Adatbázisok elmélete 21. előadás

Katona Gyula Y.
Budapesti Műszaki és Gazdaságtudományi Egyetem
Számítástudományi Tsz.
I. B. 137/b
kiskat@cs.bme.hu
http://www.cs.bme.hu/~kiskat

2005

Sorosíthatóság és zárak

Zárakat használunk, figyelünk arra, hogy legális legyen az ütemezés, és még figyelünk valamire, ami biztosítja a sorosíthatóságot.

Egyszerű tranzakció modellben vagyunk (egy fajta zár van csak és a korábbi három feltevés mindig fennáll, azaz a zárkérés legális) és még valamit felteszünk:
A sorosíthatóságról pusztán a zárkérések alapján fogunk dönteni, nem nézzük azt, hogy ezeken kívül milyen műveletek (írások/olvasások) vannak. Pontosabban:

Nem foglalkozunk azzal, hogy $LOCK_i(A)$ és $UNLOCK_i(A)$ között mi történik, feltesszük hogy valami teljesen egyedi írás és olvasás is. Ez hasonlít ahhoz a helyzethez, mint amikor a konkrét számolásokat elhanyagoltuk: feltesszük, hogy a lehető legrosszabb történik azalatt, amíg a tranzakciónál van a zár.

Így persze megint igaz lesz az, hogy olyan ütemezéseket is rossznak minősítünk, amik igazából sorosíthatók lennének, ha megnéznénk, hogy írások vagy olvasások történnek, de ez nem baj, mert szigorúbbak lehetünk, csak az a fontos, hogy olyan ne legyen sorosíthatónak minősítve, aki nem az.

Éhezés, zárak és sorosíthatóság

Másik probléma, ami zárakkal kapcsolatban előfordulhat: **éhezés**: többen várnak ugyanarra a zárra, de amikor felszabadul mindig elviszi valaki a tranzakció orra elől.
Megoldás: adategységenként FIFO listában tartani a várakozókat

Eddig azt láttuk csak, hogy mennyi baj lehet a zárak alkalmazásával (holtpont, éhezés). Most nézzük, hogy mire jók a zárak.

A zárak segítségével el lehet majd érni, hogy az ütemezések sorosíthatók legyenek, de pusztán az, hogy használjuk a zárakat, még nem ad sorosítható ütemezést.

Példa olyan legális, zárakat használó ütemezésre, ami nem sorosítható: a korábbi, nem sorosítható, írásokból és olvasásokból álló ütemezésbe zárakat rakunk:

$$l_1(A),\ r_1(A),\ w_1(A),\ u_1(A),\ l_2(A),\ r_2(A),\ w_2(A),\ u_2(A),$$
$$l_2(B),\ r_2(B),\ w_2(B),\ u_2(B),\ l_1(B),\ r_1(B),\ w_1(B),\ u_1(B)$$

Sorosítási gráf az egyszerű tranzakciómodellben

Az előbbiek értelmében tehát egy olyan legális ütemezésről akarjuk eldönteni, hogy sorosítható-e, amiben csak zárkérések és zárelengedések vannak.

Mikor lesz egy ilyen ütemezés sorosítható, függetlenül attól, hogy milyen írások és olvasások történnek valójában?

Ennek megválaszolásában segít a **sorosítási gráf**: csúcsai a tranzakciók és akkor van él T_i-ből T_j-be, ha az ütemezésben van olyan $u_i(A)\ \ldots\ l_j(A)$ rész, ahol $u_i(A)$ (T_i elengedi A zárját) és $l_j(A)$ (T_j megkapja A zárját) között A-ra senki se kap zárat.

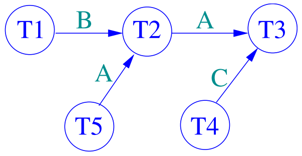
Ekkor minden olyan soros ütemezésben, ami ekvivalens lehet a miénkkel, biztos, hogy T_j-nek T_i után kell jönnie.
Ez azért van így, mert feltettük, hogy T_i is és T_j is bármit csinálhat A-val, amíg nála van a zár és ha pl. T_i írja, T_j meg olvassa A-t, akkor már csak a $T_i, \ldots, T_j$ sorrend lesz a jó.

