ساختمان داده‌ها (۲۲۸۲۲)

مدرس: مرتضی علیمی

[پاییز ۹۸]

جلسه : درخت kd

نگارنده: مهرسا پوریا

درخت kd [1] داده ساختاری برای نگهداری نقاط در فضای k بعدی است. هر نقطه در فضای k بعدی را میتوان توسط k مولفه به صورت $x = (x_۱, x_۲, \cdots, x_k)$ نمایش داد. پیشتر با درخت جست و جوی دودویی آشنا شدیم که برای نگهداری اعداد حقیقی مناسب بود. حال اگر اعداد k بعدی باشند میتوان از درخت kd که در هر عمق درخت روی یک بعد مشابه درخت جست و جوی دودویی است استفاده کرد. در ادامه با ساختار این درخت آشنا شده و چند پرسش [2] که این ساختمان داده قابل به پاسخ گویی به آن است را بررسی خواهیم کرد.

۱ آشنایی با درخت kd

در درخت جست و جوی دودویی کلید هر گره محور اعداد حقیقی را به دو قسمت تقسیم میکرد، حال اگر این ایده را به ابعاد بالاتر تعمیم دهیم، میخواهیم هنگامی که کلید یک گره k بعدی باشند فضای R^k را متناظر با آن کلید به دو قسمت تقسیم کنیم؛ برای اینکار به یک ابر صفحه $k-1$ بعدی نیاز داریم. راه های مختلفی برای اینکار وجود دارد اما ساده ترین ایده میتواند این باشد که برای هر گره متناظر با عمقش یک ابر صفحه انتخاب کنیم که خط عمود بر آن موازی محور مختصات متناظر با بعد منتسب به آن عمق باشد و با تغییر عمق گره به صورت متناوب بعد منتسب به هر عمق را عوض کنیم. به بعدی که به هر گره نسبت میدهیم، بعد برش دهنده [3] میگوییم. به علاوه هر گره صرفا زیرفضایی را افراز میکند که گره های با ارتفاع بالاتر آن محدود کرده اند. (به فضای محدود شده ای که هر گره در آن قرار میگیرد سلول آن گره [4] میگوییم.) این خاصیت این گونه به دست می آید که در هر گره با توجه به بعد برش دهنده (cd) آن مقایسه رخ می دهد و نقاطی که مولفه cd کمتری از مولفه cd ام کلید گره دارند در سمت چپ آن و در غیر این صورت در سمت راست آن گره قرار میگیرند. در ادامه یک مثال از 2dTree و تقسیم فضای دوبعدی توسط آن میبینیم. برای ساخت درخت برای یک مجموعه اعداد داده شده میتوانیم برای متوازن شدن درخت ساخته شده در هر مرحله به گونه ای فضا را برش بزنیم که تعداد داده های دو طرف تقریبا برابر باشد که این معادل انتخاب میانه از بین اعداد سلول با توجه به بعد برش دهنده است.

مثال: فرض کنید نقاط زیر را داریم :

$$(۳۵, ۴۰), (۵۰, ۱۰), (۶۰, ۷۵), (۸۰, ۶۵), (۸۵, ۱۵), (۵, ۴۵), (۲۵, ۳۵), (۹۰, ۵)$$

در این صورت درخت نشان داده شده در شکل ۱، یک 2d Tree معتبر روی این نقاط است. همانگونه که در شکل مشخص شده است در ریشه مقایسه بر اساس بعد اول (x) صورت میگیرد و در عمق بعدی مقایسه بر اساس بعد دوم (y) صورت میگیرد. همچنین در این شکل تقسیم فضای دو بعدی توسط خطوط متناظر با هر گره را میبینیم. ⊠

