
بسم الله الرحمن الرحيم

دانشگاه اصفهان

دانشکده مهندسی کامپیوتر

گروه مهندسی نرم‌افزار

پایان‌نامه کارشناسی

رشته مهندسی کامپیوتر گرایش نرم‌افزار

عنوان پروژه

ابزاری تعاملی برای آموزش و مصورسازی الگوریتم‌ها و ساختمان داده‌ها

استاد راهنما:

دکتر آرش شفیعی

پژوهشگر:

پویا سادات الحسینی

شهریور ۱۴۰۴

دانشگاه اصفهان

دانشکده مهندسی کامپیوتر

گروه مهندسی نرم‌افزار

پروژه کارشناسی رشته‌ی مهندسی کامپیوتر گرایش نرم‌افزار

آقای پویا سادات الحسینی

تحت عنوان

ابزاری تعاملی برای آموزش و مصورسازی الگوریتم‌ها و ساختمان داده‌ها

در تاریخ / / **۱۴** توسط هیأت داوران زیر بررسی و با نمره به تصویب نهایی رسید.

۱- استاد راهنمای پروژه:

دکتر امضا

۲- استاد داور :

دکتر امضا

امضای مدیر گروه

تشکر و قدردانی

جای دارد تا از تمامی تلاش‌های استاد راهنمای خود، دکتر آرش شفیعی، تشکر ویژه به عمل آورم. بی‌شک بدون راهنمایی‌های ایشان این پروژه در وضعیت کنونی خود قرار نداشت.

همچنین از خانواده که در دوران تحصیل بنده را حمایت کرده‌اند و از همکاران خود که در ارتقای دانش تخصصی بنده بسیار به بنده در این مسیر کمک کرده‌اند بسیار متشکرم.

چکیده:

در دنیای امروز، الگوریتم‌ها به‌عنوان هسته اصلی پردازش‌ها و محاسبات کامپیوتری شناخته می‌شوند و نقش اساسی در حل مسائل پیچیده و بهینه‌سازی عملکرد سیستم‌ها دارند. درک دقیق و شفاف عملکرد الگوریتم‌ها، به ویژه در مراحل مختلف اجرای آنها، برای دانشجویان، پژوهشگران و برنامه‌نویسان اهمیت فراوانی دارد. یکی از چالش‌های اصلی در آموزش الگوریتم‌ها، ارائه تصویری واضح از تغییرات داده‌ها و ساختارها در طول اجرای الگوریتم است. این پروژه با هدف ارائه یک چارچوب جامع و تعاملی، امکان مشاهده گام‌به‌گام و بصری الگوریتم‌ها را فراهم می‌کند تا کاربران بتوانند روند اجرا و تغییرات داده‌ها را به‌صورت واضح درک کنند.

پژوهش حاضر شامل طراحی و پیاده‌سازی ابزارهایی برای ثبت و ذخیره تغییرات داده‌ها، تبدیل آنها به وضعیت‌های کامل و نمایش بصری آنها در قالب تصاویر و نمودارهای قابل تنظیم است. این سیستم توانایی نمایش آرایه‌های دوبعدی، نمودارها، گراف‌ها و پیام‌های متنی مرتبط با الگوریتم‌ها را دارد و کاربران می‌توانند ابعاد، رنگ‌ها و داده‌های اولیه خروجی‌ها را مطابق نیاز خود شخصی‌سازی کنند. علاوه بر این، خروجی‌های پروژه به‌صورت فایل‌های متنی و قابل چاپ نیز تولید می‌شوند تا امکان استفاده در منابع آموزشی و تحقیقاتی فراهم گردد.

نتایج حاصل از این پروژه می‌تواند به عنوان یک منبع آموزشی غنی و ابزار تحقیقاتی مؤثر مورد استفاده قرار گیرد و به بهبود فهم و درک مفاهیم پایه علوم کامپیوتر کمک کند. همچنین این چارچوب، با ارائه تجربه‌ای بصری و تعاملی، امکان بررسی دقیق و تحلیل عملکرد الگوریتم‌ها را برای مخاطبان با سطوح مختلف فراهم می‌آورد و می‌تواند نقش مؤثری در ارتقای کیفیت آموزش و پژوهش داشته باشد.