Példa sorosítási gráfra

Az alábbi, csak zárkéréseket és zárelengedéseket tartalmazó ütemezés legális (HF: leellenőrizni):

$$l_5(A),\ \ l_1(B),\ \ u_5(A),\ \ l_4(C),\ \ u_1(B),\ \ l_2(A),\ \ l_2(B),\ \ u_2(A),$$
$$l_3(A),\ \ u_3(A),\ \ u_4(C),\ \ u_2(B),\ \ l_3(C),\ \ u_3(C)$$

Az ehhez tartozó sorosítási gráf:



Tétel a sorosítási gráfról

Tétel. *Egy csak zárkéréseket és zárelengedéseket tartalmazó egyszerű tranzakció modellbeli ütemezés pontosan akkor sorosítható, ha az előbbi módszerrel felrajzolt sorosítási gráf DAG.*

Bizonyítás: $\Rightarrow$: Ha nem DAG a gráf, akkor van benne irányított kör. Ebben a körben levő tranzakciók közül egyik sem előzheti meg a többit egy ekvivalens soros ütemezésben, amiből következik, hogy nincs ekvivalens soros ütemezés.

$\Leftarrow$: Teljes indukcióval: $n = 1$-re (1 tranzakció van csak) világos, egy ilyen ütemezés maga soros.
Legyen most az ütemezésben n tranzakció. Ha a gráf DAG, akkor létezik topologikus rendezése. Azt látjuk be, hogy a topologikus sorrend szerinti soros ütemezés ekvivalens lesz az eredeti ütemezéssel.
Ha T_i a topologikus rendezés szerinti első tranzakció, akkor nem megy bele él, vagyis ő csak olyan adategységeket használ, amiket az eredeti ütemezésben előtte senki. Így az ő összes utasítását előre mozgathatjuk, a hatás nem változik.
Ami ezután marad, az $n-1$ tranzakció utasításaiból álló ütemezés, aminek a sorosítási gráfja szintén DAG, tehát ennek az indukció szerint létezik soros ekvivalense (a maradék tranzakciók topologikus sorrendjének megfelelően), ami T_i-vel kiegészítve soros ekvivalense lesz az eredetinek.

Következmény

Következmény: A bizonyításból látszik, hogy a soros ekvivalensek és a lehetséges topologikus sorrendek megfelelnek egymásnak, vagyis annyi soros ekvivalens lesz, ahány különböző topologikus sorrend van.

Például a korábban látott sorosítási gráf esetén 8 darab topologikus sorrend van, így nyolc soros ekvivalens van:

$T_5\ T_4\ T_1\ T_2\ T_3,$
$T_4\ T_5\ T_1\ T_2\ T_3,$
$T_4\ T_1\ T_5\ T_2\ T_3,$
$T_5\ T_1\ T_4\ T_2\ T_3,$
$T_1\ T_5\ T_4\ T_2\ T_3,$
$T_1\ T_4\ T_5\ T_2\ T_3,$
$T_5\ T_1\ T_2\ T_4\ T_3,$
$T_1\ T_5\ T_2\ T_4\ T_3,$

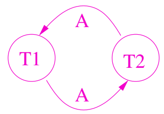
Példa, ami mutatja, hogy szigorúbbak vagyunk a kelleténél

Tekintsük az

$$l_1(A),\ \ r_1(A),\ \ u_1(A),\ \ l_2(A),\ \ r_2(A),\ \ u_2(A),\ \ l_1(A),\ \ w_1(A),\ \ u_1(A),\ \ l_2(B),\ \ r_2(B),\ \ u_2(B)$$

ütemezést.
Ha megnézzük az írás/olvasás műveleteket $(r_1(A),\ \ r_2(A),\ \ w_1(A),\ \ r_2(B))$, akkor látszik, hogy az ütemezés hatása azonos a T_2T_1 soros ütemezés hatásával, vagyis ez egy sorosítható ütemezés.

De ha felrajzoljuk a sorosítási gráfot (és ilyenkor persze nem nézzük, hogy milyen írások/olvasások vannak, hanem a legrosszabb esetre készülünk), akkor



lesz a gráf, és mivel ez nem DAG, ezért nem lesz sorosítható az az ütemezés, amiben már csak a zárak vannak benne.

Az ütemező lehetőségei a sorosíthatóság kikényszerítésére

1. Figyeli a sorosítási gráfot (amit a zárkérések alapján készít) és ha kör keletkezne, akkor az egyik körbeli tranzakciót ABORT-álja.
   (Előny: nem óvatoskodik, nem korlátoz feleslegesen;
   Hátrány: drasztikus megoldás az ABORT)
2. Protokollt ír elő a tranzakciók számára, amit minden egyes tranzakciónak be kell tartania: **2PL (two-phase locking, kétfázisú protokoll)**: a T_i tranzakció követi a kétfázisú protokollt, ha $UNLOCK_i$ után nincs $LOCK_i$, azaz ha nem kér már zárat miután elengedett már egyet.

