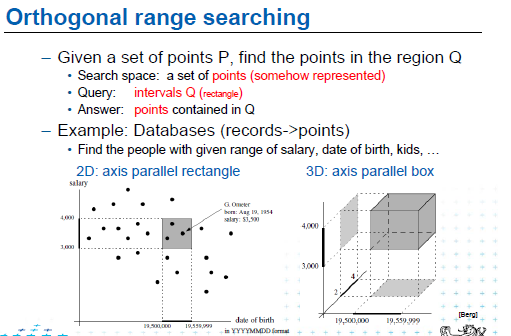
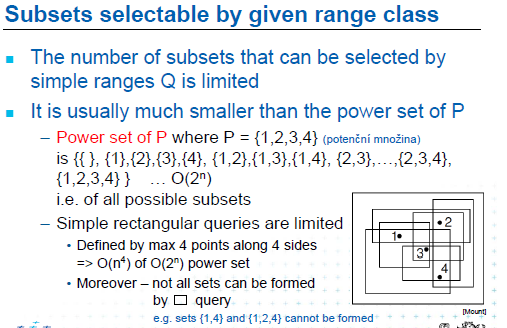
# 23. Ortogonální vyhledávání, kD strom, intervalový strom (range tree), segmentový strom.

1. **Ortogonální vyhledávání.**

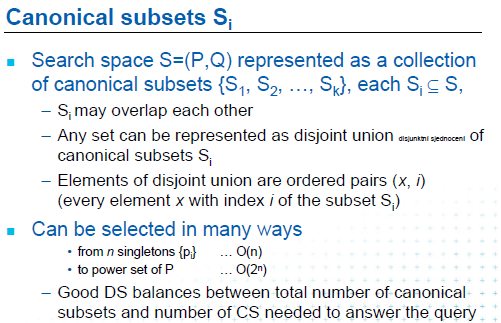
****

****

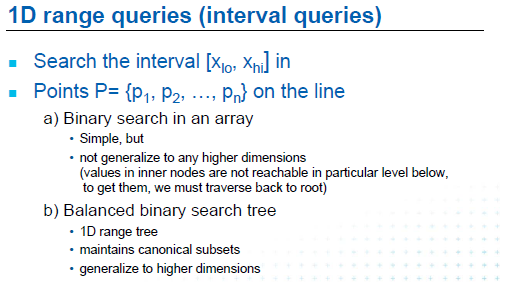
****

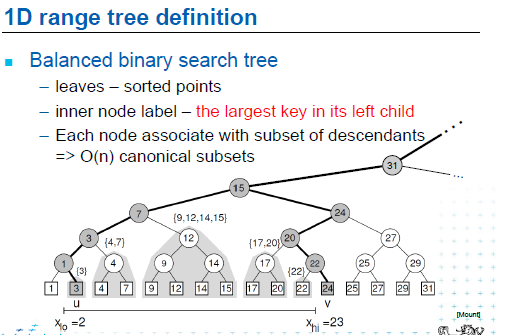
****

**(6)** vsechny možné podmnožiny**,** jako že těch možností je strašnej hafec, více než n^2

****

**(7)** <http://research.engineering.wustl.edu/~pless/506/l11w.html>

****

****

**(9) 1D range tree (Balanced BST)**

**značka uzlu = largest key - nejvetsi hodnota v levem podstromu,** treba node 7, kouknu nalevo a najdu tam nejvetsi hodnotu a zjistim, ze je to 7 uplne dole, co ten klič udává? asi neco jako **median**, proste kdyz mam mensi nebo roven 7, tak doleva, ze to najdu nalevo a zbytek napravo

**každý uzel** je asociován s **subset of descendant (potomci) = canonical subset**

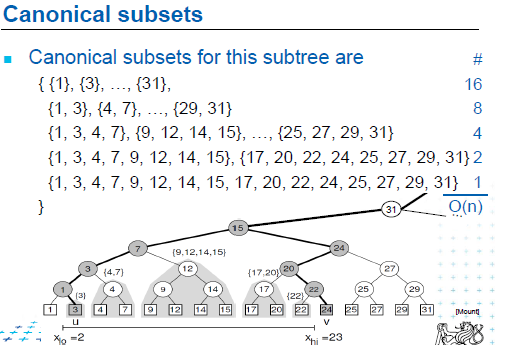
a pak se asi nejak neco vybira v zavislosti na tom intervalu co je dole ... tedy např 2 - 23

