Algoritmy pro rozsáhlá data

L07: statické R-stromy; topologické dotazy a dotazy na nejbližšího souseda

Jan Konecny

25. listopadu 2021

Statické R-stromy

Nechť množina obsahuje N obdélníků a každý uzel může obsahovat až c obdélníků.[1]

Chceme sestavit R-strom.

- Můžeme postupně všechny vkládat do původně prázdného R-stromu.
- Výhodnější je ale použít statické algoritmy stavby R-stromu
  - též nazýváno packing či bulk-loading.
  - naproti tomu, algoritmy probírané v předchozích lekcích se nazývají dynamické či inkrementální.

[1] c může být kapacita uzlu, nebo menší číslo – to když chceme nechat ve stromě volné sloty na později vkládaná data)

Naivní statický R-strom

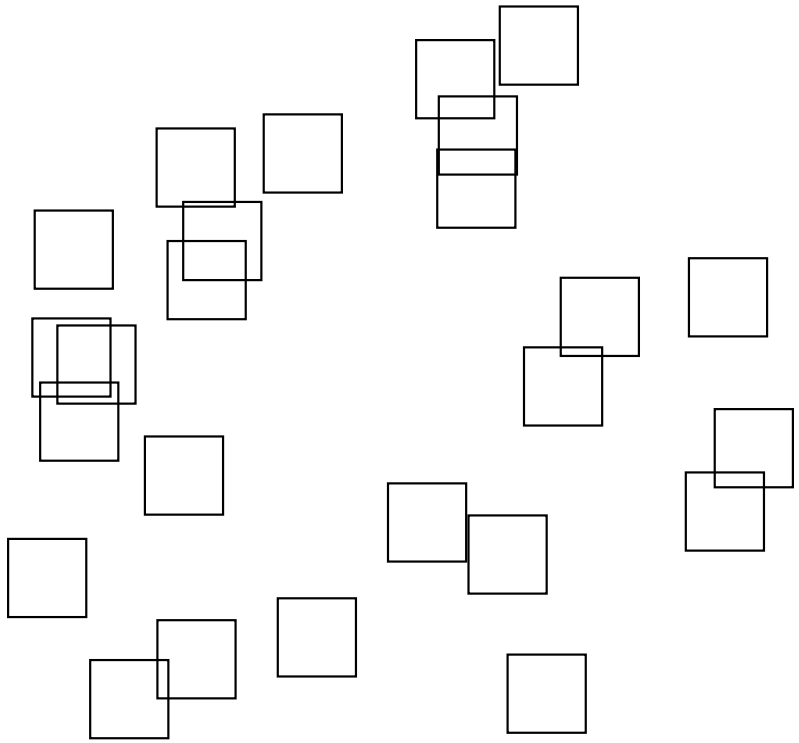
N. Roussopoulos, D. Leifker
Direct Spatial Search on Pictorial Databases Using Packed R-trees,
Proceedings ACM SIGMOD Conference on Management of Data,
pp.17-31, 1985.

Postup:

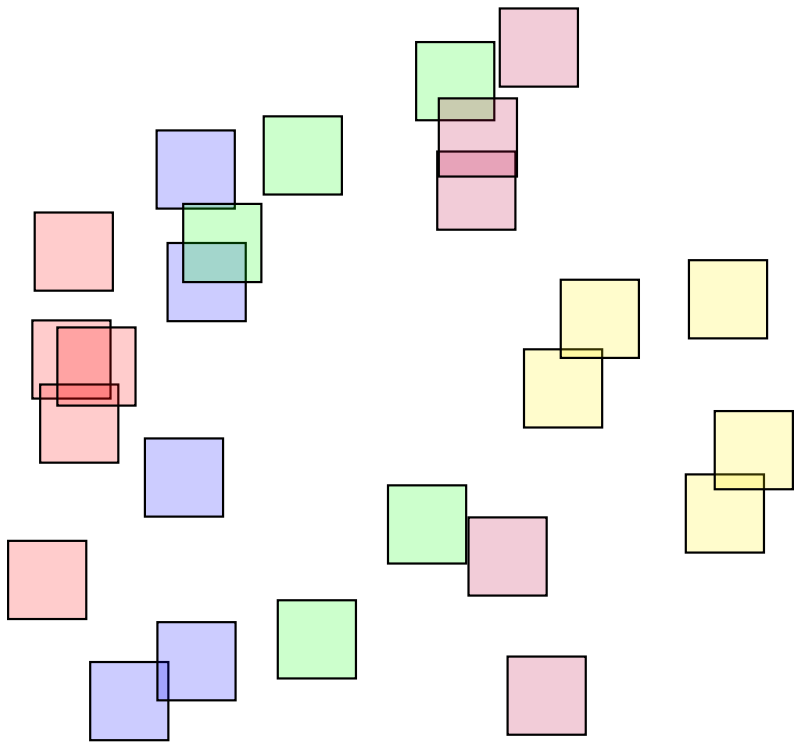
1. Setřiď obdélníky podle x-souřadnice jejich středu
   (stejně tak může být použita např. x-souřadnice levého dolního rohu).
2. Seskup obdélníky do $\lceil N/c \rceil$ po sobě jdoucích skupin o c obdélnících
   (až na poslední, ta může obsahovat 1 až c obdélníků).
3. Najdi MBR každé skupiny vytvořené v předchozím kroku a spoj každou
   s ukazatelem na uzel, ve kterém bude uložena odpovídajících množin.
4. Rekurzivně pokračuj s MBR z předchozího kroku.
   Rekurze pokračuje, dokud máne více než c obdélníků.

Příklad:

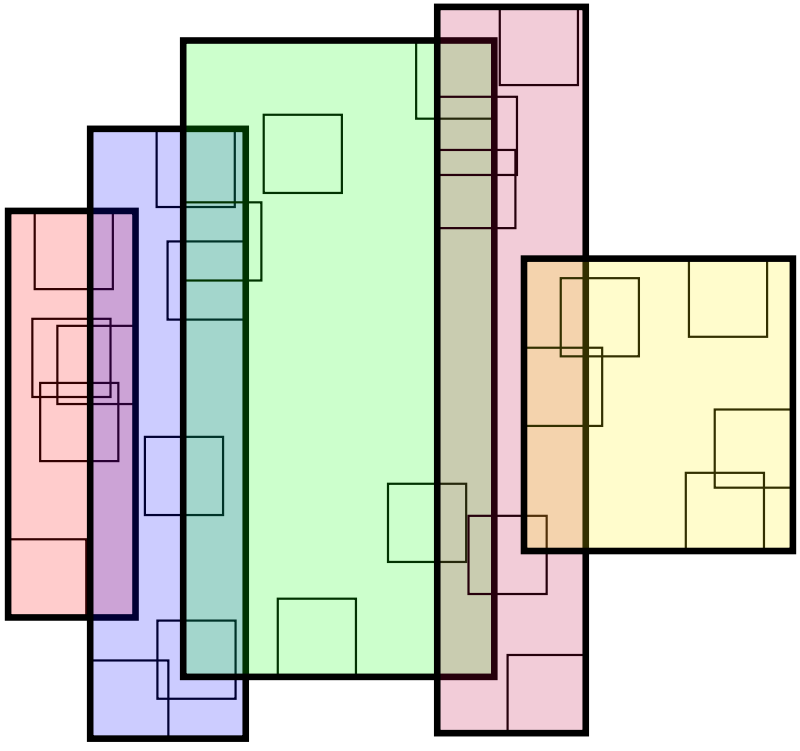
Zadání: $N = 25, c = 5$



Kroky 1 a 2: dostáváme $\lceil N/c \rceil = 5$ skupin o 5 obdélnících:



Krok 3: MBR jednotlivých skupin:



- Tento algoritmus nedává dobré výsledky.
  Data jsou vlastně „rozproužkována“ podle jedné ze souřadnic
  (x-souřadnice těžiště, některého rohu, ...).