   Tétel. *Ha az egyszerű tranzakciómodellbeli legális ütemezésben minden tranzakció követi a 2PL-t, akkor az ütemezéshez tartozó sorosítási gráf DAG, azaz az ütemezés sorosítható.*

Bonyolultabb zármodellek

Többfajta zár van, aszerint, hogy a tranzakciók mit akarnak csinálni az adattal. (Persze akkor, ha van több különböző művelet, nem csak írás és olvasás.)

Cél, hogy minél jobban tükrözzék a zárkérési lehetőségek a lehetséges műveleteket, mert így kevesebb lesz a várakozás (több olyan eset lesz, amikor lehet két tranzakciónak zárja ugyanott, ha a megfelelő műveletek mehetnek együtt) és megalapozottabb lesz a döntés a sorosíthatóságról (nem leszünk annyira feleslegesen szigorúak).

Példa: Legyen három művelet: olvasás, írás és növelés (increment).
Ez utóbbi azt jelenti, hogy az adategység aktuális értékét megnöveljük eggyel.

Ekkor bevezethetünk három zárat: RLOCK, WLOCK és INCR, a kézenfekvő használattal (a megfelelő művelet csak akkor mehet, ha a tranzakció megkapta a hozzá tartozó zárat).

Kompatibilitási mátrix

Egy mátrix segítségével adjuk meg, hogy különböző tranzakcióknak milyen zárai lehetnek egyszerre egy adategységen.
Ez a **kompatibilitási mátrix**: a sorok és az oszlopok is a lehetséges záraknak felelnek meg és a Z_i sor Z_j oszlopában pontosan akkor van `I`, ha egy tranzakció megkaphatja egy adategységre a Z_j zárat akkor, ha egy másik tranzakció Z_i zárat tart fenn ezen az adategységen. Ha nem kaphatja meg, akkor `N` áll a Z_i sor Z_j oszlopában.

Akkor lehet két különböző tranzakciónak Z_i és Z_j zárja ugyanazon az adategységen, ha mindegy, hogy a két zárnak megfelelő műveletek milyen sorrendben hajtódnak végre.

Ez alapján az RLOCK/WLOCK modell és az RLOCK/WLOCK/INCR modell mátrixai:

	RLOCK	WLOCK
RLOCK	I	N
WLOCK	N	N

	RLOCK	WLOCK	INCR
RLOCK	I	N	N
WLOCK	N	N	N
INCR	N	N	I

A kompatibilitási mátrix használata

1. Ez alapján dönti el az ütemező, hogy egy ütemezés/zárkérés legális-e, illetve ez alapján várakoztatja a tranzakciókat. Minél több az `I` a mátrixban, annál kevesebb lesz a várakoztatás.
2. A mátrix alapján keletkező várakozásokhoz elkészített várakozási gráf segítségével az ütemező kezeli a holtpontot (ami tetszőleges zármodell esetén ugyanazt jelenti és a gráfot is ugyanúgy kell felépíteni).
3. A mátrix alapján készíti el az ütemező a sorosítási gráfot egy zárkérés-sorozathoz: a sorosítási gráf csúcsai a tranzakciók és akkor van él T_i-ből T_j-be, ha van olyan A adategység, amelyre az ütemezés során Z_k zárat kért és kapott T_i, ezt elengedte, majd ezután A-ra legközelebb T_j kért és kapott olyan Z_l zárat, hogy a mátrixban a Z_k sor Z_l oszlopában `N` áll.
   Vagyis olyankor lesz él, ha a két zár nem kompatibilis egymással, nem mindegy a két művelet sorrendje.

Sorosíthatóság

A sorosíthatóságról most is pusztán a zárkérések alapján fogunk dönteni, a sorosítási gráf segítségével:
Tétel. *Egy csak zárkéréseket és zárelengedéseket tartalmazó legális ütemezés pontosan akkor sorosítható valamelyik zármodellben, ha a zármodellhez tartozó kompatibilitási mátrix alapján az előbbi módszerrel felrajzolt sorosítási gráf DAG.*

Bizonyítás: Pontosan ugyanúgy megy, ahogyan eddig.

Az ütemező egyik lehetősége a sorosíthatóság elérésére, hogy folyamatosan figyeli a sorosítási gráfot és ha irányított kör keletezne, akkor ABORT-ot rendel el.

