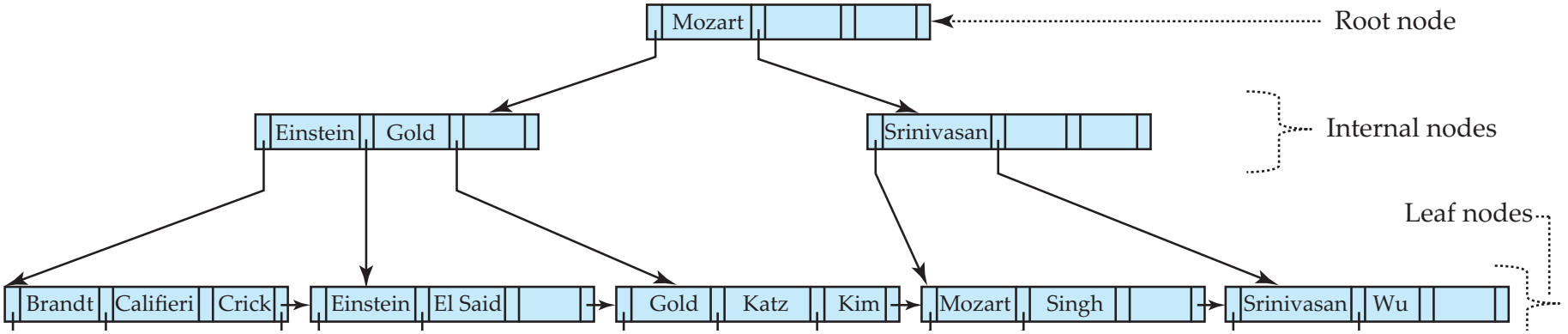
- Deljenje lista
  - Sortirati *n* parova (ključ pretrage, pokazivač) uključujući novododati par
    Ostaviti prvih $\lceil n/2 \rceil$ parova u originalni list, a ostatak prebaciti u novi list
  - Neka je novododati list *p*, neka je *k* poslednja vrednost u *p*.
    Dodati *(k, p)* u roditelja upravo podeljenog čvora
  - Ukoliko je roditelj pun, onda ga treba podeliti i propagirati deljenje naviše u stablu
- Deljenje čvorova se nastavlja naviše dok se ne naiđe na čvor koji nije pun
  - U najgorem slučaju biće potrebno podeliti koren što će povećati visinu stabla za 1

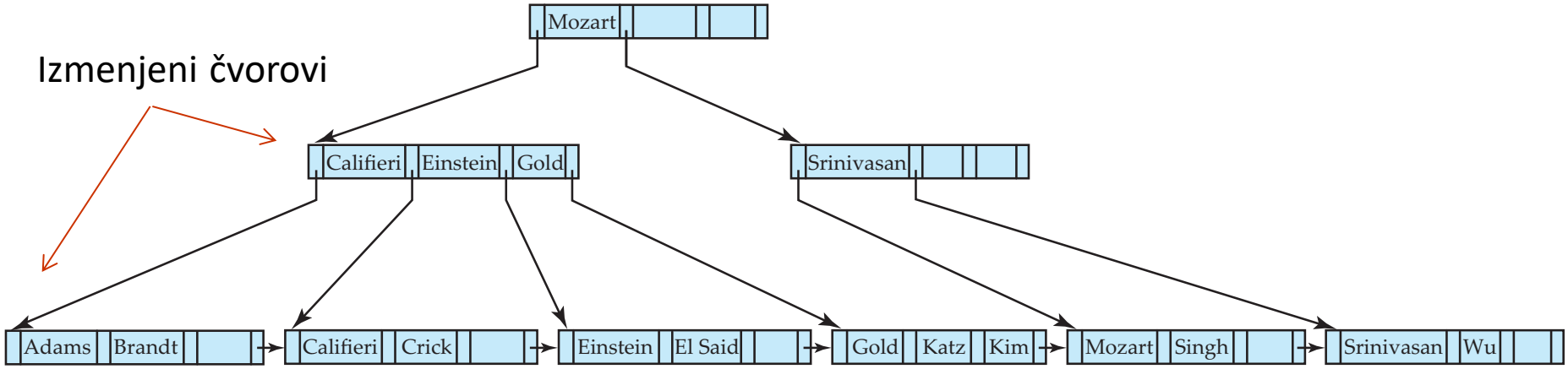


Rezultat deljenja čvora koji sadrži Brandt, Califieri i Crick nakon dodavanja Adams

