FIATAL MŰSZAKIAK TUDOMÁNYOS ÜLÉSSZAKA

Kolozsvár, 2002. március 22-23.

AZ MVP FA HATÉKONYSÁGÁNAK VIZSGÁLATA, IMPLEMENTÁCIÓJA

**Krizsán Zoltán doktorandusz
Miskolci Egyetem, Általános Informatika Tanszék**

Abstract

There are a lot of databases applications, what based on similarity search. In this case we have to find data items that are similar a query item. Our task is to minimize search time. The proximity search is defined on Metric space, where the distance is identified. To do this we have to index the data set according to distances. Number of solutions exist which solve it, but in case of large data set the computation time is long. We will review the algorithm of mvp tree, and analyse the efficiency. I have implemented a library containing objects for mvp tree in C++ with STL.

Bevezetés

Bizonyos adatbázisok alkalmazása esetén igény merül fel a közelítő keresés alkalmazására. Ezekben az esetekben egy konkrét értékhez keresünk közeli elemeket. Ilyenek a genetikus alkalmazások szöveg keresések, kép adatbázisok. Az elemek nagyon különböznek, sőt a távolság fogalma is más, és más (kép esetén például lehet alak, szín vagy textúra szerinti távolságot értelmezni).

A következőkben a metrikus terekben specifikált indexfákat vizsgáljuk meg (vp fa, mvp fa).

Metrikus a térről beszélünk, ha teljesülnek a következő feltételek:

- $d_{(x,y)} = d_{(y,x)}$
- $0 < d_{(x,y)} < \infty, x \neq y$
- $d_{(x,x)} = 0$
- $d_{(x,y)} \leq d_{(x,z)} + d_{(z,y)}$

A bináris vp fa

Struktúrája egyszerű. Minden pontban tároljuk a szükséges adatokat: S_v , M , R_{ptr} , L_{ptr} , ahol S_v (vantage point) viszonyítási pont, M a pontok távolságának a Mediánja (S_v -től), R_{ptr} , L_{ptr} pedig a jobb, és bal oldali ág mpointere. A fa felépítése rekurzív algoritmussal történik. Felülről lefelé építjük fel. Balra kerülnek azok a pontok, amelyek távolsága kisebb, mint az M Medián, jobbra pedig a többi pontok. Az előbbieket szerint két csoportra osztott halmazon ismét elvégezzük az előbbi műveletet. A felépítés számítási igénye $\log_2 n$.

Több utas vp fa

A kört tovább bontja gyűrűkre. így a fa mélysége csökken, és ezzel a számítási költség is (kevesebb távolság számítás). Ha m a felosztások száma, akkor a struktúra felépítés számítási igénye $\log_m n$

A egyszerűsített több utas vpfa

A Struktúrának 3 bemenő paramétere van:

- egy viszonyítási ponthoz tartozó távolságok száma (m),
- a levelekben tárolható maximum elemszám (k),
- levelekben elhelyezkedő adatokhoz tartozó távolságok a viszonyítási pontokhoz képest (p)

Adott továbbá egy távolság függvény ($d(H_1, H_2)$). Itt egy csomópontból több, m^2 gyerek származik. Egy csomópontban két viszonyítási pont van (S_{v1} , S_{v2}), és hozzájuk tartozó Mediánok (M_1 , $M_2[1]$, $M_2[2]$), az előbb említett struktúra összevonása.

A fa felépítésének módszere:

Legyen $H = \{H_1, H_2, H_3, \dots, H_n\}$ egy véges halmaz a metrikus térben. Véletlenszerűen kiválasztunk egy pontot a halmazból (S_{v1}), vegyük ki ezt az elemet ($H := H - \{S_{v1}\}$), majd számoljuk az összes többi pont távolságát az S_{v1} -től ($d(H_j, S_{v1})$). A Medián kiértékelése után bontsuk két csoportra a kapott halmazt ($H_1 := \{H_j \mid d(S_{v1}, H_j) \leq M_1, H_j \in H\}$, $H_2 := \{H_j \mid d(S_{v1}, H_j) > M_1, H_j \in H\}$, $H = H_1 \cup H_2$). Legyen S_{v2} az S_{v1} -től legtávolabbi pont a H_2 -ből, vegyük ki a halmazból ($H_2 := H_2 - \{S_{v2}\}$). Számítsuk ki az összes pont távolságát ($d(H_j, S_{v2})$), ahol $H_j \in H_1 \mid H_j \in H_2$. $M_2[1]$ a H_1 halmaz elemek távolságainak Mediánja, a $M_2[2]$ pedig a H_2 halmaz elemek távolságainak Mediánja az S_{v2} -től. Képezzük az alábbi halmazokat:

$$H_A := \{H_j \mid d(S_{v2}, H_j) \leq M_2[1], H_j \in H_1\}, H_B := H_1 - H_A$$

$$H_c := \{H_j \mid d(S_{v_2}, H_i \leq M_2[2], H_i \in H_2 \}, H_d := H_2 - H_c$$