Sorosíthatóság II.

Másik lehetőség a protokollal való megelőzés. Tetszőleges zármodellben értelmes a 2PL és igaz az alábbi tétel:
Tétel. *Ha valamilyen zármodellben egy legális ütemezésben minden tranzakció követi a 2PL-t, akkor az ütemezéshez tartozó sorosítási gráf DAG, azaz az ütemezés sorosítható.*

Bizonyítás: Pontosan úgy, ahogy eddig.

Megjegyzés: Minél gazdagabb a zármodell, minél több az `I` a kompatibilitási mátrixban, annál valószínűbb, hogy a sorosítási gráf DAG lesz minden külön protokoll nélkül. Ez azt jelenti, ilyenkor egyre jobb lesz az ABORT-os módszer (ritkán kellhet).

Mit látunk mi ebből?

Az adatbáziskezelő működése során az ütemező munkájába nem (nagyon) szólhatunk bele.
Miért hasznos mégis tudni, hogy hogyan működik?

- Abba beleszólhatunk, hogy mennyire törekedjen sorosíthatóságra az adatbáziskezelő (akár azt is mondhatjuk, hogy semennyire). Ehhez nem árt, ha tudjuk, hogy mi is a sorosíthatóság, mit nyerünk vele és mibe kerül (bonyolult ütemező, lassabb futás).
- Ha ismerjük a különféle ütemezési technikákat: jobban fogjuk érteni az előforduló ABORT-okat, és majd az ABORT utáni visszaállítást is.

Összefüggések az adategységek között

Eddig nem néztük azt, hogy mik azok az adategységek, amikre a zárakat lehet kérni és kapni. Hallgatólagosan feltettük, hogy ezeket egymástól függetlenül lehet zárolni, nincs közöttük semmi szervezettség, semmi összefüggés.

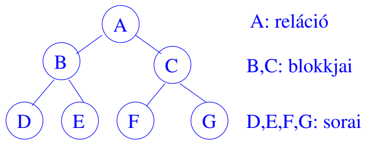
A valóságban két különböző esetben sem alkalmazható ez a megközelítés:

1. Ha az adategységek egymásba ágyazottak (pl. reláció, blokk, rekord), ekkor még további megkötéseket szeretnénk a zárolásra, az eddigi módszereket ki kell egészíteni.
2. Ha tudjuk, hogy egymáshoz képest hogyan helyezkednek el az adategységek a tárolási struktúrában, akkor jobb módszereket találhatunk, mint az eddigiek, illetve láthatjuk, hogy valamilyen eddig tanult módszer biztosan előnytelen lesz. Ez lesz például a B-fában tárolt adatok esete.

Adatbáziselemekből álló hierarchiák

A valóságban zárolhatunk teljes relációkat, de zárolhatjuk külön-külön ezek egyes blokkjait, sőt sorait is.
Minél nagyobb egységre rakunk zárat, annál könnyebb lesz a záradminisztráció, de annál több lesz a zárfeloldásra várás is és ezzel együtt a holtpont.
Alkalmazástól függ, hogy mi éri meg jobban, de egy valami mindig közös: Elvárjuk azt, hogy ha az A adategység egy reláció, B pedig ennek egy blokkja, akkor az A-ra rakott zár zárolja B-t is, azaz pl. az egyszerű tranzakciómodellben ne lehessen $l_j(B)$-t kapni, ha $l_i(A)$ után még nem volt $u_i(A)$. Ezt az eddigi technika még nem biztosítja, eddig ilyen összefüggéseket nem is vettünk figyelembe.

Egy ilyen lehetséges hierarchikus helyzet:



Figyelmeztető zármodell

Cél: olyan sorosítható ütemezés kikényszerítése, ami még az adategységek közötti hierarchiát is figyelembe veszi. Ez a hierarchia egy fával adott.

