دانشگاه صنعتی شریف
دانشکده مهندسی کامپیوتر

گزارش پروژه معماری گروه ۲۰

نگارش

مهرشاد دهقانی

یوسف سدیدی

امیرعلی شیخی

کیهان مسعودی

استاد درس

جناب آقای دکتر اسدی

مسئول پروژه

جناب آقای مرادی

تیر ۱۴۰۳

فهرست مطالب

فهرست تصاویر

فصل ۱

مقدمه

در این گزارش به بررسی فاصله استفاده مجدد در یک مجموعه از دسترسی‌ها به حافظه می‌پردازیم. در ادامه رابطه این فاصله و محلیت زمانی و فضایی در کد را مورد بررسی قرار خواهیم داد.

۱-۱ تعریف مسئله

در این پروژه به دنبال یافتن میانگین reuse distance (تعداد دسترسی به آدرس های متمایز از حافظه بین دو آدرس یکسان) بودیم تا اینکه بتوانیم به وجود ارتباطی بین reuse distance و محلیت فضایی و زمانی که در تمرین های عملی هشتم و نهم محاسبه کرده بودیم، برسیم.

۱-۲ اهمیت موضوع

به کمک نتایج حاصل از این مسئله می توان الگوریتم بهینه‌ای برای فرایند جایگزینی در سطوح مختلف حافظه نهان ارائه داد. به کمک تحلیل این رفتارها می‌توان بهینه‌سازی هایی در زمینه معماری های مرتبط با حافظه نهان انجام داد.

۱-۳ اهداف پژوهش

محاسبه average reuse distance برای فایل های ضمیمه شده (alibaba) در زمان $nlogn$ و کشف ارتباط بین reuse distance و محلیت فضایی و زمانی.

فصل ۲

مفاهیم اولیه

در اینجا به تعریف مفاهیم اولیه می‌پردازیم. فاصله استفاده مجدد یک روش کلاسیک برای مشخص کردن محلیت داده است. فاصله استفاده مجدد برای یک آدرس حافظه عبارت است از تعداد آدرس‌های حافظه‌ی متمایز که بین دو بار دسترسی به آن آدرس حافظه وجود دارد. یکی از چالش های مهم در زمینه طراحی الگوریتمی برای محاسبه فاصله‌ی استفاده مجدد، زمان اجرا و حافظه مصرفی آن است. یک روش ساده پیاده‌سازی این الگوریتم با اشتفاده از استک است که پیچیده زمانی آن $n^۲$ است. در این پروژه با استفاده از درخت های متوازن مانند AVL Tree، الگوریتمی با پیچیدگی $nlogn$ ارایه می‌دهیم.
فایل این برنامه به زبان C++ در پیوست این سند موجود است.

۲-۱ نحوه‌ی پیاده‌سازی

۲-۱-۱ داده ساختار

برای پیاده‌سازی بهینه، از یک داده ساختار مانند AVL Tree استفاده می‌کنیم. این داده ساختار به این صورت است که همواره هنگام درج و حذف عنصر، درخت با متوازن می‌کند تا ارتفاع درخت همواره $O(logn)$ باشد. در نتیجه این اعمال نیز $O(logn)$ طول خواهند کشید. عناصر این درخت شش مولفه دارند که عبارت از: اشاره‌گر به فزند چپ، راست، پدر، key، MemoryAccess و اندازه زیر درخت به این ریشه. حال یک مشکل اساسی این است که درخت ما بر اساس مولفه key یک درخت دودیی جست و جو است، درحالیکه ما نیاز داریم عناصر را بر اساس MemoryAccess پیدا

کنیم که در حالت عادی $O(n)$ است و مطلوب ما نیست. برای حل این مشکل از یک hashmap استفاده می‌کنیم و عناصر درخت را به صورت دوتایی های MemoryAccess, Key در آن نگه می‌داریم تا search در $O(1)$ انجام شود.

۲-۱-۲ نحوه کار الگوریتم

الگوریتم به این صورت کار می‌کند که از یک آرایه از دسترسی ها به حافظه (که از فایل های Alibaba خوانده می‌شود) را ورودی می‌گیرد. سپس از ابتدا با دیدن هر آدرس اگر آدرس در درخت وجود نداشت که فاصله استفاده مجدد برای آن محاسبه نمی‌گردد و صرفا به عنوان چپ‌ترین عنصر به درخت اضافه می‌شود. اگر این آدرس در درخت موجود بود، فاصله مجدد را به این صورت محاسبه می‌کنیم که از آن راس به سمت ریشه می‌رویم و هر بار که راس فعلی فرزند راست پدرش بود، آنگاه سایز زیر درخت سمت چپ پدر را به فاصله استفاده مجدد اضافه می‌کنیم (سایز زیر درخت سمت چپ خود راس را نیز در این مجموع محاسبه می‌کنیم). بعد از اتمام دسترسی ها به حافظه مجموع همه این مقادیر را به عنوان کل فاصله استفاده مجدد خروجی و میانگین آنها را به عنوان میانگین فاصله استفاده مجدد خروجی می‌دهیم.