Naredni korak: dodati ulaz sa (Califieri, pokazivač na novi čvor) u roditelja

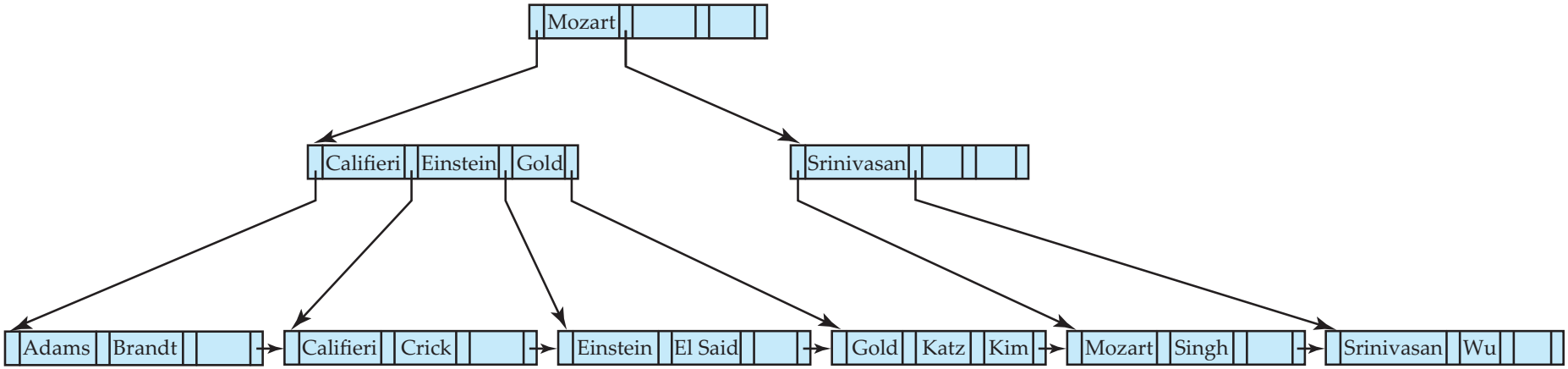
Indeksi B⁺-stabla – dodavanja





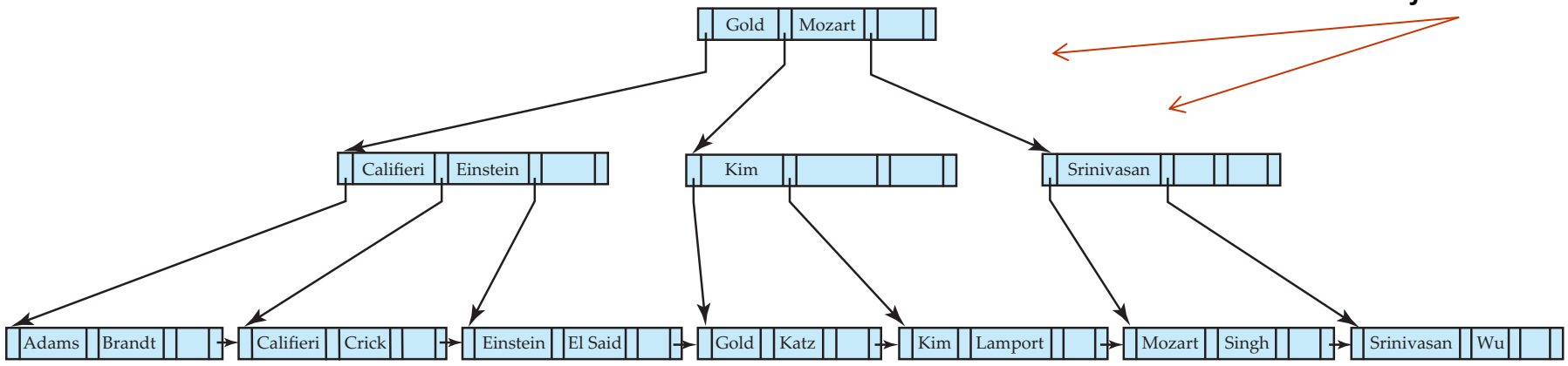
Izgled stabla pre i nakon dodavanja Adams

Indeksi B⁺-stabla – dodavanja



Izgled stabla pre i nakon dodavanja Lamport

Izmenjeni čvorovi

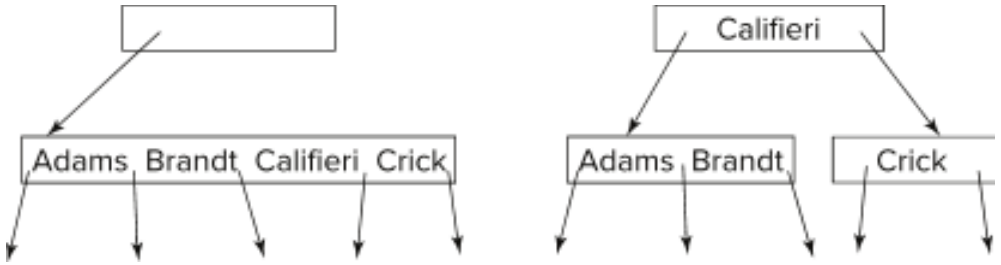


Izmenjeni čvorovi

Indeksi B$^+$-stabla – dodavanja

- Deljenje unutrašnjeg čvora: pri dodavanju *(k, p)* u već pun unutrašnji čvor *N*
  - Kopirati *N* u memoriju *M* sa prostorom za *n+1* pokazivač i *n* ključeva
  - Dodati *(k, p)* u *M*
  - Kopirati P_1, K_1, …, $K_{\lceil n/2 \rceil - 1}$, $P_{\lceil n/2 \rceil}$ iz *M* nazad u čvor *N*
  - Kopirati $P_{\lceil n/2 \rceil + 1}$, $K_{\lceil n/2 \rceil + 1}$, …, K_n, P_{n+1} iz *M* u novo-alocirani čvor *N'*
  - Dodati ($K_{\lceil n/2 \rceil}$, *N'*) u roditelja *N*

Primer



Indeksi B^+-stabla – brisanja

- Pretpostavimo da je zapis već obrisan iz datoteke
  - Neka je *pr* pokazivač na taj zapis
  - Neka je *v* vrednost ključa pretrage za taj zapis
- Potrebno je ukloniti *(v, pr)* iz lista:
  1. Ukoliko čvor ima suviše malo ulaza nakon uklanjanja, i ukoliko ulazi u čvoru i čvoru rođaku mogu da stanu u jedan čvor, onda treba **spojiti** te čvorove (rođake):

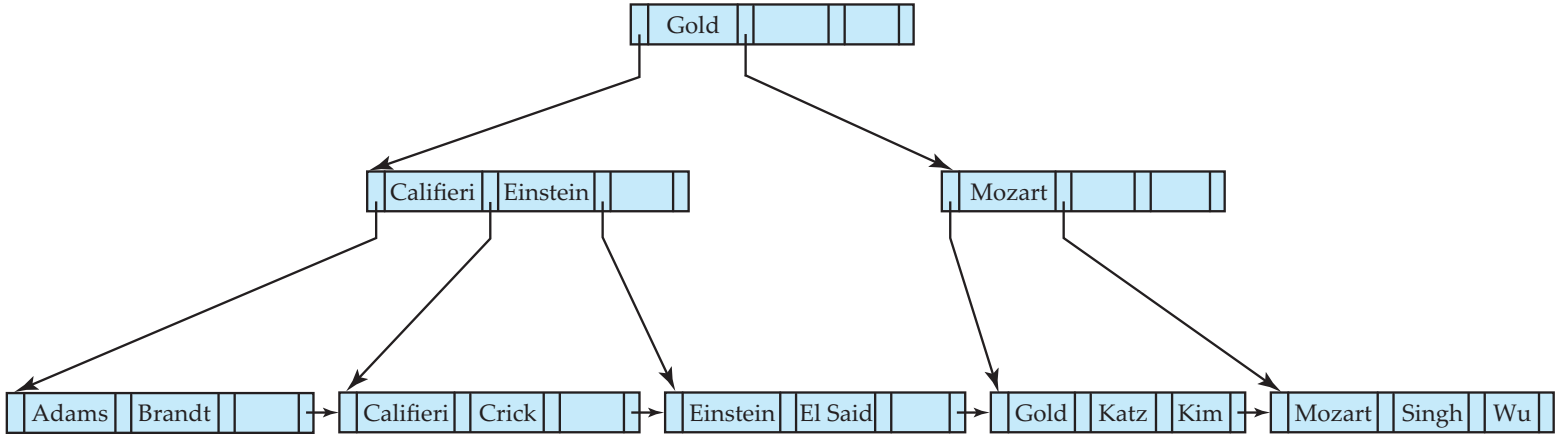
     Prebaciti sve ključeve iz ta dva čvora u jedan (onaj levi), a drugi obrisati (desni)