- Tento algoritmus nedává dobré výsledky.
  Data jsou vlastně „rozproužkována“ podle jedné ze souřadnic
  (x-souřadnice těžiště, některého rohu, ...).
- Máme ale možnost použít lepší hodnotu než jednu souřadnici:
  H-hodnotu:

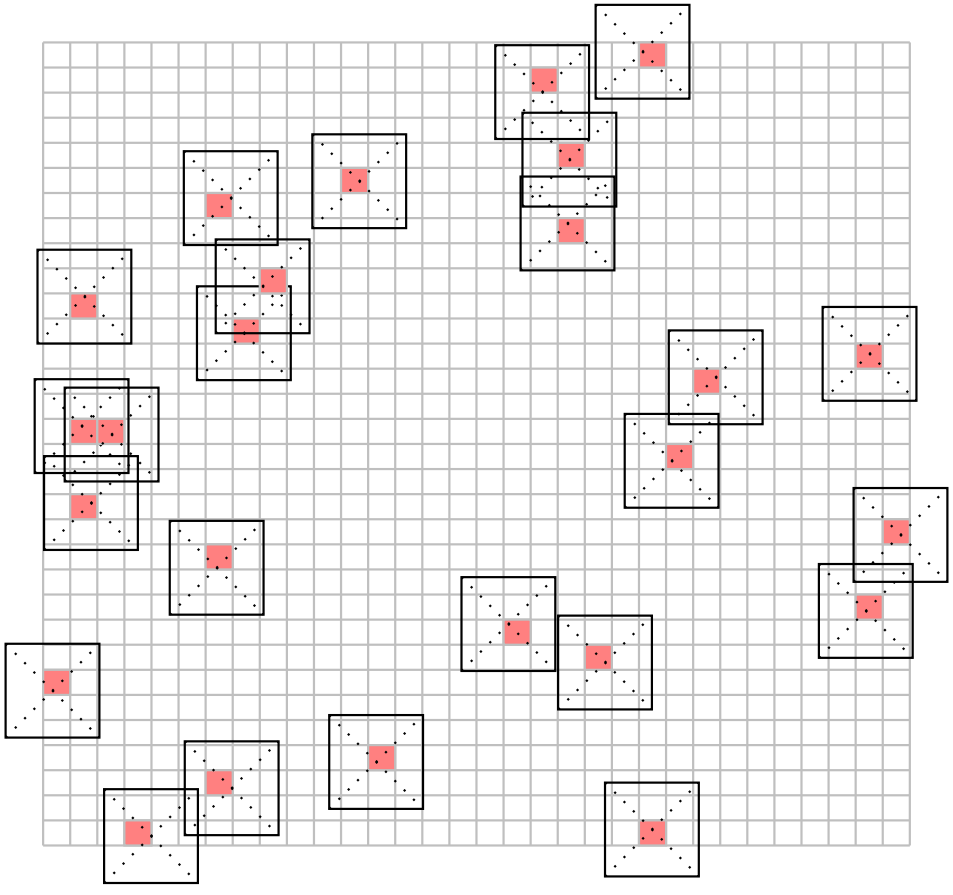
Hilbertův statický R-strom

- Algoritmus funguje stejně jako naivní algoritmus, ale místo jedné souřadnice používá H-hodnotu (tj. pozici na Hilbertově křivce).

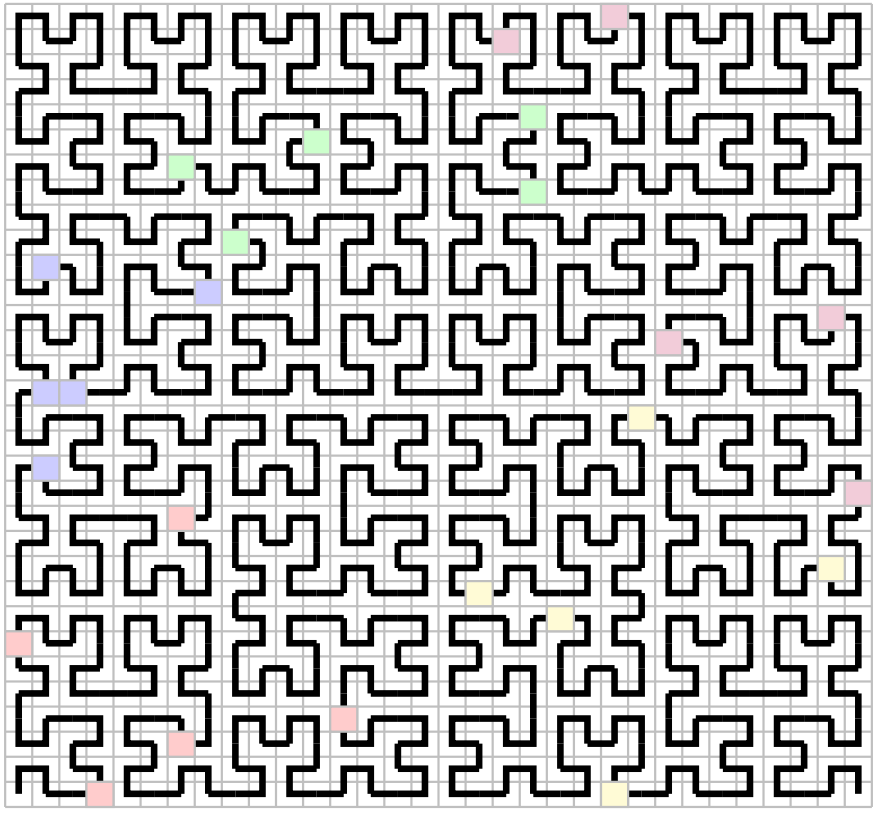
I. Kamel, C. Faloutsos
On Packing R-trees
2nd ACM International Conference on Information and Knowledge Management (CIKM'93), pp.490-499, Washington, DC, 1993.

Příklad:

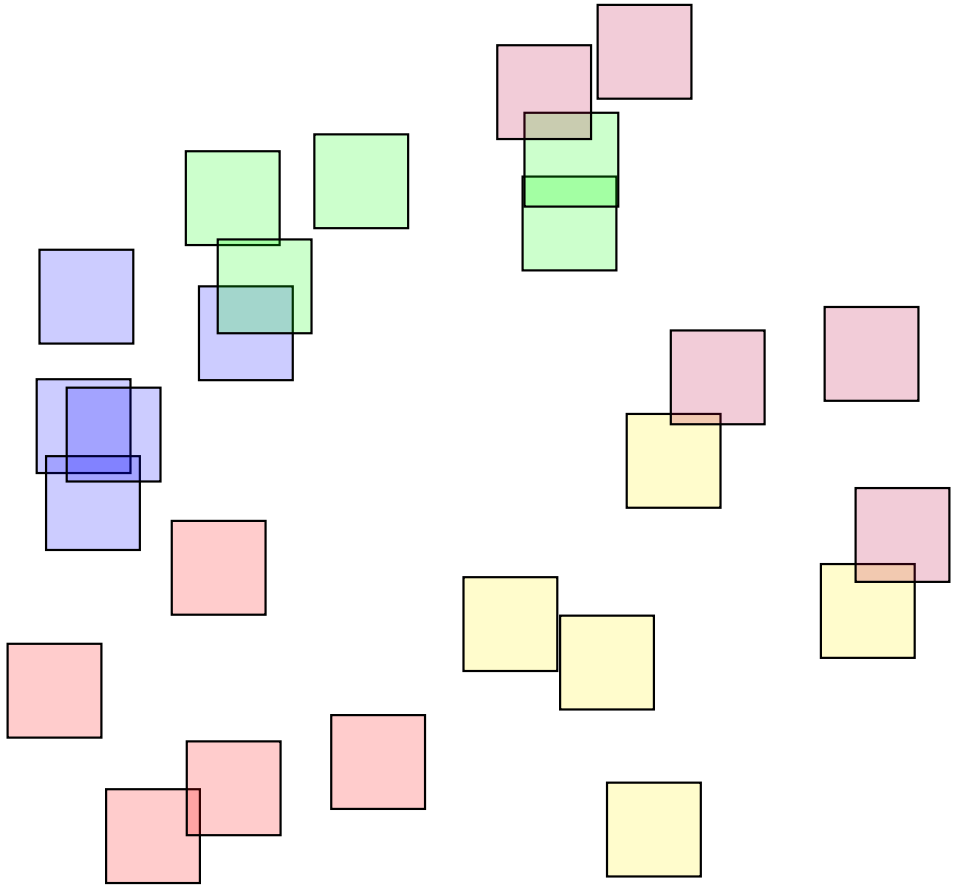
- Stejné zadání jako v prvním příkladě.
- Krok 1: Seřazení podle H-hodnot:
  Oblast proložíme dostatečně jemnou $2^n \times 2^n$ mřížkou; zde $n = 5$.
  Namapujeme těžiště obdélníků do čtverců mřížky.