۲-۱-۳ توابع پیاده سازی شده

فهرست توابعی که دراین الگوریتم نوشته شده‌اند به همراه عملکرد آنها به صورت زیراست:

- insert: یک عنصر به درخت اضافه می‌کند.
- deleteNode: یک عنصر از درخت حذف می‌کند.
- searchByMemoryAccess: با استفاده از hashmap که ذکر کردیم مقدار key برای یک رای با memoryAccess داده شده را برمی‌گرداند.
- inorderTraversal: پیمایش inorder درخت را خروجی می‌دهد.
- printInorderTraversal: پیمایش میان‌ترتیب را چاپ می‌کند.
- reuseDistance: مقدار فاصله استفاده مجدد را برای یک آدرس طبق روش ذکر شده در قسمت بالا محاسبه می‌کند.

- readMemoryAccessFromCSV: دسترسی های حافظه را از روی فایل می‌خواند و آن را در یک vector ذخیره می‌کند.

خروجی های نهایی:

۱. count: تعداد reuse distance ها است.

۲. sum: مجموع مقدار reuse distance ها برای همه آدرس ها است.

۳. average: میانگین محاسبه شده برای reuse distance ها است.

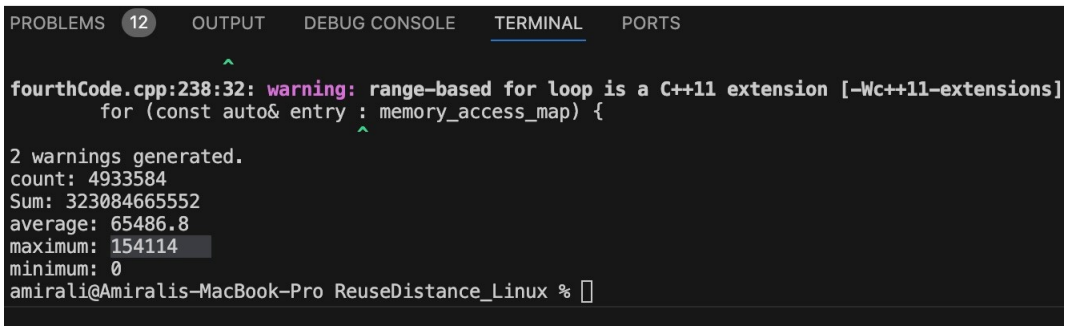
۴. maximum: بیشترین مقدار reuse distance است. (بخش امتیازی)

۵. minimum: کمترین مقدار reuse distance است. (بخش امتیازی)

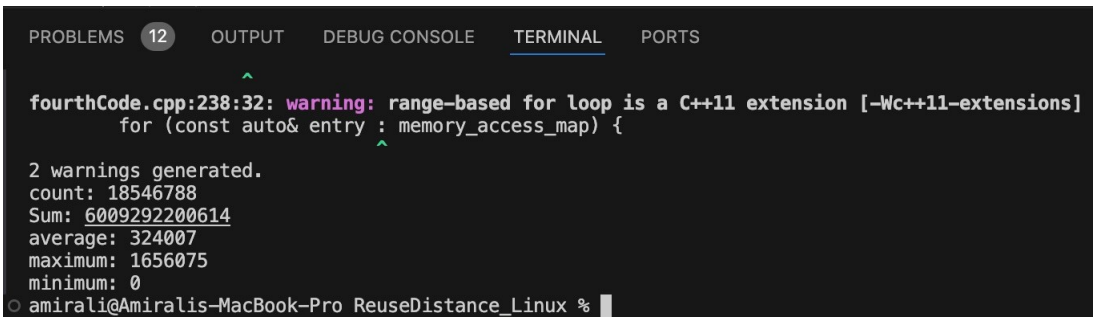
فصل ۳

نتیجه‌گیری

در ادامه تصاویر مربوط به خروجی برنامه توضیح داده شده برای فایل‌های Alibaba آمده است



