Algoritmy pro rozsáhlá data

L08: metrické stromy

Jan Konecny

10. prosince 2021

Metrické stromy

- Metrickým stromem rozumíme jakoukoli stromovou datovou strukturu specializovanou na indexování dat v metrických prostorech.
- Metrické stromy využívají vlastnosti metrických prostoru, jako trojúhelníková nerovnost efektivnějšímu přístupu k datům.
- Příklady: M-stromy, vp-stromy, MVP-stromy, a bk-stromy.

Metrika vzdálenosti $d(x, y)$ je definována následovně:

- $d(x, y) = d(y, x)$
- $0 < d(x, y) < \infty, \quad x \neq y$
- $d(x, x) = 0$
- $d(x, y) \leq d(x, z) + d(z, y) \qquad$ (trojúhelníková nerovnost)

Příklad 1

Euklidovská vzdálenost

$$d(x, y) = \sqrt{\sum_i (x_i - y_i)^2}$$

Hammingova vzdálenost

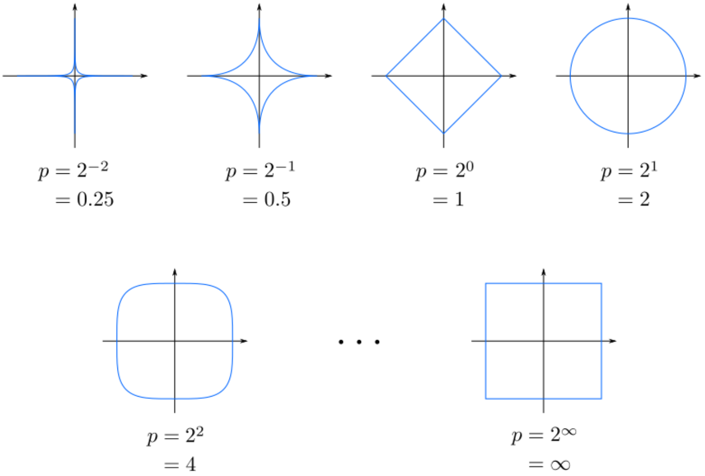
$$d(x, y) = \sum_i |x_i - y_i|$$

Čebyševova vzdálenost

$$d(x, y) = \max_i |x_i - y_i|$$

Minkowského vzdálenost

$$d(x, y) = \Big(\sum_i (x_i - y_i)^p\Big)^{\frac{1}{p}}$$



Příklad 2

Editační vzdálenost (Levenshtein distance)

$$\operatorname{lev}(a,b) = \begin{cases} |a| & \text{pokud } |b| = 0 \\ |b| & \text{pokud } |a| = 0 \\ \operatorname{lev}(\operatorname{tail}(a), \operatorname{tail}(b)) & \text{pokud } a_1 = b_1 \\ 1 + \min \begin{cases} \operatorname{lev}(\operatorname{tail}(a), b) \\ \operatorname{lev}(a, \operatorname{tail}(b)) \\ \operatorname{lev}(\operatorname{tail}(a), \operatorname{tail}(b)) \end{cases} & \text{jinak} \end{cases}$$

Near Neighbor Query: Z dané množiny datových objektů $X = \{X_1, X_2, \ldots, X_n\}$ z metrického prostoru s metrikou d(), získej všechny datové objekty, které jsou do vzdálenosti r od zadaného bodu Q.

Parametr r se obecně nazývá měření podobnosti nebo toleranční faktor.

Jsou možné i některé varianty near neighbor query:

- nearest neighbor query,
- k-nearest neighbor query
- farthest neighbor query,

vp-stromy (vantage point trees)

VP-stromy rozkládají množinu dat podle vzdáleností objektů vzhledem k referenčnímu bodu (vantage point).

Medián těchto vzdáleností je použit jako oddělovač k rokladu objektů do dvou vyvážených podmnožin, na které může být rekurzivně použita ta samá procedura.