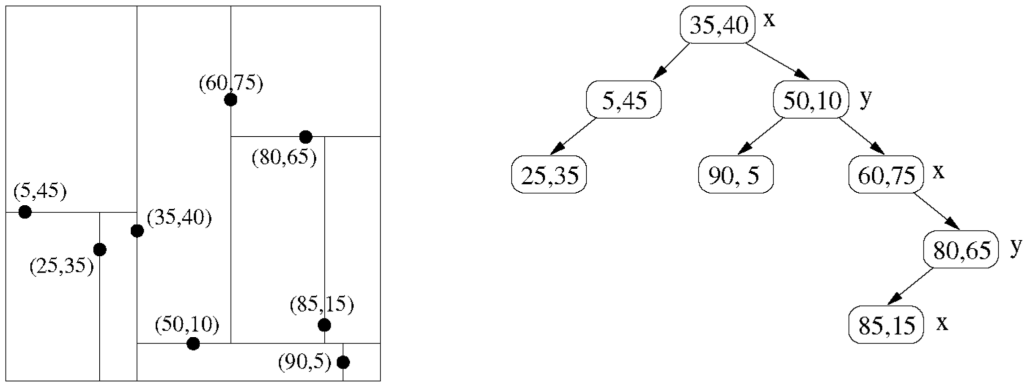
---

[1] kd Tree

[2] Query

[3] Cutting Dimension

[4] cell



شکل ۱: 2d Tree

۲ اضافه کردن

فرض کنیم یک درخت kd معتبر با ریشه $root$ داریم. میخواهیم نقطه x را به درخت اضافه کنیم، برای اینکار تابع $INSERT(x, root, 0)$ را صدا میزنیم. این تابع مشابه درخت جست و جوی دویی در هر گره با توجه به بعد برش دهنده اش (cd) مقایسه را انجام میدهید و در صورتی که مولفه cd ام x از مولفه cd ام کلید گره کوچکتر باشد آن را در سمت چپ گره و در غیر این صورت در سمت راست آن اضافه میکند و با هر بار حرکت روی درخت بعد برش دهنده را متناوبا با توجه به ساختار درخت تغییر میدهیم. در این جا فرض شده که ابعاد برش دهنده نسبت داده شده به هر گره با توجه به عمق آن و به صورت متناوب از 0 تا $Dim - 1$ است. مرتبه زمانی اضافه کردن گره از مرتبه $O(h)$ است که h عمق درخت است و در صورتی که درخت متوازن باشد از مرتبه $O(log\ n)$ است.

INSERT$(node\ x, KDNode\ t, int\ cd)$

if $(t == null)$
    $t = new \quad KDNode(x)$
else if $(x[cd] < t.data[cd])$
    $t.left = insert(x, t.left, (cd + 1) \mod Dim)$
else
    $t.right = insert(x, t.right, (cd + 1) \mod Dim)$
return

۳ یافتن کمینه در یک بعد

برای یافتن کمینه یک بعد (dim) اگر در یک گره بعد برش دهنده برابر dim باشد فقط جست و جو را در زیر درخت سمت چپ آن گره ادامه میدهیم و در غیر این صورت باید هر دو زیر درخت سمت چپ و راست آن گره را جست و جو کنیم. یافتن بیشینه نیز به همین صورت و با تغییرات جزئی انجام میشود. یافتن کمینه مشابه درخت جست و جوی دودویی در حذف درخت استفاده میشود.

```
findMin(node T, int dim, int cd)
if (T == null)
    return null
if (cd == dim)
    if (T.left == null) return t.data
    else return findMin(T.left, dim, (cd + 1) mod Dim)
else
    return minimum(
        findMin(T.left, dim, (cd + 1) mod Dim),
        findMin(T.left, dim, (cd + 1) mod Dim),
        T.data
        )
```

