به نام خدا

پروژه ی نهایی درس ساختمان داده

دکتر آبین – بهار 1404

اعضای گروه:

- آبتین رحمانی – 402243064
- ماهان بانشی – 402243042

هدف پروژه

هدف پروژه، پیاده‌سازی الگوریتم BJR-tree برای Continuous Skyline Computation است؛ الگوریتمی که از مقاله‌ی ارسالی گرفته شده و نسبت به الگوریتم های دیگر مثل BBS عملکرد بهتر و سریعتری دارد.

فایل مقاله:

در این مقاله الگوریتم BJR-tree برای محاسبه ی skyline ها بصورت پیوسته در داده های چند بعدی توضیح داده شده است. نکات مهم مقاله:

1) استفاده از درخت ریشه‌دار و متوازن (BJR-tree) که روابط غلبه (dominance) را ذخیره می‌کند.
2) استفاده از ND-cache برای کاهش محاسبات تکراری.

الگوریتم ما در بخش اصلی دارد: اضافه کردن (Injection) و حذف کردن (Ejection) نقطه

در این پروژه ما این کار های اضافه و حذف کردن نقاط را بصورت real-time انجام می دهیم تا برنامه مان حالت دینامیکی داشته باشد.

بخش های پروژه:

- تعریف مسئله و فضای داده
- پیاده‌سازی Injection و Ejection
- استفاده از lazy evaluation برای توازن بهتر درخت
- اضافه کردن ND-Cache

فایل ها:

این پروژه کار با فایل هم دارد که ما از فایل ها به برای تست کردن درستی متد های پیاده سازی شده استفاده می کنیم. ما 3 نوع فایل small , medium , large داریم.

- فایل input ورودی ها است. برنامه ی ما باید این فایل را بخواند.
- فایل setup ویژگی ها را نگه می دارد. به ترتیب تعداد نقاط، تعداد ابعاد، تعداد بازه ها زمانی و حداکثر تعداد اعداد در هر خط خروجی.
- فایل times زمان ها را نگه می دارد. هر نقطه دو زمان دارد که یکی زمان وارد شدن و یکی زمان خارج شدن است. ما برای تعریف زمان از بازه های زمانی با طول برابر استفاده کرده ایم.
- فایل out خروجی برنامه ی ماست که باید در آن فایل بنویسیم.
- فایل refout خروجی صحیح برنامه به ازای ورودی های فایل input است. برای اینکه چک کنیم که آیا برنامه درست کار می کند یا نه باید تطابق فایل out را با این فایل بررسی کنیم.

شرح پروژه :

از آنجا که میخواهیم روی دیتاست مشخصی این پرژه رو چک کنیم، در ابتدا از کاربر نام فایل های ورودی را میگیریم که با توجه به نام آن ابعاد نقاط موجود در دیتاست را مشخص کنیم

سپس با استفاده از تابع load_data مختصات نقاط را ذخیره میکنیم و چک میکنیم که کاملا درست أخیر شده باشند(از نظر ابعاد نقطه). سپس تایم های inject و eject شدن را برای هر نقطه از ابتدا از فایل لود میکنیم و یک Node میسازیم. سپس از ساختمان داده map استفاده میکنیم تا زمان های inject و eject شدن را باستفاده از Node ساخته شده ذخیره کنیم. به

عبارت دیگر به هر تایم injection یک نود نسبت میدهیم و به هر تایم ejection، آیدی آن نود را نسبت میدهیم.

سپس با تابع process_time_steps، زمان اولین injection را به عنوان زمان شروع و زمان آخرین ejection را به عنوان زمان پایان در نظر میگیریم و با یک حلقه از زمان شروع تا زمان پایان، در هر لحظه بین آن ها چک میکنیم که نقطه ای وجود دارد که باید inject یا eject شود و آن را انجام میدهیم. همچنین در هر ثانیه نقاط skyline را به وسیله تابع write output و get skyline ids در فایل خروجی مینویسیم.