واژگان کلیدی: مصورسازی الگوریتم، مصورسازی در لاتک، مصورسازی در وب

فهرست مطالب

عنوان **صفحه**

فهرست مطالب

عنوان **صفحه**

فهرست مطالب

عنوان **صفحه**

فهرست شکل‌ها

عنوان **صفحه**

فهرست شکل‌ها

عنوان **صفحه**

مخفف‌ها:

LIFO	Last In, First Out
FIFO	First In, First Out

فصل اول
مقدمه

۱-۱- هدف پروژه

امروزه الگوریتم‌ها نقش بسیار مهمی در زمینه‌های مختلف علوم کامپیوتر، مهندسی و بهینه‌سازی سیستم‌ها دارند و درک دقیق عملکرد آنها برای دانشجویان و پژوهشگران اهمیت فراوانی دارد. با این حال، فهم گام‌به‌گام تغییرات داده‌ها و ساختارهای مختلف در طول اجرای الگوریتم‌ها برای بسیاری از افراد چالش‌برانگیز است. هدف از این پژوهش، ارائه چارچوبی نوآورانه برای تسهیل یادگیری و درک الگوریتم‌ها از طریق نمایش بصری و تعاملی مراحل اجرای آنها است.

در این چارچوب، کاربران می‌توانند تغییرات آرایه‌ها، نمودارها، گراف‌ها و پیام‌های متنی مرتبط با الگوریتم‌ها را مشاهده کنند و با نحوه عملکرد الگوریتم‌ها به‌صورت مرحله‌به‌مرحله آشنا شوند. همچنین، امکان شخصی‌سازی خروجی‌ها از نظر ابعاد، رنگ‌ها و داده‌های اولیه برای کاربران فراهم شده است تا تجربه‌ی یادگیری بصری و تعاملی بهینه‌تر و جذاب‌تر شود. استفاده از این روش، به‌ویژه برای افرادی که با مفاهیم انتزاعی الگوریتم‌ها ارتباط مستقیم ندارند، یادگیری را ساده‌تر و جذاب‌تر می‌کند. این پروژه قادر است با ارائه یک دیدگاه جامع و گرافیکی از اجرای الگوریتم‌ها، به عنوان یک ابزار آموزشی و تحقیقاتی مؤثر در ارتقای فهم مفاهیم پایه علوم کامپیوتر مورد استفاده قرار گیرد.

۱-۲- کاربردهای پروژه

حاصل این پروژه می‌تواند در آموزش، یادگیری، پژوهش و مرجع به کار گرفته شود. در بخش آموزش، این چارچوب می‌تواند در کلاس‌های درس، کتاب‌های آموزشی، دوره‌های آنلاین یا به‌صورت خودآموزی برای آموزش الگوریتم‌ها و ساختارهای داده به‌کار گرفته شود. در بخش یادگیری، دانشجویان و علاقه‌مندان می‌توانند از این ابزار برای درک مفاهیم الگوریتم‌ها به‌صورت بصری و مرحله‌به‌مرحله استفاده کنند و نحوه

عملکرد هر الگوریتم را به خوبی مشاهده نمایند. در بخش پژوهش، محققان می‌توانند از این چارچوب برای تحلیل و بررسی عملکرد الگوریتم‌های مختلف بهره ببرند و نتایج اجرای الگوریتم‌ها را به‌صورت تصویری و گرافیکی تحلیل کنند. همچنین، در بخش مرجع، این ابزار می‌تواند به عنوان منبعی کاربردی برای افرادی که در حال نگارش مقالات، پروژه‌ها یا پژوهش‌های مرتبط با الگوریتم‌ها و داده‌ساختارها هستند، مورد استفاده قرار گیرد و امکان بررسی و مصورسازی دقیق مراحل اجرای الگوریتم‌ها را فراهم نماید.