     Iz roditelja obrisati par (K_{i-1}, P_i), gde je P_i pokazivač na prethodno obrisani čvor
     Rekurzivno primeniti postupak

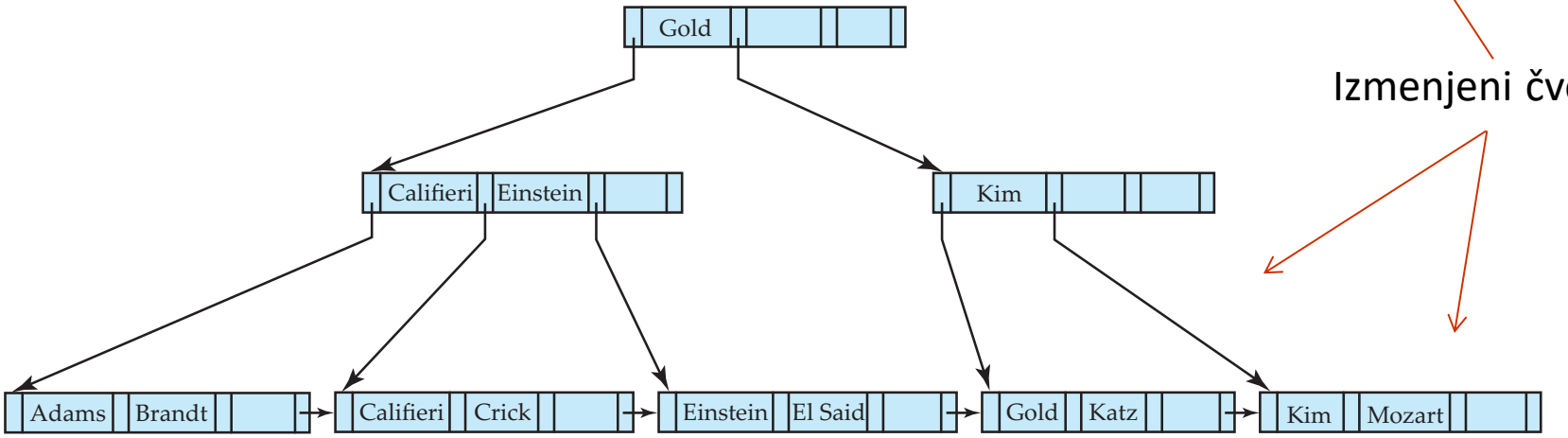
  2. Ukoliko čvor ima ima suviše malo ulaza nakon uklanjanja, ali ukoliko ulazi u čvoru i čvoru rođaku ne mogu da stanu u jedan čvor, onda treba **rasporediti** pokazivače:

     Rasporediti pokazivače između posmatranog čvora i njegovog rođaka
     tako da oba imaju više od potrebnog minimuma ulaza
     Ažurirati odgovarajuće ključeve u roditelju posmatranog čvora
- Brisanje može da dovode do kaskadnih promena naviše u stablu dok se ne naiđe na čvor koji ima $\lceil n/2 \rceil$ ili više pokazivača
- Ukoliko koren ima samo jedan pokazivač nakon brisanja, on se briše, a njegovo dete postaje

Indeksi B⁺-stabla – brisanja



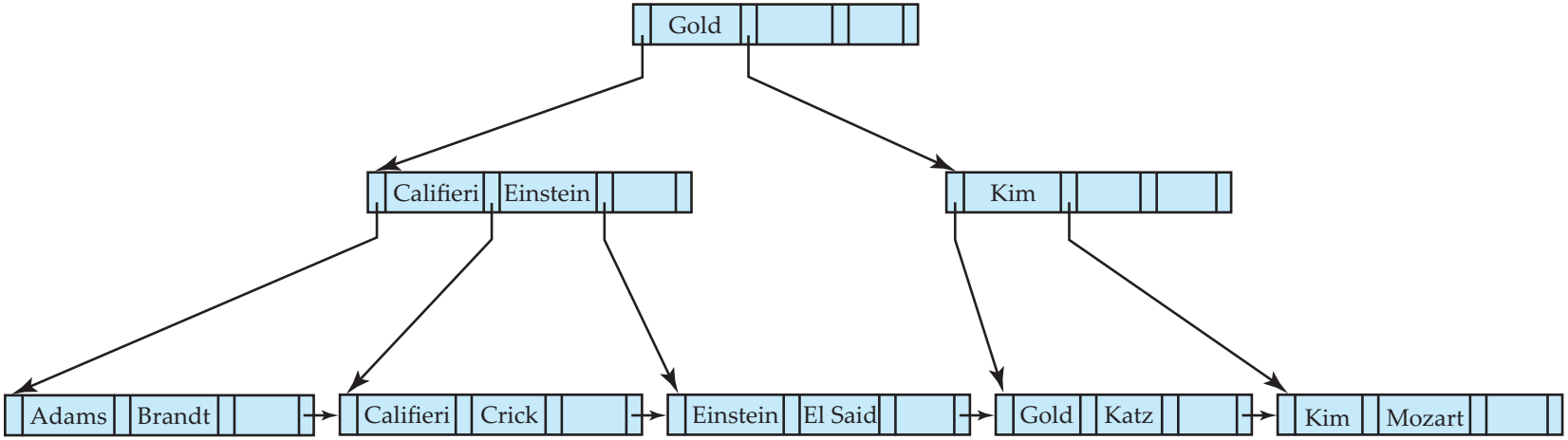
Izgled stabla pre i nakon brisanja Singh i Wu