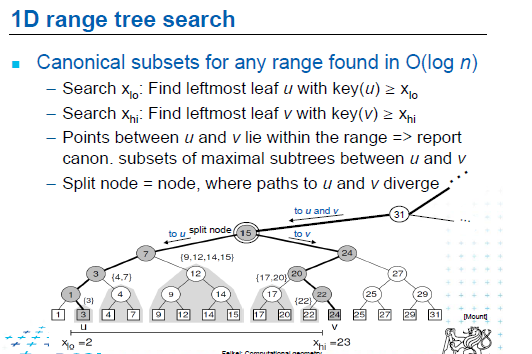
tak projizdis strom a vycucáš si ty množinky ... {3}, {4, 7}, {9,12, ..} a to jsou ty **kanonicke**

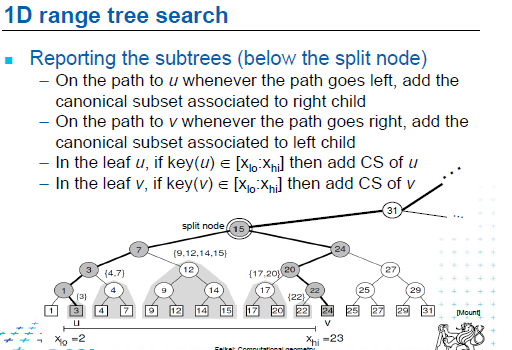
k tem **binarnim vyhledavacim stromum,** pro nalezeni bodu uvnitr nejakeho range, tak takova poznamka:  
v podstate jde o to najit **levy list a pravy list** - to je jednou log n a podruhe log n a vnitrek mezi temi konci znas

takze pocet cisel uvnitr range ma slozitost log n, stejne tak suma vsech cisel uvnitr range, tak treba pocet vsech prvku uvnitr rozsahu, najdes list **nejvice vlevo - log n**, najdes list **nejvice vpravo - log n**, a na ceste poscitas pocty cisel v tech uvedenych mnozinach nad uzly, ty uz mas predpocitane, takze **slozitost cele operace je jen log n,** totez plati pro sumu vsech listu, jen, ze jsem to ted pochopil jak to funguje a proc je ta slozitost takova jaka je

no ale musiš jít doprava i doleva, tak to by mělo být 2 log n a ta dvojka se v asymp.slo. ruší, tak to je? jj

****

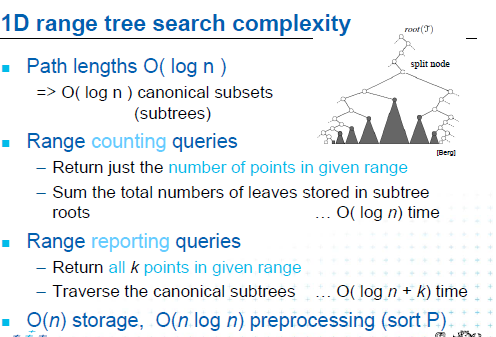
****

****

**(12)**

takže na cestě za u i přesto že jdu doleva, přidávej can.sub. pravým potomkum no a proč? protože vlevo se jde akorát po cestě dál ... nejsou tam žádné uzlíky na vyrábění can. sub. na druhé cestě doprava zase budu přidávat levým potomkum can.sub., to je prostě logika... proste can. sub. vytvarim jen z uzlu v intervalu?

ta cesta pro u je 15, 7, 3, 1 ... ty další dvě věty, pokud dojdu do u či do v, tak jestli jeho klíč, což je asi ta 3 pro u a pro v 23, tak z nich taky udělej can.sub. (jednoprvkove ... )

****

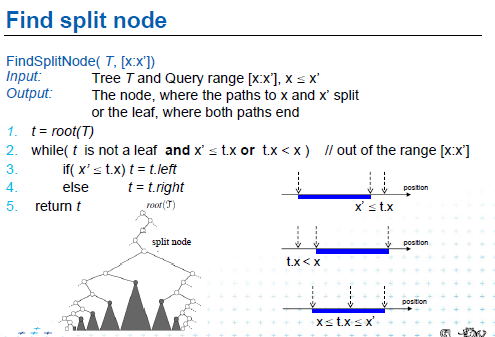
**(13) 1D range tree search complexity**

takže **délka** **cesty** je klasickej O(log n), binárka, jj, jasne

pak ten **dotaz** prostě vrací ty čísla v daným rozsahu se stejnou složitostí

a to posledni: vratí všech k bodu v rozsahu, travers can. subtrees v tam tom case ... to uz moc nedávám ...

