دانشگاه صنعتی امیرکبیر

(پلی‌تکنیک تهران)

ریاضی و علوم کامپیوتر

پایان‌نامه کارشناسی

رشته‌ی علوم کامپیوتر

عنوان

پیاده‌سازی الگوریتمی مبتنی بر خط جارو برای سلسله مراتب دوایر تو در تو

نگارش

امیررضا پوراخوان

استاد راهنما

دکتر امین غیبی

داور

دکتر علی محدث خراسانی

اسفند ۹۹

صفحه فرم ارزیابی و تصویب پایان نامه- فرم تأیید اعضاء کمیته دفاع

در این صفحه فرم دفاع یا تایید و تصویب پایان نامه موسوم به فرم کمیته دفاع - موجود در پرونده آموزشی - را قرار دهید.

تعهدنامه اصالت اثر

اینجانب امیررضا پوراخوان متعهد می‌شوم که مطالب مندرج در این پایان نامه حاصل کار پژوهشی اینجانب تحت نظارت و راهنمایی استادان دانشگاه صنعتی امیرکبیر بوده و به دستاوردهای دیگران که در این پژوهش از آنها استفاده شده است مطابق مقررات و روال متعارف ارجاع و در فهرست منابع و مآخذ ذکر گردیده است. این پایان نامه قبلاً برای احراز هیچ مدرک هم‌سطح یا بالاتر ارائه نگردیده است.

در صورت اثبات تخلف در هر زمان، مدرک تحصیلی صادر شده توسط دانشگاه از درجه اعتبار ساقط بوده و دانشگاه حق پیگیری قانونی خواهد داشت.

کلیه نتایج و حقوق حاصل از این پایان نامه متعلق به دانشگاه صنعتی امیرکبیر می‌باشد. هرگونه استفاده از نتایج علمی و عملی، واگذاری اطلاعات به دیگران یا چاپ و تکثیر، نسخه‌برداری، ترجمه و اقتباس از این پایان نامه بدون موافقت کتبی دانشگاه صنعتی امیرکبیر ممنوع است. نقل مطالب با ذکر مآخذ بلامانع است.

امیررضا پوراخوان

با تشکر از

- او که در درگاهش هزاران از ما بهتر دارد.
- پدر و مادرم که مسیر زندگیم به آن‌ها گره خورده بود و همسرم که مسیر زندگی‌شان را به من گره زدند.
- دکتر غیبی بابت مزاحمت‌هایم در این یک سال بابت پروژه و پیش‌تر بابت هزار و یک مسئله‌ی دیگر.
- دکتر محدث بابت داوری این پروژه.
- دیگر استادان عزیز گروه علوم کامپیوتر که خاطراتی شیرین از تحصیل برایم ساختند.
- و جواد کریمی که همراهی جدانشدنی‌ست.

چکیده

ما در این مقاله تلاش می‌کنیم الگوریتمی مبتنی بر خط جارو برای یافتن سلسله مراتب دوایر تو در تو [۱] ارائه دهیم. در این مسئله تعدادی دایره در ورودی به ما داده شده است، خروجی برنامه باید یک درخت باشد که سلسله مراتب دوایر را نشان دهد. یعنی اگر دایره‌ی i در صفحه درون دایره‌ی j قرار گرفته بود، در درخت نیز راس i در زیردرخت j باشد.

یافتن انگیزه‌ی حل این مسئله نیاز به تیزبینی ندارد. در دنیای اطراف ما مثال‌های بسیاری پیدا می‌شوند که ما سلسله‌ای از دوایر داریم و می‌خواهیم بدانیم هر دایره شامل چه دوایر دیگری می‌شود.

کلمات کلیدی:

خط جارو، هندسه محاسباتی، گراف درخت

فهرست مطالب

فهرست تصاویر

فهرست جداول

فصل ۱

مقدمه

ما در این مقاله تلاش می‌کنیم الگوریتمی مبتنی بر خط جارو برای یافتن سلسله مراتب دوایر تو در تو [۱] ارائه دهیم. در این مسئله تعدادی دایره در ورودی به ما داده شده است، خروجی برنامه باید یک درخت باشد که سلسله مراتب دوایر را نشان دهد. یعنی اگر دایره‌ی i در صفحه درون دایره‌ی j قرار گرفته بود، در درخت نیز راس i در زیردرخت j باشد.

در دنیای اطراف ما مثال‌های بسیاری پیدا می‌شوند که ما سلسله‌ای از دوایر داریم و می‌خواهیم بدانیم هر دایره شامل چه دوایر دیگری می‌شود.

۱-۱ کاربردها

این مسئله در مسائل کاربردی مختلفی ظاهر می‌شود. برای مثال برای تحلیل حرکت ذرات کلوئیدی در ساخت سرامیک‌ها [۲]. ذرات کلوئیدی در ابتدا در حامل (جزئی که مخلوط شونده را در بر می‌گیرد و معمولاً درصد بیشتری از مخلوط را تشکیل می‌دهد) پراکنده هستند. با گذشت زمان برخی از این ذرات ممکن است با یک‌دیگر ادغام شوند. تحلیل رفتار این ذرات برای متخصصان این حوزه حائز اهمیت است. برای مثال در ابتدا تعدادی از ذرات پخش‌شونده جدا از هم هستند و بعد از گذشت زمان در یک‌دیگر ادغام می‌شوند. با روش گفته شده می‌توان یافت که کدام مجموعه از ذرات با یک‌دیگر ادغام شده اند.

از دیگر کاربردهای مهندسی یافتن سلسله مراتب دوایر متداخل می‌توان به امولسیون آب در روغن در آب اشاره کرد [۳]. در این امولسیون ذرات آب در روغن‌هایی معلق هستند که آن‌ها هم در آب معلقند. برای تشخیص سریع این که هر ذره‌ی آب در کدام کلونی از ذرات روغن قرار دارد می‌توان از روش گفته شده استفاده کرد.

۱-۲ کارهای مشابه

۱-۲-۱ ورنوی وزن‌دار

در نمودار ساده‌ی ورنوی، مجموعه‌ای از نقاط به نام S داریم که فضا را به |S| ناحیه تقسیم می‌کنند. هر ناحیه حول یکی از اعضای S به وجود آمده است. ناحیه‌ی هر نقطه از S مثل p شامل همه‌ی نقاطی مثل q است که p نزدیک‌ترین نقطه بین اعضای S به q باشد.