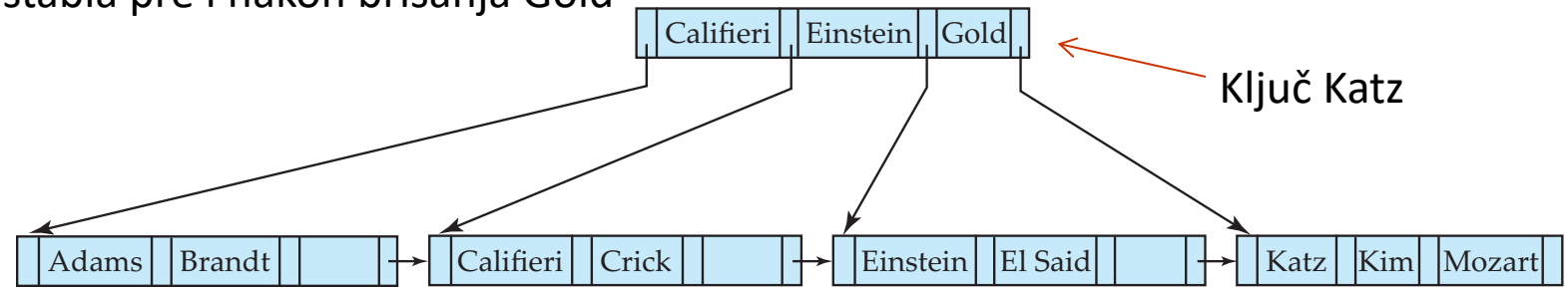
Izmenjeni čvorovi

List sa Singh i Wu postaje nedovoljno pun, pa pozajmljuje vrednost Kim od levog rođaka
Ključ se ažurira i u roditelju takođe

Indeksi B⁺-stabla – brisanja



Izgled stabla pre i nakon brisanja Gold



List sa Gold postaje nedovoljno pun, pa se spaja sa rođakom
Roditelj postaje nedovoljno pun, pa se spaja sa rođakom, pri tome se vrednost koja razdvaja dva čvora (kod roditelja) spušta dole pri spajanju
Koren koji ima samo jedno dete se briše

Indeksi B^+-stabla – algoritamska složenost izmena dodavanja i brisanja

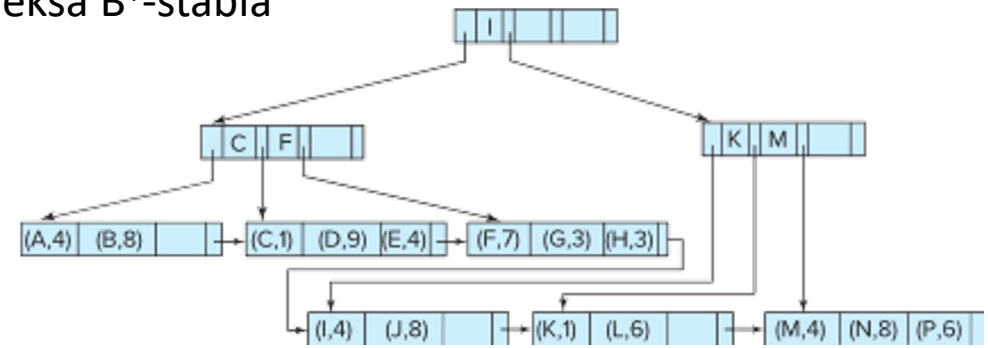
- Cena (broj potrebnih IO operacija) za dodavanje i brisanje jednog ulaza je proporcionalna visini stabla
  - Sa *K* ključa i maksimalnim granjem *n*, u najgorem slučaju kompleksnost je $O(\log_{\lceil n/2 \rceil}(K))$
- U realnim okolnostima, broj IO operacija je manji:
  - Unutrašnji čvorovi se drže u memoriji (tj. baferu)
  - Deljenja/spajanja su retka, jer većima dodavanja/brisanja utiču samo na listove
- Prosečna zauzetost čvora zavisi od redosleda dodavanja
  - Popunjenost od 2/3 pri slučajnom (random), odnosno 1/2 ½ pri sortiranom redosledu

Indeksi B$^+$-stabla – nejedinstveni ključevi pretrage (non-unique search key)

- Ako ključ pretrage a_i nije jedinstven, kreirati indeks na kompozitnom ključu (a_i, A_p) koji je jedinstven
  - A_p može biti primarni ključ ili bilo koji atribut koji je jedinstven
- Pretraga za $a_i = v$ može biti implementirana kao pretraga opsega po kompozitnom ključu sa opsegom od $(v, -\infty)$ do $(v, +\infty)$
- Više IO operacija je potrebno da bi se dohvatili konkretni zapisi
  - Ako je indeks grupišući (clustering), svi pristupi su sekvencijalni
  - Ako je indeks ne-grupišući (non-clustering), pristup do svakog zapisa može zahtevati IO operaciju (random access)
- Alternative:
  - Baketi u posebnim blokovima (loša ideja)
  - Lista pokazivača za svaki ključ pretrage
    - Dodatni kod koji obilazi liste, koje inače mogu biti zaista dugačke
    - Brisanje može biti skupo ukoliko ima mnogo duplikata (najgori slučaj → lineralno)
    - Mali utrošak dodatnog prostora, nema dodatnog troška pri pretrazi
  - Upotreba ID zapisa na disku da bi ključ postao jedinstven (umesto primarnog ključa)
    - Utrošak prostora zbog većeg (kompozitnog) ključa
    - Jednostavn kod za dodavanje i brisanje
    - Često u upotrebi

Organizacija datoteke kao B⁺-stabla

- Organizacija datotetka kao B⁺-stabla:
  - Listovi umesto sadrže zapise, a ne pokazivače na zapise
  - Omogućava čuvanje sortiranog redosleda zapisa (clustered) čak i prilikom dodavanje/brisanja/ažuriranja
- Listovi i dalje treba da budu polupopunjeni
  - Zapisi su veći od pokazivača, pa je maksimalni broj zapisa koji mogu biti sačuvani u listu daleko manji od broja pokazivača u unutrašnjim čvorovima
- Dodavanja i brisanja se sprovode na isti način kao dodavanja i brisanja ulaza kod indeksa B⁺-stabla