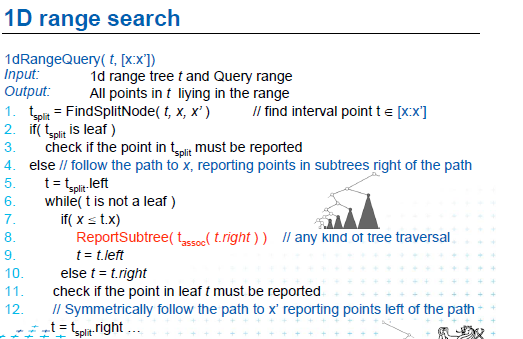
pak **storage** O(n), pak **sortovani** O(n logn)

****

**(14)**

takže při hledání toho **split uzlu** se může stát, že se nám vrátí jeden list, kde obě cesty končí ...

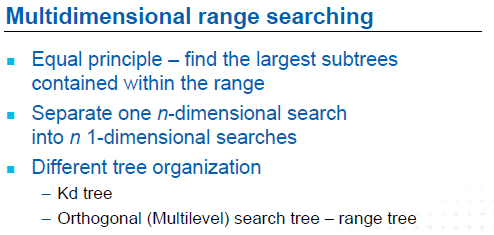
t=root(T), kde t.x a t.x' jsou hodnoty z toho rozsahu, tedy cisla daných uzlu a co je ale t.left a t.right? potomci toho t

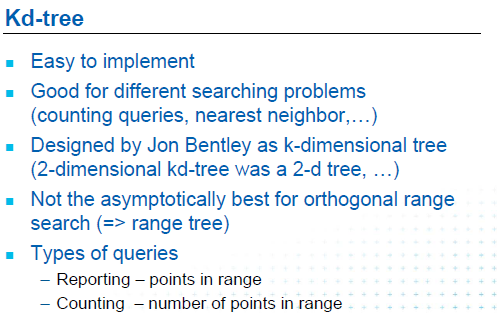
****

**(15) 1D range search**

line 3 jak chápeš? mozna, ze bud mohu vracet cisla na hranici intervalu nebo ne, takze tady je ta kontrola

tak ten **range query** vraci cisla v rozsahu ... ano ... to bude ono, proste jestli to vratit nebo ne ..

****

****

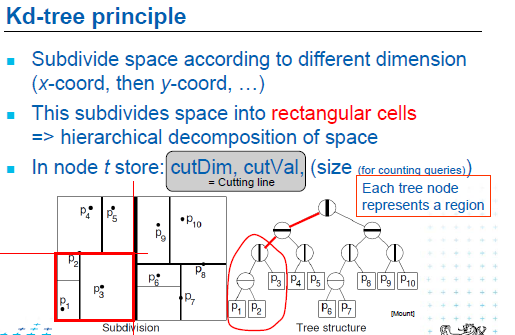
**(16,17) Multidimensional range searching**

stejný princip - najdi nejvetší podstromy obsažené v rozsahu, odděl 1 n-dimensialni vyhledavani do n 1-dimensionalnich vyhledavani – **Kd tree (2-dimensional kd-tree was a 2-d tree)**