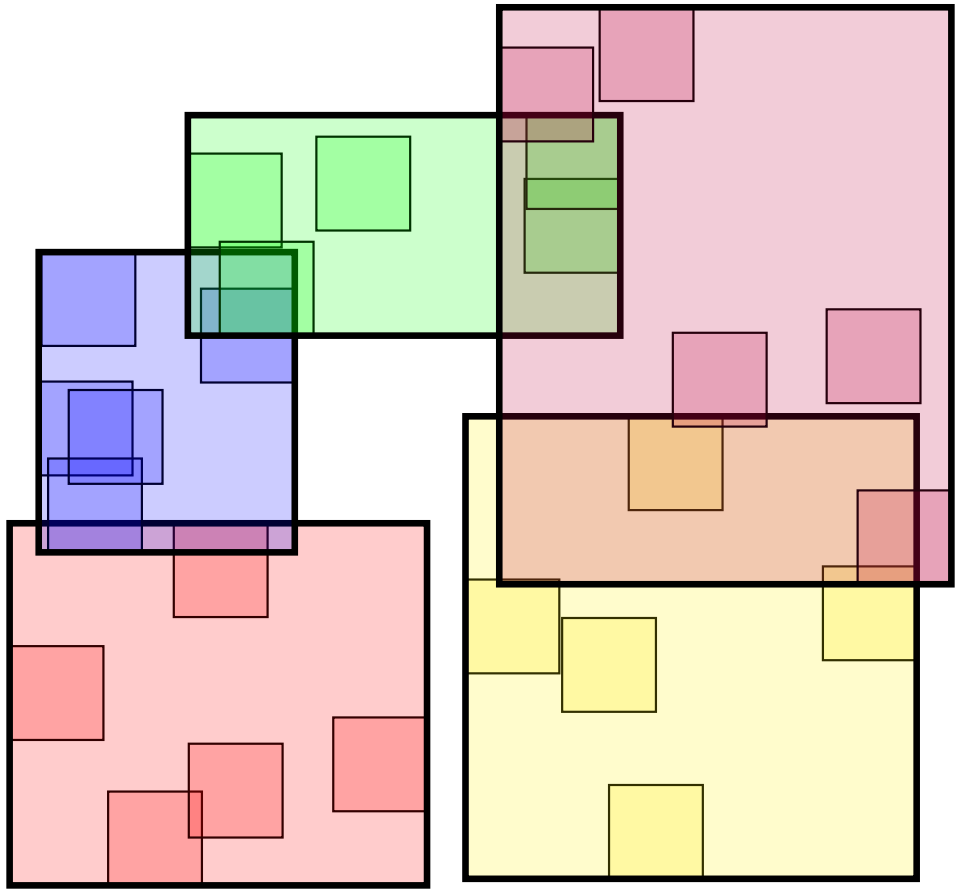
Krok 2: Obdélníky rozdělíme podle jejich H-hodnoty, tj. pozici na Hilbertově křivce (původní obdélníky opět skryty kvůli přehlednosti, zobrazeny jsou jen čtverce jejich těžišť).



naopak: skryty těžiště, mřížka a Hilbertova křivka; zobrazeny obdélníky



Nalezení MBR jednotlivých skupin.



Máme 5 obdélníků, můžeme ukončit konstrukci R-stromu.
Kdybychom dostali více obdélníků, pokračujeme rekurzivně pro tyto nové obdélníky.

STR R-strom

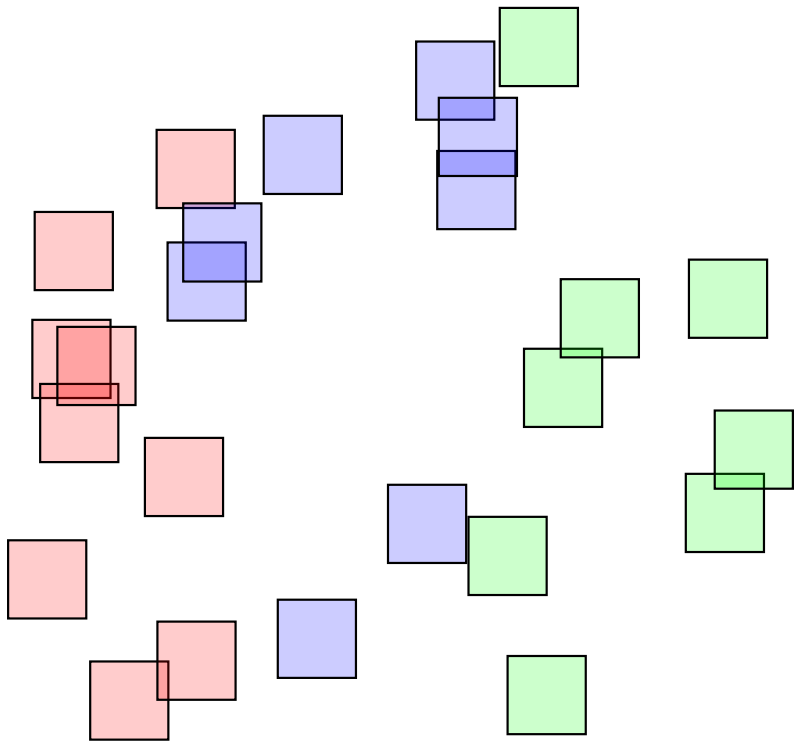
STR (Sort-Tile-Recursive, popsán pouze pro 2D)

- Nejdřív se stanoví počet listových uzlů, který je $n_l = \lceil N/c \rceil$.
- Nechť $VS = \sqrt{n_l}$.
- Obdélníky jsou seřazeny podle x-souřadnice jejich těžiště, a je vytvořeno VS pruhů.
- V každém pruhu jsou objekty seřazeny dle y-souřadnice těžiště a rozděleny do uzlů (c do každého uzlu).