در ورنوی وزن‌دار تفاوت اندکی وجود دارد. فرض کنید نقطه‌ها فروشگاه‌هایی باشند و ناحیه‌ی مربوط به هر فروشگاه خانه‌هایی را نشان بدهد که به این فروشگاه نزدیک‌ترین و لذا از آن خرید می‌کنند. اما قضیه جایی متفاوت می‌شود که برخی از فروشگاه‌ها جذاب‌تر هستند یا قیمت‌هایی پایین‌تری دارند. در این صورت ممکن است خریداری فاصله‌ی بیشتری را طی کند تا به فروشگاه بهتر برسد. این ایده به طرح ورنوی وزن‌دار منتهی می‌شود. در ورنوی وزن‌دار هر نقطه در مجموعه علاوه بر مختصات مشخصه‌ی وزن هم دارد.

حال فاصله‌ی نقطه‌ی q از یکی از اعضای S مثل p حاصل تقسیم فاصله‌ی واقعی آن‌ها بر وزن p است. یعنی هر چقدر وزن یک نقطه بیشتر شود ناحیه‌ی آن بزرگ‌تر می‌شود. در مثال فروشگاه‌ها، فروشگاه جذاب‌تر وزن بیشتری دارد.

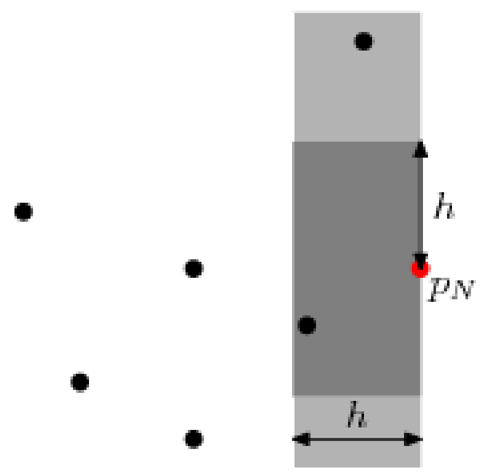
۲-۲-۱ خط جارو و کاربردهای آن

خط جارو روشی‌ست که در آن صفحه به ترتیب خاصی جارو می‌شود و تعدادی واقعه به این ترتیب بررسی می‌شود. در مسئله‌ی ما (پیدا کردن سلسله مراتب دوایر تو در تو) صفحه از چپ به راست توسط خط جارو پیمایش می‌شود. در واقع به ترتیب از x های کوچکتر به x های بزرگ‌تر. هم‌اکنون به بررسی چند مسئله‌ای که با استفاده از این روش حل می‌شود می‌پردازیم.

یافتن نزدیک‌ترین جفت نقطه

مجموعه‌ای از نقاط داده شده، نزدیک‌ترین جفت را پیدا کنید (معیار ما می‌تواند فاصله‌ی اقلیدسی یا منهتنی باشد). با در نظر گرفتن همه جفت‌ها می‌توان این مسئله را در زمان $\mathcal{O}(N^۲)$ حل کرد ، اما حربه‌ی خط جارو می‌تواند این مقدار را به $\mathcal{O}(NlogN)$ کاهش دهد.

فرض کنید ما نقاط ۱ تا N − ۱ را پردازش کرده‌ایم (به ترتیب X) و کمترین فاصله بین دو نقطه که تاکنون پیدا کرده‌ایم h است. اکنون نقطه‌ی امN را پردازش می‌کنیم و سعی می‌کنیم نقطه نزدیک‌تر از h را به آن پیدا کنیم. همان‌طور که در مستطیل خاکستری روشن نشان داده شده است، ما مجموعه‌ای از تمام نقاط پردازش شده‌ای را که مختصات X آنها در فاصله‌ی h از نقطه N قرار دارند، نگه می‌داریم. وقتی هر نقطه پردازش می شود، به مجموعه اضافه می‌شود و وقتی به نقطه‌ی بعدی برویم یا وقتی h کاهش یابد، تعدادی نقطه از مجموعه حذف می شوند – آن‌هایی که فاصله‌شان با خط جارو از h بیشتر شده است. مجموعه توسط مختصات y مرتب می‌شود. یک درخت دودویی خودمتوازن برای این کار مناسب است و ضریب log N را اضافه می‌کند.



شکل ۱-۱: یافتن نقطه‌ای با فاصله‌ی کمتر از h با نقطه‌ی N

برای جستجوی نقاط نزدیک‌تر از h به نقطه N ، فقط باید نقاط مجموعه فعال (خاکستری) را در نظر بگیریم و علاوه بر این فقط باید نقاطی را در نظر بگیریم که مختصات y آنها در محدوده yN − h تا yN + h قرار داشته باشند (در مستطیل خاکستری تیره). این محدوده را می توان از مجموعه مرتب شده در زمان $\mathcal{O}(logN)$ استخراج کرد ، اما مهمتر اینکه تعداد عناصر $\mathcal{O}(۱)$ است (به سیستم اندازه‌گیری فاصله‌ی مورد استفاده بستگی دارد) ، زیرا فاصله‌ی بین هر دو نقطه در مجموعه حداقل h است. نتیجه این است که جستجوی هر نقطه به زمان $\mathcal{O}(logN)$ نیاز دارد که در مجموع پیچیدگی زمانی $\mathcal{O}(NlogN)$ را به دست می‌دهد.

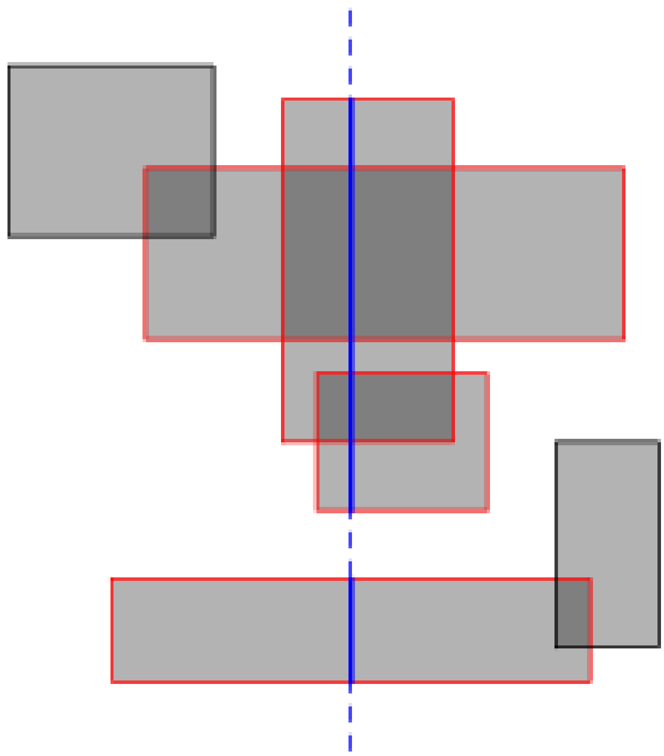
یافتن اجتماع مستطیل‌ها

تعدادی مستطیل با اضلاع موازی با محورها در صفحه به ما داده شده اند، هدف یافتن اجتماع این مستطیل‌هاست. برای مثال بایستی مساحت اجتماع آن‌ها را بیابیم.

