Vektoros grafikát tároló adatbázisok

Katona Endre „Térképi adatbázisok”
diasorozata alapján

Vektoros adatábrázolás

Kép = rajzelemek sorozata, koordinátageometriai leírással.
CAD rendszerekre jellemző (pl. AutoCAD).

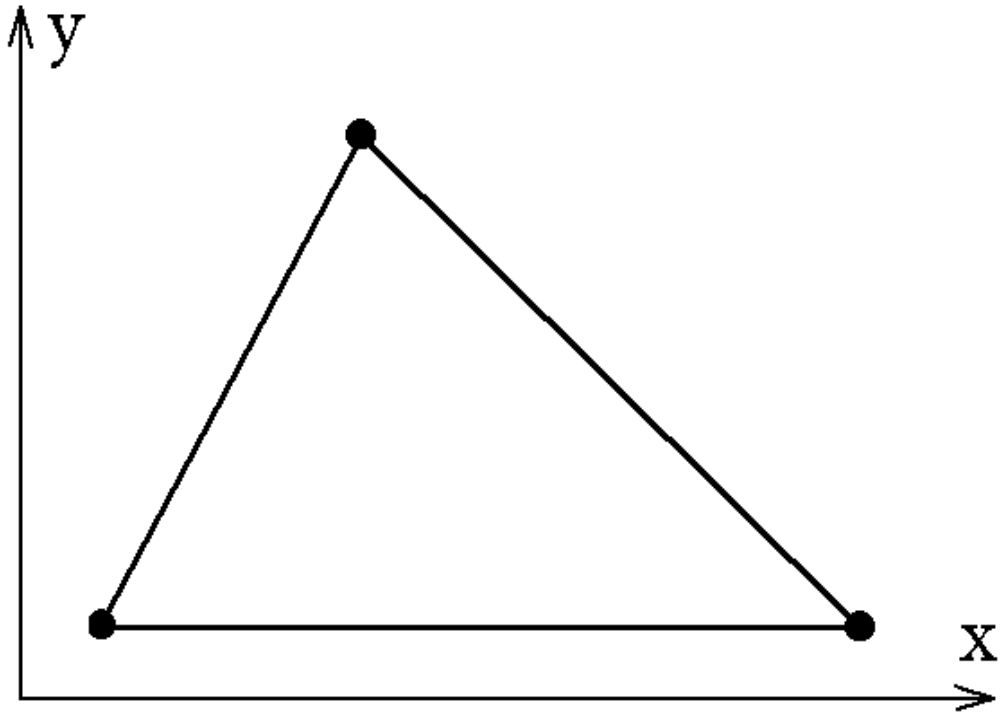
Vektor: irányított egyenesszakasz: LINE x_1,y_1, x_2,y_2

Jellemzők:

- A kép struktúráját kódolja.
- Monitoron való megjelenítéshez raszteresre kell konvertálni (számítógépes grafikai algoritmusok).

Grafikus formátumok: DXF, IGES, DWG (AutoCAD), ArcView shapefile

Példa vektoros rajzra



LINE (1,1), (7,12); LINE (1,1), (16,1); LINE (7,12), (16,1)

POLYGON (1,1), (7,12), (16,1)

Rajzelem típusok és megadási módjuk

Pontszerű (0D) objektumok:

- ***Pont:*** POINT x, y
- ***Felirat:*** TEXT x, y, méret, irány, szöveg

Vonalas (1D) objektumok:

- ***Egyenesszakasz:*** LINE x_1, y_1, x_2, y_2
- ***Vonallánc (töröttvonal, line string, polyline, „ív”)***:
  POLYLINE x_1, y_1,..., x_n, y_n

Területi (2D) objektumok:

- ***Alakzat (zárt poligon)***:
  POLYGON x_1, y_1,..., x_n, y_n ahol $x_{n+1} = x_1$, $y_{n+1} = y_1$
- ***Kör:*** CIRCLE x, y, r

Vektoros adatmodellek

Milyen módon kezeljük a térbeli objektumok kapcsolatrendszerét?
(pl. "kapcsolódik", "szomszédos", "tartalmazza", stb.)

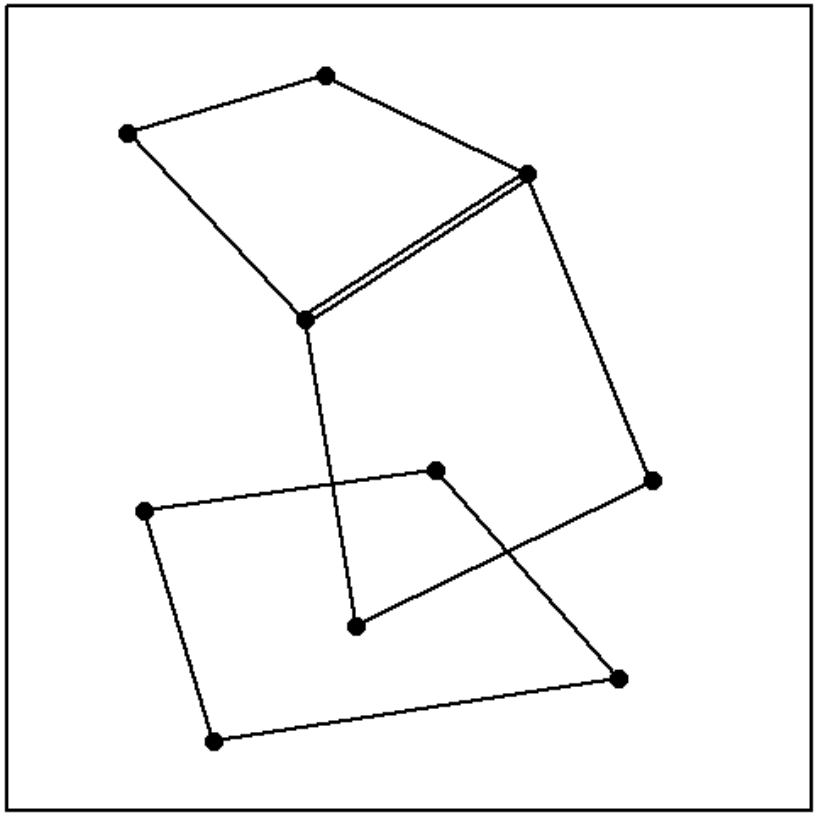
Két megközelítés:

Spagetti modell: nem kezeljük a kapcsolatokat.

Topológikus modellek: térbeli kapcsolatok tárolása.

Spagetti modell

- Térbeli objektumok egyszerű halmaza, például
  LINE (11, 5), (23, 45)
  LINE (4, 13), (11, 5)
  CIRCLE (11, 14), 20
  ...
- Objektumok között nincs hivatkozási kapcsolat.
- Könnyen kezelhető.
- Adatintegritás ellenőrzése nehéz.



Példa: telekosztás

Topológikus modellek

Topológia: térbeli kapcsolódási struktúra.

Rajzelem azonosító (*identifier*, *id*).

Az egyes rajzelemek egymásra hivatkoznak.

Példák:

- tartománytérkép
- hálózat

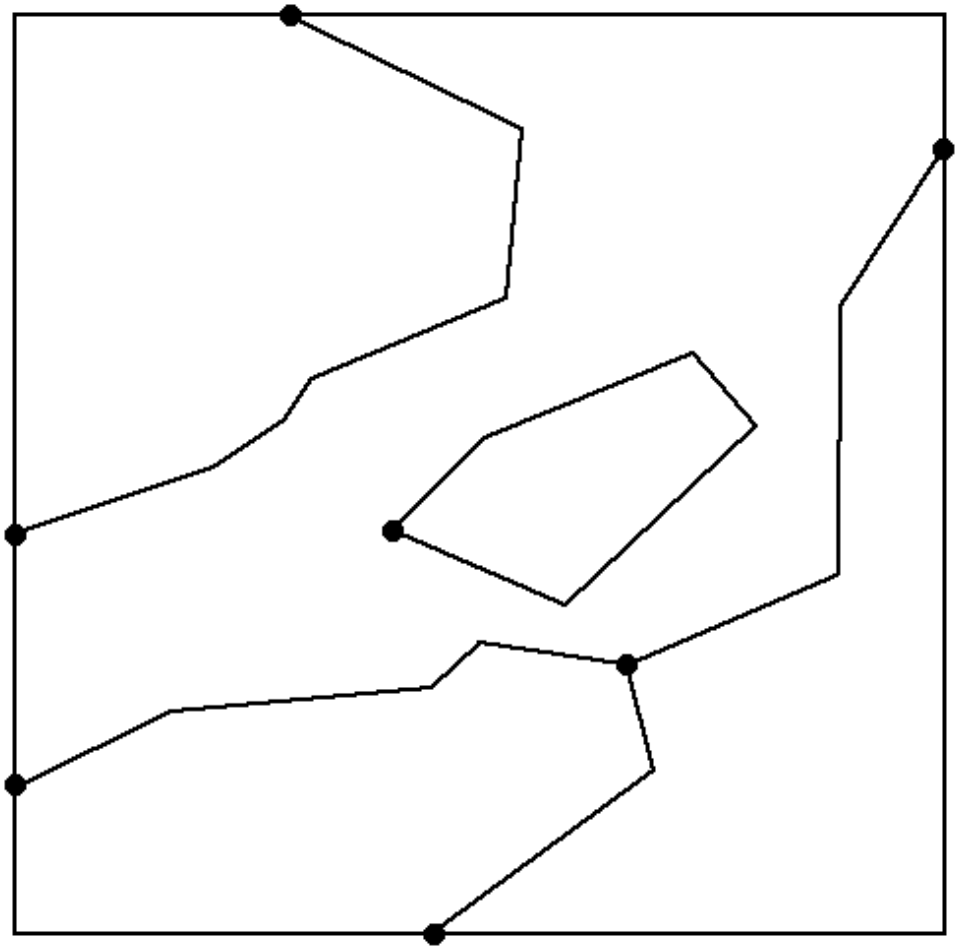
Tartománytérkép

Példák:
- talajtérkép
- megyetérkép

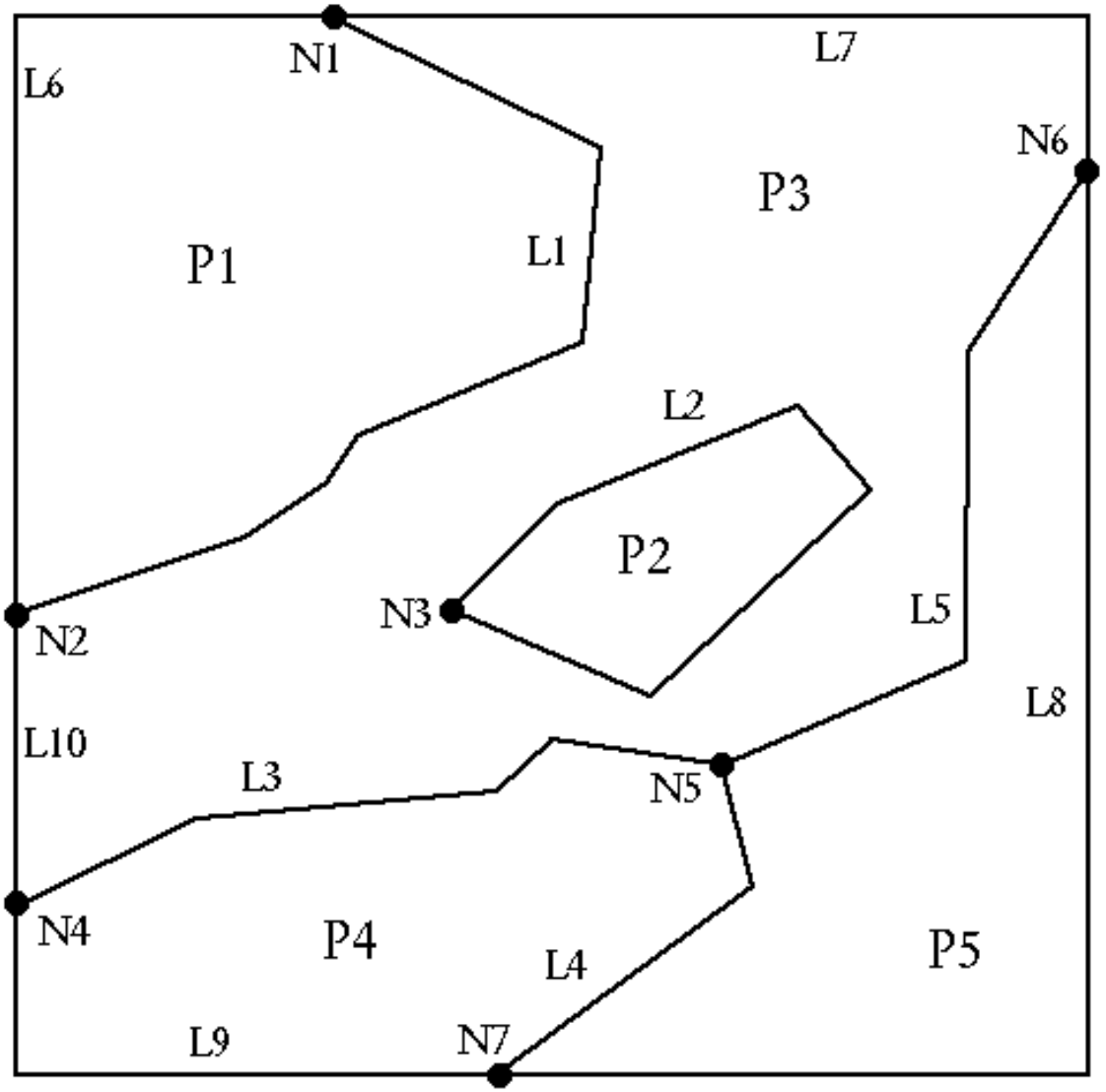
0D: *csomópontok*

1D: *vonalak* (nem metszik egymást)

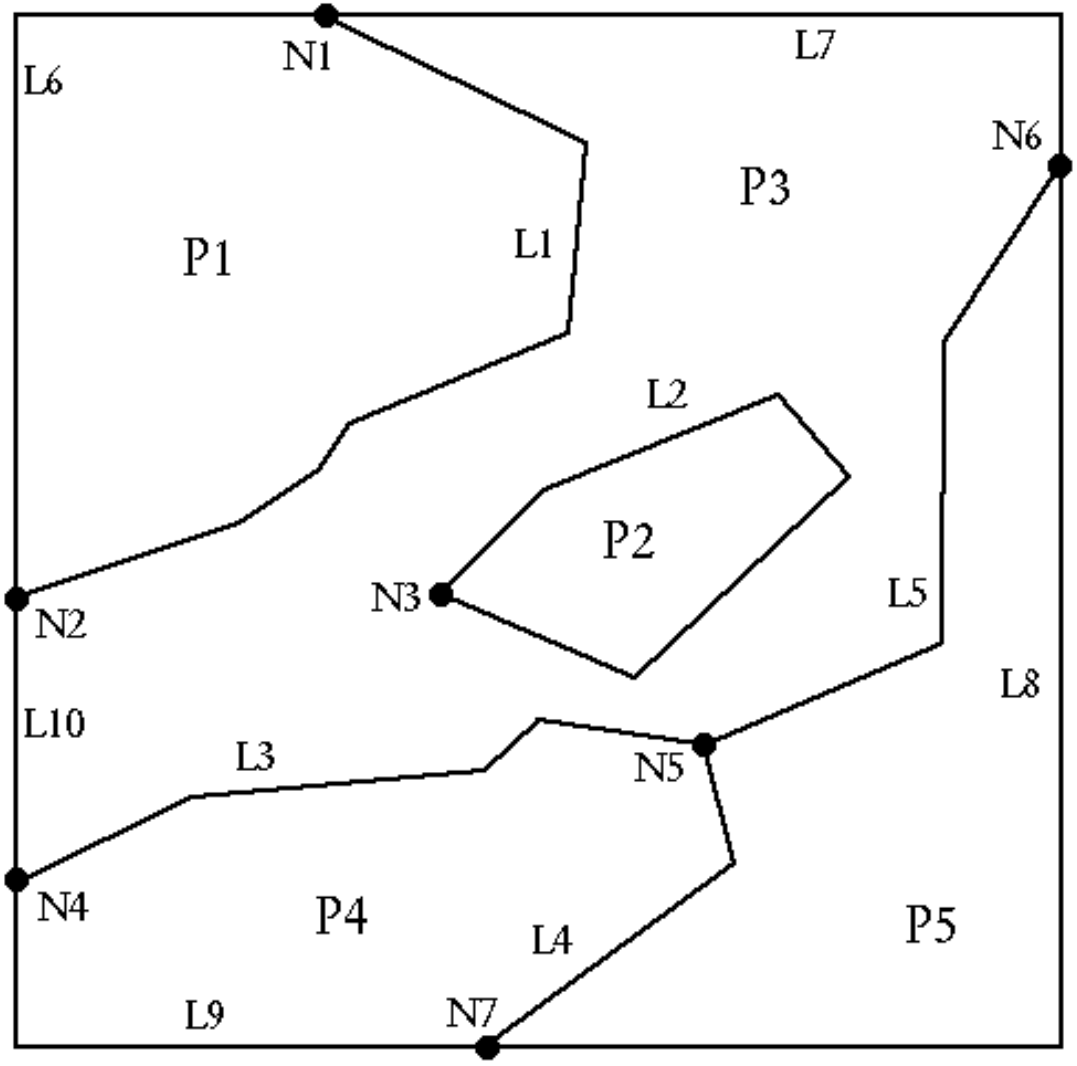
2D: *tartományok* (diszjunktak, de szigetek lehetnek)



Tartománytérkép adatstruktúra

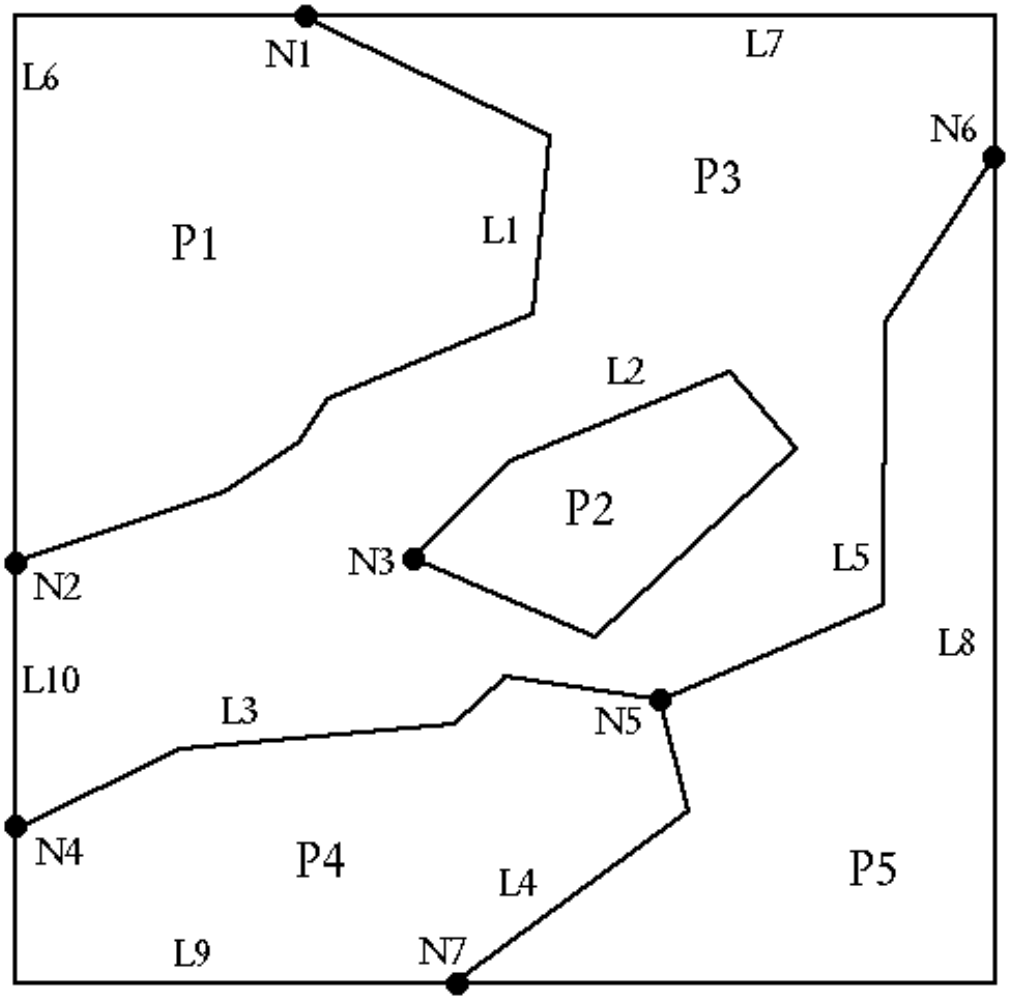


NODE:	*id*	*x*	*y*
	N1	x1	y1
	N2	x2	y2
	N3	x3	y3
	N4	x4	y4
	N5	x5	y5
	N6	x6	y6
	N7	x7	y7