۱-۳- ساختار پایان نامه

در این پایان‌نامه به ارائه چارچوبی برای ثبت، قاب‌سازی و مصورسازی الگوریتم‌ها پرداخته شده است. ساختار پایان‌نامه به‌صورت زیر است:

- **فصل دوم:** مروری بر مفاهیم پایه الگوریتم‌ها، ساختمان داده‌ها و دیگر مفاهیم مرتبط با معماری و پیاده‌سازی پژوهش ارائه خواهد شد.
- **فصل سوم:** پیشینه پژوهش و بررسی کارهای مشابه در زمینه پیاده‌سازی و مصورسازی الگوریتم‌ها و ابزارهای تعاملی ارائه خواهد شد.
- **فصل چهارم:** شرح کامل پروژه، شامل طراحی چارچوب ثبت رخدادها، تبدیل آنها به وضعیت‌های کامل و روش‌های مصورسازی بصری در قالب تصاویر و نمودارها بررسی خواهد شد.
- **فصل پنجم:** نتایج، بررسی کامل نحوه مصورسازی یک الگوریتم توسط چارچوب ارائه شده.
- **فصل ششم:** نتیجه‌گیری، ارزیابی عملکرد چارچوب، بیان مزایا و محدودیت‌ها، و ارائه پیشنهادهایی جهت بهبود و توسعه‌ی بیشتر پژوهش در آینده.

فصل دوم
مفاهیم

۲-۱- مقدمه

در این قسمت مفاهیم پژوهش مورد بررسی قرار میگیرد. این قسمت شامل معرفی مفاهیم، ابزارها و کتابخانه‌هایی است که در انجام این پژوهش از آنها استفاده شده است.

۲-۲- الگوریتم

الگوریتم مجموعه‌ای از دستورالعمل‌های مشخص و گام‌به‌گام است که برای حل یک مسئله یا انجام یک وظیفه طراحی شده است. هر الگوریتم دارای ورودی مشخص، پردازش مرحله‌ای و خروجی معین است. الگوریتم‌ها اساس برنامه‌نویسی و طراحی نرم‌افزار محسوب می‌شوند و پایه‌ای برای ساختاردهی منطقی برنامه‌ها هستند. طراحی الگوریتم بهینه و صحیح باعث می‌شود برنامه‌ها سریع‌تر و با مصرف منابع کمتر اجرا شوند. الگوریتم‌ها علاوه بر کامپیوتر، در مسائل ریاضی، علوم داده، شبکه‌های اجتماعی و حتی کاربردهای روزمره مانند مسیریابی و مرتب‌سازی اطلاعات نیز استفاده می‌شوند.

۲-۲-۱- ویژگی‌های یک الگوریتم

یک الگوریتم بهینه باید دارای ویژگی‌های زیر باشد:

- **وضوح و دقت:** تمام مراحل و دستورالعمل‌ها باید به‌طور روشن و بدون ابهام مشخص شوند، به طوری که هر فرد یا سیستم بتواند آن را اجرا کند.
- **قابلیت اجرا:** الگوریتم باید قابلیت پیاده‌سازی بر روی کامپیوتر یا اجرای دستی را داشته باشد.
- **پایان‌پذیری:** الگوریتم باید پس از تعداد محدودی گام به نتیجه برسد و از حلقه‌های بی‌پایان جلوگیری شود.

- **کارایی:** الگوریتم باید از نظر زمان و حافظه بهینه باشد، به‌طوری که حداقل منابع را مصرف کند و بیشترین عملکرد را ارائه دهد.
- **قابلیت تعمیم:** الگوریتم باید بتواند برای مجموعه‌های مختلف داده ورودی اعمال شود و محدود به یک نمونه خاص نباشد.

۲-۲-۲- انواع الگوریتم‌ها از نظر عملکرد

الگوریتم‌ها از نظر عملکرد و کاربرد در مسائل مختلف به دسته‌های زیر تقسیم می‌شوند:

- **مرتب‌سازی :**الگوریتم‌هایی که داده‌ها را بر اساس معیار مشخصی مرتب می‌کنند، مانند مرتب‌سازی حبابی یا سریع.
- **جستجو :**الگوریتم‌هایی که برای یافتن عناصر مشخص در مجموعه داده‌ها استفاده می‌شوند، مانند جستجوی خطی یا دودویی.
- **گراف :**الگوریتم‌هایی که بر ساختارهای گرافی اعمال می‌شوند، مانند یافتن کوتاه‌ترین مسیر، بررسی اتصال یا مسیریابی.
- **دیگر الگوریتم‌های محاسباتی:**شامل الگوریتم‌های عددی، الگوریتم‌های رمزنگاری، و الگوریتم‌های بهینه‌سازی.