دادهساختاری داریم که آرایه‌ای فرضی به نام a دارد و دو نوع پرسش را پاسخ می‌دهد:

- به ازای L, R ورودی، به ازای هر i در بازه‌ی L تا R به a_i یک واحد اضافه یا کم می‌کند.
- تعداد عناصری از آرایه که از صفر بزرگ‌ترند را برمی‌گرداند.

با فرض داشتن این داده‌ساختار به حل مسئله می‌پردازیم. خط جارو را از چپ به راست حرکت می‌دهیم. اتفاق‌ها به دو نوعند؛ اضافه شدن یا حذف شدن یک مستطیل. مستطیل (sx, ex, sy, ey) را در نظر بگیرید که گوشه‌های مخالف آن (sx, ex) و (sy, ey) هستند. هنگامی که به sx می‌رسیم بایستی این مستطیل را اضافه کنیم. بدین منظور در داده‌ساختار به بازه‌ی sy تا ey یکی اضافه می‌کنیم. همچنین وقتی به ey می‌رسیم یکی از این بازه می‌کاهیم.



شکل ۲-۱: حرکت خط جارو بر روی اجتماع مستطیل‌ها. به ناحیه‌ی پوشیده شده از خط‌جارو توجه کنید.

بپردازیم به محاسبه‌ی مساحت. می‌دانیم بین هر دو اتفاق متوالی طولی از خط جارو که توسط اجتماع مستطیل‌ها پوشیده می‌شود تغییری نمی‌کند. لذا می‌توانیم هنگام جا به جا شدن از یک اتفاق به اتفاق بعدی طول پوشیده شده از خط جارو را در فاصله‌ی بین این دو اتفاق ضرب کنیم تا مساحت ناحیه‌ی پیموده شده به دست آید. برای یافتن طول پوشیده شده از خط جارو از پرسش نوع دوم داده‌ساختار کمک می‌گیریم.

یافتن پایین‌پوش

در این مسئله تعدادی خط به ما داده شده است که هیچ‌یک عمودی نیستند. هدف یافتن پوش پایینی (lower envelope) این خطوط است. به بیان دیگر می‌خواهیم به ازای هر x بدانیم کدام خط کمترین y را به ازای این x دارد.

خطوط را بر حسب شیب‌شان از زیاد به کم مرتب می‌کنیم. برای حل این مسئله خط جارو را از چپ به راست حرکت می‌دهیم. یک اتفاق برابر است با این که خط بعدی که در لیست مرتب شده‌ی خطوط قرار دارد گوی رقابت را از خطی که در حال حاضر پایین‌ترین است ببرد. بدین ترتیب تا آخر پیش می‌رویم و بعد از رسیدن خط جارو به آخرین عضو (که کمترین شیب را دارد) به کار خود پایان می‌دهیم.

یافتن تعداد چهارضلعی‌های شامل مبدا

تعدادی نقطه به ما داده می‌شود و تعداد چهارضلعی‌هایی که رئوس‌شان از این نقاط انتخاب شده باشند و مبدا درون آن‌ها قرار بگیرد پرسیده می‌شود.

این یک کاربرد متفاوت از خط جاروست. برای حل این مسئله حرکت دادن خط جارو از چپ به راست مفید واقع نمی‌شود. در عوض باید خط جارو را به شکل یک نیم‌خط که یک سر آن در مبدا قرار دارد و سر دیگر آن که تا بی‌نهایت ادامه دارد صفحه را جارو کرد. در واقع نقاط بر حسب زاویه‌ای که با محور x ها می‌سازند مرتب می‌شوند.

برای حل این مسئله توجه به این نکته لازم است که شرط لازم و کافی برای این که چهار نقطه مبدا را نپوشانند این است که همه‌ی آن‌ها در یک نیم‌صفحه‌ی مبداگذر باشند. در واقع باید خطی مبداگذر وجود داشته باشد که همه‌ی این چهار نقطه یک سمت آن قرار بگیرند. پس ما حالات نامطلوب را می‌شماریم و از حالات کل کم می‌کنیم.

