12. METODY INDEXOVÁNÍ V PROSTOROVÝCH DB

Prostorové databáze – pracují nad jednoduchými prostorovými objekty, které v nějaké podobě také uchovávají (persistentně)
– jednoduché geometrické entity
– body, úsečky, lomené úsečky, polygony, ...

Typy:
- body
- úsečky (či lomené)
- regiony
  - polygony 
  - plochy 

– vše v prostoru je spojité ale my to potřebujeme uložit do databáze v diskrétní podobě (potřeba PC)

⇒ problém diskretizace spojitých prostorových objektů

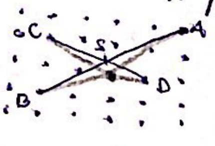
– řešení:

1) d-simplexy – nejmenší nevyplněné objekty dané d dimenze ze kterých se ostatní objekty skládají

0 – body
1 – úsečky
2 – trojúhelník
⋮

– každý d-simplex se skládá z d+1 (d-1)-simplexů

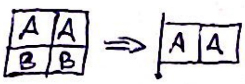
2) deskriptory (mapy na síť bodů) – množina bodů/úseček/vyšších objektů které jsou souhrnně popsány
– žádné objekty se nekříží a pokud ano, se v bodě křížení rozdělí
– zaznamenávají se do databáze jen koncové body
– každé koncové body jsou body sítě prostoru
– každý bod který je potřeba zaznamenat a není bodem sítě se na ten bod sítě „posune" – tu vzniká nepřesnost



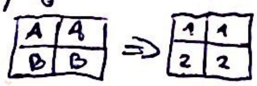
– posune se ten jejich průsečík na bod sítě a rozdělí se na 4 úsečky AS, BS, CS, DS a uloží se do DB jen ty body ⇒ zkreslení

Operace
1) predikátové – je uvnitř, sousedí s, rovná se, nerovná se, ...
2) geometrické relace – průnik, překrytí, ...
3) výpočetně náročné – plocha, vzdálenost, ...
4) další

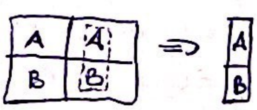
• selekce



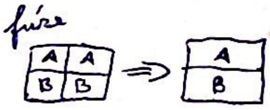
• projekce



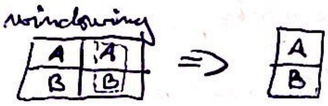
• ořezání



• fúze



• windowing



Indexace bodů

- jak už víme tak v DB neukládáme celé spojité objekty ale většinou pouze například koncové body úseček atd.
- index je nějaká datová struktura sloužící k akcelerování vyhodnocování dotazů nad daty
- v prostorových databázích jsou indexy velmi důležité protože bez nich by rychlé dotazy (nejbližší soused, plocha, ...) byly velmi obtížné

1D

- v 1D prostoru se dají využít rozšířené hashovací tabulky nebo B-stromy → všechny listy jsou na stejné úrovni
- je to klasické indexování nad kterým je definované uspořádání

↳ prostor se rozdělí na buňky a ty se mapují do hashovací tabulky

↳ data jsou seřazena v listech
- když se uzel v B-stromu přeplní je to dost pracné na to aby se složil
- listy obsahují jen 50%

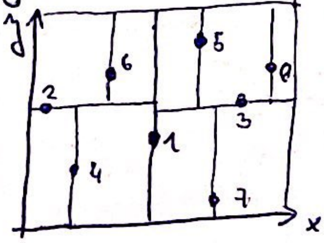
ND

- dá se mapovat do 1D a indexovat pomocí rozšířených hashovacích funkcí nebo B-stromů - není to vždy vhodné protože se ztrácí informace
- využívají se stromy dělící prostor a speciální hashovací algoritmy a hybridní
  (K-D Tree, BSP tree, Quad tree) (Grid file, Two-level Grid file, Twin Grid File) (BANG File) - strom i hash