شکل ۳-۱: خروجی برنامه برای فایل A۴۲



شکل ۳-۲: خروجی برنامه برای فایل A۱۰۸

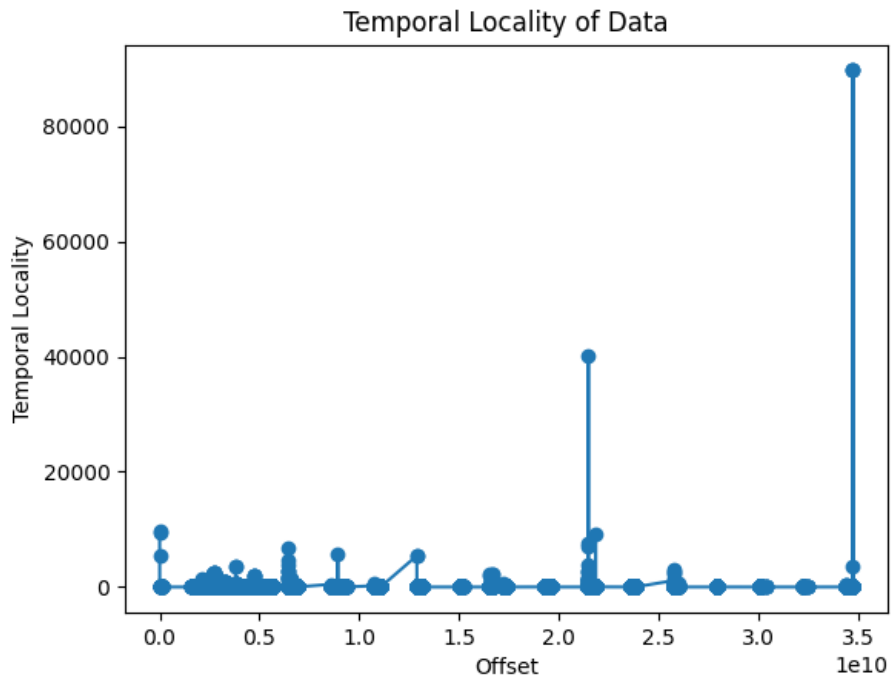
```
                ^
fourthCode.cpp:238:32: warning: range-based for loop is a C++11 extension [-Wc++11-extensions]
        for (const auto& entry : memory_access_map) {
                               ^
2 warnings generated.
count: 14713296
Sum: 3461403564695
average: 235257
maximum: 2973720
minimum: 0
amirali@Amiralis-MacBook-Pro ReuseDistance_Linux %
```

شکل ۳-۳: خروجی برنامه برای فایل A۱۲۹

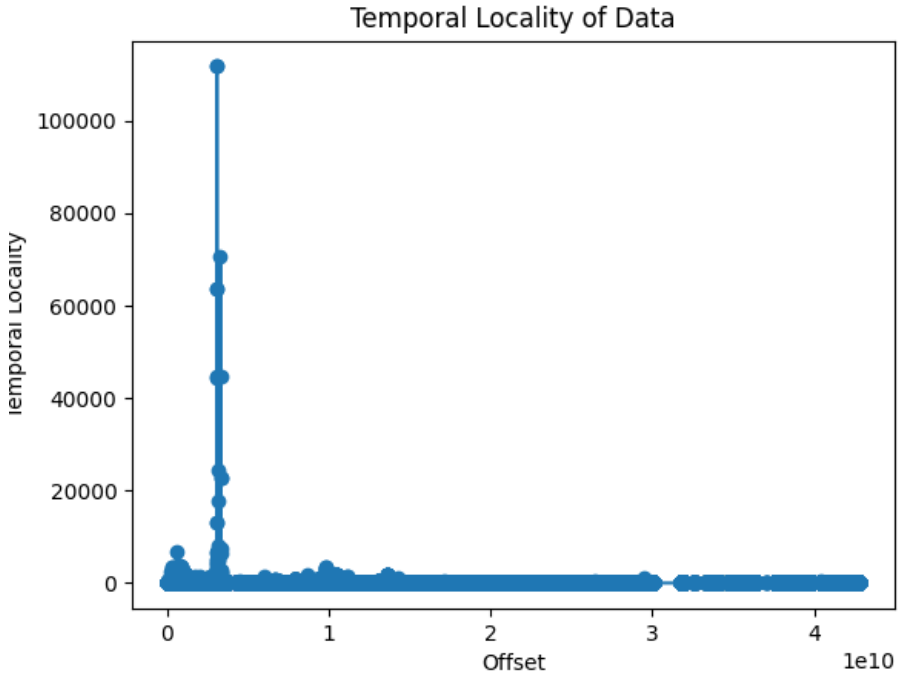
```
PROBLEMS 12   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

                ^
fourthCode.cpp:238:32: warning: range-based for loop is a C++11 extension [-Wc++11-extensions]
        for (const auto& entry : memory_access_map) {
                               ^
2 warnings generated.
count: 27065802
Sum: 197374542265
average: 7292.4
maximum: 178209
minimum: 0
```