- Neophodno je dobro iskorišćenje prostora (zapisi zauzimaju više prostora od pokazivača)
- Kako bi se poboljšalo iskorišćenje prostora, uključiti više rođaka pri prerasporeživanju prilikom deljenja i spajanja čvorova - izbegavati deljenja/spajanja kad je to moguće tako da svaki čvor ima najmanje $\lfloor 2n/3 \rfloor$ ulaza

Ostala razmatranja kod indeksiranja

- Relokacija zapisa i sekundarni indeksi
  - Ukoliko se zapis relocira (pomera), svi sekundarni indeki koji čuvaju pokazivač na taj zapis moraju biti ažurirani
  - Deljenje čvorova kod organizacije datoteke kao B^+-stabla postaje skupo
  - Rešenje: umesto pokazivača na zapise u sekundarnim indeksima čuvati ključeve B^+-stabla (iz primarnog, grupišućeg indeksa)
    Dodati ID zapisa ukoliko ključ B^+-stabla nije jedinstven
    Potrebni su dodatni prolazi kroz datoteku da bi se locirao zapisa što dovodi do veće cene pretrage, ali deljenje čvorova je jeftinije
- B-stablo i B^+-stablo nije isto, ali se često koristi isti skraćeni termin (B-stablo, a misli na B^+)
  - Unutrašnji čvorovi mogu da sadrže ključeve koji se ne pojavljuju u listovima
  - Potreban je dodatni pokazivač u unutrašnjim čvorovima (pokazuje na zapis ili baket)
  - Prednosti: mogu da zauzimaju manje prostora i pretraga može biti brža (ne ide do lista)
  - Mane: unutrašnji čvorovi veći → veća dubina stabla, kompleksnije izmene
- Indeksiranje stringova
- Obimna dodavanja zapisa (**bulk loading**)
- Indeksiranje na fleš memoriji (SSD disk, umesto magnetnog diska)
- Indeksiranje u memoriji (operativnoj memoriji)

Indeksiranje stringova

- Stringovi varijabilne dužine kao ključevi
  - Varijabilan broj grananja u čvorovima
  - Iskorišćenje prostora kao kriterijum za deljenje čvora, a ne broj pokazivača
- Kompresija prefiksa
  - Unutrašnji čvorovi ne čuvaju kompletnu vrednost ključa, već njegov prefiks
    Čuva se dovoljno karaktera tako da se pojedini ulazi za podstabla mogu razlikovati
    Npr. Za Silas i Silberschatz se može iskoristiti prefiks Silb
  - Ključevi u listovima mogu biti komprimovani tako što dele zajednički prefiks

Obimna dodavanja zapisa (bulk loading)

- Dodavanje ulaza jedan-po-jedan u postojeće B^+-stablo zahteva više od 1 IO po zapisu
  - Pod pretpostavkom da listovi nisu mogli da stanu u memoriju
  - Može biti vrlo neefikasno dodavanje velikog broja zapisa odjednom (**bulk loading**)
- Efikasno rešenje 1:
  - Najpre sortirati ulaze
  - Dodavati u sortiranom redosledu
    (dodavanje u postojeće strane ili dolazi do deljenja)
    (znatno bolje IO performanse, ali je većina listova polupopunjena)
- Efikasno rešenje 2: - konstrukcija B^+-stabla odozdo-naviše (**bottom-up B^+-tree construction**)
  - Najpre sortirati ulaze
  - Kreirati stablo nivo-po-nivo, polazeći od nivoa listova
  - Većina DBMS poseduje alata za bulk loading

Indeksrianje na fleš memoriji

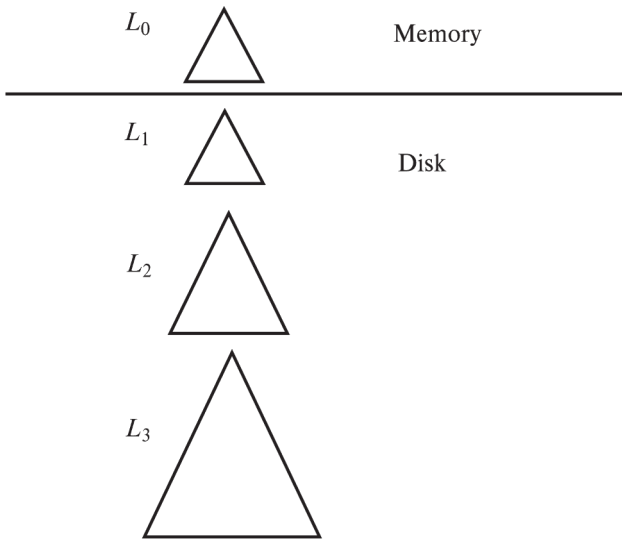
- Cena slučajnog (random) pristupa mnogo niža na fleš nego magnetnom disku
  - 20 do 100 mikrosekundi za čitanje/upis
- Upisi nisu trenutni, a eventualno mogu zahtevati skupo brisanje (više strana istovremeno)
- Optimalna veličina strane je mnogo manja
- Obimna dodavanja (bulk loading) i dalje korisna jer minimiziraju broj brisanja strana
- Koriste se indeksi optimizovane za upis (**write-optimized index**)
  kako bi se minimizirao broj strana u koje se vrši upis

Indeksi optimizovani za upis – write-optimized index

- Performanse B^+-stabla mogu biti skromne kada se radi o opterećenju sa velikim brojem dodavanja
  - Jedan IO po listu, uz pretpostavku da su unutrašnji čvorovi u memoriji
  - Magnetni disk (HDD) ostvaruje manje od 100 dodavanja po sekundi po disku
  - Fleš disk (SSD), jedna prepisivanje strane po dodavanju
- Dva pristupa za optimizaciju upisa (**write-optimized tree**):
  - Dnevnik strukturirana stabla spajanja (**Log-structured merge tree - LSMT**)
  - Bafer stablo (**Buffer tree**)

Dnevnik strukturirano stablo spajanja – Log Structured Merge Tree (LSM Tree)