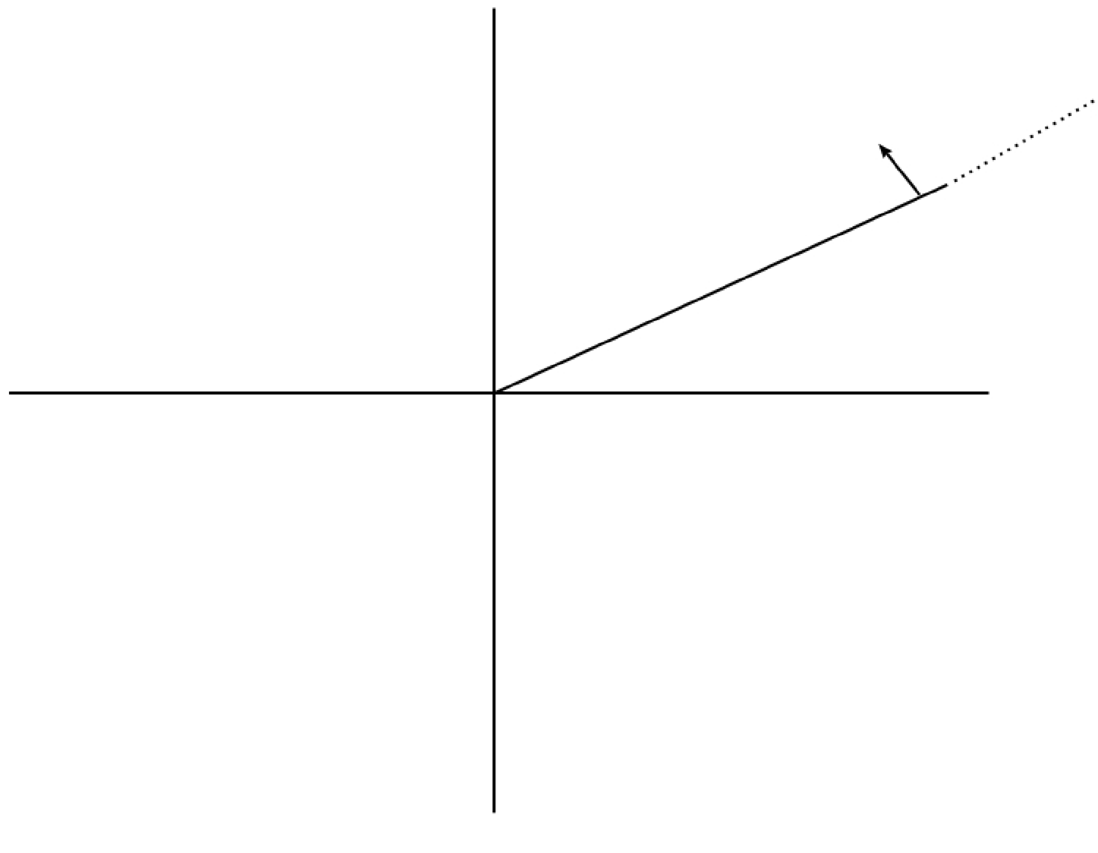
برای شمردن حالات نامطلوب کافی‌ست تعداد چهارتایی‌هایی از نقاط را بشماریم که در یک نیم‌صفحه قرار می‌گیرند. حربه‌ی خط جارو در این جا به ما کمک می‌کند. ما به ترتیب زاویه نقاط را با خط جارو می‌بینیم و تعداد آن‌ها را به خاطر می‌سپاریم. خط جاروی دیگری موسوم به خط ساحلی پس از این که نیمی از صفحه را طی کردیم (در واقع زاویه‌ی خط جارو به ۱۸۰ درجه رسید) کار خود را آغاز می‌کند و نقاطی که زاویه‌ای که با خط جارو می‌سازند از ۱۸۰ بیشتر است را حذف می‌کند. بعد از هر مرحله تعداد ۴ تایی‌هایی از نقاط که درون نیم صفحه‌ی ما هستند را به حالات نامطلوب اضافه می‌کنیم.

یافتن برخورد پاره‌خط‌ها

تعدادی پاره‌خط در صفحه داده شده و بایستی نقاط برخورد آن‌ها را گزارش کنیم. خط جارو از چپ به راست حرکت می‌کند. هر لحظه پاره‌خط‌هایی که با خط جارو برخورد دارند را به صورت مرتب شده از پایین‌ترین به بالاترین نگه‌داری می‌کنیم. برای این کار از یک درخت دودویی خودمتوازن استفاده می‌کنیم.

اتفاق‌ها از سه نوعند.

- یک پاره‌خط افزوده می‌شود.
- یک پاره‌خط حذف می‌شود.
- دو پاره خط برخورد می‌کنند.



شکل ۳-۱: نحوه‌ی حرکت خط جارو

به هنگام اضافه شدن یک پاره‌خط آن را در داده‌ساختار اضافه می‌کنیم. حذف هم بدیهی‌ست. به هنگام برخورد دو پاره‌خط جای آن‌ها در درخت دودویی با هم عوض می‌شود و نیاز داریم که این دو کلید را در داده‌ساختار جا به جا کنیم.

فصل ۲

شرح مسئله

ورودی مسئله دوایر $c_1, c_2, \ldots, c_n$ هستند که هیچ برخوردی یا لمسی با یک‌دیگر ندارند اما ممکن است یکی درون دیگری باشد. خروجی راه‌حل یک درخت $n + 1$ راسی‌ست که رئوس آن از صفر تا n شماره‌گذاری شده‌اند. راس صفر متناظر با دایره‌ای فرضی‌ست که تمام صفحه را در برمی‌گیرد و رئوس یک تا n متناظر با دوایر ۱ تا n هستند. راس i در زیردرخت راس j قرار دارد اگر و فقط اگر دایره‌ی j در درون دایره‌ی i قرار گرفته باشد. بدین ترتیب واضح است که همه‌ی رئوس ۱ تا n در زیردرخت راس صفر قرار می‌گیرند پس راس صفر ریشه‌ی درخت خروجی‌ست.

برای درک بهتر یک راه‌حل شهودی برای مسئله را شرح می‌دهیم. ابتدا یک راس به عنوان ریشه قرار می‌دهیم. دوایری در صفحه که دایره‌ی دیگری شامل‌شان نیست فرزندان ریشه هستند. سپس دوایری که مستقیماً درون این دوایر قرار دارند نوه‌های ریشه هستند. در واقع به ازای هر راس، فرزندان آن دوایری هستند که مستقیماً و بدون واسطه درون دایره‌ی متناظر این راس قرار می‌گیرند.

برای دیدن چند مثال به فصل ۵ رجوع کنید.

فصل ۳

داده‌ساختار مورد استفاده

در مقاله‌ی اصلی به عنوان داده ساختار برای نگه‌داری بازه‌ها از درخت قرمز سیاه [۴] استفاده شده است. درخت قرمز سیاه می‌تواند عملیات‌های حذف، جست‌وجو و افزودن را در زمان لگاریتمی انجام دهد. به همین دلیل است که این درخت برای نگه‌داری بازه‌ها استفاده شده است.

این در حالی‌ست که هر داده‌ساختاری که بتواند سه عملیات افزودن، جست‌وجو و حذف را در زمان لگاریتمی انجام دهد می‌تواند مورد استفاده قرار گیرد. برای مثال می‌توان به درخت‌های خودمتوازن زیر اشاره کرد:

- درخت ای‌وی‌ال [۵]

  در این درخت دودویی هر راس یک ضریب توازن دارد که ارتفاع زیردرخت راست منهای ارتفاع زیردرخت چپ است. منظور از ارتفاع، یک زیردرخت، بیشینه فاصله‌ی یک راس در این زیردرخت از ریشه‌ی زیردرخت است.

  درخت ای‌وی‌ال تلاش می‌کند ضریب توازن را همیشه عددی بین ۱ و −۱ نگه دارد. بدین منظور با چرخش‌هایی همیشه ارتفاع زیردرخت چپ و راست را متوازن نگه می‌دارد. در این درخت ارتفاع همیشه از $\mathcal{O}(\log n)$ می‌ماند.