J. Uhlmann
Satisfying General Proximity/Similarity Queries with Metric Trees.
Information Processing Letters. 40 (4): 175–179. (1991)

vp-strom: definice

Každý vnitřní uzel je ve tvaru $(Sv, M, L_{\mathrm{ptr}}, R_{\mathrm{ptr}})$, kde

- S_v je vantage point,
- M mediánová vzdálenost všech bodů (od S_v) indexovaných pod tím uzlem
- L_{ptr} a R_{ptr} jsou ukazatele na levý a pravý podstrom.
    - levý podstrom uzlu indexuje ty body, jejichž vzdálenost od S_v je menší nebo rovna M
    - pravý podstrom uzlu indexuje ty body, jejichž vzdálenost od S_v je větší než M.

V listových uzlech jsou reference na datové body místo ukazatelů na podstromy.

vp-strom: konstrukce (bulk-loading)

Je-li dána konečná množina $\mathcal{S} = \{S_1, S_2, \ldots, S_n\}$ objektů, a metrika $d(\cdot)$, binární vp-strom V na S je konstruován následovně:

1. Pokud $S = \emptyset$, vytvoř prázdný strom.
2. Jinak, nechť S_v je libovolný objekt z $\mathcal{S}$ (vantage point)
   $M = \mathrm{median}\{d(S_i, S_v) \mid \forall S_i \in \mathcal{S}\}$
   $\mathcal{S}_\mathrm{l} = \{S_i \mid d(S_i, S_\mathrm{v}) \leq M,$ kde $S_i \in \mathcal{S}$ a $S_i \neq S_v\}$
   $\mathcal{S}_\mathrm{r} = \{S_i \mid d(S_i, S_\mathrm{v}) > M,$ kde $S_i \in \mathcal{S}$ a $\}$
3. Rekurzivně vytvož vp-strom na $\mathcal{S}_\mathrm{l}$ a $\mathcal{S}_\mathrm{r}$ jako levý a pravý podstrom.

Poznámka

- Tento iterativní proces je podobný jako u k-d stromu, ale používá kruhové (nebo sférické, hyperkulové atd.) dělení místo přímočarého.
- Ve dvourozměrném euklidovském prostoru to lze vizualizovat jako sérii kruhů oddělujících data.

- Binární vp-strom je vyvážený, takže může být snadno stránkován pro uložení v sekundární paměti. Konstrukce vyžaduje $O(n \log_2(n))$ výpočtů vzdáleností.
- Pro daný objekt Q, množina datových objektů, které jsou ve vzdálenosti r od Q je nalezena vyhledávacím algoritmem:

1. Pokud $d(Q, S_v) \leq r$, pak S_v je ve výsledné množině,
2. Pokud $d(Q, S_v) + r \geq M$, prohledej pravý podstrom,
3. Pokud $d(Q, S_v) - r \leq M$, prohledej levý podstrom,

Poznámka

Všimněme si, že 2) a 3) mohou nastat současně a pak se prohledávají oba podstromy.

Poznámka: Zobecnění binární vp-stromů na m-ární vp-stromy

Binární vp-strom může být snadno zobecněn na m-ární stromovou strukturu.

- Konstrukce vp-stromu řádu m je velmi podobná konstrukci binárního vp-stromu.
- Namísto hledání mediánu vzdáleností mezi vantage pointem a datovými body, jsou body seřazeny a rozloženy do m skupin o stejné kardinalitě.
- Hodnoty vzdáleností použité pro ten rozklad jsou zaznamenány v uzlu.

MVP-stromy (multiple vantage point trees)

- MVP-stromy, rozkládají prostor do sférických řezů okolo vantage pointů (podobně jako vp-stromy), rozklady vytváří vzhledem k více než jednomu vantage pointu na každé úrovni a udržuje si informaci navíc v listech pro kvůli efektivnímu odfiltrování kandidátních bodů.
- MVP-strom používá dva vantage pointy v každém uzlu.
  Na každý uzel v MVP-stromu může být nahlíženo jako na dvě úrovně VP-stromu (rodič a jeho přímí potomci) kde všechny uzly v potomcích na nižší úrovni používají ten samý vantage point.
- To umožňuje mít více odfiltrovaných dat v každém uzlu při vyhledávání a menší počet vantage pointů v nelistových úrovních.