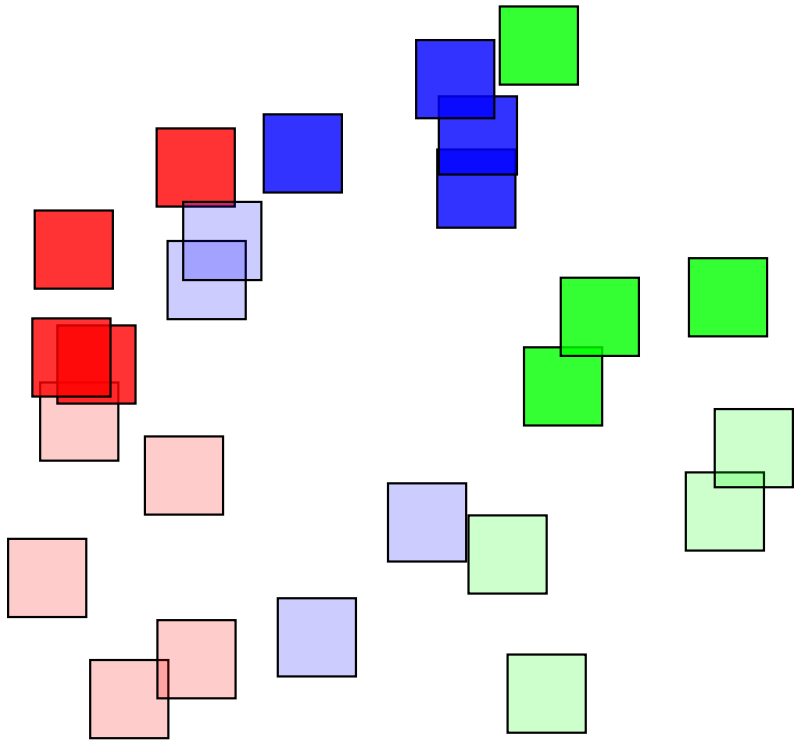
S.T. Leutenegger, M.A. Lopez, J. Edgington.
STR: A simple and efficient algorithm for R-tree packing.
13th international conference on data engineering. IEEE, 1997.

Příklad:

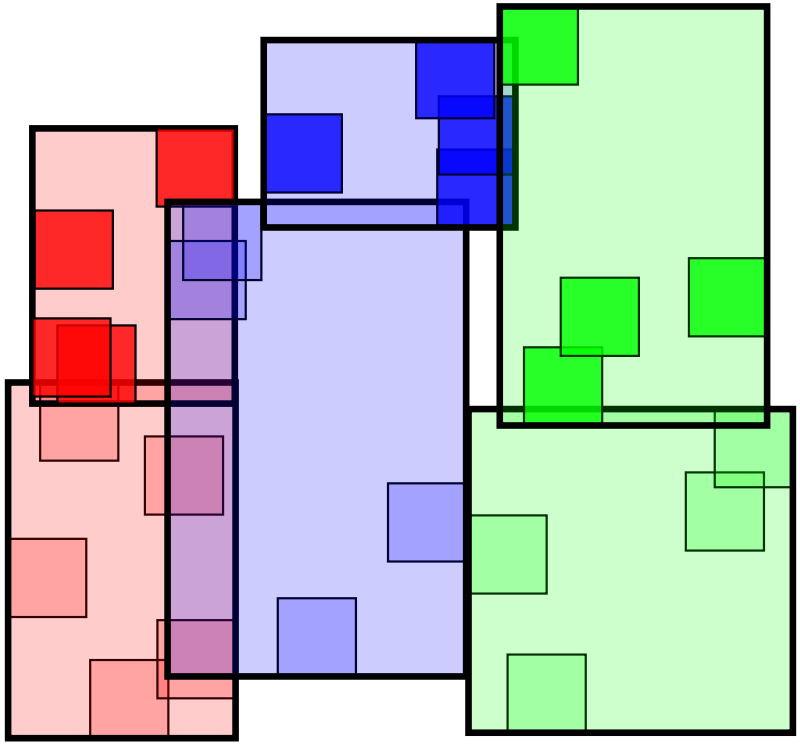
- Zadání stejné jako v předchozích příkladech (tento příklad nebude vycházet úplně hezky kvůli nízkým hodnotám parametrů).
- Máme $n_l = 5,\ VS = \sqrt{5} \approx 2.236$, rozdělme tedy obdélníky rovnoměrně (9,8,8) do tří pruhů dle x-souřadnice těžiště.



Následně každý z těch tří pruhů rozdělíme rovnoměrně na dvě množiny:



Najdeme MBR vzniknuvších množin.

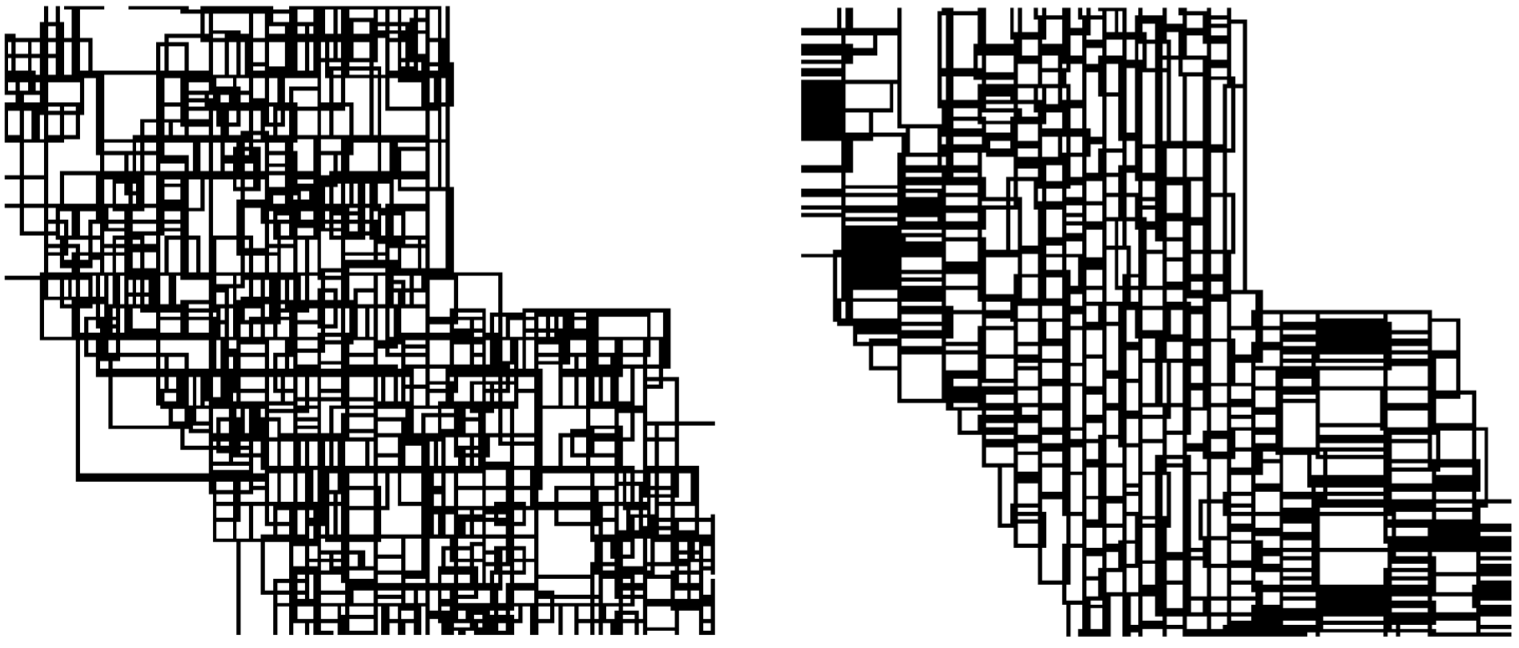


- Dostali jsme 6 množin (více než c)
- Měli bychom rekurzivně pokračovat pro tyto nalezené obdélníky.