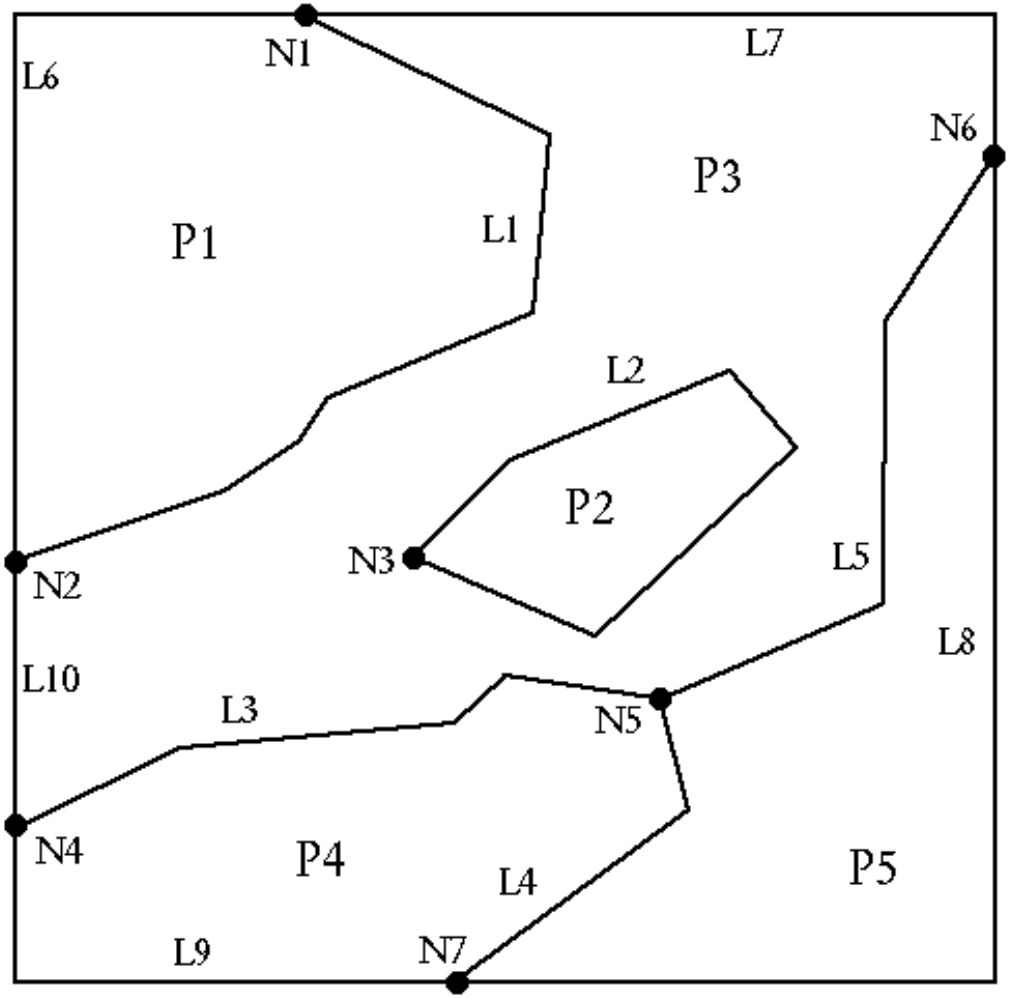
LINE:

id	*node*$_1$	*node*$_2$	*lpoly*	*rpoly*	x_1,y_1, …, x_n,y_n	
L1	N1	N2	P3	P1	...	...
L2	N3	N3	P3	P2	...	...
L3	N4	N5	P3	P4		
L4	N5	N7	P5	P4		
L5	N5	N6	P3	P5		
L6	N1	N2	P1	P0		
L7	N1	N6	P0	P3		
L8	N6	N7	P0	P5		
L9	N4	N7	P4	P0		
L10	N2	N4	P3	P0		



POLYGON: *id* *line$_1$, ..., line$_n$*

P1	L1, L6
P2	L2
P3	L1, L7, L5, L3, L10, L2
P4	L3, L4, L9
P5	L4, L5, L8



Tartománytérkép adatstruktúra összefoglalva

NODE: node_id, x, y

LINE: line_id, $node_1$, $node_2$, lpoly, rpoly, x_1, y_1, ..., x_n, y_n

POLYGON: poly_id, $line_1$, ..., $line_n$

Hálózat

Jellemzői:

0D és 1D objektumok rendszere.

A vonalláncok csomópont nélkül is keresztezhetik egymást (pl. felüljáró).

Alkalmazási példák:

úthálózat, vasúthálózat

csővezeték hálózat (víz, gáz, csatorna)

kábelhálózat

folyóhálózat

Geometriai hálózat

NODE:

id : csomópont azonosító

x, y : koordináták

$e_1,...,e_n$: kiinduló élek azonosítói

[*attribútumok,* pl. van-e közlekedési lámpa]

LINE:

id : vonallánc azonosító

$node_1$: kezdő csomópont azonosítója

$node_2$: záró csomópont azonosítója

$x_1, y_1, ..., x_n, y_n$: töréspontok koordinátái

[*attribútumok,* pl. forgalom iránya]

Logikai hálózat (gráf)

NODE:

id : csomópont azonosító

$e_1,...,e_n$: kiinduló élek azonosítói

[*attribútumok*]

EDGE:

id : él azonosító

$node_1$: kezdő csomópont azonosítója

$node_2$: záró csomópont azonosítója

[*attribútumok,* pl. hossz]

Félig geometriai hálózat

NODE:

id : csomópont azonosító

x, y : koordináták

$e_1,...,e_n$: kiinduló élek azonosítói

[*attribútumok,* pl. aluljáró, közlekedési lámpa]

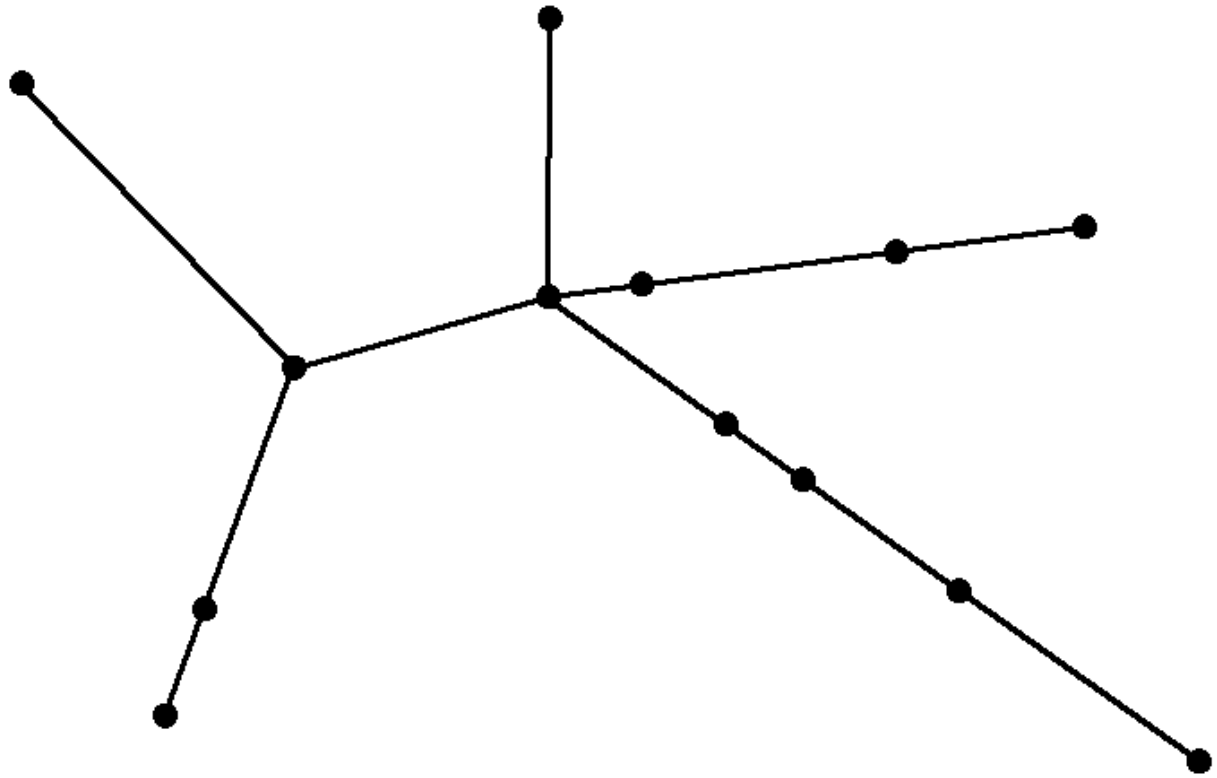
EDGE:

id : él azonosító

$node_1$: kezdő csomópont azonosítója

$node_2$: záró csomópont azonosítója

[*attribútumok,* pl. él tényleges hossza]



Lineáris címzés módszere:

Objektum (obj_id, megnev, edge_id, dist1, dist2)

Példa "objektum" táblára

obj_id	*megnev*	*edge_id*	*dist1*	*dist2*
O1	híd	E3	145	147
O2	útátjáró	E3	192	192
O3	őrház	E4	48	48
O4	vágányjavítás	E5	216	231

Adatbázismodellek

Kérdés: vektoros adatstruktúrákat hogyan tudunk kezelni adatbázisban?

Szétválasztott modell:

- Adatbázis linkek (MicroStation).

Integrált modell:

- Tisztán relációs megközelítés.
- Objektum-relációs modell.
- Térbeli adattípusok.

Szétválasztott modell

A reprezentálandó adatok szétválasztása:

Grafikus adatok: térképen ábrázolandók,

Leíró adatok: táblázatokban tárolhatók.

A kezelő szoftver:

Grafikus rendszer (Graphics System = GS).

Adatbázis-kezelő rendszer (Database Management System = DBMS).

A kapcsolatteremtésről a GS gondoskodik.

MicroStation módszer: adatbázis-linkek

A DBMS oldaláról: táblák bővítése LINK mezővel.

A GS oldaláról: rajzelemekhez adatbázis link(ek).

Rajz file (GS)	***Adattáblák (DBMS)***	
rajzelem	$R(A_1,...,A_n,LINK)$	$S(B_1,...,B_m,LINK)$
link=(R,3)	a_{11} ... a_{1n} 1	b_{11} ... b_{1m} 1
rajzelem	a_{21} ... a_{2n} 2	b_{21} ... b_{2m} 2
rajzelem	a_{31} ... a_{3n} 3	b_{31} ... b_{3m} 3
link=(R,1)	...	...
link=(S,2)		
rajzelem		
link=(S,2)		

A szétválasztott modell értékelése

Előnyök:

Nagy múltú szoftverek (GS, DBMS).