**Types of queries**

– Reporting – points in range

– Counting – number of points in range

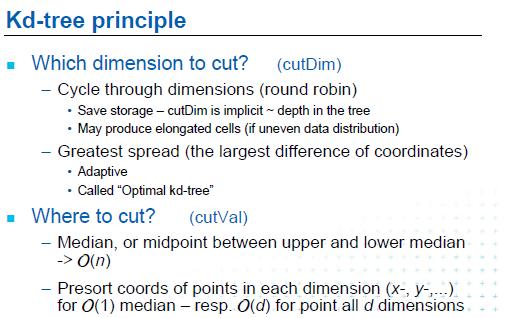
****

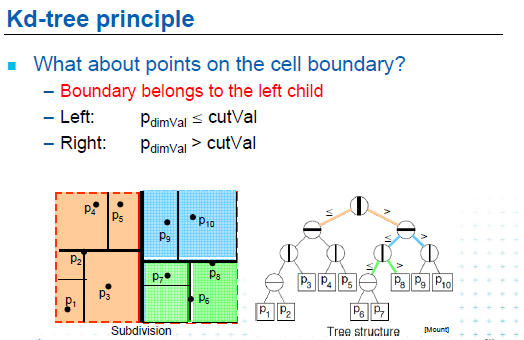
**(18)**

tohle jsme meli na MPV, jj, kd-tree, ale ne moc detailne, jen ze to existuje, pro nalezeni nejblizsich sousedu

**elongated** **cells** – prodloužené cely, **uneven** – nestejná, **spread –** rozšíření

Kde rozdělit? takže přesortíš všechnybody v x a y dimenzich a vybreš medián

****

****

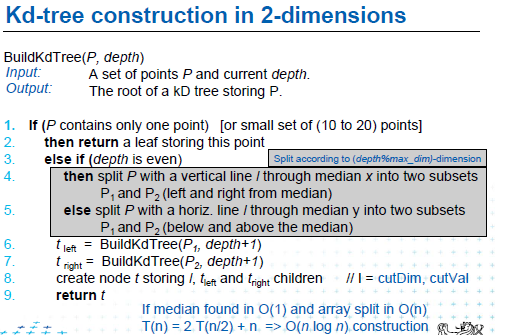
**(20)**

**takze na hrane doleva**

a rezes to tedy primo tim **medianem** jo? pres ten bod? no asi ano... ale nebylo by vice bodu na hrane ... kdyby jsi to rezal vzdy skrz bod? zda se ze jo, barvy - jo on to pak přebarvil napravo pro ukazku, levou ponechal ...

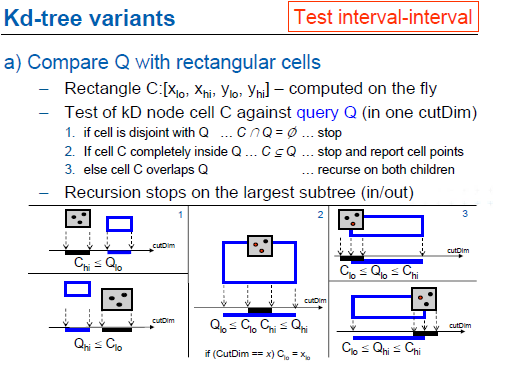
proste mas **vsechny body** v prostoru, tak si je seradis podle x a najdes bod uprostred – **median**, tim to oddelis **vertikalne na dve pulky - levou a pravou**, pak si vezmes tu **levou pulku** a seradis si to **podle y,** najdeš median a udelas **horizontalni rez** a oddelis na horni a dolni, a tak pokracujes

**!** Pro nalezení **mediánu** daného souboru stačí hodnoty seřadit podle velikosti a vzít hodnotu, která se nalézá uprostřed seznamu. Pokud má soubor sudý počet prvků, obvykle se za medián označuje [aritmetický průměr](http://cs.wikipedia.org/wiki/Aritmetick%C3%BD_pr%C5%AFm%C4%9Br) hodnot na místech *n*/2 a *n*/2+1.