- درخت ۳-۲ [۶، صفحه‌ی ۵۰۴]

  این درخت سه نوع راس دارد. راس ۲، راس ۳ و برگ. محتویات فقط در برگ‌ها ذخیره می‌شوند و راس‌های دیگر برای مسیریابی هستند. راس ۲، دو بچه دارد و راس ۳، سه بچه دارد. در راس ۲، یک مقدار نگه‌داری می‌شود که مقادیر کوچک‌تر از آن در زیردرخت فرزند چپ نگه‌داری می‌شوند و مقادیر بزرگ‌تر در زیردرخت فرزند راست. در راس ۳ دو مقدار نگه‌داری می‌شود که بازه‌ی مقادیر نگه‌داری شده در سه فرزند را نشان می‌دهد.

  ارتفاع این درخت همیشه لگاریتمی باقی می‌ماند.

همچنین می‌توان از داده‌ساختارهایی استفاده کرد که به صورت سرشکن یا احتمالاتی در زمان لگاریتمی جواب می‌دهند:

- تریپ [۷]

  این درخت ترکیبی از هرم (Heap) و درخت جست‌وجوی دودویی‌ست. هر راس دارای دو کلید است که با کلید اول هرم ساخته می‌شود و با کلید دوم درخت جست‌وجوی دودویی. کلید اول به صورت تصادفی تولید می‌شود تا درخت را متوازن نگه دارد. تریپ دو عملیات اصلی دارد: جداسازی و ادغام. در عملیات جداسازی با یک مقدار خاص تریپ به دو تریپ بریده می‌شود؛ مقادیر کوچک‌تر در یک تریپ و مقادیر بزرگ‌تر در تریپ دیگر قرار می‌گیرند. در عملیات ادغام دو تریپ به یک تریپ تبدیل می‌شوند. ارتفاع این ساختمان داده به صورت احتمالاتی همیشه $\mathcal{O}(\log n)$ باقی می‌ماند. دیگر عملیات‌ها را می‌توان با استفاده از این دو عملیات ساخت. برای مثال برای درج، می‌توان ابتدا تریپ را برید و سپس این دو تریپ را با تریپ تک‌عنصری شامل عنصر جدید ادغام کرد.

- فهرست پرشی [۸]

یک لیست پیوندی را در نظر بگیرید. به صورت تصادفی نیمی از این لیست را انتخاب کنید و لیست پیوندی جدیدی بسازید که هر عنصر دارای یک اشاره‌گر به لیست قبلی باشد. این کار را تکرار کنید تا لیستی ساخته شود که دارای هیچ عنصری نیست. حال جست‌وجو روی این ساختمان داده به صورت احتمالاتی $\mathcal{O}(\log n)$ است. همچنین با همین حربه می‌توان عملیات‌های حذف و اضافه را هم به صورت احتمالاتی در زمان لگاریتمی انجام داد.

فصل ۴

راه‌حل و پیاده‌سازی

کدهای مربوطه در مخزن گیت‌هاب [1] قرار دارند.

۴-۱ ایده‌ی کلی

از چپ به راست خط جارو را حرکت می‌دهیم. هر دایره زمانی شروع می‌شود (توسط خط جارو دیده می‌شود) و زمانی پایان می‌یابد (یعنی خط جارو از روی آن رد می‌شود). داده‌ساختاری داریم که در آن بازه‌هایی از y که دوایر به خود اختصاص داده‌اند را در آن مدیریت می‌کنیم. همان‌طور که در بالا گفته شد یک دایره‌ی مجازی با شماره‌ی صفر نیز داریم که تمامی دوایر را در بر می‌گیرد. این دایره توسط خط جارو دیده نمی‌شود. یعنی نه زمانی شروع می‌شود و نه زمانی پایان می‌یابد. این دایره‌ی مجازی از همان ابتدا در داده‌ساختار قرار دارد.

با اضافه شدن یک دایره می‌توان تضمین کرد این دایره در بازه‌ی یکی از دوایر قبلی قرار می‌گیرد. فرض کنید دایره‌ی i ام به هنگام اضافه شدن در بازه‌ی دایره‌ی j قرار بگیرد. آن‌گاه در درخت نهایی ما راس j را پدر راس i قرار می‌دهیم. همچنین بازه‌ی مربوط به دایره‌ی i را می‌افزاییم و بازه‌ی مربوط به دایره‌ی j را به دو قسمت می‌شکنیم؛ قسمتی که بالای دایره‌ی i قرار دارد و قسمتی که پایین دایره‌ی i قرار دارد.

۴-۲ معرفی اشیاء استفاده شده

- ***Interval***

  این نشان دهنده‌ی یک بازه است که مجموعه‌ی intervals (که در ادامه توضیح داده می‌شود) هم حاوی اشیائی از این جنس است. موجودیت بازه حاوی یک owner_id است که شماره‌ی دایره‌ای‌ست که صاحب این بازه‌ست. در واقع اگر دایره‌ای در این بازه بیوفتد، مشخص می‌شود که پدر این دایره در درخت نهایی owner_id خواهد بود. همچنین موجودیت Interval شامل دو موجود از جنس Semicircle به نام‌های up و down است. این دو در واقع نشان‌دهنده‌ی مرزهای این بازه هستند.

- ***Semicircle***

  نشان‌دهنده‌ی یک نیم‌دایره است.

- ***Circle***

  نشان‌دهنده‌ی یک دایره است.

۴-۳ توضیح کامل

ابتدا اتفاقات خط جارو را به ترتیب x مرتب می‌کنیم. اتفاقات خط جارو شروع و یا پایان یک دایره هستند.

---

[1] ***https://github.com/ar-pa/circles-hierarchy***

حال یک std::set به نام intervals می‌سازیم که در آن بازه‌های تشکیل شده توسط دوایر را می‌ریزیم. اشیای درون این مجموعه از جنس Interval هستند. یک دایره‌ی مجازی داریم که شماره‌ی آن صفر است و به شکلی مقداردهی شده که تمامی دوایر دیگر را می‌پوشاند. برای شروع مجموعه‌ی intervals تنها حاوی بازه‌ی این دایره است. یعنی owner_id آن برابر صفر و up و down آن به نیم‌دایره‌ی بالا و پایین دایره‌ی شماره‌ی صفر اشاره می‌کنند.