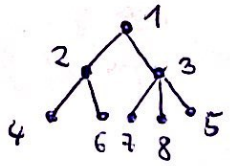
1) Stromy dělící prostor

K-D Tree

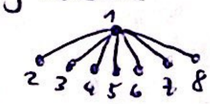
- celý prostor je rozdělen hyperplochami (plochy, úsečky). Zde každá hyperplocha prochází bodem a zároveň jsou hyperplochy rovnoběžné s kartézským souřadným systémem
- při odebírání prvků se musí celý strom znovu sestavit ↳ nebo alespoň ten podstrom kterého se to týká
- špatné přidávání může strom degradovat
- body jsou tedy v uzlech i listech
- dobrý pro vyhledávání sousedů atd...
- rychlé vyhledávání



=>



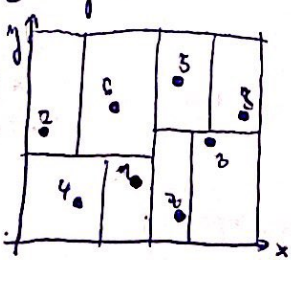
Kdybychom ukládali blbě tak to může tady nastat

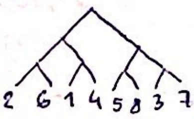


- pokud se ukládá bod tak se jím ten strom vždy dělí
- strom je tedy závislý na pořadí ukládání - adaptivní není

Adaptivní K-D Tree

- zase hyperplochy rovnoběžné s kartézským souřadným systémem
- nyní hyperplochy neprochází body ale mezi nimi
- body jsou tedy jen v listech
- v každé ploše 1 bod!
- snaží se mít na každé straně hrany přibližně stejný počet bodů
- odpadá problém s ukládáním





- jednoduché mazání, jen odebrán list

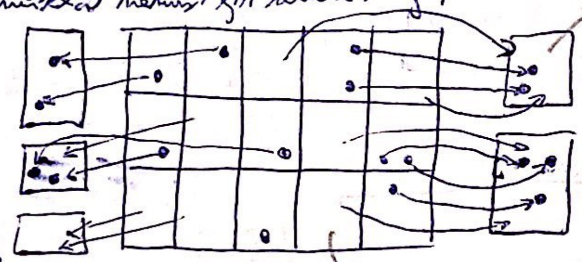
BSP Tree

- pouze hyperplochy (plochy, přímky / úsečky)
- stejně jako u adaptivního K-D Tree by plochy neprochází body
- dělí prostor na poloviny
- snaží přibližně stejný počet prvků v každé polovině
- hyperplochy nejsou rovnoběžné s kartézským souřadným systémem
- nemusí dělit tak aby v každém oblasti byl 1 bod - může nechat víc (u K-D adaptivního jen 1 bod v oblasti)
- nezaručuje žádné zlepšení, protože složitost může být větší

2) Adaptivní hashování - to mapování dělá hashovací funkce!

Grid File

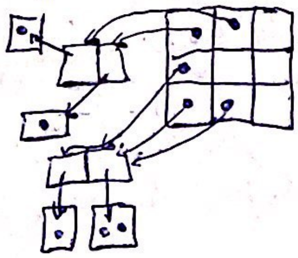
- na prostor je přiložena mřížka a ten prostor bodů je tak rozdělen do buněk
- n-rozměrný prostor rozdělen n-rozměrnou mřížkou — hyperplochy (úsečky, plochy)
- v paměti je adresář který jednotlivé buňky mapuje na datové jednotky (buckety) kde se nachází přímo ty body ↳ může i více buněk na jednu jednotku
- pokud se přidá bod do prostoru a ta buňka mřížky mu přetéče (je tam moc bodů) tak se musí rozdělit - rozštěpí se adresář ↳ (může třeba omezena buňka na 4 body)
- pokud se vyprázdní datová jednotka tak se sloučí s jinou a musí se třeba i přepočítat mřížka
- v datových jednotkách se hledá už sekvenčně
- mřížka nemusí být tabulka (?) hezky pravidelná



- mazání ani vkládání není lokální → může být globální
  ↳ protože při přeplnění buňky mřížky se musí vložit další hyperplocha a při vyprázdnění datové jednotky (odebere se poslední prvek) se sloučí s jinou datovou jednotkou a eventuálně se zjistí že se musí odstranit hyperplocha
- mapování buňky mřížky na datovou jednotku je ve velkém adresáři na HDD