۲-۲-۳- انواع الگوریتم‌ها از نظر ساختمان داده

الگوریتم‌ها همچنین بر اساس داده‌هایی که پردازش می‌کنند و ساختار داده‌ای که استفاده می‌کنند، طبقه‌بندی می‌شوند. این دسته‌بندی باعث می‌شود بتوان عملکرد الگوریتم و میزان مصرف منابع آن را بهتر تحلیل کرد:

- **آرایه‌ها و لیست‌های پیوندی:** الگوریتم‌هایی که بر داده‌های متوالی یا پیوندی اعمال می‌شوند.
- **پشته و صف:** الگوریتم‌هایی که بر ساختارهای داده‌ای با نظم خاص (LIFO و FIFO) عمل می‌کنند.
- **درخت‌ها:** الگوریتم‌هایی که بر ساختارهای سلسله‌مراتبی داده‌ها اعمال می‌شوند، مانند درخت جستجوی دودویی.
- **گراف‌ها:** الگوریتم‌هایی که بر مجموعه‌ای از گره‌ها و یال‌ها اعمال می‌شوند، مانند الگوریتم‌های مسیریابی و بررسی اتصال.

این دسته‌بندی‌ها به دانشجویان و پژوهشگران کمک می‌کند تا الگوریتم‌ها را بهتر درک کنند و بدانند هر الگوریتم برای چه نوع داده‌ها و مسئله‌ای مناسب است.

۲-۳- زبان و ابزارها

در این پروژه، برای پیاده‌سازی چارچوب ثبت، قاب‌سازی و مصورسازی الگوریتم‌ها از ترکیبی از ابزارها و زبان‌های برنامه‌نویسی استفاده شده است. انتخاب این زبان‌ها و ابزارها به دلیل توانایی آن‌ها در پردازش داده‌ها، تولید خروجی‌های بصری و امکان تعامل با کاربر بوده است.

۲-۳-۱ لاتک

لاتک یک زبان نشانه‌گذاری و سیستم آماده‌سازی اسناد است که به ویژه برای تولید متون علمی و ریاضی با کیفیت بالا استفاده می‌شود. از ویژگی‌های مهم لاتک می‌توان به توانایی تولید نمودارها و تصاویر پیچیده با دقت بالا، پشتیبانی از فرمول‌ها و معادلات ریاضی و قابلیت خروجی PDF استاندارد اشاره کرد. در این پروژه، لاتک برای مصورسازی الگوریتم‌ها و تولید خروجی‌های گرافیکی مرحله‌به‌مرحله استفاده شده است. این امکان به کاربران اجازه می‌دهد تا مراحل اجرای الگوریتم‌ها را به صورت تصویری و قابل چاپ مشاهده کنند.

۲-۳-۲- انگولار

انگولار یک چارچوب توسعه وب متن‌باز است که توسط گوگل ارائه شده و برای ساخت برنامه‌های تک‌صفحه‌ای (SPA) تعاملی استفاده می‌شود. در پروژه حاضر، انگولار برای تولید کامپوننت‌های تعاملی وب مورد استفاده قرار گرفته است. این کامپوننت‌ها شامل قابلیت نمایش مرحله‌به‌مرحله الگوریتم‌ها، پخش خودکار یا دستی مراحل، و امکان شخصی‌سازی خروجی‌ها از نظر رنگ، ابعاد و داده‌های اولیه هستند. استفاده از انگولار امکان توسعه رابط کاربری پویا و قابل تعامل را فراهم می‌کند که تجربه یادگیری بصری و تعاملی را برای کاربران بهبود می‌بخشد.

۲-۳-۳- تایپ‌اسکریپت

تایپ‌اسکریپت یک زبان برنامه‌نویسی سطح بالا است که بر پایه جاوااسکریپت توسعه یافته و ویژگی‌هایی مانند تایپ استاتیک و ابزارهای پیشرفته برای برنامه‌نویسی بزرگ‌مقیاس را ارائه می‌دهد. در این پروژه،

تایپ‌اسکریپت برای پیاده‌سازی منطق اصلی چارچوب، مدیریت داده‌ها، ثبت رخدادها و کنترل جریان اجرای الگوریتم‌ها در محیط وب استفاده شده است. استفاده از تایپ‌اسکریپت باعث افزایش قابلیت نگهداری، خوانایی و قابلیت توسعه سیستم می‌شود و با همکاری انگولار، اجرای امن و قابل اعتماد الگوریتم‌ها را تضمین می‌کند.

