Hledání nejbližších sousedů v R-stromech

V předchozích lekcích jsme v R-stromech měli vyhledávání, ve kterém vstupem bylo vyhledávací okénko, a algoritmus měl najít všechny objekty, které měla s tímto okénkem neprázdný průnik. Nyní se zaměříme na jiný druh vyhledávání – vyhledávání nejbližších sousedů. Tj. vstupem bude bod Q a nalezeny mají být objekty, které jsou k němu nejblíže.

Budeme teď uvažovat jen Euklidovskou vzdálenost.

Potřebujeme uvést následující metriky:

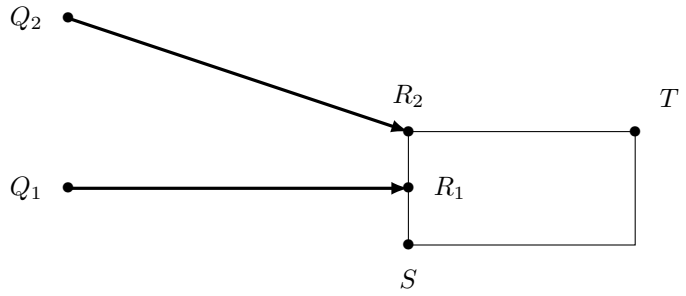
Definition 1 (MINDIST). *Je-li dán bod Q a obdélník $R = (S, T)$, minimální vzdálenost, označovaná jako* MINDIST(Q, R) *je definována jako:*

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

kde

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

MINDIST(Q, R) je tedy vzdálenost od Q k nejbližšímu bodu obdélníku R.



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.

Definition 2 (MINMAXDIST). *Je-li dán bod Q a obdélník $R = (S, T)$, min-maximální vzdálenost, označovaná jako* MINMAXDIST(Q, R) *je definována jako:*

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

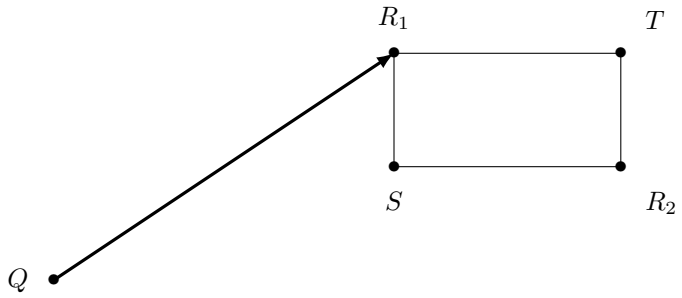
kde

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

a

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

Tato poněkud komplikovaná definice říká, že MINMAXDIST(Q, R) je vzdálenost k nejbližšímu rohu, který je sousedem nejvzdálenějšího rohu (viz obrázek).



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.

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

Algoritmus Branch-and-Bound výše uvedené metriky k uspořádání a prořezání vyhledávacího stromu. Pořadí hledání určuje návštěvy uzlů během průchodu stromu. MINDIST produkuje optimističtější řazení než MINMAXDIST, ale mohou existovat případy dat (v závislosti na velikostech a rozložení MBR), kde MINMAXDIST produkuje méně nákladné procházení. Pro bod Q je prořezávání návštěv uzlů během vyhledávání se provádí podle následujících heuristik:

H1: MBR M s MINDIST(Q, M) větší, než 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 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 MINDIST(Q, M) větší, než aktuální vzdálenost od Q k danému objektu Q je zahozena, protože nemůže obsahovat objekt bližší než O. Toto je používáno v *prořezávání směrem nahoru*.

Pseudokód algoritmu pro nalezení jednoho nejbližšího souseda je uveden v Algoritmu 1. Algoritmus začíná to od kořenového uzlu a inicializuje proměnnou *Nearest* na nekonečné (tento krok není v pseudokódu zobrazen). V nelistovém uzlu procedura najde MINDIST (lze použít i MINMAXDIST) pro MBR potomků uzlu (krok 10) a seřadí je spolu s odpovídajícími ukazateli na podřízené uzly, do seznamu aktivních větví (krok 11). Potom se provede prořezávání typů H1 a H2 (krok 12). Pro každý prvek aktuálního seznamu větví je tento postup aplikován rekurzivně (krok 15). Na úrovni listů (kroky 1-8) je použita funkce skutečné vzdálenosti (krok 3) a je aktualizována proměnná *Nearest* (kroky 4-7). Nakonec je použito prořezávání směrem nahoru typu H3 (krok 16).

```
   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  |   |   |   end
8  |   |   end
9  |   else
10 |   |   genBranchList(branchList)
11 |   |   sortBranchList(branchList)
12 |   |   last ← pruneBranchList(Node, Point, Nearest, branchList)
13 |   |   for i ← 1 to last do
14 |   |   |   newNode ← Node.branch[branchList[i]]
15 |   |   |   NNSearch(newNode, Point, Nearest)
16 |   |   |   last ← pruneBranchList(Node, Point, Nearest, branchList)
17 |   |   end
18 |   end
```

Algorithm 1: Hledání nejbližšího souseda

Pro nalezení k-nejbližších sousedů ($k > 1$) lze použít předchozí postup lze snadno upravit tak, aby udržovaly aktuálních k nejbližších objektů (spolu s odpovídajícími vzdálenostmi) a prořezáváním vzhledem k nejvzdálenějšímu objektu.

Vylepšení původního algoritmu

Ten algoritmus výše může být zefektivněn (vzhledem k počtu navštívených uzlů) takto: budeme používat jenom H3, a budeme ho aplikovat na trochu jiném místě v kódu. Přesněji, v Algoritmu 1 vynecháme krok 12 (to byla aplikace ořezávání H1 a H2) a for cyklus v krocích 13-17 nahradíme takto:

```
for i ← 1 to last do
    Apply H3
    newNode ← Node.branch[branchList[i]]
    NNSearch(newNode, Point, Nearest)
end
```

Motivace pro tu modifikaci je následující. Heuristika H2 zahodila objekt O pokud je jeho aktuální vzdálenost od bodu Q větší než 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 MINMAXDIST a dostaneme algoritmus, který je závislý pouze na MINDIST, který se počítá snadněji.

Domácí úkoly

- Proveďte experimentální srovnání těchto dvou algoritmů.
- Proveďte experimentální srovnání Hilberovových a STR statických R-stromů (metriku nechámvám na vás: rychlost vyhledávání, množství překryvů, počet navštívených uzlů při náhodném vyhledávání, počet uzlů...)