Gyakran a grafikus és leíró adatok külön keletkeznek.

Bizonyos rajzi információk könnyebben kezelhetők
(pl. vízrajz, domborzat).

Hátrányok:

Térbeli adatok esetén elvész az adatbázis-funkcionalitás.

A grafikus és leíró adatokat elkülönülten kezeli.

Integrált modell: Tisztán relációs megközelítés

- Grafikus és leíró adatok egyaránt relációs adatbázisban.
- Topológikus jellegű hivatkozások külső kulcsokkal.
- Változó hosszúságú listák kezelése problematikus.

Félig geometriai hálózat: relációs megvalósítás

Node (id, x, y, *edges*)

Edge (id, *node1*, *node2*)

```
CREATE TABLE Node
(id INTEGER PRIMARY KEY,
 x REAL, y REAL,
 edges CLOB);

CREATE TABLE Edge
(id INTEGER PRIMARY KEY,
 node1 INTEGER REFERENCES Node(id),
 node2 INTEGER REFERENCES Node(id));
```

Az adatbázis feltöltése:

```
INSERT INTO Node VALUES (1, 123,456, '11 21');
INSERT INTO Node VALUES (2, 234,567, '11');
INSERT INTO Node VALUES (3, 345,678, '21');
INSERT INTO Edge VALUES (11, 1, 2);
INSERT INTO Edge VALUES (21, 1, 3);
```

Hátrány: a CLOB-beli hivatkozások SQL eszközökkel nem elérhetők.

Tartománytérkép – spagetti modell

Példa: Telek (helyrajziszám, terület, poligon)

```
CREATE TABLE Telek
(helyrajziszám INTEGER PRIMARY KEY,
 terület REAL,
 poligon CLOB);

INSERT INTO Telek VALUES (123, 105.6,
   '11 21, 12 22, 13 23');
```

Hátrány: a CLOB-beli hivatkozások SQL eszközökkel nem elérhetők, például SQL SELECT utasítással nem tudjuk lekérni egy poligon szögpont-koordinátáit.

Ok: a séma nincs 1. normálformában → normalizálás

Tartománytérkép
spagetti modell normalizálva

Példa: Telek (helyrajziszám, terület, poligonID)
PolKoord (*poligonID*, sorszám, x, y)

```
CREATE TABLE Telek
(helyrajziszám INTEGER,
 terület REAL,
 poligonID INTEGER PRIMARY KEY);

CREATE TABLE PolKoord
(poligonID INTEGER REFERENCES Telek(poligonID),
 sorszám INTEGER,
 x REAL, y REAL,
 PRIMARY KEY(poligonID, sorszám));
```

A 987-es helyrajzi számú telek szögpont koordinátáinak lekérése:

```
SELECT x,y FROM Telek,PolKoord
  WHERE helyrajziszám=987 AND
    Telek.poligonID=PolKoord.poligonID
  ORDER BY sorszám;
```

Tartománytérkép – topológikus modell

Node (id, x, y)
Line (id, *node1, node2, lpoly, rpoly*, breakpoints)
Polygon (id, *lines*)

```
CREATE TABLE Node
 (id INTEGER PRIMARY KEY, x REAL, y REAL);
CREATE TABLE Polygon
 (id INTEGER PRIMARY KEY, lines CLOB);
CREATE TABLE Line
 (id INTEGER PRIMARY KEY,
  node1 INTEGER REFERENCES Node(id),
  node2 INTEGER REFERENCES Node(id),
  lpoly INTEGER REFERENCES Polygon(id),
  rpoly INTEGER REFERENCES Polygon(id),
  breakpoints CLOB);
```

Hátrány: egyszerű SQL SELECT-tel nem tudjuk lekérni egy vonallánc vagy poligon szögpont-koordinátáit.

Ok: a CLOB-beli hivatkozások → normalizálás

A tisztán relációs megközelítés értékelése

Előnyök: a grafikus és leíró adatokat egy közös adatbázisban tároljuk:

- azok nem szakadhatnak el egymástól,
- a teljes adatbázis-funkcionalitás érvényesül valamennyi adatra.

Hátrányok: a térbeli adatok kezelése nehézkes:

- lekérdezéskor ismernünk kell a térbeli adatok pontos tárolási struktúráját,
- az adatkezelés hatékonyabbá tételéhez több-kevesebb redundancia bevitelére lehet szükség.

Térbeli adatok objektum-relációs modellben

Változó hosszúságú listák:

- dinamikus tömb (VARRAY)
- beágyazott tábla

Félig geometriai hálózat

Node (id, x, y, *edges*)

Edge (id, *node1*, *node2*)

```
CREATE TYPE EdgesTípus AS VARRAY(100)
 OF INTEGER;

CREATE TABLE Node
(id INTEGER PRIMARY KEY,
 x REAL, y REAL,
 edges EdgesTípus);

CREATE TABLE Edge
(id INTEGER PRIMARY KEY,
 node1 INTEGER REFERENCES Node(id),
 node2 INTEGER REFERENCES Node(id));
```

Speciális példa: utcahálózat

Utca (utcaId, név)

Utcaszakasz (szakaszId, *utcaId*, sorszám, vonallánc, attr)

- Egy utca több szakaszból állhat.
- Az egyes szakaszokhoz eltérő "attr" attribútumok tartozhatnak (például forgalmi rend).
- A szakaszok sorrendjét a "sorszám" attribútum határozza meg.

Megoldás VARRAY segítségével

```
CREATE TABLE Utca
( utcaId INTEGER PRIMARY KEY,
  név VARCHAR(30)
);
CREATE TYPE Vonaltip AS VARRAY(100) OF REAL;
CREATE TABLE Utcaszakasz
( szakaszId INTEGER PRIMARY KEY,
  utcaId INTEGER REFERENCES Utca(utcaId),
  sorszám INTEGER,
  vonallánc Vonaltip,
  attr VARCHAR2(20)
);
```

A "Virág" utca teljes hosszán a vonallánc koordináták lekérdezése:

```
SELECT vonallánc FROM Utca, Utcaszakasz
  WHERE Utca.utcaId=Utcaszakasz.utcaId
  AND név='Virág'
  ORDER BY sorszám;
```

Megoldás beágyazott táblával

```
CREATE TABLE Utca
( utcaId INTEGER PRIMARY KEY,
  név VARCHAR(30)
);
CREATE TYPE KoordTipus AS OBJECT
  (num NUMBER, x REAL, y REAL);
CREATE TYPE KoordTabla AS TABLE OF KoordTipus;
CREATE TABLE Utcaszakasz
( szakaszId INTEGER PRIMARY KEY,
  utcaId INTEGER REFERENCES Utca(utcaId),
  sorszám INTEGER,
  vonallánc KoordTabla
) NESTED TABLE vonallánc STORE AS KoordTab;
```

A "Virág" utca teljes hosszán a vonallánc koordináták lekérdezése:

```
SELECT vonallánc FROM Utca, Utcaszakasz
  WHERE Utca.utcaId=Utcaszakasz.utcaId AND
  név='Virág'
  ORDER BY sorszám;
```

Tartománytérkép – spagetti modell

Példa: Telek (helyrajziszám, terület, Poligon(num,x,y))
Megoldás: beágyazott tábla.

```
CREATE TYPE KoordTipus AS OBJECT
   (num NUMBER, x REAL, y REAL);

CREATE TYPE KoordTabla
  AS TABLE OF KoordTipus;

CREATE TABLE Telek
(helyrajziszám INTEGER PRIMARY KEY,
 terület REAL,
 poligon KoordTabla)
 NESTED TABLE poligon STORE AS KoordTab;
```

Az adatbázis feltöltése:

```
INSERT INTO Telek VALUES
(987, 101.6, KoordTabla(
  KoordTipus(1, 11,21),
  KoordTipus(2, 32,42),
  KoordTipus(3, 53,63))
);
```

Egy telek koordinátáinak lekérése:

```
SELECT x,y FROM TABLE( SELECT poligon
  FROM Telek WHERE helyrajziszám=987 )
  ORDER BY num;
```

Tartománytérkép topológikus megvalósítása

Node (id, x, y)

Line (id, *node1, node2, lpoly, rpoly,* Breakpoints(num,x,y))

Polygon (id, *lines*)

```
CREATE TABLE Node
 (id INTEGER PRIMARY KEY, x REAL, y REAL);
CREATE TYPE LinesTipus
  AS VARRAY(100) OF INTEGER;
CREATE TABLE Polygon
 (id INTEGER PRIMARY KEY, lines LinesTipus);
```

```
CREATE TYPE KoordTipus AS OBJECT
   (num NUMBER, x REAL, y REAL);
CREATE TYPE KoordTabla
  AS TABLE OF KoordTipus;
CREATE TABLE Line
 (id INTEGER PRIMARY KEY,
  node1 INTEGER REFERENCES Node(id),
  node2 INTEGER REFERENCES Node(id),
  lpoly INTEGER REFERENCES Polygon(id),
  rpoly INTEGER REFERENCES Polygon(id),
  breakpoints KoordTabla)
 NESTED TABLE breakpoints STORE AS KoordTab;
```

A 987-es számú vonallánc szögpont koordinátáinak lekérése:

```
SELECT x,y FROM Node WHERE
  id=(SELECT node1 FROM Line WHERE id=987);
SELECT x,y FROM
  TABLE(SELECT breakpoints FROM Line
  WHERE id=987) ORDER BY num;
SELECT x,y FROM Node WHERE
  id=(SELECT node2 FROM Line WHERE id=987);
```

Az objektum-relációs megközelítés értékelése

Előnyök: Az objektum-relációs modell lényegesen jobb lehetőségeket biztosít a térbeli adatok kezelésére (VARRAY, beágyazott táblák).

Hátrányok: az adatstruktúrák használata még mindig kissé nehézkes, és a megoldások hatékonysága is kérdéses.

Megoldás: beépített térbeli adattípusok.

Integrált modell: Térbeli adattípusok

Szabványosítás: Open Geospatial Consortium: OGC modell

Térbeli adattípusokat támogató rendszerek:

- PostgreSQL
- PostGIS (az OGC modellt követi)
- MySQL (az OGC modellt követi)
- Oracle Spatial

Térbeli indexek

Cél: térbeli feltétel szerinti lekérdezések gyorsítása.