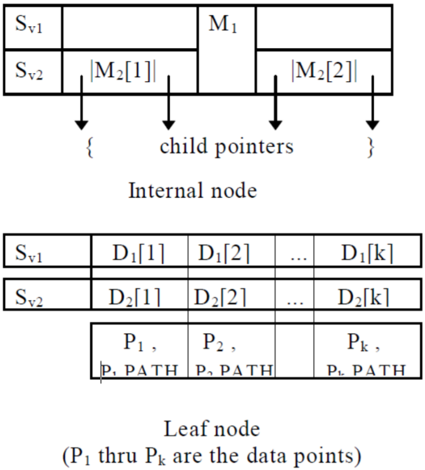
MVP-strom: definice

MVP-strom má tři parametry:

- počet tříd rozkladu vytvořený každým vantage pointem (m),
- maximum dat pro listové uzly (k),
- a počet vzdáleností pro datové body v listech (p).

- V binárních MVP-stromech rozděluje první vantage point S_{v1} prostor na dvě části, a druhý vantage point S_{v2} rozděluje každou z těchto částí na dvě.
- Máme tedy 4 potomky v binárním případě. Obecně počet potomků vnitřního uzlu je m^2,

- V každém vnitřním uzlu jsou udržovány mediány M_1, $M_2[1]$, $M_2[2]$ pro rozklad vzhledem k S_{v1} a S_{v2}.
- V listových uzlech uchováváme přesné vzdálenosti mezi datovými body a vantage pointy toho listu.
- $D_1[i]$ a $D_2[i]$ $(i = 1, 2, \ldots, k)$ jsou vzdálenosti z prvního a druhého vantage pointu, a k je počet dat v listech (může být zvoleno vyšší než m^2).
- Pro každý datový bod x v listech uchovává pole x.PATH[p] vypočtených vzdálenosti mezi datovým bodem x a prvními p vantage pointy na cestě od kořene do listového uzlu.



Struktura vnitřního uzlu a listových uzlů
v binárním MVP-stromu.

MVP-strom konstrukce

Je-li dána konečná množina $\mathcal{S} = \{S_1, S_2, \ldots, S_n\}$ objektů a metrika $d()$, je binární MVP-strom s parametry $m = 2$, k a p konstruován na $\mathcal{S}$ takto:

1. Pokud $\mathcal{S} = \emptyset$, vytvoř prázdný strom a skonči.
2. Pokud $|\mathcal{S}| \leq k + 2$, tak
   2.1 Vyber libovolný objekt z $\mathcal{S}$ – to je S_{v1}, první vantage point
   2.2 $\mathcal{S} \leftarrow \mathcal{S} - \{S_{v1}\}$
   2.3 Spočítej všechny $d(S_i, S_{v1})$, kde $S_i \in \mathcal{S}$ a ulož je v poli D_1
   2.4 Vyber nejvzdálenější objekt od S_{v1} – to je S_{v2}, druhý vantage point
   2.5 $\mathcal{S} \leftarrow \mathcal{S} - \{S_{v2}\}$
   2.6 Spočítej všechny $d(S_i, S_{v2})$, kde $S_i \in \mathcal{S}$ a ulož je v poli D_2
   2.7 Skonči

3. Jinak (pokud $|\mathcal{S}| > k + 2$)
   3.1 Vyber libovolný objekt z $\mathcal{S}$ – to je S_{v1}, první vantage point
   3.2 $\mathcal{S} \leftarrow \mathcal{S} - \{S_{v1}\}$
   3.3 Spočítej všechny $d(S_i, S_{v_1})$
   Pokud (*level* $\leq p$), nastav S_i.PATH[*level*] $\leftarrow d(S_i, S_{v1})$
   3.4 Uspořádej objekty v $\mathcal{S}$ vzhledem k jejich vzdálenosti od S_{v1}
   $M_1 \leftarrow \mathrm{median}(\{d(S_i, S_{v1}) \mid S_i \in \mathcal{S}\})$
   Rozlož objekty do dvou seznamů $\mathcal{S}_1$ a $\mathcal{S}_2$ stejné délky (podle M_1)
   3.5 Vyber libovolný objekt z $\mathcal{S}_2$ – to je S_{v2}, druhý vantage point
   3.6 $\mathcal{S}_2 \leftarrow \mathcal{S}_2 - \{S_{v2}\}$
   3.7 Vypočítej všechny $d(S_j, S_{v2})$, kde $S_j \in \mathcal{S}_1 \cup \mathcal{S}_2$
   Pokud *level* $< p$, tak S_j.PATH[*level* $+ 1$] $= d(S_j, S_{v2})$.
   3.8 $M_2[1] \leftarrow \mathrm{median}(\{d(S_j, S_{v2}) \mid S_j \in \mathcal{S}_1\})$
   $M_2[2] \leftarrow \mathrm{median}(\{d(S_j, S_{v2}) \mid S_j \in \mathcal{S}_2\})$
   3.9 Rozlož $\mathcal{S}_1$ i $\mathcal{S}_2$, každý do dvou seznamů stejné délky
   (podle $M_1[1]$ a $M_2[2]$).
   3.10 rekurzivně pokračuj pro všechny 4 množiny.