- Dodavanje zapisa se najpre obavlja u stablo L_0 koje se čuva u memoriji
- Kada se L_0 napuni zapisi se prebacuju na diska (u L_1 stablo)
  - To je B+-stablo koje se konstruiše odozdo-naviše tako što se spaja postojeće L_1 stablo sa zapisima iz L_0
- Kada L_1 pređe određenu granicu popunjenosti, spaja se sa L_2
  - Isti princip za više nivoe LSM
  - Granica popunjenosti za L_{i+1} stablo je k puta veća od granice popunjenosti za L_i stablo
- Brisanje zapisa se sprovodi kao dodavanje specijalnog zapisa „brisanja“
  - Pretraga pronalazi originalne zapise i zapise „brisanja“, ali vraća samo one koji nemaju zapise „brisanja“
  - Prilikom spajanja stabala, ako se naiđe na zapis „brisanja“, briše se i on i originalni zapis
- Ažuriranje se sprovodi kao dodavanje+brisanje

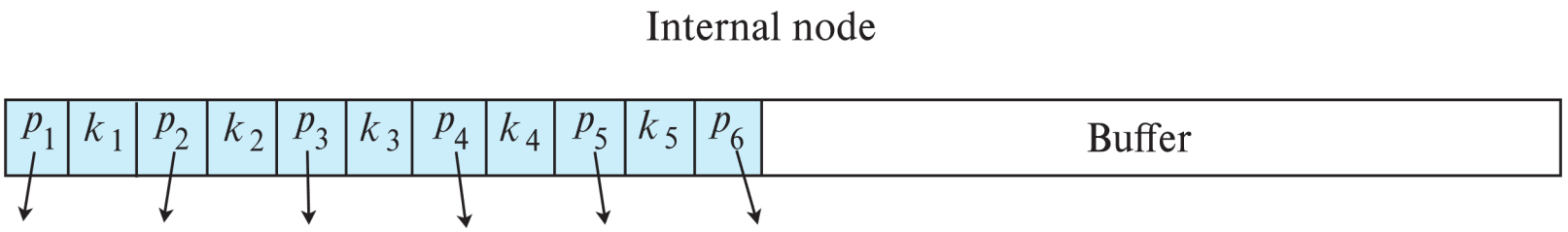


Dnevnik strukturirano stablo spajanja – Log Structured Merge Tree (LSM Tree)

- Prednosti LSM su:
  - Dodavanja se sprovode koristeći samo sekvencijalne IO
  - Listovi su puni, te je iskorišćenje prostora dobro
  - Redukovan broj IO po dodatom zapisu u odnosu na klasično B^+-stablo
- Nedostaci LSM su:
  - Pretraga treba da pretražuje više stabala
  - Celokupan sadržaj svakog nivoa se kopira više puta
- Dodatna optimizacija (**Stepped-Merged index**)
  - Varijanta LSM kod koje postoji više stabala na svakom nivou
  - Dodatno redukuje potreban broj upisa u odnosu na LSM indeks
  - Pretraga tj. upiti dodatno poskupljuju (upotreba **bloom filter** da se izbegne pretraga)
  - Koriste se kod BigData sistema (Google BigTable, Apache Cassandra, MongoDB)

Bafer stablo spajanja – Buffer Tree

- Bafer stablo predstavlja alternativu upotrebi LSM stabla
- Ključna ideja: svaki unutrašnji čvor B$^+$-stabla ima bafer koji čuva dodavanja
  - Dodavanja se spuštaju na niže nivoe tek kada se bafer napuni
  - Ako je bafer veliki (manji broj pokazivača), dubina stabla raste
  - Broj IO po zapisu se smanjuje
- Prednosti:
  - Manja kompleksnost pretrage
  - Može da se koristi sa bilo kojom indeksnom strukturom zasnovanom na stablu
  - Koristi PostgreSQL kod generalisanih stabala pretrage (Generalized Search Tree – GiST)
- Nedostatak:
  - Potreba za većim brojem IO u odnosu na LSM stablo



Heš indeksi – Hash Index

- Baket je prostor smeštanja koji čuva jedan ili više zapisa (baket je najčešće jedan blok)
  - Za dati ulaz baket se pronalazi tako što se nad ključem pretrage za taj ulaz primeni heš funkcija (**hash function**)
- Heš funkcija *h* je funkcija koja skup svih ključeva *K* mapira na skup svih baketa *B*
- Heš funkcija se koristi da locira zapise prilikom pristupa, dodavanja i brisanja
- Ulazi sa različitim ključem mogu biti mapirani na isti baket
  To znači da se prilikom pretrage mora sekvencijalno pretražiti ceo baket
- Kod heš indeksa, baket čuva ulaze sa pokazivačima na zapise
- Kod datoteke organizovane kao heš, baket čuva zapise

  Npr. Relacija *Instructor*
  ako se *dept_name* koristi kao ključ

bucket 0

bucket 1

15151	Mozart	Music	40000

bucket 2

32343	El Said	History	80000
58583	Califieri	History	60000

bucket 3

22222	Einstein	Physics	95000
33456	Gold	Physics	87000
98345	Kim	Elec. Eng.	80000

bucket 4

12121	Wu	Finance	90000
76543	Singh	Finance	80000

bucket 5

76766	Crick	Biology	72000

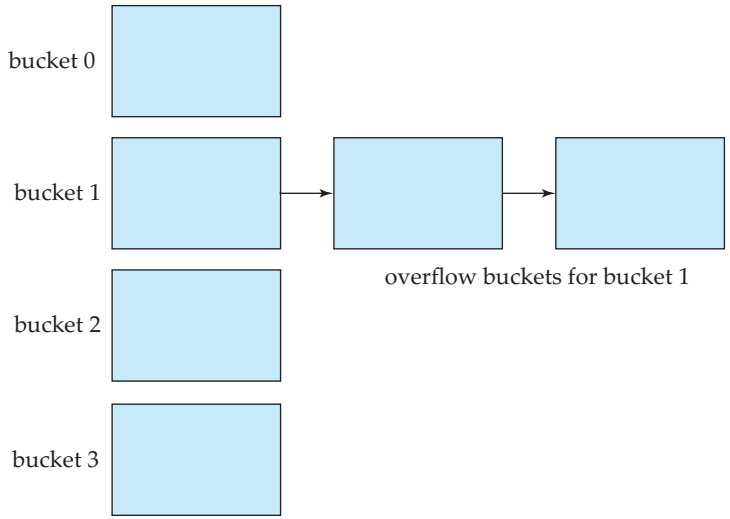
bucket 6

10101	Srinivasan	Comp. Sci.	65000
45565	Katz	Comp. Sci.	75000
83821	Brandt	Comp. Sci.	92000

bucket 7

Heš indeksi – prelivanje (overflow)