حال شرح می‌دهیم که اعضای intervals به چه ترتیبی مرتب‌شده نگه‌داری می‌شوند. در هر لحظه ما یک sweep_line_x داریم که نشان می‌دهد خط جارو در چه نقطه‌ای است. در ابتدای کار این مقدار برابر با مقداری بسیار منفی‌ست. حال برای ترتیب دادن به بازه‌ها در مجموعه‌مان، ابتدا نیم‌دایره‌ی پایینی آن‌ها را با هم مقایسه می‌کنیم و سپس در صورت تساوی نیم‌دایره‌ی بالایی‌شان را با هم مقایسه می‌کنیم. برای مقایسه‌ی دو نیم دایره، به مقدار sweep_line_x نیاز داریم. با استفاده از این مقدار درمی‌یابیم که به ازای sweep_line_x = x کدام یک از این نیم‌دایره‌ها پایین‌ترند. در واقع بازه‌ها به ترتیب از پایین به بالا نگه‌داری می‌شوند.

هنگامی که یک دایره را اضافه می‌کنیم، ابتدا با استفاده از lower_bound پدر این دایره را می‌یابیم. دایره به هنگام اضافه شدن، روی خط جارو فقط یک نقطه می‌سازد. همان‌طور که پیش‌تر گفتیم، خط جارو به تعدادی بخش تقسیم شده است که این بخش‌ها در مجموعه‌ی intervals به ترتیب از پایین به بالا موجودند. هر یک از این بخش‌ها مربوط به یکی از دایره‌ها هستند. کافی‌ست بیابیم دایره‌ی جدید (که اکنون فقط یک نقطه است) در کدام یک از این بازه‌ها قرار می‌گیرد. منظور از lower_bound تابع std::set::lower_bound است. این تابع اولین عنصری در مجموعه که از عنصر داده‌شده‌ی ما بزرگتر-مساوی باشد را برمی‌گرداند.

ما دایره‌ی جدید را ایجاد می‌کنیم و همچنین شی Interval مربوط به آن را هم می‌سازیم. سپس با استفاده از این Interval (که در واقع در sweep_line_x فعلی تنها یک نقطه است) تابع intervals.lower_bound را صدا می‌کنیم. اشاره‌گری که برگردانده می‌شود دقیقاً بازه‌ی بعدیِ بازه‌ی پدر این دایره است. چرا؟ بازه‌ها به ترتیب و بدون فاصله از مقدار بسیار منفی تا بسیار مثبت در intervals گسترش یافته‌اند. پایان یک بازه دقیقاً شروع بازه‌ی بعدی‌ست. با ترتیب ما، اولین (کوچک‌ترین).

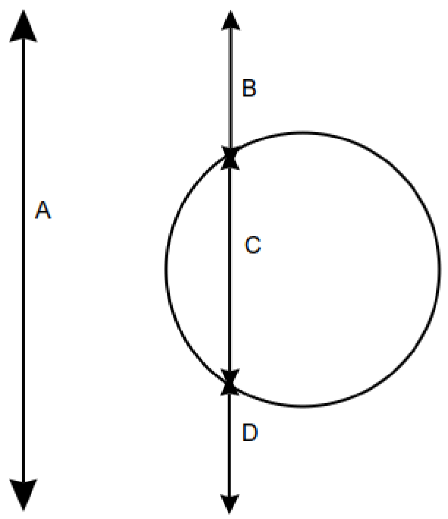
توجه کنید به دلیل اضافه شدن یک دایره‌ی مجازی همه‌ی دوایر پدر دارند. حال بازه‌ی مربوط به پدر (بازه‌ی A در شکل ۱-۴) را حذف کرده و سه بازه به جای آن اضافه می‌کنیم. بازه‌ای از پدر که پایین دایره‌ی جدید قرار دارد، بازه‌ای از پدر که بالای دایره‌ی جدید قرار دارد و بازه‌ی جدیدی که توسط این دایره تشکیل شده است. به صورت دقیق‌تر، سه بازه‌ی اضافه شده بدین شرحند:

- بازه‌ی بالایی پدر: up این بازه همان up بازه‌ی قبلی پدر است. down این بازه برابر است با نیم‌دایره‌ی بالایی دایره‌ی جدید. بازه‌ی B در شکل ۱-۴ را ببینید.

- بازه‌ی دایره‌ی جدید: up این بازه نیم‌دایره‌ی بالایی دایره‌ی جدید است. down این بازه برابر است با نیم‌دایره‌ی بالایی دایره‌ی جدید. بازه‌ی C در شکل ۱-۴ را ببینید.

- بازه‌ی پایینی پدر: down این بازه همان down بازه‌ی قبلی پدر است. up این بازه برابر است با نیم‌دایره‌ی پایینی دایره‌ی جدید. بازه‌ی D در شکل ۱-۴ را ببینید.

هنگامی که یک دایره را حذف می‌کنیم نیز ابتدا با استفاده از lower_bound پدر آن را می‌یابیم. بازه‌های مربوط به پدر دو طرف بازه‌ی دایره‌ی در حال حذف شدن را گرفته‌اند. کاری که ما باید انجام بدهیم این است که بازه‌ی فرزند



شکل ۱-۴: اضافه شدن یک دایره

را حذف و بازه‌های پدر را ادغام کنیم. برای این کار بازه‌های پدر را حذف و یک بازه‌ی جدید که ادغام این دو بازه باشد را اضافه می‌کنیم.

پس از پایان این فرآیند ما به ازای هر دایره پدر آن را داریم و می‌توانیم درخت را بسازیم.

۴-۴ تحلیل زمانی

مرتب‌سازی اولیه $\mathcal{O}(n \log n)$ طول می‌کشد. فرآیند افزودن و حذف هر دایره $\mathcal{O}(\log n)$ طول می‌کشد. در مجموع زمان اجرای راه‌حل برابر است با $\mathcal{O}(n \log n)$.

۴-۵ شبه‌کد

ورودی: مجموعه دوایر $C = c_1, c_2, \ldots, c_n$ در صفحه.

خروجی: سلسله مراتب دوایر در صفحه.

۱: یک مجموعه‌ی بازه به نام I بساز. ابتدا بازه‌ی منفی بی‌نهایت تا مثبت بی‌نهایتِ متعلق به ریشه را در آن قرار بده. این مجموعه یکی از داده‌ساختارهای مطرح شده در فصل ۳ است.

۲: نقاط ابتدایی (چپ‌ترین) و انتهایی (راست‌ترین) دوایر را در آرایه‌ی E قرار بده.