Vyhledávání v MVP-stromech

Množinu objektů, které jsou ve vzdálenosti r od Q je nalezena vyhledávacím algoritmem:

1. Vypočítej vzdálenosti $d(Q, S_{v1})$ a $d(Q, S_{v2})$
   - pokud $d(Q, S_{v1}) \leq r$, pak S_{v1} je ve výsledné množině
   - pokud $d(Q, S_{v2}) \leq r$, pak S_{v2} je ve výsledné množině
2. Pokud je aktuální uzel list, tak pro všechny datové body S_i uzlu:
   2.1 Najdi $d(S_i, S_{v1})$ a $d(S_i, S_{v2})$ v polích D_1 a D_2.
   2.2 pokud $[d(Q, S_{v1}) - r \leq d(S_i, S_{v1}) \leq d(Q, S_{v1}) + r]$ a současně $[d(Q, S_{v2}) - r \leq d(S_i, S_{v2}) \leq d(Q, S_{v2}) + r]$ a pro všechna $i = 1, \ldots, p$ platí $PATH[i] - r \leq S_i.PATH[i] \leq PATH[i] + r$, vypočítej $d(Q, S_i)$. $d(Q, S_i) \leq r$, pak S_i je ve výsledné množině.

3. jinak, (pokud je aktuální uzel interní)
   3.2 pokud $l \leq p$, $PATH[l] = d(Q, S_{v1})$,
   pokud $l \leq p$, $PATH[l+1] = d(Q, S_{v2})$.
   3.2 pokud $d(Q, S_{v1}) + r \leq M_1$, pak
   - pokud $d(Q, S_{v2}) + r \leq M_2[1]$, rekurzivně prohledej první podstrom s $l \leftarrow l + 2$
   - pokud $d(Q, S_{v2}) - r \geq M_2[1]$, rekurzivně prohledej druhý podstrom s $l \leftarrow l + 2$

   3.3 pokud $d(Q, S_{v1}) + r \geq M_1$, pak
   - pokud $d(Q, S_{v2}) + r \leq M_2[1]$, rekurzivně prohledej třetí podstrom s $l \leftarrow l + 2$
   - pokud $d(Q, S_{v2}) - r \geq M_2[1]$, rekurzivně prohledej čtvrtý podstrom s $l \leftarrow l + 2$

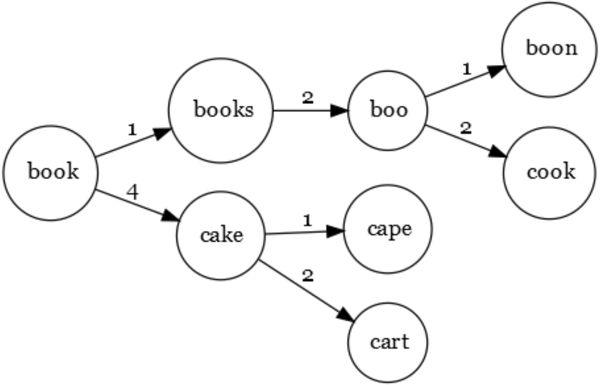
Bk-stromy

W. Burkhard, R. Keller
Some approaches to best-match file searching
CACM, 1973

- stromová struktura adaptovaná na diskrétní metriky (třeba Levenshteinova vzdálenost).
- Definován následovně: libovolný prvek je zvolen jako kořen, k-tý podstrom prvku a je rekurzivně vytvořen ze všech prvků b

$$d(a, b) = k$$

- Obvykle používány k hledání blízkých slov ve slovníku.