****

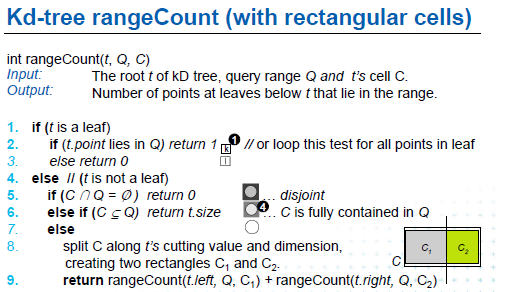
**(21)**

Split according to **(depth%max\_dim)-dimension,** hloubku znam i dimenzi ... ale to max dim? urcis jestli rezat podle x nebo podle y, max\_dim je pocet dimenzi, v nasem pripade 2 - x a y

****

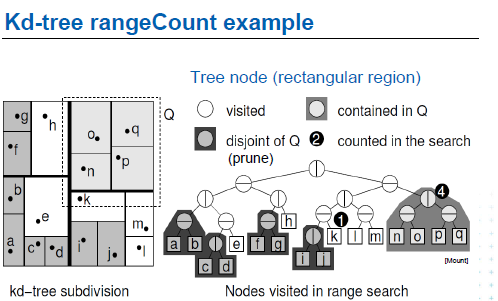
**(22)**

**a) Compare Q with rectangular cells,** prostě **dotaz Q je ractangle (modrý)** a vracíš body z toho diagramu ...   
tak Q je query (rectangle) ... a C je cell (množina bodu)

****

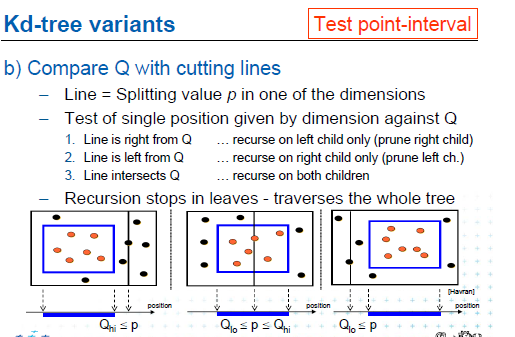
**(23)**

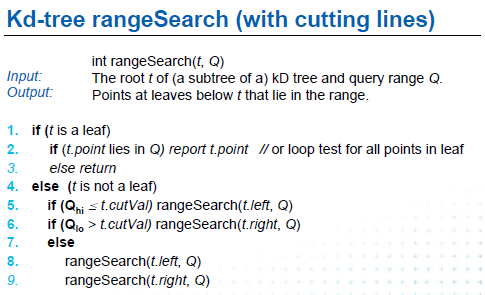
ale Q by tady melo asi nejak z casti zasahovat do C, tedy existuje prunik C a Q, ale C neni podmnozina Q, to je to if a else if

****

**(24)**

ten **dotaz** zasahuje i do **jiných oblastni**, jako je h, e, k, m ... to l se mi už nezdá, to je asi chyba, no ale tyto množiny nejsou vybrány celé ... takže ty nebere v potaz, dotaz tedy vrátí pouze ty světle šedé, ty jenž pokrývá celé

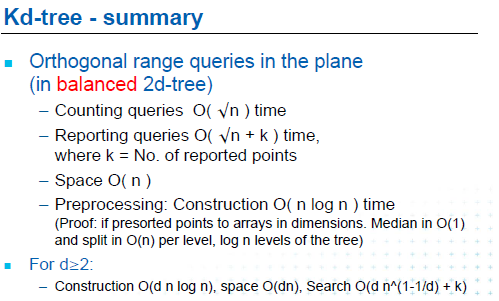
****

****

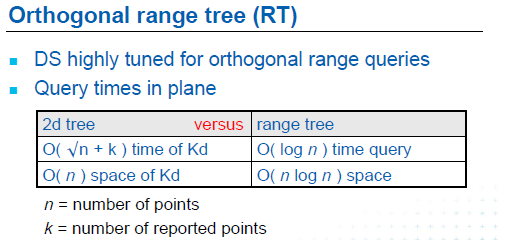
**(26)**

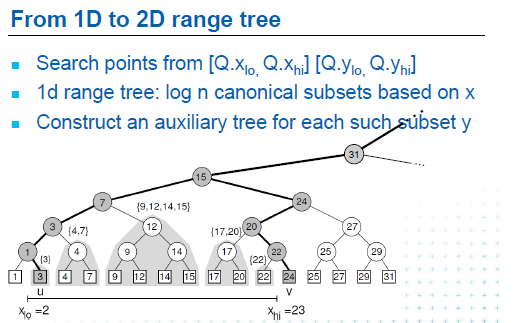
takze pokud je t leaf, tak vratime, ze je intersect nebo neni intersect podle toho zda lezi v Q ci ne, kdyz to neni leaf

tak zkontrolujeme, jestli je **Q nalevo od delici cary**, tak budu hledat v lefem potomkovi, kdyz je Q **napravo** od cary, tak v pravem potomkovi a proc je tam to posledni else? rekurse aha, asi kdyz je **uvnitr intervalu (tedy neni ani napravo ani nalevo)**, tak se to prohleda cele i v levem i vpravem potomkovi