Példák: tekintsük a T1(a1, g1) és T2(a2, g2) relációs sémákat a g1 ill. g2 térbeli oszlopokkal.

Térbeli szelekció: adott P pont közelében lévő elemek:

```
SELECT * FROM T1
  WHERE distance(P,g1) < 10;
```

Térbeli összekapcsolás:

```
SELECT * FROM T1,T2 WHERE intersect(g1,g2);
```

Térbeli keresés hagyományos indexszel

Adott a **Pont(id, x, y)** tábla, az **(a, b)** pont környezetében keresünk:

```
CREATE INDEX xKoord ON Pont(x);
SELECT * FROM Pont
  WHERE x BETWEEN a-10 AND a+10;
```

Probléma: csak egy dimenzió szerint indexelünk.

```
CREATE INDEX xKoord ON Pont(x);
CREATE INDEX yKoord ON Pont(y);
SELECT * FROM Pont
  WHERE x BETWEEN a-10 AND a+10
    AND y BETWEEN b-10 AND b+10;
```

Probléma: vagy az egyik, vagy a másik index szerint gyorsít.

Kapcsolat a hagyományos és a térbeli indexelés között

Hagyományos index

(indexkulcs, rekord_id)

Indexkulcs szerint sorbarendezhető.

Adatstruktúrák:

B-fa

Hash (tördelőtáblázat)

Térbeli index

0D: ((x, y), objektum_id),

1D, 2D:

(befogl. téglalap, objektum_id)

Nem rendezhető sorba.

Adatstruktúrák:

Négyesfa, R-fa

négyzetrács-index

Térbeli indexelés elve

Adott térbeli feltételnek eleget tevő elemek kiválasztása két lépésben történik (ha pl. 100 000 elemből 50-et kell kiválasztani):

- ***Előszűrés*****:** térbeli indexszel (pl. 100 000-ből 100-at). Igen gyors.
- ***Kiválasztás:*** egyenkénti ellenőrzés (pl. 100-ból 50-et). Lassú, de csak kevés elemre kell végrehajtani.

Térbeli indexek csoportosítása

Kérdés: lehet-e kiegyensúlyozott térbeli indexet csinálni?

Térvezérelt (space-driven) indexek:

A tér felosztása az adatoktól többé-kevésbé független.
Például: ***grid, négyesfa.*** Nem kiegyensúlyozott.

Adatvezérelt (data-driven) indexek:

A tér felosztása az adatoktól függ.
Például: ***R-fa.*** Kiegyensúlyozott.

Négyzetrács index (grid index)

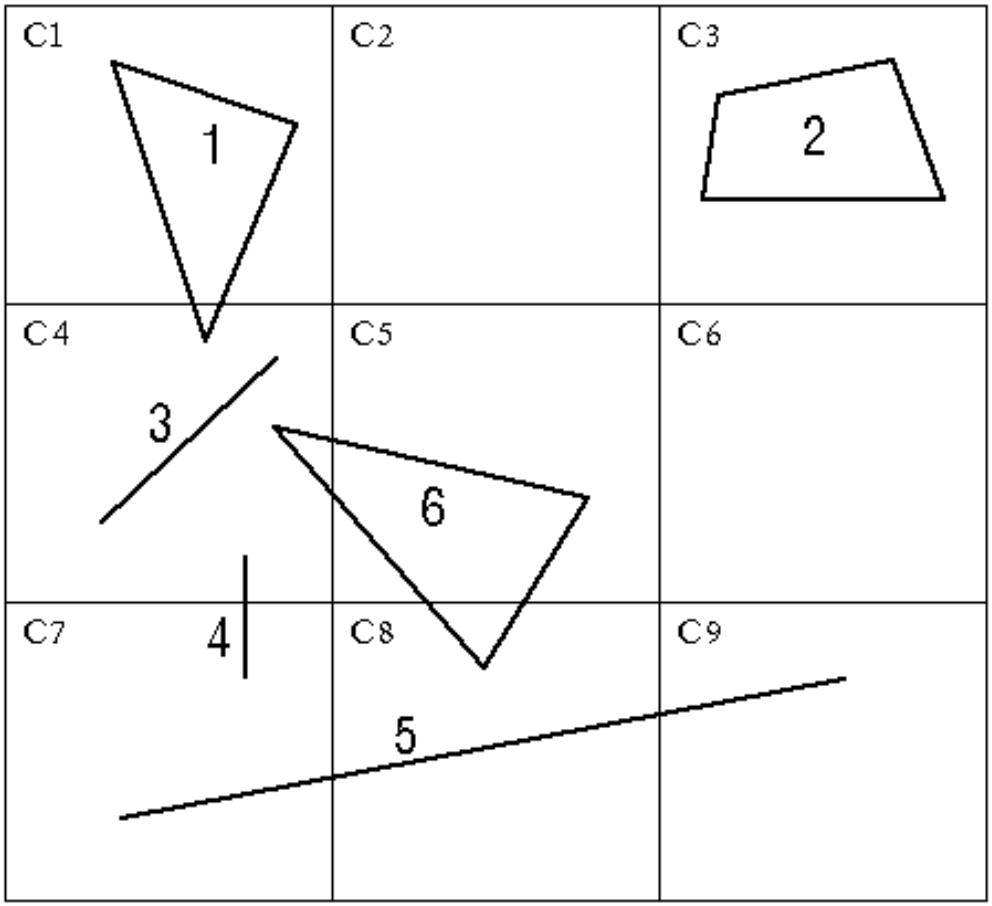
A teljes terület felosztása *n* x *m* négyzetre (téglalapra).

Minden négyzethez indexlista.

Ha a térbeli objektum (részben vagy egészben) beleesik egy négyzetbe, akkor id-je felkerül az indexlistára.

Egy id több indexlistán is szerepelhet (redundáns tárolás).

Nem kiegyensúlyozott.

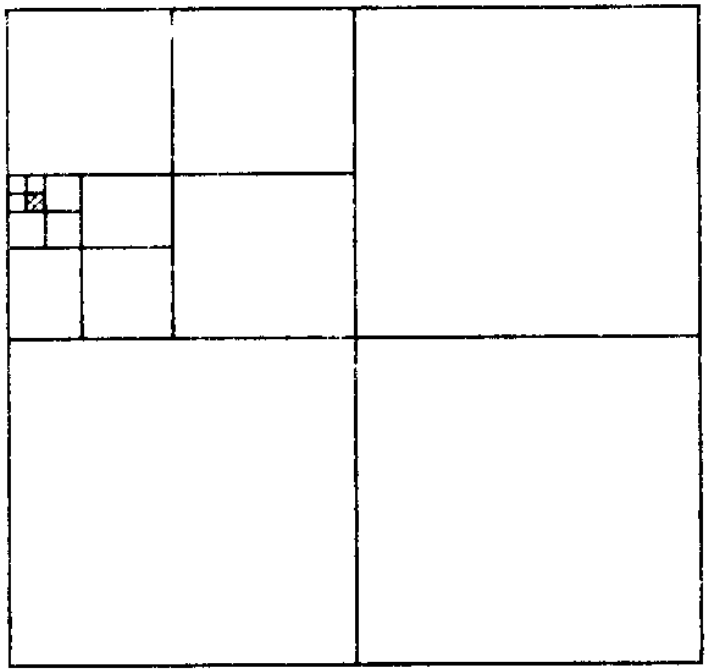


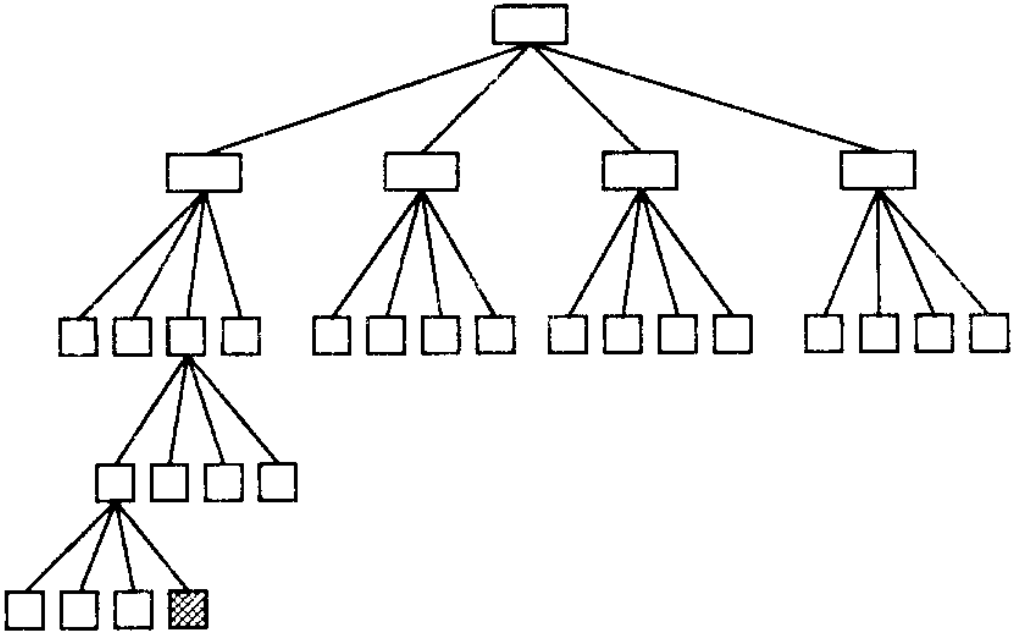
C1	***C2***	***C3***	***C4***	***C5***	***C6***	***C7***	***C8***	***C9***
R1		R2	R1	R6		R4	R5	R5
			R3			R5	R6	
			R4					
			R6					

Megjegyzések

- Amikor azt vizsgáljuk, hogy egy objektum beleesik-e egy grid négyzetbe, elegendő az objektum befoglaló téglalapját vizsgálni. Előfordulhat, hogy egy objektum olyan négyzet listájára is felkerül, amelybe valójában nem esik bele (például a 6. alakzatot fel kellene venni a C7 négyzet listájára is). Mivel az indexelés csak előszűrést végez, az utána következő ellenőrzésnél az ilyen objektumok kiesnek.
- A rajz módosításakor nem szükséges a listákról törölni a törölt/módosított objektumokra való hivatkozásokat, elegendő csak az új/módosított objektumokat felvenni. Az így bennmaradó hibás hivatkozások az előszűrés utáni ellenőrzésnél kiesnek.

Négyesfa index (quadtree)





Minden szögponthoz indexlista.

Megjegyzés: nem kiegyensúlyozott.