book
books
boo
boon
cook
cake
cape
cart
1
4
2
1
2
1
2

- konstrukce, inkrementální vkládání a vyhledávání je zjevné.
- vyhledávání NN je prováděno takto:

- pokud je strom prázdný, vrať $\emptyset$
- Vytvoř množinu uzlů S ke zpracování, a vlož do ní kořen.
- Dokud $S \neq \emptyset$
  - Vyber lib. uzel u z S
  - Vypočítej $d_u \leftarrow d(q, u)$, pokud je menší, než doposud nalezené, zapamatuj si ho
    $d_{\mathrm{best}} \leftarrow d_u$
  - Pro každého potomka v uzlu u:
    pokud $|d_v - d_u| < d_{\mathrm{best}}$
    vlož v do S.

M-stromy

- M-stromy jsou stromové datové struktury, které jsou podobné R-stromům a B-stromům.
- M-strom konstruován pomocí metriky a staví na trojúhelníkové nerovnosti pro efektivní vyhledávání rozsahu a vyhledávání k-nejbližších sousedů (k-NN).
- Zatímco M-stromy mohou dobře fungovat za mnoha podmínek, strom může mít také velké překrytí a neexistuje jasná strategie, jak se překrývání nejlépe vyhnout.
- Kromě toho je lze použít pouze pro funkce vzdálenosti, které splňují trojúhelníkové nerovnost[1].

[1] mnoho pokročilých funkcí nepodobnosti to nesplňuje

- V podstatě si je můžeme představit jako R-stromy, kde nemáme MBR ale n-dimenzionální koule dané objekty a poloměrem.
- strom primárně slouží k podobnostním dotazům (nejbližší sousedi).

M-strom má tyto komponenty a podkomponenty:

- **Nelistové uzly** – obsahují
  - množinu záznamů N_{RO} navigačních objektů
  - ukazatel na rodičovský objekt O_{p}.
- **Listové uzly** – obsahují
  - množinu záznamů N_{O} objektů
  - ukazatel na rodičovský objekt O_{p}.

- **Navigační objekty** – záznam navigačního objektu obsahuje:
  - (hodnoty) navigačního objektu O_r.
  - pokrývaný poloměr $r(O_r)$
  - ukazatel na pokrývaný strom $T(O_r)$ – všechny objekty v podstromu jsou ve vzdálenosti maximálně $r(O_r)$ od O_r.
  - vzdálenost $d(O_r, P(O_r))$ objektu O_r od jeho rodičovského objektu (používá se k optimalizaci při vyhledávání)
- **Objekty** – záznam objektu obsahuje:
  - (hodnoty) objektu O_j.
  - identifikátor objektu $oid(O_j)$.
  - vzdálenost $d(O_j, P(O_j))$ objektu O_j od jeho rodičovského objektu (používá se k optimalizaci při vyhledávání)

Hledání rozsahu

Algoritmus `Range`(Q,$r(Q)$), kde Q je objekt a $r(Q)$ je poloměr, vybere všechny objekty takové, že $d(Oj, Q) < r(Q)$.

- začíná od kořenového uzlu a rekurzivně prochází všechny cesty, u kterých nelze vyloučit, že vedou k objektům splňujícím tuto nerovnost.

```
def Range(N, E):
    input : N: uzel
            Q objekt vyhledávání
            r(Q) poloměr.
    Nechť O_p je rodičovský objekt uzlu N
    if N není list then
        for O_r ∈ N do
            if |d(O_p, Q) − d(O_r, O_p)| ≤ r(Q) + r(O_r) then
                Vypočti d(O_r, Q)
                if d(O_r, Q) ≤ r(Q) + r(O_r) then
                    Range(T(O_r),Q,r(Q))
    else
        for O_j ∈ N do
            if |d(O_p, Q) − d(O_j, O_p)| ≤ r(Q) then
                Vypočti d(O_j, Q)
                if d(O_j, Q) ≤ r(Q) then
                    Přidej O_j do výsledku
```