A keresési algoritmus az [1], és [2] cikkekben.

Implementáció

A feladatot C++-ban az STL(Standard Template Library) segítségével oldottam meg. Három osztály szükséges a feladat megvalósításához: mvpTree, treeNode és a oneltem, melyek az 1. ábrán találhatóak. Az első kettő egy könyvtárban van, megvalósítása rejtett, míg a oneltem.hpp szerkeszthető, tartalmazza az adatelemet reprezentáló osztály definícióját. A feladattól függően a felhasználó módosítja a forrást (oneltem.hpp), az általános megoldást specializálja.

1. ábra Az implementációhoz használt osztályok interfészei

Az ábrán látható oneltem osztály csak egy példa, tetszőlegesen módosítható, felépítésében lehet más. A felhasználónak feltétlenül implementálnia kell a tagfüggvényeket, azok prototípusán nem változtathat, viszont az adattagok száma és típusa lehet más. A treeNode osztály mind a belső adatpont, mind a levél tárolására is alkalmas, az m_bType adattag jelöli ki mely adattagok érvényesek, így kényelmesen használható. A felépítéskor létrehozunk egy csomópontot, majd a separate metódussal szétbontjuk. Itt ha a csomópontban tárolt elemek száma még több, mint a kívánt ($H=\{H_1, H_2, H_3, \dots, H_i\} \mid i > k+2, k = m_iMaxFanout$), akkor belső csomópont és meghívódik a SetSv1 egy H_i -el , egyébként meghívódik a SetLeafeSv1 H_i -el. Az előző lépést addig ismételjük rekurzívan, amíg a halmaz elemeinek száma le nem csökken a megfelelő számúra. A fa mélységét eltároljuk a későbbi helytelen csomópont kérések kiszűrésére. Az eredeti specifikációban (lásd [1],[2]) egy levél esetén tároljuk az adatpontok távolságát az első p viszonyítási ponttól, a keresés gyorsítása érdekében. így azonban az indexfa felépítése hosszadalmasabb, és tárolásához több memória szükséges.

Konklúzió

Az [1], és [2] cikkekben specifikált mvp fa, ahol az első viszonyítási pont ismeretében választjuk ki a második viszonyítási pontot helytakarékos, hatékony, a probléma az első viszonyítási pont kiválasztása. Jelenleg egy véletlen eleme a halmaznak. Ennél jobb megoldás, ha egyszerre jelöljük ki a mind a két pontot. Menete a következő: veszünk egy véletlen pontot, majd attól legtávolabbit, ez lesz az S_{v_1} , majd az attól legtávolabbit, ez lesz az S_{v_2} . Ez viszont számítás igényes, amit minden lépésben meg kell ismételni. Ha létezik az adathalmazban átlag érték, vagy globális viszonyítási pont, akkor az adathalmazt rendezve, ismét alkalmazhatnánk az előző megoldást felezési pontokkal. A jövőben szeretném kiterjeszteni az mvp fa implementációját, hogy a viszonyítási pontokat 1 lépésben lehessen kiválasztani. Szeretném továbbá optimalizálni az algoritmust és a kezdeti paraméterek (p,k,m) hatásait vizsgálni a fa hatékonyságára.

Irodalom

[1]Tolga Bozkaya, Meral Ozsologlu: DISTANCE-BASED INDEXING FOR HIGH-DIMENSIONAL METRIC SPACES

[2] Tolga Bozkaya, Meral Ozsologlu: INDEXING LARGE METRIC SPACES FOR SIMILARITY SEARCH QUERIES

Krizsán Zoltán Doktorandusz

Miskolci Egyetem, Általános Informatika Tanszék Miskolc Egyetemváros

Telefon: +36-46-565-111/1758