Egyszerű változat esetén háromféle zárművelet lesz: (lényegében az egyszerű tranzakciómodellnek felel meg):

- $LOCK_i(A)$: T_i zárolja A-t (explicit lock) és minden leszármazottját (implicit lock), kizárólagosan, azaz ezek után más tranzakció se A-ra, se ennek leszármazottjára nem kaphat zárat.
- $WARN_i(A)$: T_i figyelmeztetést rak A-ra (gyerekeire nem), ez annak jelzésére szolgál, hogy T_i majd zárat akar kapni A valamely leszármazottjára
- $UNLOCK_i(A)$: felszabadítja az A-ra rakott $LOCK$-ot és $WARN$-t, az implicit zár is lekerül A leszármazottjairól

A zárak használata

1. Az i-edik tranzakció, T_i, csak akkor olvashatja vagy írhatja az A adategységet, ha előtte zárat kért és kapott rá ($LOCK_i(A)$ vagy $LOCK_i$ A valamelyik ősén) és ezt a zárat még azóta nem engedte fel.
2. $LOCK_i(A)$ és $WARN_i(A)$ után mindig van $UNLOCK_i(A)$.
3. Ha $LOCK_i$ van A-n, akkor se $WARN_j$ se $LOCK_j$ nem kerülhet már rá (ha $j \neq i$), de két különböző tranzakciónak lehet $WARN$-ja ugyanott. Vagyis a kompatibilitási mátrix:

	LOCK	WARN
LOCK	N	N
WARN	N	I

A figyelmeztető protokoll

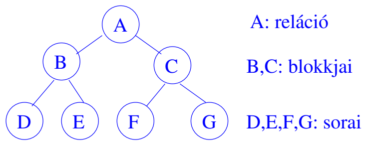
A T_i tranzakció követi a figyelmeztető protokollt, ha zárkérései a fentieken kívül még a következőket is tudják:

(a) T_i első zárkérése $WARN_i$ vagy $LOCK_i$ a gyökérre
(b) Ezután $LOCK_i$ vagy $WARN_i$ csak akkor kérhető egy adategységre, ha $WARN_i$ már van az apján.
(c) $UNLOCK_i$ csak akkor kérhető egy adategységre, ha már nincs sem explicit $LOCK_i$, sem $WARN_i$ az adategység leszármazottjain
(d) Kétfázisú zárkérés van: $UNLOCK_i$ után nincs se $LOCK_i$, se $WARN_i$.

Az (a) és (b) pontok miatt a zárkérések felülről lefelé kúsznak a fában, a zárelengedések pedig a (c) miatt alulról felfele mennek az egyes tranzakciók esetén.

Példa

Az adategységek hierarchiája legyen (mint előbb):



Az alábbi zárkérésekből és zárelengedésekből álló sorozat legális és mindhárom tranzakció követi a figyelmeztető protokollt.

$WARN_1(A)$, $WARN_2(A)$, $WARN_3(A)$, $WARN_1(B)$, $LOCK_2(C)$, $LOCK_1(D)$, $UNLOCK_2(C)$, $UNLOCK_1(D)$, $UNLOCK_2(A)$, $UNLOCK_1(B)$, $LOCK_3(B)$, $WARN_3(C)$, $LOCK_3(F)$, $UNLOCK_1(A)$, $UNLOCK_3(B)$, $UNLOCK_3(F)$, $UNLOCK_3(C)$, $UNLOCK_3(A)$

Azért legális, mert minden LOCK és WARN fel van engedve később és egyszerre csak több WARN van ugyanott, más nem.

A tranzakciók zárkérései pedig egyrészt 2PL szerint mennek, másrészt a zárkérések a fában felülről lefele mennek, a zárelengedések pedig alulról felfele.

Figyelmeztető protokoll II.

A figyelmeztető protokollra igaz az alábbi tétel:
Tétel. *Ha a figyelmeztető zármodellben, egy legális ütemezésben minden tranzakció követi a figyelmeztető protokollt, akkor az ütemezés sorosítható és soha nem lesz egyszerre két különböző tranzakciónak zárja (se explicit, se implicit) ugyanazon az adategységen.*

Bizonyítás: A sorosíthatóságot a kétfázisúság biztosítja (pontosabban nem bizonyítjuk).

Zárkonfliktus pedig azért nem lesz, mert

- Két különböző tranzakciónak explicit LOCK-ja nem lehet soha ugyanott, ha az ütemezés legális.
- Egy explicit és egy implicit LOCK (két különböző tranzakciótól) nem lehet ugyanott: nem lehet, hogy egy A adatelemen $LOCK_i$ van és ezzel egyidejűleg egy leszármazottján (akin így implicit T_i lock van) van $LOCK_j$ is, mert ekkor A-n $WARN_j$-nek is kell lennie, de az nem lehet egy legális ütemezésben.
- Két különböző tranzakciónak implicit LOCK-ja sem lehet ugyanazon a C adategységen, mert ekkor C két különböző felmenőjén a két különböző tranzakció két LOCK-jának kellene lennie, ami az előző pont értelmében nem lehet.