Lemma 3

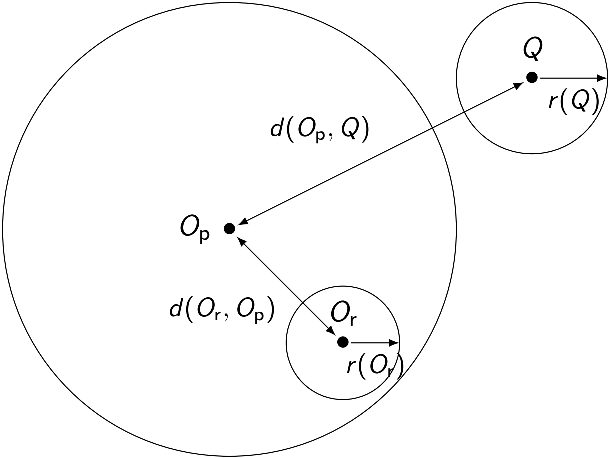
Pokud $d(O_r, Q) > r(Q) + r(O_r)$, pak pro každý objekt O_j v $T(O_r)$ platí $d(Q_j, Q) > r(Q)$.
Takže $T(O_r)$ může být bezpečně ořezáno z prohledávání.

- Ovšem abychom mohli použít Lemma 3, musíme vypočítat $d(Q_r, Q)$.
- Tomu se dá předejít s využitím následujícího lemmatu.

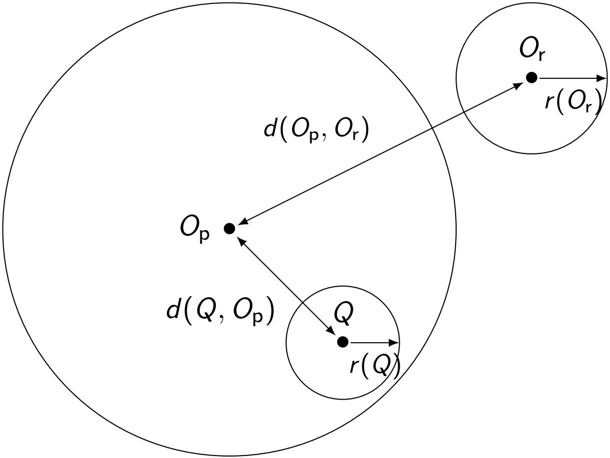
Lemma 4

Pokud $d(O_p, Q) - d(O_r, O_p) > r(Q) + r(O_r)$, pak $d(O_r, Q) > r(Q) + r(O_r)$.

- Obě ta lemmata jsou důsledky trojúhelníkové nerovnosti.



Q
$r(Q)$
$d(O_p, Q)$
O_p
$d(O_r, O_p)$
O_r
$r(O_r)$


O_r
$r(O_r)$
$d(O_p, O_r)$
O_p
$d(Q, O_p)$
Q
$r(Q)$

Hledání sousedů

- Algoritmus `k_NN_Search` najde *k* nejbližších sousedů k objektu *Q* – předpokládá se, že minimálně *k* objektů je v M-stromu.
- Použijeme techniku *branch-and-bound*, celkem podobnou té, kterou jsme tu měli pro R-stromy.
- Využijeme dvě globální struktury:
  - prioritní fronta *PR*,
  - *k*-prvkové pole *NN*, které na konci provedení obsahuje výsledek.

- PR je fronta ukazatelů na aktivní podstromy, tj. podstromy, které mohou obsahovat blízké sousedy.
- Společně s podstromy $T(O_r)$ je uložena dolní hranice $d_{\min}(T(O_r))$ vzdálenosti k objektům v $T(O_r)$.
- Ta dolní hranice je

$$d_{\min}(T(O_r)) = \max\{d(O_r, Q) - r(O_r), 0\}$$

  protože žádný objekt v $T(O_r)$ nemůže mít menší vzdálenost než $d(O_r, Q) - r(O_r)$.
- Tyto hranice jsou používány funkcí `ChooseNode` k výběru následujícího zkoumaného uzlu z PR.