****

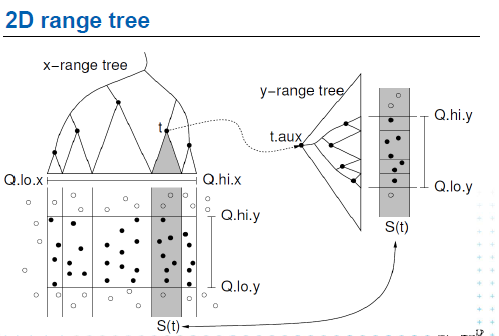
**(27) Orthogonal range queries in the plane (in balanced 2d-tree)**

****

****

**(29) From 1D to 2D range tree**

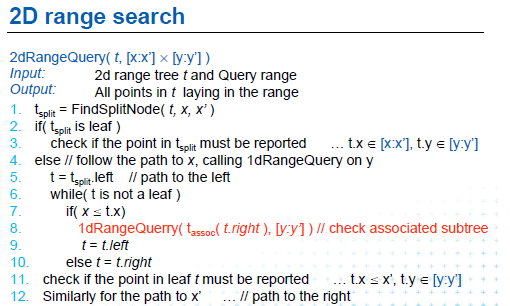
**dotaz je 2D**, tedy rozsah je nad osou x i y, a pro kazdou podmnozinu y vytvor **pomocný strom**

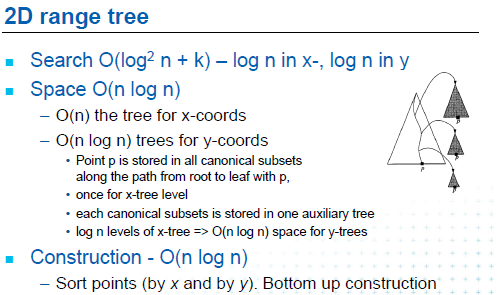
****

**(30) 2D range tree**

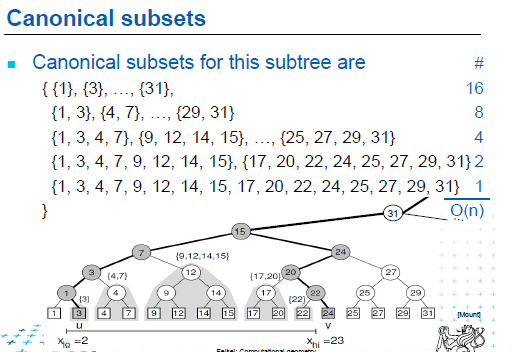
takze je to nejprve rozsekane mnoha vertikalnimi carami a pak kazdy kousek horizontalnimi, S(t) - ze by ten rizek? jj

**no prostě pro každy podstrom x existuje y strom, ktery to dále horiz. deli ...**

****

****

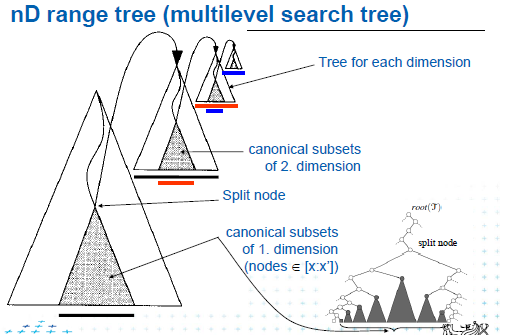
**(32)** proc ten strom pro x ma mensi space nez y? aha **pro ty x je jeden strom a pro y vice stromů**