Všechny tři algoritmy popsané výše se dají zobecnit na vyšší počet dimenzí:

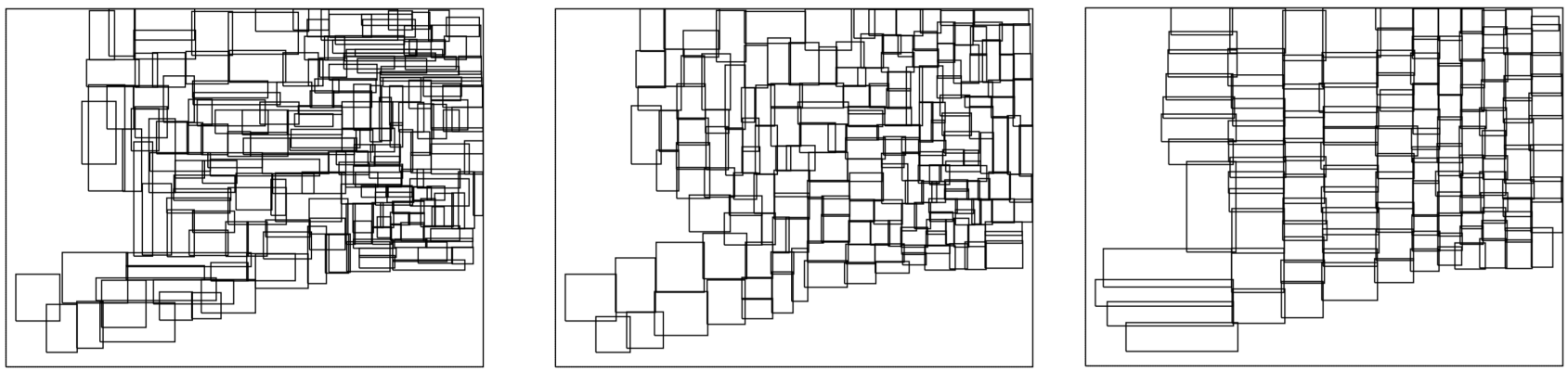
- U naivního algoritmu je to triviální.
- U Hilbertova statického R-stromu potřebujeme rozšíření Hilbertovy křivky na více dimenzí – není zrovna jednoduché.
- U STR je to snadné: pro $d > 2$ dimenzí, nejdříve rozdělí (nad-)obdélníky na $VS = n_l^{\frac{1}{d}}$ skupin podle x-souřadnice, kde $n_l = \lceil N/c \rceil$. Každá skupina je následně zpracována rekurzivně pro zbývajících $d - 1$ dimenzí.

Grafická ukázka výsledku algoritmu Hilbertova statického R-stromu (vlevo) a STR (vpravo); pouze úroveň listů:



Vidíme, že v STR je méně překryvů MBR.

Podobný závěr můžeme udělat v následujícím obrázku o R-stromu (vlevo), R*-stromu (uprostřed) a STR (vpravo):



Předchozí algoritmy fungovaly směrem zdola nahoru.
Teď se podíváme na jeden algoritmus, který funguje směrem shora dolu:

VAMsplit R-tree

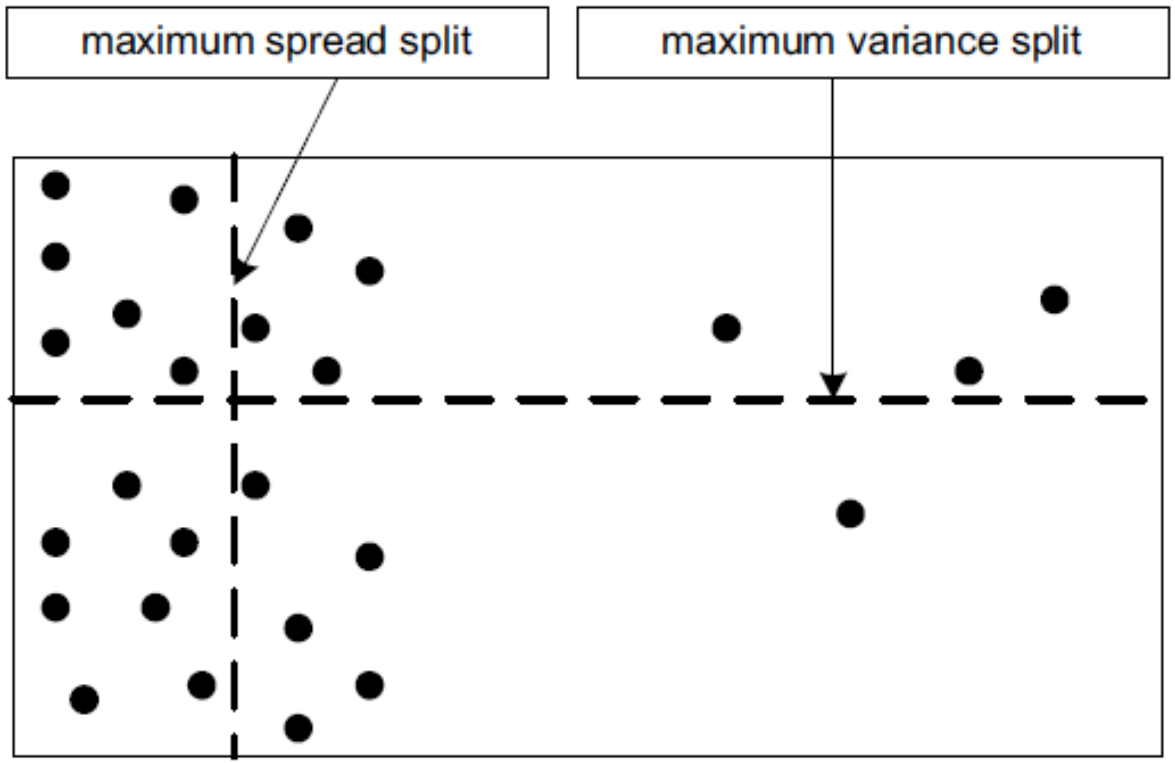
- v podstatě varianta k-d stromu (resp. VAM k-d stromu)
- v každém kroku se vybere *dimenze splitu* podle největší variance;
  a *pozice splitu*:

$$s_l = s_p - s_h = \begin{cases} \left\lfloor \frac{s_p}{2} \right\rfloor & \text{pokud } s_p \leq 2b. \\ b \cdot \left\lfloor \frac{s_p}{2b} \right\rfloor & \text{jinak,} \end{cases}$$

kde

- b je velikost bucketu,
- s_p počet prvku pro daný podstrom
- s_l a s_h jsou počty prvků v levém a pravém podstromu

D.A. White, R. Jain
Similarity Indexing: Algorithms and Performance
Proceedings 4th International Conference on Storage and Retrieval for Image and Video Databases (SPIE'96), pp.62-73, San Diego, CA, 1996.


maximum spread split
maximum variance split

Zpracování prostorových dotazů

- jeden z hlavních důvodů popularity R-stromových indexů z jejich všestrannosti, protože mohou efektivně podporovat mnoho typů prostorových operátorů.

Nejběžnější z nich:

Topologické dotazy
najdi všechny objekty, které se např. překrývají s daným objektem nebo ho pokrývají.