- Ořezávací kritérium v `k_NN_Search` je dynamické – vyhledávaný poloměr je vzdálenost mezi Q a aktuálním k-tým nejbližším sousedem.
- Proto pořadí, ve kterém jsou uzly zkoumány má vliv na výkon.
- Heuristické kritérium využívané ve funkci `ChooseNode` je následující:

vybere se ten uzel, pro který je minimální hranice $d_{\min}$ minimální.

---

def `ChooseNode(`PR`):`

input : PR: prioritní fronta

Nechť $d_{\min}(T(O_r^*)) = \min\{d_{\min}(T(O_r))\}$ uvažujíce všechny záznamy v PR

Odstraň záznam $[T(O_r^*), d_{\min}(T(O_r))]$ z PR

vrať $T(O_r^*)$

- Na konci vykonávání `k_NN_Search` bude i-tá položka pole NN mít hodnotu $NN[i] = [O_j, d(O_j, Q)]$, kde O_j je i-tý nejbližší soused Q.

- Vzdálenost, od i-tého záznamu bude označena d_i, takže d_k je největší vzdálenost v NN.

- Zjevně d_k hraje roli *dynamického poloměru vyhledávání*, protože jakýkoli podstrom splňující $d_{\min}(T(O_r)) > d_k$ může být bezpečně ořezán.

- Záznamy v NN jsou iniciálně nastaveny na $NN[i] = [\_, \infty]$ $(i = 1, \ldots, k)$.
- Jak začne vyhledávání a přistupuje se k interním uzlům, aplikujeme následující: pro každý podstrom $T(O_r)$ vypočteme horní hranici $d_{\max T(O_r)}$ vzdálenosti objektů v $T(O_r)$ od Q.
- Horní hranice je

$$d_{\max T(O_r)} = d(O_r, Q) + r(O_r).$$

Příklad 5

Uvažme nejednodušší případ, $k = 1$, dva podstromy $T(O_{r_1})$ a $T(O_{r_2})$ a předpokládejme, že $d_{\max T(O_{r_1})} = 5$ $d_{\min T(O_{r_2})} = 7$.

Protože je garantováno, že $T(O_{r_1})$ obsahuje objekt, jehož vzdálenost od Q je nejvýše 5, můžeme $T(O_{r_2})$ ořezat z vyhledávání.

def `k_NN_NodeSearch(`N, Q, k`):`

input : N – uzel
Q – objekt, k němuž jsou vyhledáváni sousedi
k – počet sousedů

Nechť O_p je rodičovský objekt uzlu N
if N *není list* **then**
  for $O_r \in N$ **do**
    if $|d(O_p, Q) - d(O_r, O_p)| \leq d_k + r(O_r)$ **then**
      Vypočítej $d(O_r, Q)$
      if $d_{\min}(T(O_r)) \leq d_k$ **then**
        přidej $[T(O_r), d_{\min}(T(O_r))]$ do PR
        if $d_{\max}(T(O_r)) < d_k$ **then**
          $d_k \leftarrow$ `NN_Update(`$[\_, d_{\max}(T(O_r))]$`)`
          Odstraň z PR všechny záznamy, pro které $d_{\min}(T(O_r)) > d_k$

for $O_j \in N$ **do**
  if $|d(O_p, Q) - d(O_j, O_p)| \leq d_k$ **then**
    Vypočítej $d(O_j, Q)$
    if $d(O_j, Q) \leq d_k$ **then**
      $d_k \leftarrow$ `NN_Update(`$[O_j, d(O_j, Q)]$`)`
      Odstraň z PR všechny záznamy, pro které $d_{\min}(T(O_r)) > d_k$

Vkládání do M-stromu

- Nejprve najdeme listový uzel N, kam nový objekt O_n patří.
- Pokud N není plný, stačí přidat O_n do N. Pokud N je plný, pak vyvolat metodu splitu N.

- Základní idea, použitá ke stanovení „nejvhodnějšího“ listového uzlu listu, je sestoupit v každé úrovni stromu do toho podstromu $T(O_\mathrm{r})$, pro který není nutné zvětšovat poloměr pokrytí, tj.

$$d(O_\mathrm{r}, O_n) \leq r(O_\mathrm{r}) \tag{1}$$

- Pokud existuje více podstromů s touto vlastností, vybereme ten, pro který je objekt O_n nejblíže k O_r.
- Tato heuristika se snaží dostat dobře seskupené (shluknuté) podstromy, což má přínosný vliv na výkon.