- Do prelivanje baketa (**bucket overflow**) može doći usled:
  - Nedovoljno baketa
  - Loše distribucije podataka
    (više zapisa sa istim ključem ili heš funkcija ne daje uniformnu raspodelu)
- Iako se verovatnoća prelivanja baketa može smanjiti, ne može se eliminisati
- Povezivanje prelivanja (overflow chaining)
  baketi prelivanja posmatranog baketa se povezuju u listu
  tzv. zatvoreno adresiranje (closed addressing)
  alternativna tzv. otvoreno adresiranje (open aaddressing)
  ne koristi bakete prelivanja, a inače se i ne koristi kod DBMS



Heš indeksi – statičko i dimaničko heširanje

- Statičko heširanje mapira ključeve K na fiksni skup baket adresa B:
  - Ukoliko je inicijalni skup baketa mali, a datoteka raste performanse opadaju
  - Ako se prostor alocira za očekivani rast datoteke, iskorišćenje prostora je nisko na početku
  - Ako se količina podataka smanjuje, iskorišćenje prostora je sve niže
- Potrebno je povremeno reorganizovati datoteku koristeći novu heš fukncju
  - Ometa uobičajan rad DMBS tokom trajanje reorganizacije
- Koristiti dinamičko heširanje koje dozvoljava dinamičku promenu broju baketa
  - Periodično reheširanje ako broj ulaza postane 1,5 puta veči od veličine heš tabele
    Kreiranje nove heš tabele koja je 2 puta veća od trenutne
    Ponovo heširati sve ulaze iz postojeće tabele u novu tabelu
  - Linerano heširanje: raditi reheširanje na inkrementalan način
  - Proširivo heširanje
    Prilagođeno za disk heširanje time što se baketi dele za različite ključeve
    Dupliranje broja ulaza u heš tabelu, bez dupliranja broja baketa

Poređenje sa heš indeksa i uređenih (sortiranim) indeksa

- Cena povremenog reorganizovanja
- Relativna frekvencija dodavanja i brisanja
- Da li je poželjno optimizovati prosečno vreme pristupa
  na štetu vremena pristupa u najgorem slučaju?
- Očekivani tipovi upita:
  - Heš je generalno bolji za pretragu za datu vrednost ključa
  - Za pretragu po opsegu (range queries), neophodni su uređeni indeksi
- U praksi:
  - PostgreSQL podržava heš indekse, ali obeshrabruje njihovu upotrebu (zbog loših performansi)
  - Oracle podržava datoteke sa heš organizacijom, ali ne i heš indekse
  - SQL Server podržava samo B^+-stabla

Indeksiranje po više ključeva

- Upotreba više indeksa za određene vrste upita
- Npr. **select** ID
  from instructor
  where dept_name=„Finance“ **and** salary=80000
- Moguće strategije ako postoje samo indeksi nad jednim ključem:
  - Upotrebiti indeks nad *dept_name* da se pronađu instruktori, a potom za svakog od njih proveriti uslov za *salary*
  - Upotrebiti indeks nad *salary* da se pronađu instruktori, a potom za svakog od njih proveriti uslov za *dept_name*
  - Upotrebiti *dept_nam*e da se nađu pokazivači na sve zapise instruktora iz “Finance“, potom na isti način upotrebiti *salary*, a iskoristiti presek ta dva skupa da bi se došlo do pokazivača na isntruktore koje treba dohvatiti
- Indeksi sa više ključeva (kompozitni ključevi), a pretraga leksikografksa
  $(a_1, a_2) < (b_1, b_2)$ ako je ili $a_1 < b_1$ ili ako je $a_1 = b_1$ i $a_2 < b_2$
  - U datom primeru bi bio korsitan indeks nad *(dept_name, salary)*
    koji bi takođe bio koristan i kod **where** dept_name=„Finance“ **and** salary<80000
    a koji ne bi bio koristan kod **where** dept_name < „Finance“ **and** salary=80000
    (jer bi bili dohvaćeni mnogi zapisi koji zadovoljavaju prvi, ali ne i drugi uslov)

Ostala razmatranja

- Pokrivajući indeksi (**covering index**)
  - Dodavanje dodatnih atributa u indeks kako se za određene upite ne mora pristupati zapisima već samo ulazima indeksa
  - Čuvati dodatni atribute samo u listovima
  - Naročito korisni kod sekundarnih indeksa
- Indeksi za upite nad više ključeva – bitmap indeksi (**bitmap index**)
- Indeksi za prostorne i vremenske podatke (**k-d tree, R-tree**)

Bitmap indeksi

- Dizajnirani za upite nad više ključeva, u situacijama kada se podaci dodaju, a nikada ili retko menjaju (npr. višedimenzioni upiti u skladištima podataka)
- Pretpostavka da su zapisi u relaciji sekvencijalno numerisani polazeći od 0
  - Za dati broj *n* se lako dohvata zapis *n*
  - Naročito upotrebljivo ako su zapisi fiksne veličine
- Upotrebljivo kod atributa koji imaju relativno mali broj mogućih različitih vrednosti
  - Npr. Atributi poput: pol, zemlja, država
  - Npr. Atributi koji su klasifikovani u klase: nivo zarade
- Bitmap indeksi su nizovi bitova – postoji bitmapa za svaku moguću vrednost atributa
  - U bitmapi za vrednost *v*, bit za neki zapis je 1 ukoliko taj zapis ima vrednost *v* za posmatrani atribut, a u suprotnom je 0

Npr. Upit muškarci sa zaradom nivoa L1:

10010 AND 10100 = 10000

Dohvata se samo jedan zapis
Brojanje zapisa još efikasnije

record number	*ID*	*gender*	*income_level*
0	76766	m	L1
1	22222	f	L2
2	12121	f	L1
3	15151	m	L4
4	58583	f	L3

Bitmaps for *gender*

m	10010
f	01101

Bitmaps for *income_level*

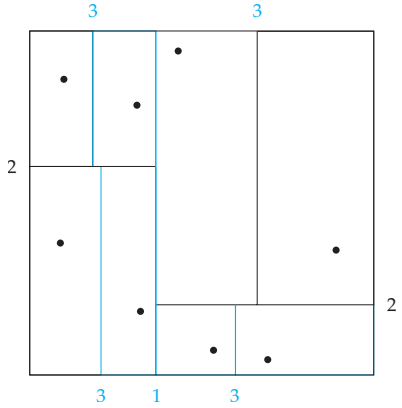
L1	10100
L2	01000
L3	00001
L4	00010
L5	00000