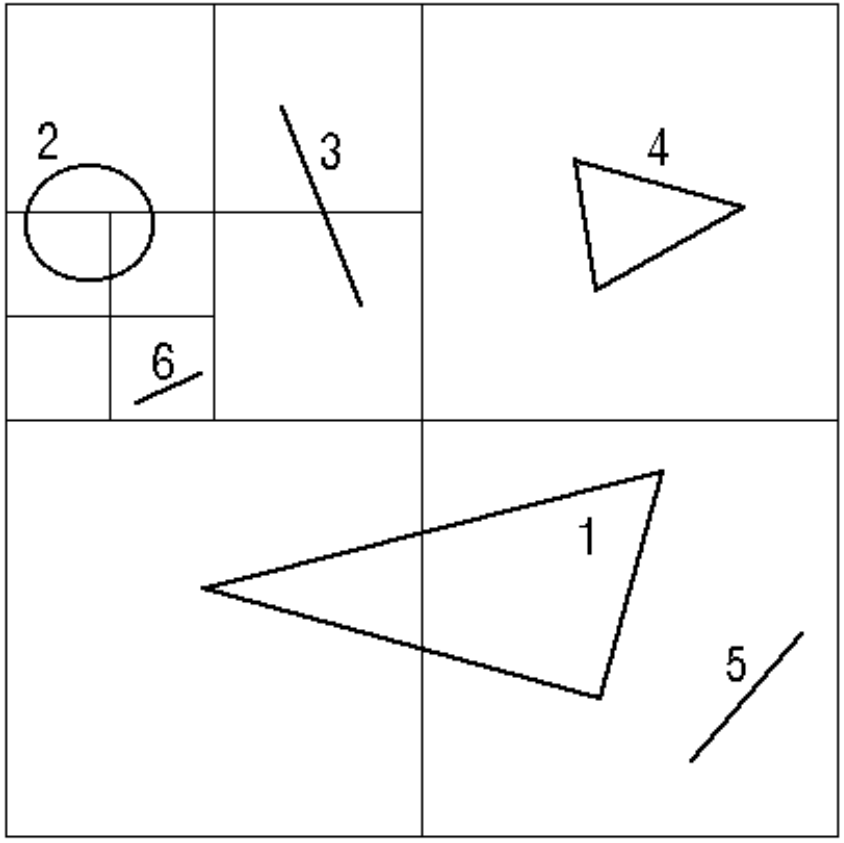
Négyesfa index - 2

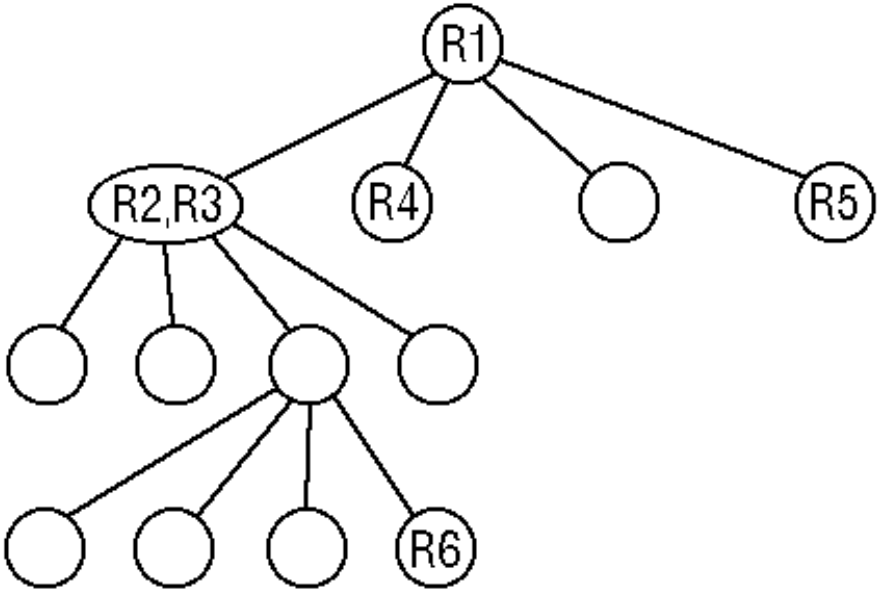
Több lehetséges felépítés (pl. pontszerű ill. területi objektumokra optimalizálva).

Itt bemutatott változat: egy objektum-id egy és csak egy indexlistára kerül: amely négyzetbe teljesen elfér, de egyik résznégyzetbe sem (redundanciamentes tárolás).

Nem kiegyensúlyozott.

Négyesfa index - 3





Adatstruktúra:

NODE (n1, n2, n3, n4, objektum-id-lista).

R-fa

- Kiegyensúlyozott fa.
- Az R-fa (régiófa) a B+ fa adaptációja több dimenzióra. Nem számokat, hanem téglalapokat rendez.
- N-dimenzióra is működik, de 2D-re tárgyaljuk.
- A fa minden szögpontjának egy lemezblokk felel meg. Ha *n* index-bejegyzést tartalmaz, akkor a szögpontnak *n* leszármazottja van. (*n* nagy lehet, pl. *n*=100).
- A fa mélysége általában nem több 3-4 szintnél.

Az R-fa definíciója

Keresőkulcs: egy téglalap.

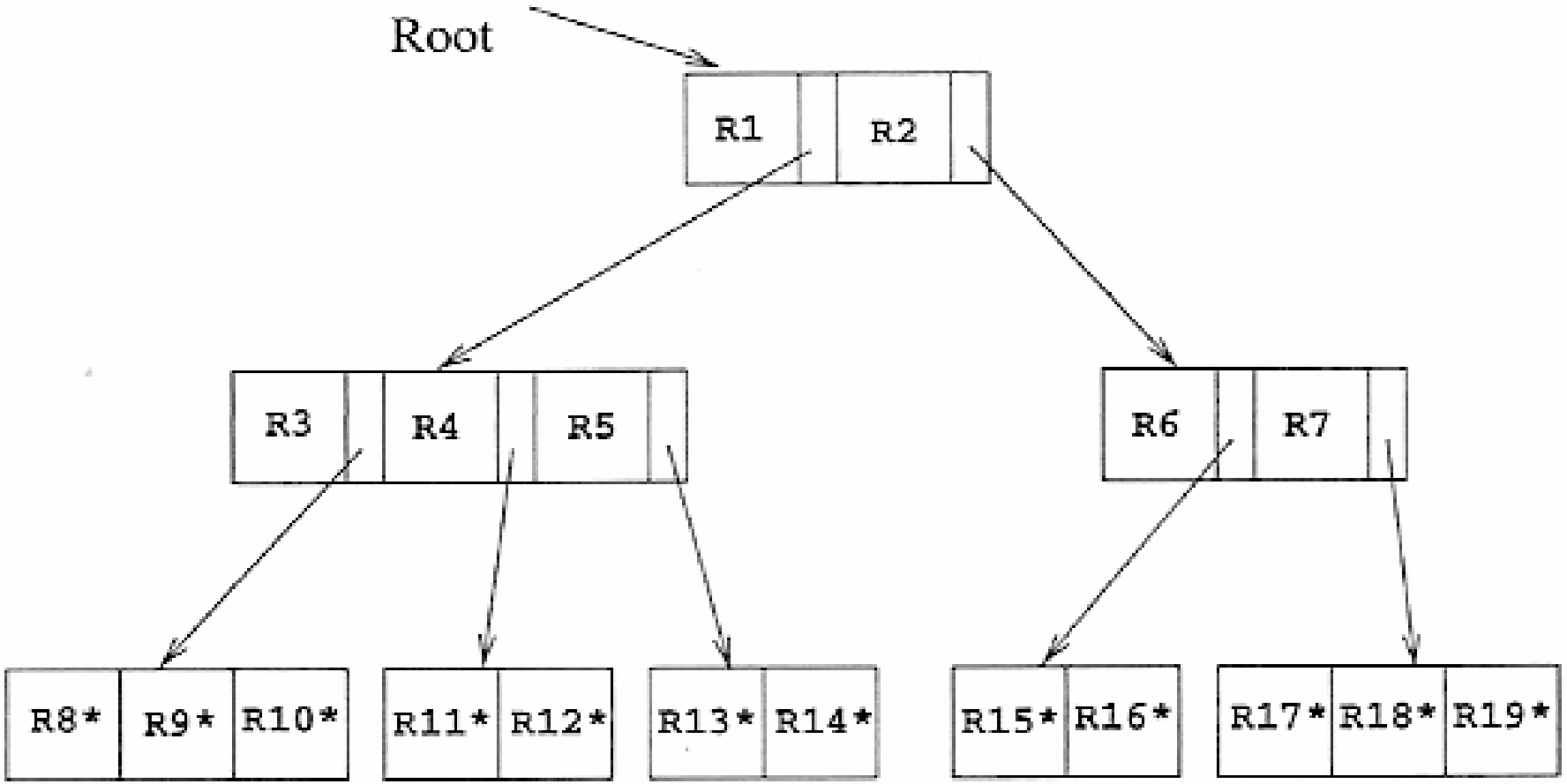
Index elem: (téglalap, pnt), ahol pnt egy gyermek szögpontra mutat, a téglalap pedig *a gyermek szögponthoz rendelt összes téglalap minimális befoglaló téglalapja* (amely tehát tartalmazza az adott szögponthoz tartozó teljes részfát).

Közbülső szögpont: index elemek halmaza.