در تابع get_skyline_ids فرزندان ریشه درختمان را پیدا میکنیم، این فرزندان همان نقاط skyline ما هستند. آیدی هر node را استخراج کرده و ذخیره میکنیم.

Algorithm • `dominates()` in reference

```
1: procedure dominates(dataset, a: index, b: index)
2: flag ← 0;
3: for d = 0 to dimensions − 1 do
4:     if dataset[a][d] > dataset[b][d] then
5:         return 0
6:     else if dataset[a][d] < dataset[b][d] then
7:         flag ← 1;
8: return flag
```

الگوریتم Domination :

الگوریتم inject : برای این الگوریتم همچنان باید متد dominate را تعریف کنیم.

Algorithm 1 Injection (base)

```
1: procedure inject(r: root, v: new vertex)
2: C ← children of r;
3: for all c ∈ C do
4:    if c dominates v then
5:       inject(c, v);
6:       return
7: set v to r's child;
8: for all c ∈ C do
9:    if v dominates c then
10:      move c to v's child;
```

الگوریتم Ejection :

Algorithm 2 Ejection

```
1: procedure eject(v: ejected vertex ≠ O)
2: p ← parent of v;
3: C ← children of v;
4: remove v from p;
5: for all c ∈ C do
6:    inject(p, c);
```

بخش امتیازی:

ND-Cache: Non-Dominated Cache ساختاری ساده و کارآمد است که با آن نقاط غیرچیرگی‌شده (Skyline) در هر لحظه از زمان نگهداری می کنیم.

در این روش، تنها نقاطی که توسط سایر نقاط سلطه داده نشده‌اند، به‌صورت سلسله‌مراتبی در یک ساختار درختی‌گونه نگهداری می‌شوند، و هنگام ورود یا خروج نقاط، وضعیت Skyline به‌روز می‌گردد.

مراحل اجرای کار در برنامه ای که نوشته ایم:

1: خواندن فایل small.setup:

- شامل تعداد نقاط، تعداد ابعاد، تعداد بازه‌های زمانی و پیشوند فایل‌ها است.

2: خواندن مختصات نقاط از فایل input:

- هر نقطه شامل تعدادی مقدار عددی است (به اندازه‌ی ابعاد مشخص‌شده).

3: خواندن زمان ورود و خروج هر نقطه از فایل times:

- زمان‌ها به‌صورت زوج (in, out) مشخص شده‌اند و در طول بازه‌های زمانی اجرا می‌شوند.

4: تعریف ساختار Node:

- هر نقطه داخل یک گره (Node) نگهداری می‌شود که شامل مختصات نقطه، اشاره‌گر به والد و لیستی از فرزندان است.

5: فرآیند تزریق (injection):

- در هر زمان، نقاطی که وارد می‌شوند به ND-Cache اضافه می‌شوند.
- هنگام اضافه شدن، اگر نقطه‌ی جدید:
  - تحت سلطه‌ی گرهی باشد، به زیر آن گره تزریق می‌شود (به‌صورت بازگشتی)
  - دیگر نقاط را سلطه دهد، آن‌ها فرزند گره جدید می‌شوند
  - ین ساختار کاملاً مشابه تزریق در درخت BJR است.

6: فرآیند حذف (ejection):

- نقاطی که زمان خروج‌شان فرارسیده، از ساختار حذف می‌شوند.
- اگر نقطه‌ای حذف شود، فرزندان آن دوباره تزریق می‌شوند تا سلسله‌مراتب حفظ شود.

7: تولید فایل خروجی:

- در انتهای هر بازه‌ی زمانی، گره‌های مستقیم زیر ریشه به‌عنوان نقاط skyline (خروجی) ثبت می‌شوند.

- خروجی به صورت مرتب‌شده (بر اساس id) در فایل small.ndcache.out ذخیره می‌شود.

8: مقایسه با فایل refout:

- فایل خروجی تولید شده با فایل small.refout مقایسه می‌شود. (در مجموعه داده ی small)