۲-۴- طراحی

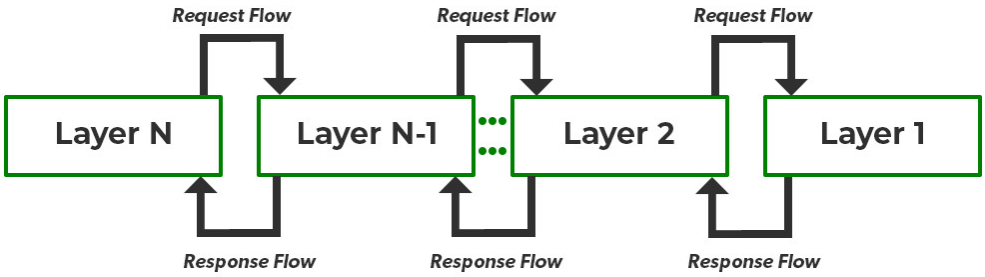
در این بخش مفاهیم مرتبط با طراحی که در ادامه مورد استفاده قرار گرفته است شرح داده میشود.

۲-۴-۱- معماری چندلایه

معماری چندلایه (Multi-Layer Architecture) یک الگوی طراحی نرم‌افزار است که سیستم را به چند لایه مجزا تقسیم می‌کند، به طوری که هر لایه وظایف و مسئولیت‌های مشخصی دارد و مستقل از دیگر لایه‌ها عمل می‌کند. این معماری بر اصل جداسازی مسئولیت‌ها تأکید دارد و هدف آن افزایش قابلیت نگهداری، توسعه و مقیاس‌پذیری سیستم‌های نرم‌افزاری است.

در این مدل، هر لایه دارای مجموعه‌ای از وظایف مشخص و محدود است و با لایه‌های دیگر تنها از طریق رابط‌های تعریف‌شده (Interface) یا قراردادهای مشخص ارتباط برقرار می‌کند. چنین رویکردی باعث می‌شود که پیچیدگی سیستم کاهش یافته و تغییر یا بهبود هر بخش بدون تأثیرگذاری مستقیم بر سایر بخش‌ها امکان‌پذیر باشد.

معماری چندلایه همچنین امکان تفکیک دقیق مسئولیت‌ها و سازماندهی منطقی کدها را فراهم می‌کند. هر لایه می‌تواند مستقل توسعه، نگهداری و تست شود. این ساختار همچنین توسعه موازی تیم‌ها را آسان‌تر می‌کند، زیرا هر تیم می‌تواند بر روی یک یا چند لایه خاص کار کند بدون اینکه با لایه‌های دیگر تداخل داشته باشد.



شکل ۲-۱: معماری چند لایه

مزایای استفاده از معماری چندلایه بسیار گسترده است و می‌تواند اثر مستقیمی بر کیفیت نرم‌افزار داشته باشد:

- **جداسازی مسئولیت‌ها:** هر لایه به وظیفه خاص خود محدود می‌شود و وابستگی‌ها به حداقل می‌رسد. این ویژگی باعث می‌شود که تغییرات در یک بخش از سیستم به سادگی انجام شده و پیچیدگی مدیریت کاهش یابد.
- **قابلیت نگهداری بالا:** با تفکیک منطقی کدها، توسعه‌دهندگان می‌توانند خطاها و مشکلات را سریع‌تر شناسایی و اصلاح کنند. سیستم‌هایی که از معماری چندلایه استفاده می‌کنند، معمولاً طول عمر بیشتری دارند.
- **سهولت در تست و اعتبارسنجی:** هر لایه را می‌توان به طور جداگانه تست کرد، بدون آنکه نیاز باشد تمام سیستم را اجرا کنیم. این موضوع باعث افزایش کیفیت و اطمینان از صحت عملکرد سیستم می‌شود.
- **انعطاف‌پذیری و قابلیت ارتقا:** تغییر در یک لایه یا بهبود عملکرد آن بدون تأثیر مستقیم بر سایر لایه‌ها امکان‌پذیر است. این انعطاف‌پذیری باعث می‌شود سیستم بتواند با نیازها و تکنولوژی‌های جدید سازگار شود.
- **امکان استفاده مجدد از کد:** لایه‌ها می‌توانند به صورت مستقل مورد استفاده مجدد قرار بگیرند، چه در همان سیستم و چه در پروژه‌های دیگر، که توسعه نرم‌افزار را سریع‌تر و اقتصادی‌تر می‌کند.
- **کاهش پیچیدگی و افزایش سازماندهی:** با تفکیک منطقی وظایف، ساختار کلی سیستم ساده‌تر و قابل فهم‌تر می‌شود. این سازماندهی باعث می‌شود توسعه‌دهندگان تازه‌کار سریع‌تر با سیستم آشنا شوند.