۳: نقاط موجود در E را به ترتیب از چپ به راست (افزایش x) مرتب کن و یک فهرست اتفاق از آن‌ها بساز: $P = \{p_1, p_2, \ldots, p_{2n}\}$.

۴: به ازای هر نقطه در P مثل p انجام بده

۵: بازه‌ی $i = (s, e)$ که شامل نقطه‌ی p است را بیاب.

۶: اگر p یک نقطه‌ی ابتدایی‌ست آنگاه

۷: چاپ کن دایره‌ی صاحب i پدر دایره‌ای‌ست که p چپ‌ترین نقطه‌ی آن است.

۸: بازه‌ی i را حذف کن. دو بازه‌ی $(s, y(p))$ و $(y(p), e)$ را اضافه کن که صاحب‌شان همان صاحب i باشد. همچنین بازه‌ی $(y(p), y(p))$ را اضافه کن که صاحبش دایره‌ای‌ست که p متعلق به آن است.

۹: وگرنه

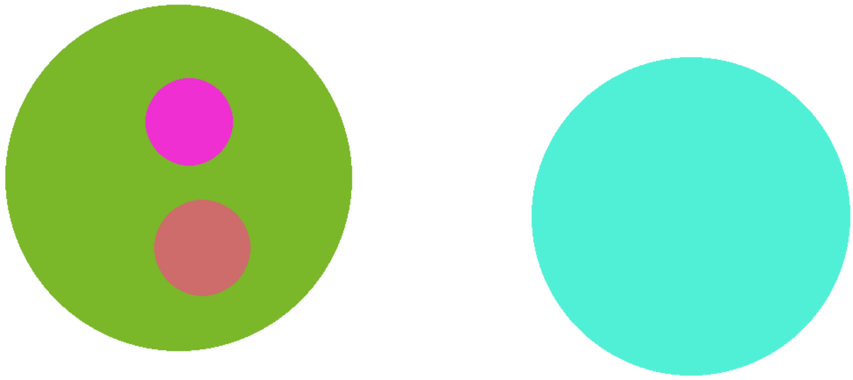
۱۰: بازه‌ی i را حذف کن. بازه‌ی قبل و بعد از i را ادغام کن.