شکل ۳-۴: خروجی برنامه برای فایل A۶۶۹

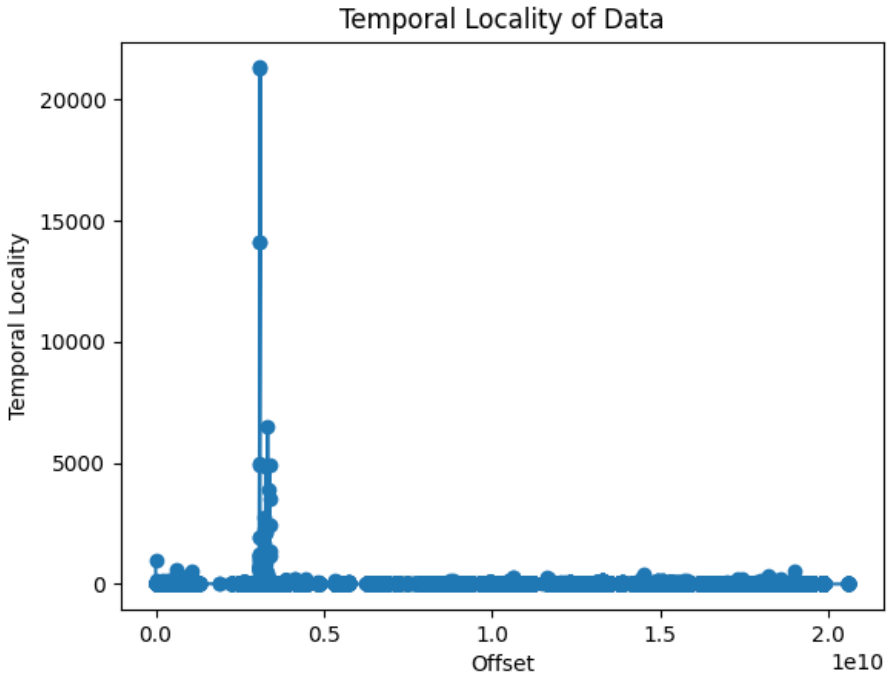


شکل ۳-۵: نمودار محلیت زمانی فایل A۴۲

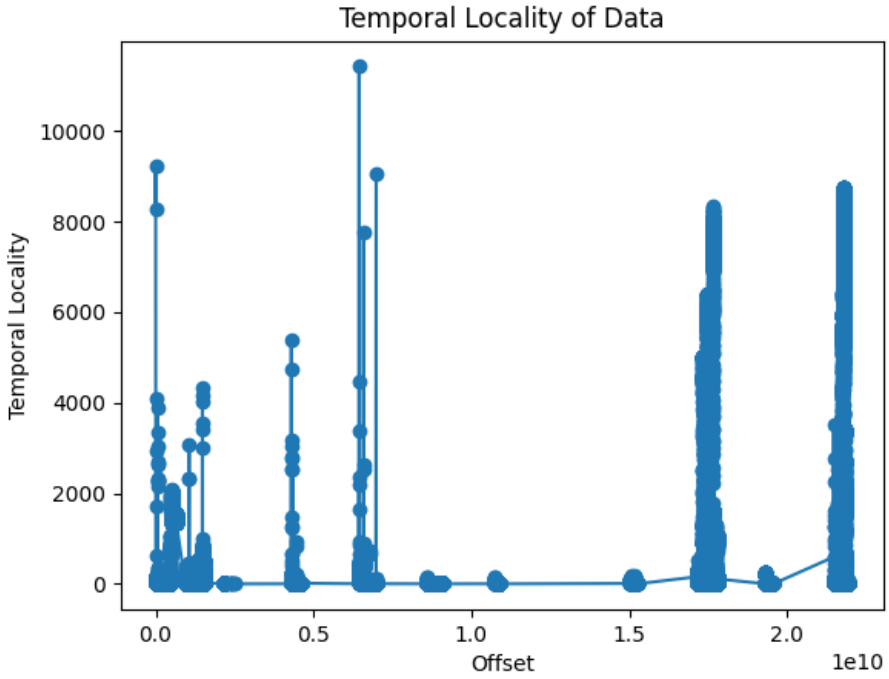


شکل ۳-۶: نمودار محلیت زمانی فایل ۱۰۸A

از تصاویر فوق می توان استنباط کرد که هر چه average reuse distance بیشتر باشد آنگاه temporal locality یا همان محلیت زمانی کمتر است زیرا می دانیم هر چه عدد محاسبه شده بیشتر باشد به این معنی است که به طور میانگین فاصله دو آدرس یکسان بیشتر بوده است که در واقع طبق تعریف بر کم شدن محلیت زمانی حکم می کند. همچنین می دانیم reuse distance هیچ ارتباطی با spatial locality یا محلیت فضایی ندارد زیرا دسترسی به آدرس های مجاور باعث افزایش محلیت فضایی می شود درحالیکه reuse distance مرتبط با آدرس های یکسان است پس این دو هیچ ارتباطی با یکدیگر ندارند.



شکل ۳-۷: نمودار محلیت زمانی فایل A۱۲۹



شکل ۳-۸: نمودار محلیت زمان فایل A۶۶۹