- **پشتیبانی از توسعه موازی و تیمی:** تیم‌های مختلف می‌توانند به صورت همزمان روی لایه‌های مختلف کار کنند، بدون آنکه تغییرات یک تیم عملکرد تیم دیگر را مختل کند.

۲-۴-۲- الگوهای طراحی

الگوهای طراحی (Design Patterns) راه‌حل‌های استاندارد و قابل استفاده مجدد برای مسائل رایج در طراحی نرم‌افزار هستند. این الگوها به توسعه‌دهندگان کمک می‌کنند تا کدهایی انعطاف‌پذیر، قابل نگهداری و مقیاس‌پذیر بنویسند و از تکرار بی‌رویه کد جلوگیری کنند.

الگوهای طراحی معمولاً بر اساس هدف و کاربرد به سه دسته کلی تقسیم می‌شوند:

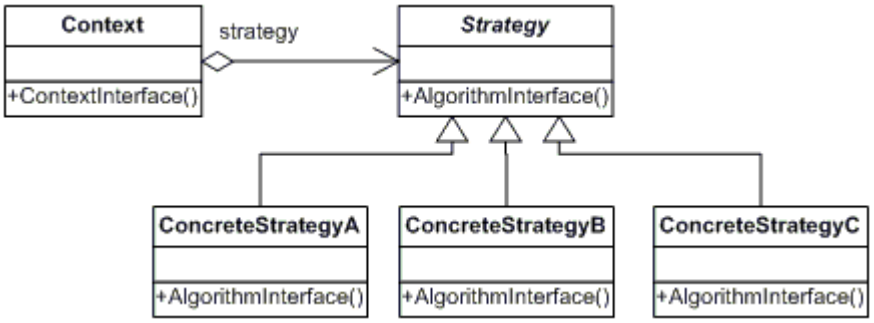
- **الگوهای خلقی (Creational Patterns) :** مربوط به نحوه ایجاد اشیاء و مدیریت چرخه حیات آن‌ها هستند.
- **الگوهای ساختاری (Structural Patterns) :** نحوه سازماندهی کلاس‌ها و اشیاء برای ایجاد ساختارهای پیچیده را مشخص می‌کنند.
- **الگوهای رفتاری (Behavioral Patterns) :** نحوه تعامل و ارتباط بین اشیاء و مدیریت رفتارها را مشخص می‌کنند.

۲-۴-۲-۱- الگوی استراتژی

الگوی استراتژی یک الگوی رفتاری است که به شما امکان می‌دهد الگوریتم‌ها یا رفتارهای مختلف را به صورت جداگانه تعریف و در زمان اجرا انتخاب کنید .این الگو به جای داشتن شرط‌های پیچیده برای انتخاب رفتار، الگوریتم‌ها را به کلاس‌های مستقل تقسیم می‌کند و از طریق رابط مشترک قابل استفاده می‌کند.

مزایای اصلی این الگو عبارتند از:

- افزایش انعطاف‌پذیری و امکان تغییر الگوریتم‌ها بدون دستکاری کلاس‌های دیگر.
- جداسازی منطق الگوریتم‌ها از کد اصلی سیستم.
- کاهش وابستگی‌ها و جلوگیری از استفاده از شرط‌های پیچیده و تو در تو.