Two level Grid File

- je jako grid file - pouze mřížka ale datové jednotky jsou pouze gridfile a ty až ty buckety - dvouúrovňový grid file



- změny jsou častěji pouze lokální ale pořád může docházet i k nelokálním

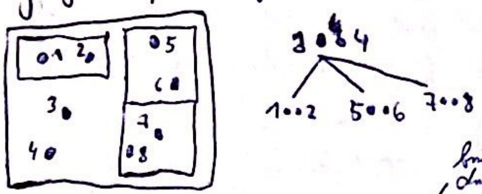
Twin Grid File

- dva grid filey a když ten primární přetéče tak se data vkládají do toho druhého
- jeden je primární a druhý přetokový ↳ aby se nemuselo dělit
- 2 grid fily vedle sebe

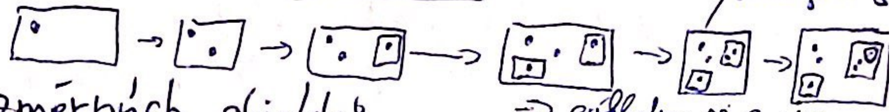
3) Hybridní algoritmy (hashování a stromy)

BANG File

– základem je Grid File kde se ale buňky překrývají a pokud je v buňce více jak 2 body tak se vytvoří podbuňka
– v buňka je stále datová jednotka
– datové jednotky jsou ve vyvažovaném stromě
– data nejsou jen v listech ale i v uzlech



Př.: max 2 body



⇒ základem je Grid File + vyvažovaný strom

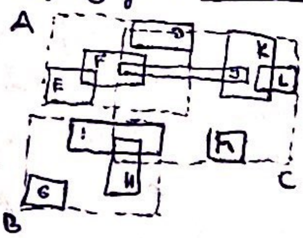
Indexování celých vícerozměrných objektů

– většinou se indexuje jen bod ale v některých případech je nutné indexovat celé vícerozměrné objekty
– indexují se vícerozměrné struktury (mapy, polygony, obdélníky) třeba v GIS
– kvůli komplexnosti se řeší bounding boxy (BOUNDING BOX)
– máme tři způsoby:

1) Transformace / mapování – mapují se objekty popsané k body v n-dimenzionálním prostoru na 1 bod v k·n-dimenzionálním prostoru
   – obdélník ve 2D na bod ve 4D – trojúhelník ve 2D na bod v 6D atd..
   – má to problémy převodu zpět a některé dotazy nejsou proveditelné

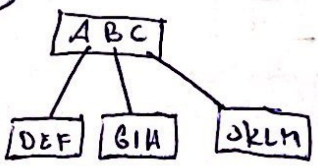
2) Překrývání

R-Tree
– obrazce se mohou překrývat a jednotlivé buňky se také překrývají
– buňky obalují objekty a pokud jedna buňka obsahuje moc objektů tak se dělí
– vždy může být objekt ve více buňkách ale v 1 buňce je vždy celý
– buňky se mohou překrývat, objekty také a jeden objekt může zasahovat do více buněk ale v jedné je kompletně celý
– **objekty** jsou **v listech stromu** a v **uzlech jsou buňky**



– vidíme že 1 buňka max 4 objekty a pak se dělí
– pro každý objekt se vytvoří buňka která je obalí celé
– třeba A obaluje celé D, E, F

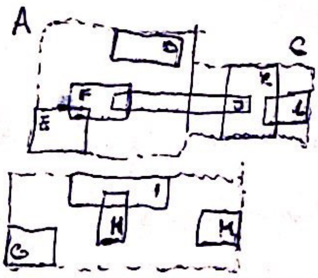
⇒



– není jen obdélník – polygony má P-tree – používá konvexní obálku

3) Ořezávání – buňky se nepřekrývají a objekty jsou na hranicích buněk rozděleny a jedna část může spadat do jedné buňky a druhá do druhé
   – pokud jsou buňky od sebe a mezi nimi je část objektu tak se musí ke vložení nová buňka nebo stávající rozšířit ⇒ nebezpečí deadlocku

R⁺ Tree



⇒

– vidíme že J je ve dvou buňkách