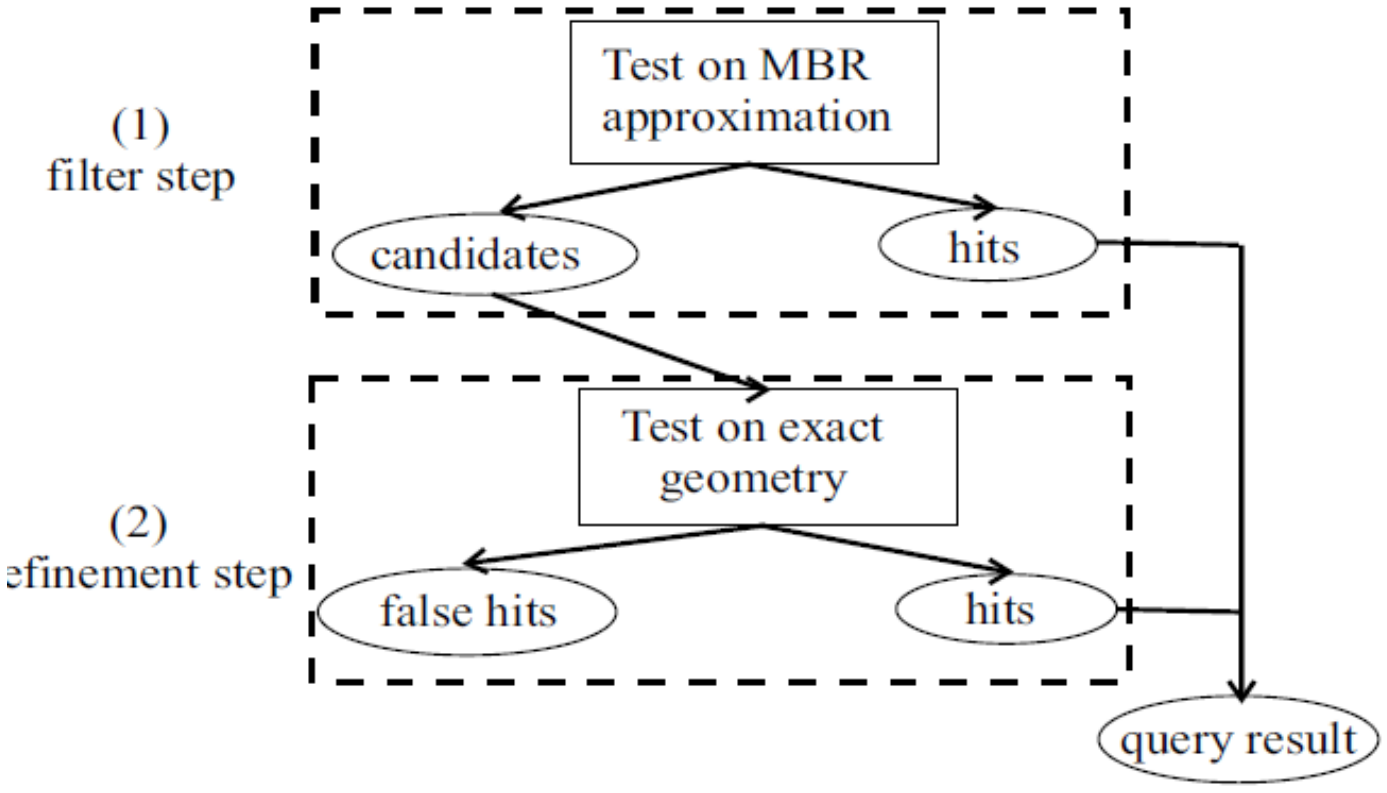
Směrové dotazy
najdi všechny objekty, které se např., leží severně od daného objektu.

Vzdálenostní dotazy
najdi všechny objekty, v menší než v dané vzdálenosti od daného objektu;
najdi k objektů nejblíže k danému objektu.

R-tree abstrahuje objekty složitých tvarů, tím, že používá jejich MBR aproximace.

Aby odpověděl na ty dotazy (předpředchozí slajd), provede tuto dvou-krokovou proceduru:

- filtrovací krok – najde kolekci objektů, jejichž MBR splňují daný dotaz – množina kandidátů
- finalizační krok – řešíme skutečnou geometrii každého kandidáta, a odstraníme ty, které nevyhovují zadané geometrii.


(1)
filter step
Test on MBR
approximation
candidates
hits
(2)
efinement step
Test on exact
geometry
false hits
hits
query result

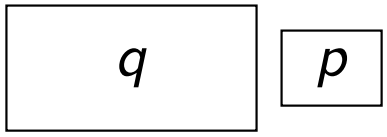
Rozsahové a topologické operátory

- Nejběžnější operací s indexem R-tree je dotaz na rozsah (range query), který najde všechny objekty, které oblast dotazu protíná.
- Algoritmus uveden v

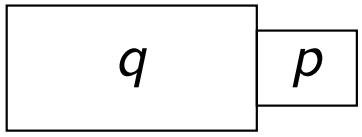
  

  A. Guttman,
  R-Trees: A dynamic index structure for spatial searching,
  in Proc. of ACM SIGMOD, June 1984, pp. 47–57

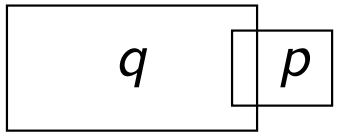
  a my jsme ho tu už popsali.
- Point query – vlastně totéž, bod je degenerovaný obdélník.



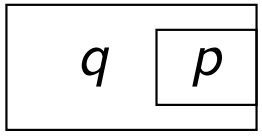
disjoint(q,p)



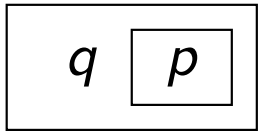
meet(q,p)



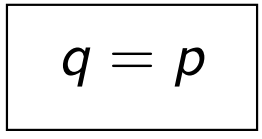
overlap(q,p)



covers(q,p)

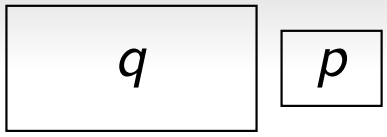


inside(q,p)

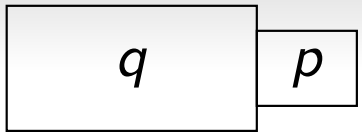


equal(q,p)

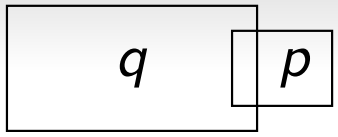
Topologický vztah mezi dvěma MBR se nemusí nutně shodovat s topologickým vztahem mezi odpovídajícími objekty, protože MBR jsou aproximace.



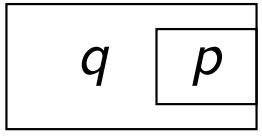
disjoint(q,p)



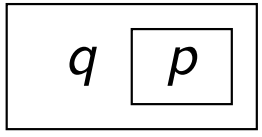
meet(q,p)



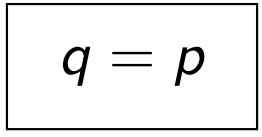
overlap(q,p)



covers(q,p)



inside(q,p)



equal(q,p)

Příklad 1

Abychom našli:

- objekty, které pokrývají (*cover*) objekt q, musíme projít všechny objekty, jejichž MBR splňují *cover*, *contains*, *equal* vzhledem k MBR objektu q.
- objekty, které jsou stejné (*equal*), stačí projít všechny objekty, jejichž MBR je roven (*equal*) MBR objektu q.

Pro topologický dotaz potřebujeme uvažovat obecnější relační dotazy nad R-stromem.

Označme:

- $RN.mbr$ – MBR celého uzlu
- $e.mbr$ – MBR záznamu e v uzlu
- q – referenční objekt, a $q.mbr$ je jeho MBR.

Possible relation	Relations to be tested
equal($e.mbr$,$q.mbr$)	equal($RN.mbr$,$q.mbr$) $\vee$ covers($RN.mbr$,$q.mbr$) $\vee$ contains($RN.mbr$,$q.mbr$)
contains($e.mbr$,$q.mbr$)	contains($RN.mbr$,$q.mbr$)
inside($e.mbr$,$q.mbr$)	overlap($RN.mbr$,$q.mbr$) $\vee$ covers($q.mbr$,$RN.mbr$) $\vee$ inside($RN.mbr$,$q.mbr$) $\vee$ equal($RN.mbr$,$q.mbr$) $\vee$ covers($RN.mbr$,$q.mbr$) $\vee$ contains($RN.mbr$,$q.mbr$)
covers($e.mbr$,$q.mbr$)	covers($RN.mbr$,$q.mbr$) $\vee$ contains($RN.mbr$,$q.mbr$)
covers($q.mbr$,$e.mbr$)	overlap($RN.mbr$,$q.mbr$) $\vee$ covers($q.mbr$,$RN.mbr$) $\vee$ equal($RN.mbr$,$q.mbr$) $\vee$ covers($RN.mbr$,$q.mbr$) $\vee$ contains($RN.mbr$,$q.mbr$)
disjoint($e.mbr$,$q.mbr$)	disjoint($RN.mbr$,$q.mbr$) $\vee$ meet($RN.mbr$,$q.mbr$) $\vee$ overlap($RN.mbr$,$q.mbr$) $\vee$ covers($RN.mbr$,$q.mbr$) $\vee$ contains($RN.mbr$,$q.mbr$)
meet($e.mbr$,$q.mbr$)	meet($RN.mbr$,$q.mbr$) $\vee$ overlap($RN.mbr$,$q.mbr$) $\vee$ covers($RN.mbr$,$q.mbr$) $\vee$ contains($RN.mbr$,$q.mbr$)
overlap($e.mbr$,$q.mbr$)	overlap($RN.mbr$,$q.mbr$) $\vee$ covers($RN.mbr$,$q.mbr$) $\vee$ contains($RN.mbr$,$q.mbr$)

Dotaz na nejbližšího souseda (Nearest-neighbor query)

První algoritmus `branch-and-bound` představen v:

N. Roussopoulos, S. Kelley, F. Vincent
Nearest Neighbor Queries,
Proceedings ACM SIGMOD Conference on Management of Data, pp.71-79, San Jose, CA, 1995.