۱۱: پایان شرط

۱۲: پایان حلقه

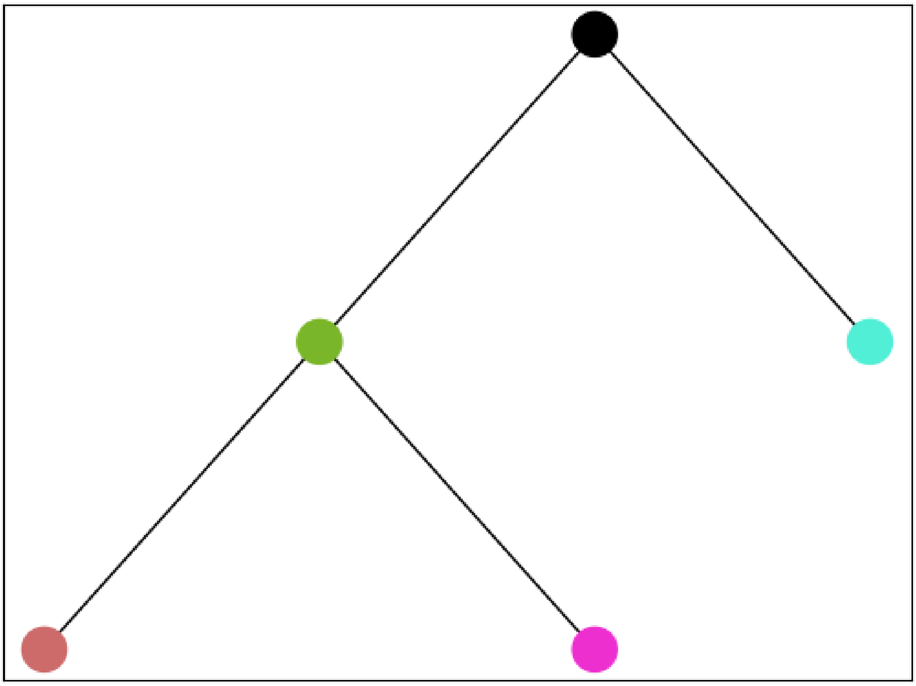
فصل ۵

تصاویری از محیط برنامه

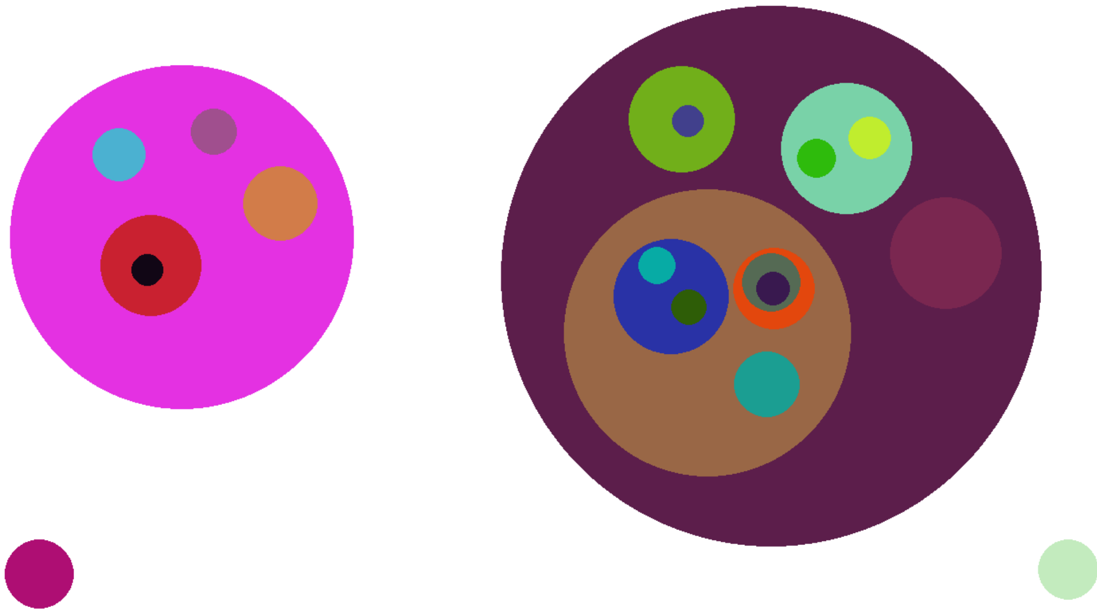


شکل ۱-۵: چهار دایره‌ی ساده

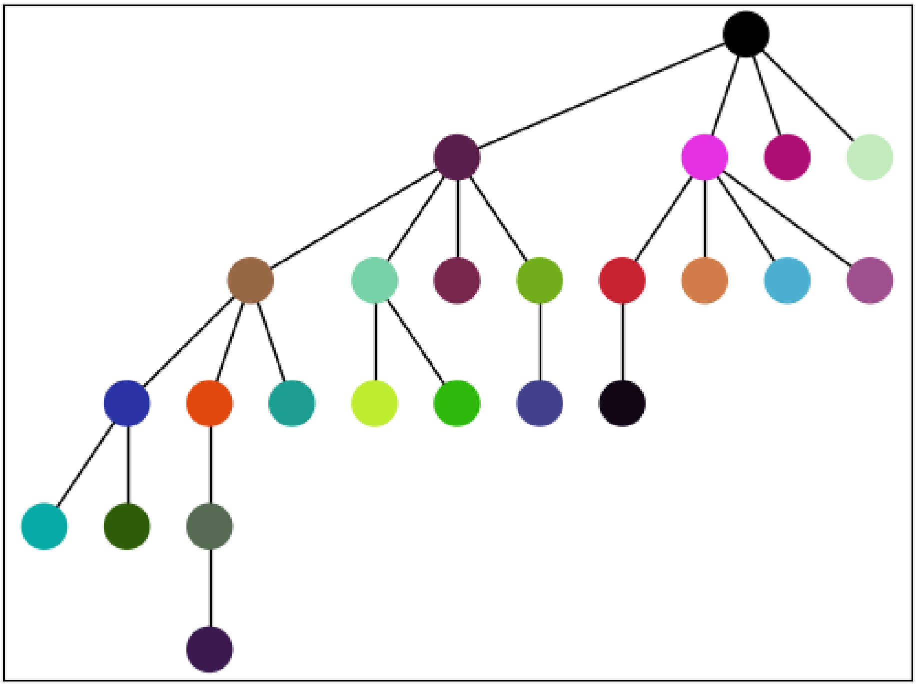
یک رابط کاربری پایتونی تهیه شده است که کار با برنامه را ساده‌تر کند. با رابط کاربری می‌توان دوایر را کشید و درخت نهایی را مشاهده کرد. شکل‌های ۱-۵، ۲-۵، ۳-۵ و ۴-۵ تصاویری از اجرای برنامه هستند.



شکل ۲-۵: درخت متناظر با شکل ۱-۵ . دقت کنید ریشه (که با رنگ سیاه نمایش داده می‌شود) برابر دایره‌ای بسیار بزرگ است که تمام شکل را در بر می‌گیرد.



شکل ۳-۵: تعداد بسیار زیادی دایره‌ی تو در تو



شکل ۴-۵: درخت متناظر با شکل ۳-۵

فصل ۶

نتیجه‌گیری

یافتن سلسله مراتب دوایر تو در تو مسائل متنوعی را در حوزه‌های مختلف حل می‌کند؛ از تحلیل حرکات ذرات کلوئیدی در ساخت سرامیک تا یافتن کلونی‌های آب در روغن.

ما در این مقاله سعی کردیم این مسئله را با الگوریتمی که در مقالهٔ اصلی [۱] آمده است پیاده‌سازی کنیم. ما در پیاده‌سازی خود از خط جارو کمک گرفتیم و با حرکت دادن خط جارو راس‌ها و یال‌هایی را به درخت افزودیم تا به حالت نهایی خود تبدیل شود.

دیدیم که داده‌ساختارها تا چه می‌توانند ما را در حل مسائل یاری دهند؛ به هنگام افزودن یک نیم دایره به مجموعه و پرسش این که آیا یک نقطهٔ دلخواه درون بازه‌ای از بازه‌های موجود در داده‌ساختار قرار دارد یا نه؟ اگر با داده‌ساختارها آشنا نبودیم حل این بخش از مسئله فقط در زمان خطی میسر می‌شد حال آن که ما این بخش را به کمک درخت‌های خودمتوازن در زمان لگاریتمی حل کردیم.

فهرست مراجع

[1] *Kim, Deok-Soo, Byunghoon Lee, and Kokichi Sugihara. ”A sweep-line algorithm for the inclusion hierarchy among circles.” Japan journal of industrial and applied mathematics 23.1 (2006): 127-138.*

[2] *Hong, Chu-Wan. ”From long-range interaction to solid-body contact between colloidal surfaces during forming.” Journal of the European Ceramic Society 18.14 (1998): 2159-2167.*

[3] *Oh, Chul, et al. ”O/W/O Multiple Emulsions via One-Step Emulsification Process.” Journal of dispersion science and technology 25.1 (2004): 53-62.*

[4] *Bayer, Rudolf. ”Symmetric binary B-trees: Data structure and maintenance algorithms.” Acta informatica 1.4 (1972): 290-306.*

[5] *Adelson-Velsky, E. M., and E. M. Landis. ”An algorithm for the organization of information. Soviet Math.” (1962).*

[6] *Cormen, Thomas H., et al. Introduction to algorithms. MIT press, 2009.*

[7] *Seidel, Raimund, and Cecilia R. Aragon. ”Randomized search trees.” Algorithmica 16.4 (1996): 464-497.*

[8] *Pugh, William. Concurrent maintenance of skip lists. 1998.*

Abstract

In this article, we try to present a sweep-line-based algorithm for nested circle hierarchy [1]. We are given a number of circles at the input, the output of the program should be a tree that shows the hierarchy of circles. That is, if the i-th circle was inside the j-th circle, in the output tree the i-th vertex must be inside the sub-tree of j-th vertex.

Finding the motivation to solve this problem does not require sharpness. There are many examples in the world around us that we have a series of circles and we want to know what other circles each circle contains.

Keywords:

Sweep-line, Computational geometry, Tree-graph

Amirkabir University of Technology

(Tehran Polytechnic)

Department of Mathematics and Computer Science

Bachelor Thesis

Computer Science Field

Title

Implementation of A Sweep-Line Algorithm for the Inclusion Hierarchy among Circles

By

AmirReza PourAkhavan

Supervisor

Dr. Amin Gheibi

Jury

Dr. Ali Mohades

May 2021