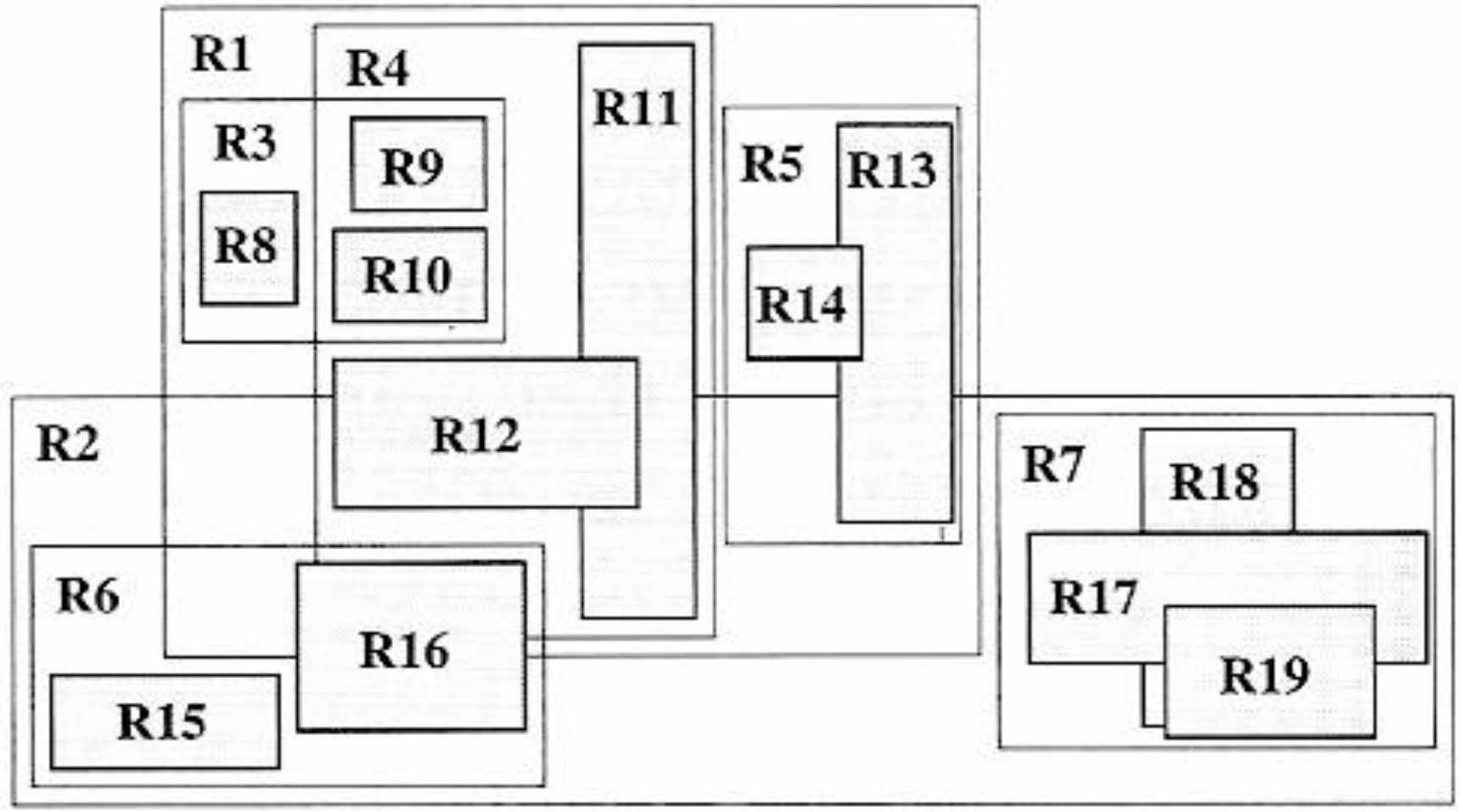
Adatelem: (téglalap, id), amely az *id* azonosítójú objektumra hivatkozik a befoglaló téglalapjával. Ha az objektum pont, akkor a téglalap is ponttá zsugorodik.

Levél: adatelemek halmazát tartalmazza.

Példa R-fára


Root
R1
R2
R3
R4
R5
R6
R7
R8*
R9*
R10*
R11*
R12*
R13*
R14*
R15*
R16*
R17*
R18*
R19*

Példa R-fára (folytatás)



Szabályok

- Egy objektum csak egy levélen szerepelhet.
- Az R-fa rangja *r*, ha egy lapon (lemezblokkon) legfeljebb 2*r* indexbejegyzés fér el.
- Az indexbejegyzések minimális száma *m*, ahol $0 < m \leq r$. *m* értéke a konkrét R-fa-kezelő algoritmustól függ, egy szokásos érték például $m = 0.8r$.
- A gyökér legalább 2 bejegyzést tartalmaz.

Pont szerinti lekérdezés

Adott pontot tartalmazó objektumok keresése:

- A gyökértől lefelé keresünk.
- Ha egy gyermek téglalapja tartalmazza a pontot, akkor a megfelelő részfában keresünk tovább.
- Ha több gyermek is tartalmazza a pontot, akkor mindegyik részfát végig kell nézni.
  Következmény: a keresési idő logaritmikusnál több is lehet, a legrosszabb esetben a teljes fát be kell járni!
- Ha egy levél téglalapja tartalmazza a pontot, akkor meg kell vizsgálni, hogy maga az objektum is tartalmazza-e.

A gyakorlatban általában logaritmikus idővel számolhatunk.

Téglalap szerinti lekérdezések

Adott téglalapba eső objektumok lekérdezése:

- Mint a pont szerinti lekérdezésnél, de mindig azt vizsgáljuk, hogy a keresési téglalap ***metszi-e*** az adott szögpont téglalapját.
- A levelek szintjén azt vizsgáljuk, hogy a keresési téglalap ***tartalmazza-e*** az adott levél téglalapját.

Adott téglalapot metsző objektumok lekérdezése:

- Mint az előző esetben, de a levelek szintjén is ***metszést*** vizsgálunk.
- Ha a kereső téglalap metszi egy objektum befoglaló téglalapját, akkor ellenőrzendő, hogy az objektumot is metszi-e.

Ponthoz legközelebbi objektum keresése

- Kis négyzettel vesszük körül a pontot, és ezt metsző téglalapokat keresünk.
- Ha egyet sem találunk, a kereső négyzet méretét növeljük mindaddig, amíg nem találunk metsző téglalapot.
- ***Megjegyzés:*** valójában négyzet helyett kört kellene venni, de négyzettel könnyebb számolni.

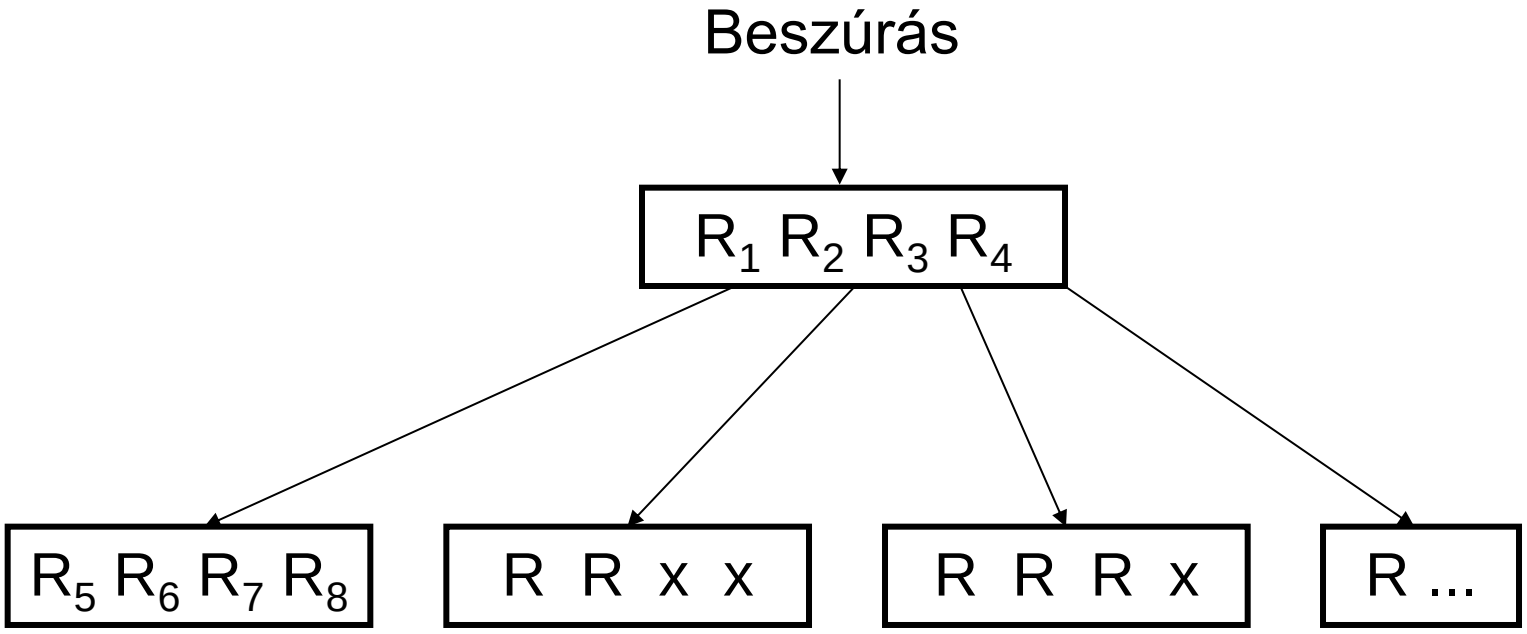
Beszúrás

- A gyökértől süllyesztjük a (R, oid) objektumot.
- Olyan leszármazottat keresünk, amelynek téglalapjába R belefér. Ha több leszármazott téglalapjába is belefér (átfedő téglalapok), akkor a bal szélső szögpontnál folytatjuk a beillesztést.
- Ha egyik leszármazott téglalapjába sem fér bele R, akkor azt a leszármazottat választjuk, amelynek téglalapja a legkevesebb nagyítást igényli (területi értelemben) ahhoz, hogy lefedje R-t.

Beszúrás (folytatás)

- Levél szinten felvesszük az új objektumot, és ha kell, növeljük a levél téglalapját, és visszamenőleg az ősök téglalapjait is.
- Ha a levélen már nincs hely az új objektumnak, akkor kettévágjuk a levelet. A kettévágásánál arra kell törekedni, hogy minimalizáljuk a téglalapok átfedését, mert az többszörös keresést okozhat.
- Ha a kettévágás miatt a szülő szögpont is betelik, akkor azt is kettévágjuk, és szükség esetén ezt a műveletet továbbvisszük az ősök felé (lásd B-fa). Ha a gyökér kettévágása is szükséges, akkor a fa szintszáma eggyel nő.