****

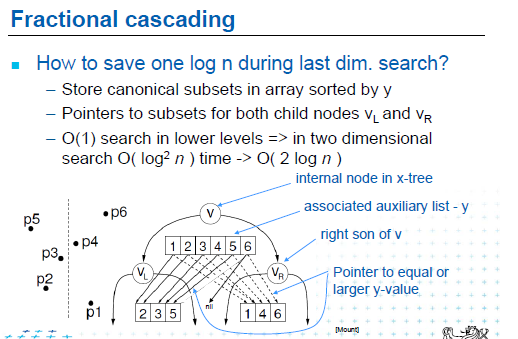
**(33)**

**canonical sub. = to je mnozina, ktera obsahuje listy z nejakeho podstromu daneho stromu, na 33 mame strom, tak vyberu si libovolny podstrom, treba uzel 12 a nize, tak kanonicka podmnozina bude napr.: {9,12,14,15}**

ale dobry je na tom to, ze kdyz chces vypsat seznam vsech cisel v intervalu, tak nemusis otevriat uzel 12 a spoustet se po nem dolu, ale staci se jen kouknout do 12 a vzit tu mnozinu kterou uzel 12 zna, takhle, kdyby ses spustil do listu 9, tak tam bys mel kanonickou mnozinu {9}

****

**(34)** nd-rande tree - asi neco podobneho jako 2d-range tree, pro **kazdy strom existuje strom v dalsi dimenzi**

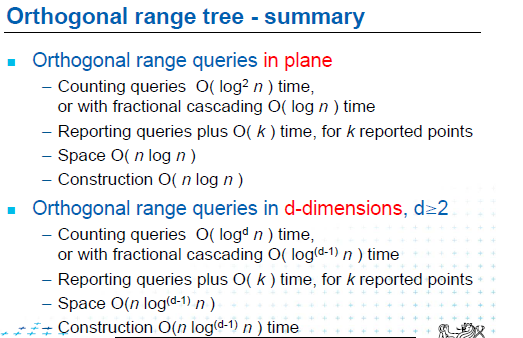
****

**(35) Fractional cascading** <http://research.engineering.wustl.edu/~pless/506/l12.html>

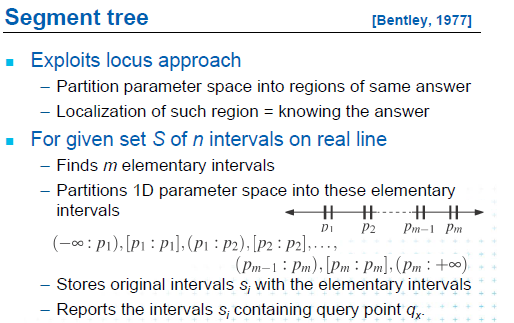
2d je specialni pripad nd, takze staci si pamatovat nd, proste snazi se podle y hledat v case O(1) a to tak, ze to podle y prohledaji pouze jednou, ne pro kazde x, jakoze vyberou ten dany y tree, jak jsme meli ze pro jeden x tree je nekolik y trees, rekl bych ze vsechny y tree nejak upravi, aby slo pouzit jen jeden a v nem vyhledat pouze jednou  
to bereš z té stránky jak jsem poslal jo? u toho **fractional cascaing**

takze na 35: V, VL a VR je klasickej strom podle x, takze kdyz vyhledavam treba p4, tak pujdu napravo a ted bych mel mit dalsi strom, ktery mi tu pravou oblast rozrizne podle y a mel bych hledat v nem jenze, tady je to vylepseni  
v **uzlu V jsme seradili body podle y** a udelali si ty **pointery na potomky**, pritom pointer ukazuje na stejne nebo vetsi cislo, takze na zaklade tech pointeru uz nemusime prohledavat y-novej strom, ale staci se podivat na ty pointery

treba vlevo mezi 5 a 3 je pointer ze 4, takze z toho nejak zjistim, ze 5 je nad 4-kou

****

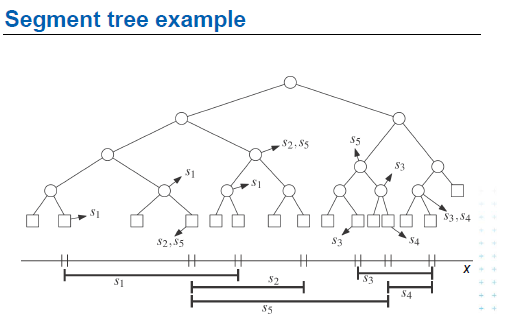
1. **Segmentový strom**

****

**(34)**

jak je to rozdelene, tak tam je prvni interval -inf;p1, nasleduje bod [p1: p1], pak zase interval p1-p2 a zase bod p2