Pokud neexistuje navigační objekt, který by splňoval (1), vybereme ten, který minimalizuje zvětšení pokrývajícího poloměru

$$d(O_{\mathrm{r}}, O_n) - r(O_{\mathrm{r}}).$$

Tímto se snažíme minimalizovat celkový „objem“ pokrytý navigačními objekty v aktuálním uzlu.

```
def Insert(N, E):
    input : N: uzel
            O_n nový záznam objektu
    Nechť 𝒩 je množina záznamů v N
    if N není list then
        Nechť 𝒩_in jsou v 𝒩 ty záznamy splňující d(O_r, O_n) ≤ r(O_r)
        if 𝒩_in ≠ ∅ then
            Nechť O_r^* ∈ 𝒩_in, pro který je d(O_r^*, O_n) minimální
        else
            Nechť O_r^* ∈ 𝒩, pro který je d(O_r^*, O_n) − d(O_r^*) minimální
            r(O_r^*) = d(O_r^*, O_n)
        Insert (T(O_r^*),O_n)
    else
        if N není plný then
            ulož O_n do N
        else
            Split(N,O_n)
```

Split

- M-strom (opět) roste zdola nahoru.
- s přeplněním uzlu N se vypořádáme opět tak, že vytvoříme nový uzel N' na stejné úrovni, rozdělením objektů z N mezi tyto dva uzly a posláním dvou nových navigačních objektů, které se odkazují na vzniklé uzly, do předka N_p (split).
- Pokud dojde ke splitu kořene, je vytvořen nový kořen a M-strom vyroste o jednu úroveň.

def `Split`(N, E):

    input : N: uzel
              E nový záznam objektu
    $\mathcal{N} \leftarrow$ záznamy uzlu $N \cup \{E\}$
    if N *není kořen* **then**
        Nechť O_p je rodičovský objekt uzlu N, uložený v uzlu N_p
    $N' \leftarrow$ nový uzel
    `Promote`($\mathcal{N}$,O_{p_1},O_{p_2})
    $\mathcal{N}_1, \mathcal{N}_1 \leftarrow$ `Partition`($\mathcal{N}$, O_{p_1}, O_{p_2})
    ulož objekty z $\mathcal{N}_1$ do N
    ulož objekty z $\mathcal{N}_2$ do N'
    if N *je kořen* **then**
        $N_\mathrm{p} \leftarrow$ nový kořenový uzel
        ulož objekty O_{p_1} a O_{p_2} do N_p
    else
        Nahraď objekt O_p objektem O_{p_1} v N_p
        if N_p *je plný* **then**
            `Split`(N_p, O_{p_2})
        else
            ulož O_{p_2} do N_p

- Metoda `Promote` vybere a dle specifického kritéria dva navigační objekty O_{p_1} a O_{p_2}, které budou vloženy do rodičovského uzlu N_p.
- Metoda `Partition` rozdělí objekty z přeplněného uzlu (množiny $\mathcal{N}$) do dvou disjunktní podmnožiny $\mathcal{N}_1$ a $\mathcal{N}_2$, které jsou pak uloženy do uzlů N a N'.
- Specifické implementace metod `Promote` a `Partition` definují strategii splitu.

Bez ohledu na specifickou strategii splitu, sémantika pokrývajících poloměrů musí být zachována.

- Pokud je splitovaný uzel list, pokrývající poloměr povyšovaného objektu O_{p_1} je nastavena na

$$r(O_{p_1}) = \max\{d(O_j, O_{p_1}) \mid O_j \in \mathcal{N}_1\}.$$

- Pokud splitujem interní uzel, pak

$$r(O_{p_1}) = \max\{d(O_r, O_{p_1}) + r(O_r) \mid O_j \in \mathcal{N}_1\}.$$

  To zajišťuje, že $d(O_j, O_{p_1}) \leq r(O_{p_1})$ platí pro libovolný objekt v podstromu v (O_{p_1}).