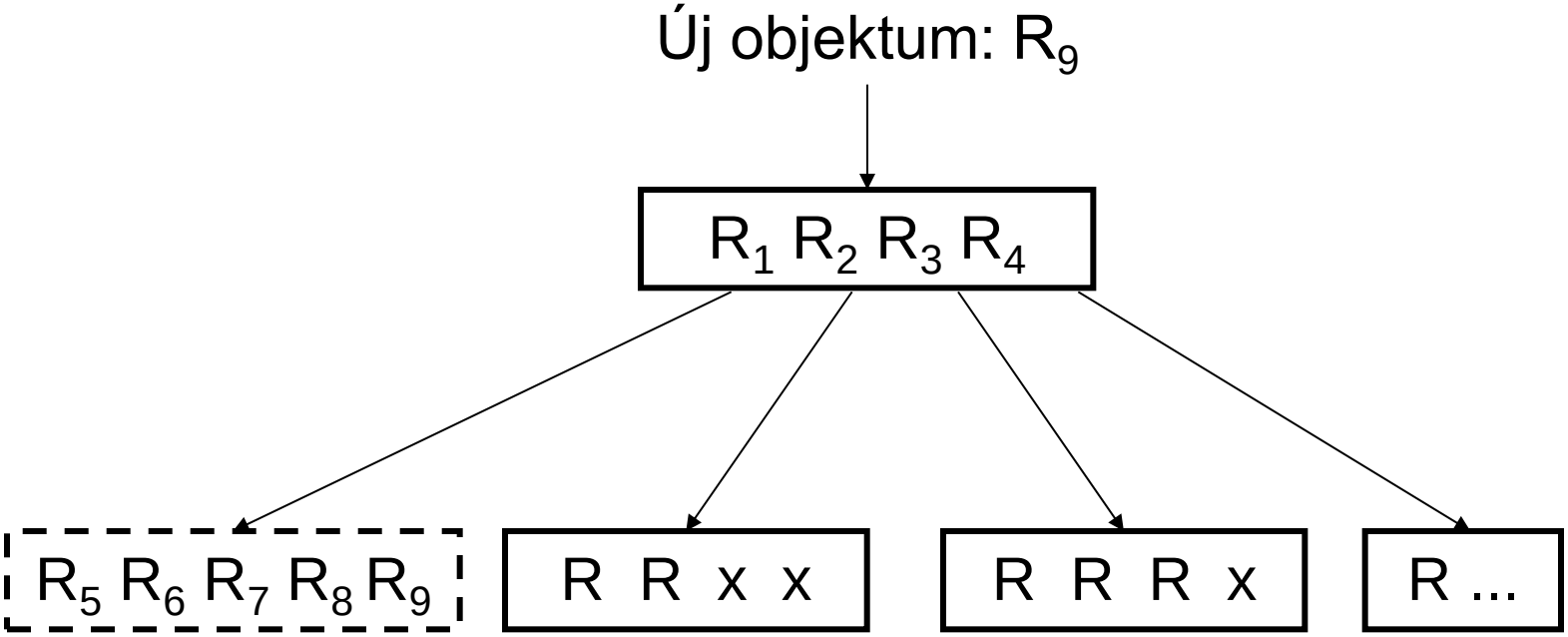
Beszúrás R-fába: kezdőállapot



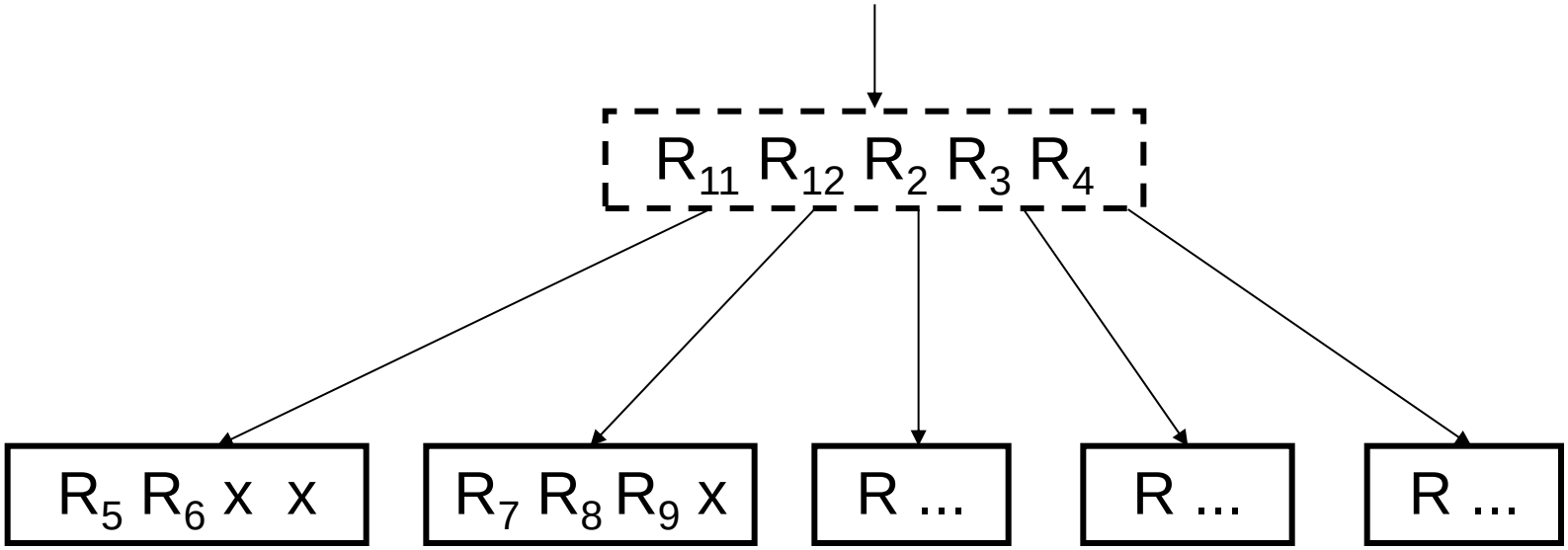
R: téglalap (rectangle), ***x:*** üres hely, ***nyilak:*** pointerek

***r* = 2**, azaz max. 4 indexbejegyzés fér el egy szögpontban.

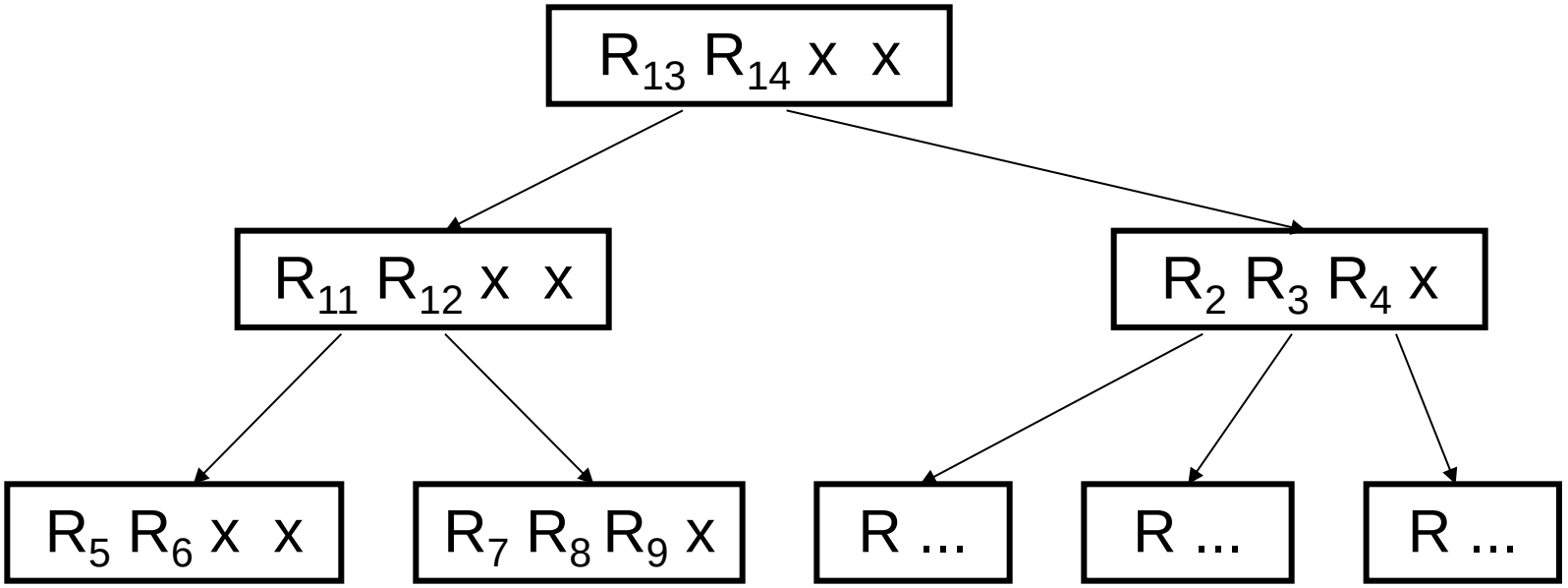
Beszúrás R-fába (folytatás)



Beszúrás folytatás – levél-lap szétvágása



Beszúrás folytatás – gyökérlap szétvágása



Törlés

- Az objektum törlése után a levél és az ősök téglalapjait lehet, hogy zsugorítani kell.
- Ha egy levél-lap *m*-nél kevesebb bejegyzést tartalmaz, akkor törlésre kerül, és az objektumokat újra be kell szúrni az R-fába.

R-fa változatok

R*-fa: kényszerített újra beszúrás elve: ha egy szögpont betelik, az elemek kb. 30%-át újra beszúrjuk a fába, ezzel csökkentve a szögpont-kettéosztás valószínűségét.

R+-fa:

- A téglalapok adott szinten nem metszik egymást.
- Következmény: a keresési idő mindig logaritmikus.
- Többszörös beszúrás: új elem beszúrásra kerül minden olyan részfába, amelynek téglalapját metszi.
- Ezért az R+-fa jelentősen nagyobb lehet, mint ugyanarra az adathalmazra felépített R-fa.

Alkalmazás: térbeli összekapcsolás

T1(a1, g1) és T2(a2, g2) relációsémák, ahol a megfelelő táblák g1 ill. g2 szerint R-fa-indexeltek.

Lekérdezés:

```
SELECT * FROM T1,T2 WHERE intersect(g1,g2);
```

Indexelés nélkül: minden T1-beli g1-et minden T2-beli g2-vel össze kell hasonlítani.

Indexeléssel: Legyen N1 és N2 a két R-fa aktuálisan vizsgált szögpontja (kezdetben a két gyökér). Rekurzív algoritmus a következő dián.

Rekurzív algoritmus (vázlat)

```
SpatialJoin(N1,N2) returns (id1,id2) párok halmaza
begin
 result := kezdetben üres;
 for all R1 in N1
  for all R2 in N2
   if (R1 ∩ R2 nemüres) then
    if (levél szinten vagyunk) then
     result += (id1,id2);
    else
     N1 := ReadPage(R1.pnt);
     N2 := ReadPage(R2.pnt);
     result += SpatialJoin(N1,N2);
   end if; end if;
 end for; end for;
end
```

Gyorsítás

A kettős ciklus lassú lehet: ha egy szögpontban 100 indexbejegyzés van, 100*100 = 10 000 összehasonlítás szükséges.

Gyorsítás: legyen R az N1-hez és N2-höz tartozó befoglaló téglalapok metszete: R = N1.mbb ∩ N2.mbb. N1-nek és N2-nek csak azokat az index-bejegyzéseit kell vizsgálni, amelyek téglalapja metszi R-t.