Megjegyzés: Lehetne bonyolultabb zármód esetén is nézni figyelmeztető zárolást és figyelmeztető protokollt (az RLOCK/WLOCK modelnek megfelelően): lenne külön írási és olvasási figyelmeztetés is.

B-fában tárolt adategységek

Tekintsük most azt a helyzetet, amikor a zárolható adategységek egy fa csúcsaiban helyezkednek el, de a fa most nem az adategységek egymásba ágyazottságát mutatja (az adategységek most diszjunktak), hanem azt, hogy hogyan lehet elérni az adatokat.
Például a B-fa esetén a levelekhez csak úgy juthatunk el, ha a gyökértől indulva végigjárunk egy lefele vezető utat. Ahhoz, hogy beolvashassuk azt a levelet, ami nekünk kell, előtte be kell olvasnunk az összes felmenőjét (és ha csúcsvágás vagy csúcsösszevonás úgy kívánja, írnunk is kell őket).

Ilyenkor a szokásos technikák mennek ugyan, de nagyon előnytelenek lehetnek. Például a 2PL esetén egész addig kell tartani a zárat a gyökéren, amíg le nem értünk a levélhez, ami indokolatlanul sok várakozáshoz vezet.

Kéne másik módszer, ami ebben a speciális esetben biztosítja a sorosíthatóságot, de nem olyan szigorú, mint a 2PL. Ez lesz a faprotokoll.

Faprotokoll

Egyszerű tranzakciómodellben vagyunk (de ezt is lehetne RLOCK/WLOCK modellre kibővíteni), azaz

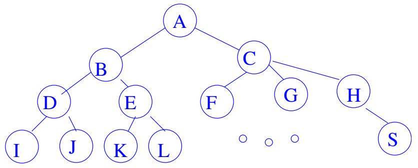
- egy zár van csak, ezt meg kell kapni íráshoz és olvasáshoz is
- zár után mindig van UNLOCK
- nincs két különböző tranzakciónak zárja ugyanott

A T_i tranzakció követi a faprotokollt, ha

1. Az első zárat bárhova elhelyezheti.
2. A későbbiekben azonban csak akkor kaphat zárat A-n, ha ekkor zárja van A apján.
3. Zárat bármikor fel lehet oldani (nem 2PL).
4. Nem lehet újrazárolni, azaz ha T_i elengedte egy A adategység zárját, akkor később nem kérhet rá újra (még akkor sem, ha A apján még megvan a zárja).

Tétel. (*Bizonyítás nélkül*) *Ha minden tranzakció követi a faprotokollt egy legális ütemezésben, akkor az ütemezés sorosítható lesz, noha nem feltétlenül lesz 2PL.*

Példa



Tekintsük ezt a B_3-fát. Az A-tól H-ig levő adategységek a fa belső csúcsai, itt mutatók és keresést segítő kulcsok vannak, a levelekben (I-től S-ig) pedig a keresési kulcs szerint rendezetten vannak a tárolt adatok, amik között keresünk. Most feltesszük, hogy egy levélben egy tárolt elem van.

Ha mondjuk az I-ben, J-ben és K-ban tárolt elemek keresési kulcsa $1, 3$ és 10, és be akarunk szúrni egy olyan elemet, ahol a kulcs értéke 4, akkor először olvasni kell A-t, B-t és D-t, majd írni is kell D-t.

Ekkor a megfelelő (faprotokoll szerinti, legális) ütemezés eleje
$LOCK_i(A)$, $LOCK_i(B)$, $UNLOCK_i(A)$ mert B beolvasása után látjuk, hogy neki csak két gyereke van, ha kell is csúcsvágás, az A-t biztos nem érinti, A-t nem kell majd írni. Csak addig kellett fogni A-n a zárat, amíg B-re is megkaptuk.

Ezután $LOCK_i(D)$, $UNLOCK_i(B)$, mert látjuk, hogy D-nek csak két gyereke van, ezért B-t biztos nem kell írni.

Innen tovább: $UNLOCK_i(D)$, amikor már megtörtént az új levél beszúrása és D-ben is beállítottuk a mutatókat.

Példa II.

Tanulság:

- Faprotokoll szerint ment az ütemezés $\Longrightarrow$ jó lesz
- Nem 2PL és ezzel nyertünk is sokat, mert amint megvolt $UNLOCK_i(A)$, akkor rögtön indulhat a következő beszúrás, ha az a fa jobb oldali ágán fut le. Ha 2PL lett volna, akkor $LOCK_i(D)$-ig kellene várni ezzel.