۴ یافتن نزدیک ترین همسایه

یکی از مهمترین query ها که توسط kd tree میتوانیم به آن پاسخ دهیم یافتن نزدیک ترین همسایه به یک نقطه در فضاست. پاسخ به این پرسش کاربردهای زیادی از جمله در الگوریتم های یادگیری ماشین مانند KNN [5] دارد. فرض کنیم مجموعه نقاط S در یک درخت kd ذخیره شده است. میخواهیم نقطه ای از S که کمترین فاصله با نقطه ورودی Q را دارد بیابیم. فرض کنیم نزدیکی را بر اساس کم بودن فاصله اقلیدسی بین نقاط تعریف کنیم. فرض کنید که در ابتدا حدسی برای نزدیکترین نقطه ($best$) داریم. اگر فرض کنیم نقاط دو بعدی هستند اگر نقاط نزدیک تر به این نقطه وجود داشته باشد باید در داخل دایره به مرکز q و شعاع اندازه فاصله بین q و نقطه حدس زده شده ($bestDist$) می افتند. در فضای d بعدی در داخل کره d بعدی می افتد که به آن ابرکره نامزد [6] میگوییم. این مشاهده از آنجا اهمیت دارد که توسط آن میتوانیم بخش هایی از درخت که مطمئنیم جواب نزدیکترین همسایه در آن نیست را حذف کنیم. به این کار هرس کردن [7] گفته میشود. حال به استفاده از این شهود به توضیح الگوریتم NN می پردازیم. در ابتدا $best = null$ و $bestDist = \infty$ می باشد. cd همان بعد برش دهنده است که در قسمت های قبل به آن اشاره کردیم و BB سلول آن گره است. این پارامتر از این جهت اهمیت دارد که اگر فاصله ی همه ی نقاط داخل یک محدوده از Q بیشتر از بهترین فاصله فعلی بود دیگر نباید آن محدوده را جست و جو کنیم. وقتی وارد گره T میشویم اول تعیین میکنیم که سلول مربوط به آن آیا میتواند جوابی بهتر از بهترین جواب فعلی تولید کند یا خیر اگر نه از آن گره خارج میشویم. سپس اگر خود گره فاصله کمتر از $bestDist$ داشت مقادیر مربوط به جواب را با مقدار خود گره به روز رسانی میکنیم. در نهایت به جست و جو در زیر درخت های آن گره میپردازیم در ابتدا باید بررسی کنیم که در کدام سمت ابر صفحه مربوط به آن گره قرار داریم اگر در سمت چپ قرار داشتیم ابتدا سلول سمت چپ و سپس سلول سمت راست و در غیر این صورت برعکس عمل میکنیم. این کار در واقع برای این است که بخش هایی که احتمالا جواب بهینه در آن قرار دارد را زودتر بررسی کنیم. فرض میکنیم توابع $trimLeft$ و $trimRight$ سلول مربوط به چپ و راست را بر میگردانند. اگر بخواهیم مرتبه ی زمانی این الگوریتم را بررسی کنیم در بدترین حالت هر گره یکبار بازدید میشود مرتبه زمانی جست و جوی نزدیک ترین همسایه از $O(n)$ است. اما میتوان نشان داد به $O(2^k + log\ n)$ نزدیک تر است، شهود آن این است که برای هر نقطه انتظار داریم همسایه های سلول مربوط به آن را در فضا بررسی کنیم که تعداد آنها 2^k است و زمان $O(logn)$ برای پایین آمدن در درخت و یافتن این گره ها نیاز داریم.

[5] K nearest neighbours
[6] Candidate hypersphere
[7] Pruning

```
NN(node Q, kdTree T, int cd, Rect BB)
if (T == null) return
if (distance(Q, BB) >= bestDist)
    return
(dist = distance(Q, T.data))
if dist < bestDist
    best = T.data
    bestDist = dist
if (Q[cd] < T.data[cd])
    NN(Q, T.left, nextCd, BB.trimLeft(cd, t.data))
    NN(Q, T.right, nextCd, BB.trimRight(cd, t.data))
else
    NN(Q, T.right, nextCd, BB.trimRight(cd, t.data))
    NN(Q, T.left, nextCd, BB.trimLeft(cd, t.data))
```

مراجع

[1] https://www.cs.umd.edu/users/meesh/420/ContentBook/FormalNotes/MountNotes/lecture17-quadkd.pdf.

[2] https://www.cs.umd.edu/users/meesh/420/ContentBook/FormalNotes/MountNotes/lecture18-kd2.pdf

[3] http://stanford.edu/class/archive/cs/cs106l/cs106l.1162/handouts/assignment-3-kdtree.pdf