Budeme uvažovat jen Euklidovskou vzdálenost.

Potřebujeme uvést následující metriky:

Definice 2 (MINDIST)

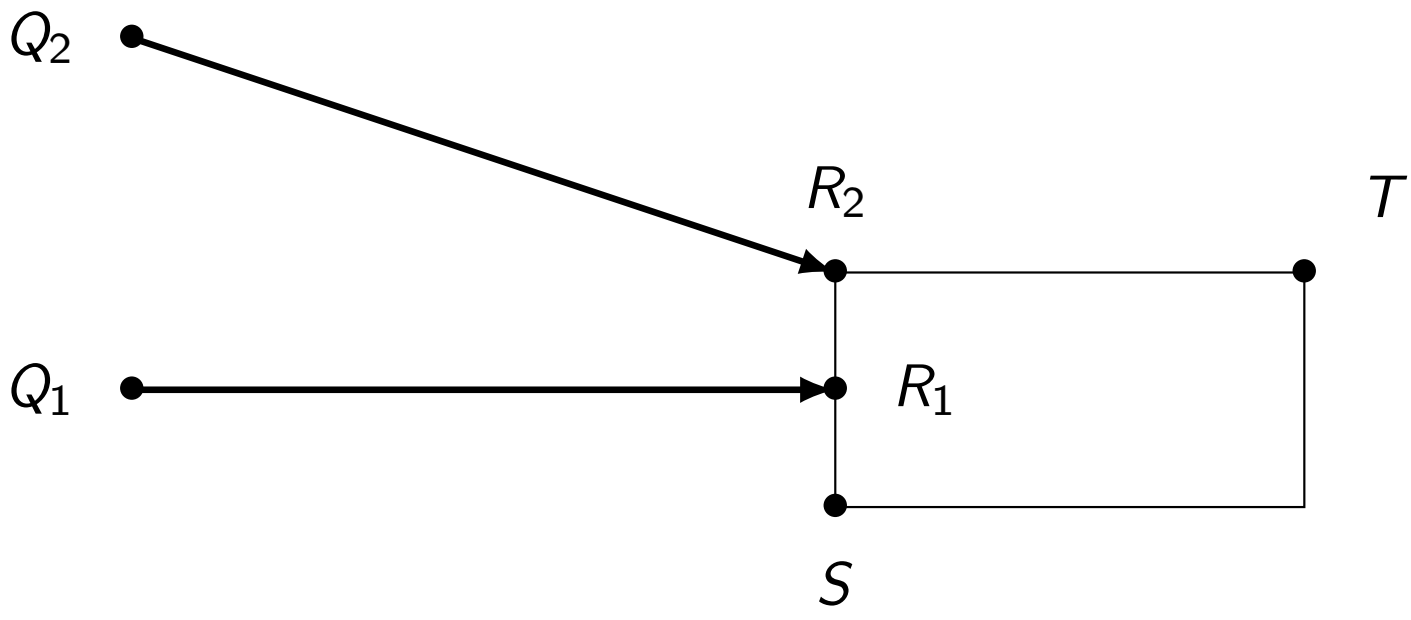
Je-li dán bod Q a obdélník $R = (S, T)$, minimální vzdálenost, označovaná jako $\mathrm{MINDIST}(Q, R)$ je definována jako:

$$\mathrm{MINDIST}(Q, R) = \sum_{i=1}^{d} |q_i - r_i|^2$$

kde

$$r_i = \begin{cases} s_i & \text{pokud } q_i < s_i \\ t_i & \text{pokud } q_i > t_i \\ q_i & \text{jinak.} \end{cases}$$

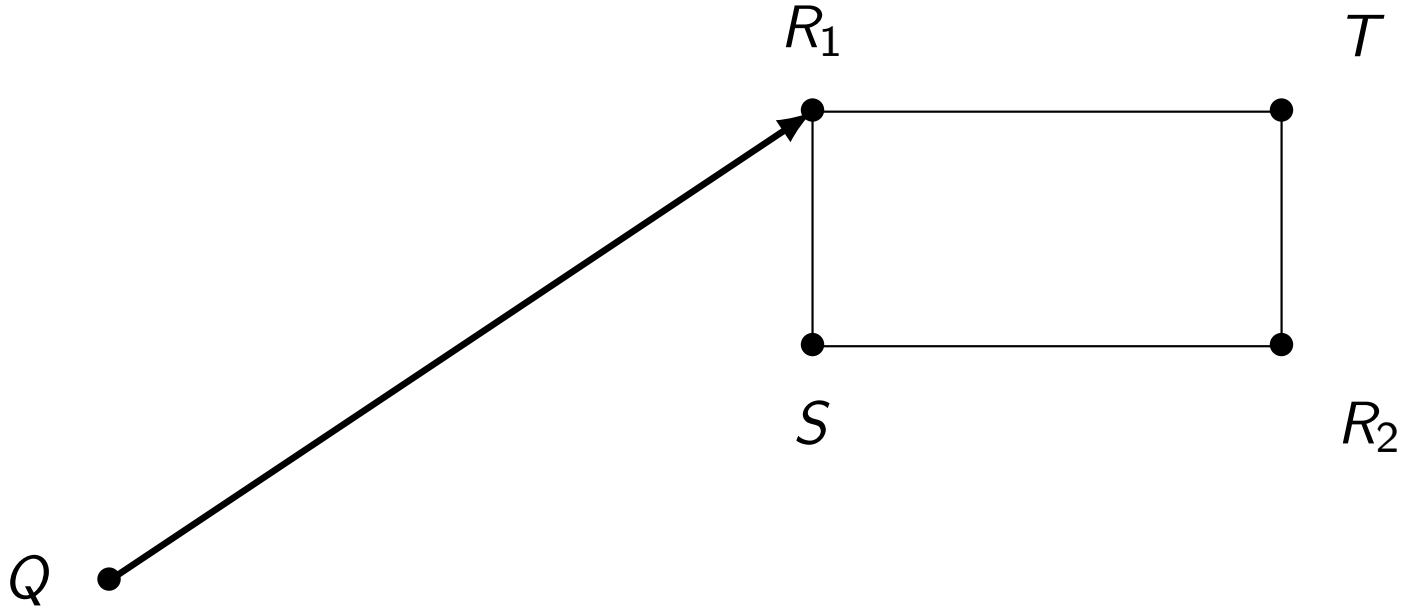
$\mathrm{MINDIST(Q, R)}$ je tedy vzdálenost od Q k nejbližšímu bodu obdélníku R:



$\mathrm{MINDIST(Q, R)}$ představuje optimistický odhad vzdálenosti nejbližších dat v obdélníku R od bodu Q – v optimistickém případě jsou nejbližší data v obdélníku v nejbližším bodě obdélníku.

Co pesimistický případ?

Co pesimistický případ?
$\mathrm{MINMAXDIST}(Q, R)$ je vzdálenost k nejbližšímu rohu, který je sousedem nejvzdálenějšího rohu.



V tomto obrázku je od Q nejvzdálenějším bodem obdélníka bod T, jeho dva sousedi jsou R_1 a R_2. Z nich R_1 je bližší.