Bitmap indeksi

- Bitmap indeksi su generalno veoma mali u poređenju sa veličinom relacije
  - Npr. Ako je zapis 100B, prostor koji zauzima jedna bitmapa je 1/800 prostora relacije
  - Ako je broj različitih vrednostri atributa 8, onda bitmapa zauzima koliko i 1% relacije
- Bitmape se pakuju u reči (pa se porede po 32 ili 64 bita istovremeno)
  - Npr. Bitmapa sa 1 milion bitova se and/or sa svega 31,250 instrukcija
- Brojanje bitova sa vrednošću 1 se može sprovesti brzo:
  - Iskoristiti svaki bajt da se indeksira unapred sračunati niz od 256 elemenata od kojih svaki čuva broj jedinica u binarnoj reprezentaciji (poput brojača) Čak se može dodatni ubrzati ukoliko se koriste parovi bajtova
  - Potom se saberu dohvaćeni brojači
- Bitmape se mogu koristiti umesto liste ID zapisa u listovima B^+-stabla, za ključeve koji imaju veliki broj odgovarajućih zapisa
  - Upotrebljivo ukoliko više od 1/64 zapisa ima datu vrednost, ako se pretpostavi da je ID zapisa veličine 64 bita
  - Ova tehnika objedinjuje prednosti bitmap i B^+-stablo indeksa

Indeksiranje prostornih podataka

- Baza može čuvati podatke o linijama, poligonima, kao i dvodimenzionim slikama
  - Upiti mogu imati prostorne uslove (npr. sadržanje ili preklapanje objekata)
  - Upiti o najbližim susedima (traženje objekata koji su najbliži datoj tački)
  - Upiti nad opsezima koji koriste prostorne regione
    (objekti potpuno ili delimično u zadatom regionu)
  - Upiti koji određuju presek ili uniju regiona
  - Prostorno spajanje dve relacije po atributima koji čuvaju informaciju o lokaciji
- Najranije strukture za indeksiranje više dimenzija (**k-d tree**)
  - Svaki nivo k-d stabla particioniše prostor na dva dela
    Prvi nivo stabla particioniše prostor po jednoj dimenziji
    Naredni nivo stabla particioniše prostor po drugoj dimenziji i tako naizmenično
  - Svaki čvor deli trenutno sačuvane lokacije na po pola u dva podstabla
  - Particionisanje staje kada čvor ima manje
    od određenog broja tačaka
- Proširenje k-d-B stabla koje podržava
  veći broj dece za asvaki od unutrašnjih čvorova
  što je pogodno za sekundarne indekse

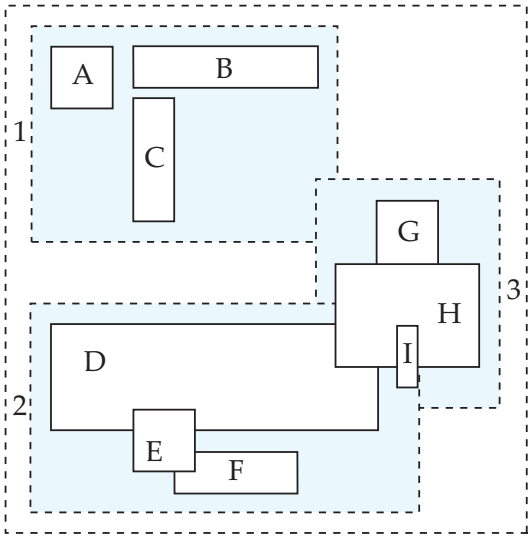


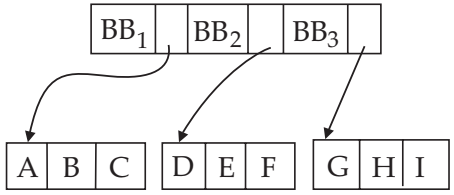
Indeksiranje prostornih podataka

- R-stabla (R-tree) su N-dimenziono proširenje B^+-stabla, što je korisno za indeksiranje skupova pravouganika i poligona
- Osnovna ideja: generalizovati pojam jedno-dimenzionog intervala pridruženom svakom od čvorova u B^+-stablu na N-dimenzioni interval
  - Radi dobro samo za mali broj N, tako da se često korsti za N=2
  - Granični okvir (**bounding box**) nekog čvora je minimalni okvir (pravouganik) koji sadrži sve pravougaonike/poligone pridružene tom čvoru
    Granični okviri dece posmatrganog čvora se smeju preklapati
- Npr. Skup pravouganika (puna linija) i skup graničnih okvira (isprekidana linija)

Pretraga za objektima koji se preklapaju sa datom tačkom:

1. Ako je čvor list, onda vratiti sve objekte čiji ključevi se preklapaju sa datom tačkom
2. U suprotnom, svako dete posmatranog čvora čiji se granični okvir preklapa sa datom tačkom rekurzivno pretražiti





Indeksiranje vremenskih podataka

- Vremenski podaci su podaci koji imaju pridruženi vremenski period (interval validnosti)
- Vremenski interval ima početno i krajnje vreme
  - Kranje vreme se postavlja na beskonačno ako je podatak trenutno važeći ili na neki veoma veliki datum (npr. 31.12.9999)
- Upit može tražiti sve podatke koji su bili validni u nekom trenutku u vremenu
  - Indeksiranje po periodu validnosti ubrzava ovu pretragu
- Da bi se kreirao vremenski indeks po atributu *a* potrebno je:
  - Upotrebiti prostorni indeks, poput R-stabla, sa atributom *a* kao jednom dimenzijom a vremenom kao drugom dimenzijom (vreme validnosti formira interval)
  - Redovi koji su trenutno validni predstavljaju problem (zato što je granica jako velika) Što se može rešiti čuvanjem svih trenutno validnih redova u posebnom indeksu koji se formira nad *(a, start-time)*, a kako bi se pronašli svi redovi u nekom trenutku vremena t potrebno je pretražiti redove u intervalu od *(a, 0)* do *(a, t)*
- Vremenski indeks na primarnim ključem može pomoći obezbeđivanju ograničenja vremenskog primarnog ključa