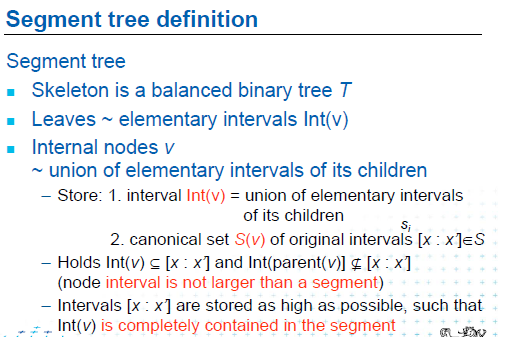
a tak dale ... takze male intervaly a body mezi nimi

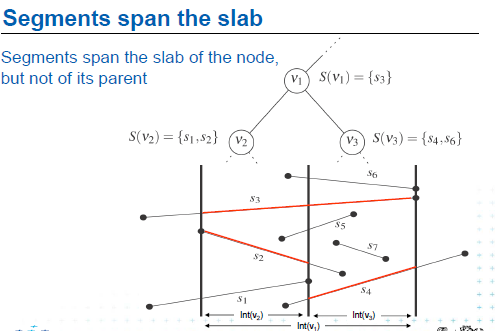
****

**(35)**

**si** jsou originalni intervaly - zadane asi, **pi** jsou elementarni intervaly, si popisujeme pomoci pi

tak ctverecky dole (leaves) jsou jednotlive intervaly (popr. bodove intervaly), šipky nevím

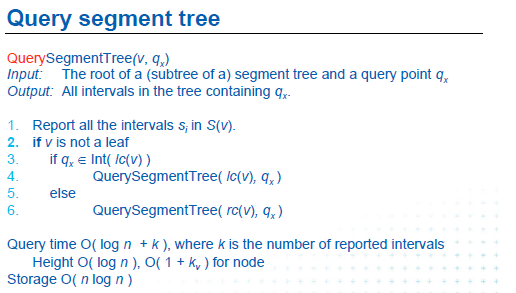
****

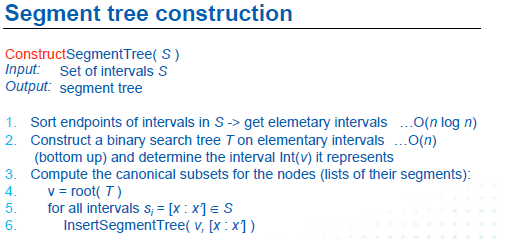
****

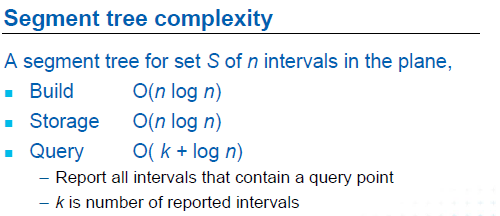
**(37)**

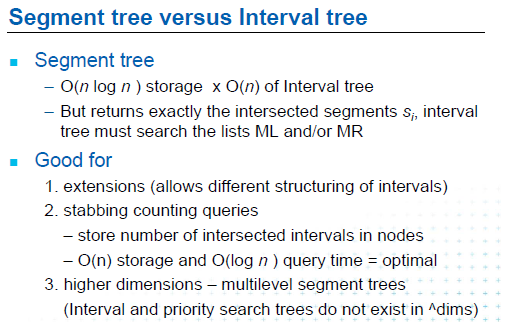
barva akorát ukazuje 3 případy pro 3 uzly, tedy pro rodiče a dva potomky, ale je třeba se na to koukat bez barev, pak je to jasné, segment musí protínat celý interval ...

jeste proc neni s3 v v2 nebo v3 ? protože protíná obojí ... to by pak nemuseli vubec dělat rodiče ... bo by byly k ničemu a v listech by bylo všechno ... chápeš? jde o to to rozložit ... po tom stromě ...

****

****

****

****