Definice 3 (MINMAXDIST)

Je-li dán bod Q a obdélník $R = (S, T)$, min-maximální vzdálenost, označovaná jako $\mathrm{MINMAXDIST}(Q, R)$ je definována jako:

$$\mathrm{MINMAXDIST}(Q, R) = \min_{1 \leq k \leq = d} (|q_k - rm_k|^2 + \sum_{\substack{i \neq k \\ 1 \leq i \leq d}} |q_i - rMi|^2$$

kde

$$rm_k = \begin{cases} s_k & \text{pokud } q_k \leq \frac{s_k + t_k}{2}, \\ t_k & \text{jinak,} \end{cases}$$

a

$$rM_i = \begin{cases} s_i & \text{pokud } q_i \geq \frac{s_i + t_i}{2}, \\ t_i & \text{jinak.} \end{cases}$$

- $\mathrm{MINMAXDIST}(Q, R)$ představuje pesimistický odhad vzdálenosti nejbližších objektů v obdélníku R od bodu Q.
- Pozor, nejde o nejvzdálenější bod od obdélníka R, protože ten je MBR nějakých obdélníků či objektů:

Musí existovat obdélníky či objekty, které leží u dvou ke Q nejbližších stran obdélníka R, protože ten je těsně obepíná. Pesimistický odhad je, že tyto objekty jsou v těch krajních bodech těchto stran, které jsou dále od Q. Bližší z nich tedy představuje pesimistický odhad ke Q nejbližších objektů v R.

Algoritmus `Branch-and-Bound`

- výše uvedené metriky k uspořádání a prořezání vyhledávacího stromu.
- Pořadí hledání určuje návštěvy uzlů během průchodu stromu:
  - $\mathrm{MINDIST}$ produkuje optimističtější řazení než $\mathrm{MINMAXDIST}$, ale mohou existovat případy dat (v závislosti na velikostech a rozložení MBR), kde
  - $\mathrm{MINMAXDIST}$ produkuje méně nákladné procházení.

Pro bod Q je prořezávání návštěv uzlů během vyhledávání se provádí podle následujících heuristik:

H1: MBR M s $\mathrm{MINDIST}(Q, M)$ větší, než $\mathrm{MINMAXDIST}(Q, M')$ jiného MBR M', je zahozen, protože nemůže obsahovat nejbližšího souseda.
Toto je používáno v *prořezávání směrem dolů*.

H2: aktuální vzdálenost Q k danému objektu O, který je větší než vzdálenost $\mathrm{MINMAXDIST}(Q, M)$ pro MBR M, může být zahozena, protože M obsahuje objekt O', který je ke Q bližší.
Toto je také používáno v *prořezávání směrem dolů*.

H3: každý MBR M s $\mathrm{MINDIST}(Q, M)$ větší, než aktuální vzdálenost od Q k danému objektu O je zahozena, protože nemůže obsahovat objekt bližší než O.
Toto je používáno v *prořezávání směrem nahoru*.

```
def NNSearch(Node, Point, Nearest):
    input : Node – actual node
            Point – query point
            Nearest – the nearest object

1   if Node is a leaf then
2       for i ← 1 to Node.count do
3           dist ← objectDIST(Point, Node)
4           if dist < Nearest.dist then
5               Nearest.dist ← dist
6               Nearest.rect ← Node.branch[i].rect
7   else
8       genBranchList(branchList)
9       sortBranchList(branchList)
10      last ← pruneBranchList(Node, Point, Nearest, branchList)
11      for i ← 1 to last do
12          newNode ← Node.branch[branchList[i]]
13          NNSearch(newNode, Point, Nearest)
14          last ← pruneBranchList(Node, Point, Nearest, branchList)
```

`Branch-and-Bound`

- začíná to od kořenového uzlu a inicializuje proměnnou *Nearest* na nekonečno (tento krok není v pseudokódu zobrazen).
- V nelistovém uzlu najde $\mathrm{MINDIST}$ (lze použít i $\mathrm{MINMAXDIST}$) pro MBR potomků uzlu (krok 8) a seřadí je spolu s odpovídajícími ukazateli na podřízené uzly, do seznamu aktivních větví (krok 9).
- Potom se provede prořezávání typů H1 a H2 (krok 10).
- Pro každý prvek aktuálního seznamu větví je tento postup aplikován rekurzivně (krok 13).
- Na úrovni listů (kroky 1-6) je použita funkce skutečné vzdálenosti (krok 3) a je aktualizována proměnná *Nearest* (kroky 4-7).
- Nakonec je použito prořezávání směrem nahoru typu H3 (krok 14).

K.L. Cheung, A. Fu
Enhanced Nearest Neighbor Search on the R-tree
ACM SIGMOD Record, Vol.27, No.3, pp.16-21, 1998.

`Branch-and-Bound` být zefektivněn (vzhledem k počtu navštívených uzlů) takto:

- budeme používat jenom H3, a budeme ho aplikovat na trochu jiném místě v kódu.
- přesněji, v pseudokódu `Branch-and-Bound` vynecháme krok 10 (to byla aplikace ořezávání H1 a H2) a for cyklus v krocích 11-14 nahradíme takto:

---

for $i \leftarrow 1$ **to** *last* **do**
    Apply H3
    $newNode \leftarrow Node.branch[branchList[i]]$
    `NNSearch(`*newNode, Point, Nearest*`)`

Motivace pro tu modifikaci:

- Heuristika H2 zahodila objekt O pokud je jeho aktuální vzdálenost od bodu Q větší než $\mathrm{MINMAXDIST}$ bodu Q od jiného MBR.
- Evidentně, ten uzel, který obsahuje O nemůže být v tomto okamžiku ořezán, protože už byl přečten; takže heuristika H2 neredukuje počet návštěv uzlů.
- Navíc se dá ukázat, že cokoli by bylo ořezáno pomocí H1, bude taky ořezáno pomocí H3, když to bude použito na tom novém místě (navíc H1 je použitelná jenom, když hledáme jen jednoho nejbližšího souseda).
- Dále, pokud je zbavíme H1 a H2, vyhneme se tím úplně výpočtu $\mathrm{MINMAXDIST}$ a dostaneme algoritmus, který je závislý pouze na $\mathrm{MINDIST}$, který se počítá snadněji.

Inkrementální NN Algoritmus

- postupně vrací objekty podle jejich vzdálenosti od Q
- Např., Máme města C, a dotaz: „najdi město v C nejbližší ke Q, aby jeho populace byla větší než 10M.“
- že dostáváme sousedy postupně, můžou být testnuty vzhledem ke specifikovaným kritériím – tomu se říká distance browsing
- Je to něco jiného než hledání k nejbližších sousedů pro předem dané k, protože k tady neznáme předem.

G. Hjaltason, H. Samet
Distance Browsing in Spatial Databases,
ACM Transactions on Database Systems, Vol.24, No.2, pp.265-318, 1999.

- Algoritmus udržuje sadu uzlů, které mají být navštíveny, v prioritní frontě.
- Položky ve frontě jsou seřazeny podle metriky MINDIST.
- Předpokládá se, že skutečné objekty (např. polygony) jsou uloženy odděleně na úrovni dat a každý objekt zcela obsažen ve svém odpovídajícím ohraničujícím obdélníku.

```
def IncNNSearch(Q):
    input : Q – query point
1   enqueue(PriorityQueue, root's children)
2   while PriorityQueue not empty do
3       element ← dequeue(PriorityQueue)
4       if element is an object O or an object bounding rectangle O.mbr then
5           if element is O.mbr and PriorityQueue not empty and Dist(q,O) >
              first(PriorityQueue).key then
6               enqueue(PriorityQueue, O, objDist(q,O))
7           Output element
8       else if element is a leaf node then
9           for each object bounding rectangle O.mbr in element do
10              enqueue(PriorityQueue, O.mbr, dist(q,O.mbr))
11      else
12          for each entry e in element do
13              enqueue(PriorityQueue, e, dist(q,O.mbr))
```