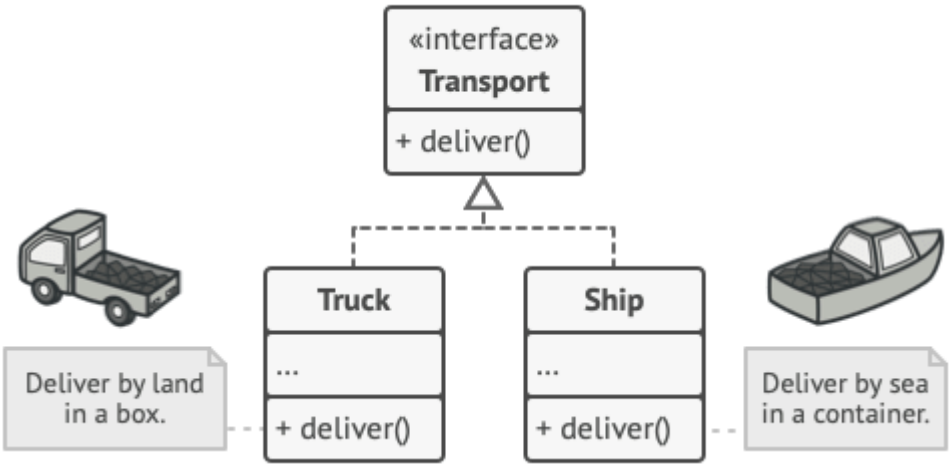
شکل ۲-۲: الگوی استراتژی

۲-۴-۲-۲- الگوی کارخانه

الگوی کارخانه یک الگوی خلقی است که برای ایجاد اشیاء بدون مشخص کردن کلاس دقیق آن‌ها در زمان کدنویسی استفاده می‌شود. این الگو یک رابط یا کلاس پایه ارائه می‌دهد و زیرکلاس‌ها یا کارخانه‌ها مسئول ایجاد نمونه‌های خاص هستند.

مزایای اصلی الگوی کارخانه عبارتند از:

- ایجاد اشیاء به صورت پویا و مستقل از کد مصرف‌کننده.
- کاهش وابستگی مستقیم به کلاس‌های مشخص و افزایش انعطاف‌پذیری.
- تسهیل اضافه کردن انواع جدید اشیاء بدون تغییر کد موجود.



شکل ۲-۳: الگوی کارخانه

۲-۴-۳- اصول SOLID

اصول SOLID مجموعه‌ای از پنج اصل طراحی شیءگرا هستند که برای افزایش انعطاف‌پذیری، قابلیت نگهداری و توسعه سیستم‌های نرم‌افزاری ارائه شده‌اند. رعایت این اصول باعث می‌شود کدها قابل فهم‌تر، قابل تست و مقیاس‌پذیرتر باشند.

۲-۴-۳-۱- اصل تک مسئولیتی

هر کلاس باید تنها یک مسئولیت یا دلیل برای تغییر داشته باشد. به عبارت دیگر، هر کلاس باید تنها یک وظیفه مشخص را انجام دهد و تمام قابلیت‌های آن مرتبط با همان وظیفه باشند. رعایت این اصل باعث کاهش پیچیدگی، ساده شدن نگهداری و افزایش خوانایی کد می‌شود.

۲-۴-۳-۲- اصل باز - بسته

کلاس‌ها و ماژول‌ها باید برای توسعه باز و برای تغییر بسته باشند. یعنی بتوان عملکرد کلاس‌ها را بدون تغییر کد موجود، با افزودن قابلیت‌های جدید گسترش داد. این اصل باعث می‌شود تغییرات آینده بدون ایجاد خطا در بخش‌های دیگر سیستم امکان‌پذیر باشد.

۲-۴-۳-۳- اصل جایگزینی لیسکوف

اشیاء زیرکلاس باید بتوانند به جای اشیاء کلاس پایه استفاده شوند بدون آنکه رفتار سیستم تغییر کند . به عبارت دیگر، زیرکلاس‌ها باید تعهدات و رفتارهای تعریف‌شده توسط کلاس پایه را نقض نکنند. رعایت این اصل تضمین می‌کند که سیستم با ارث‌بری کلاس‌ها پایدار و قابل پیش‌بینی باشد.

۲-۴-۳-۴- اصل جداسازی رابط

به جای استفاده از یک رابط بزرگ و جامع، بهتر است چندین رابط کوچک و تخصصی ایجاد شود، به طوری که هر کلاس تنها با رابط‌هایی تعامل داشته باشد که واقعاً نیاز دارد. این اصل باعث کاهش وابستگی‌ها و ساده‌تر شدن پیاده‌سازی و نگهداری کلاس‌ها می‌شود.

۲-۴-۳-۵- اصل وارونگی وابستگی

ماژول‌های سطح بالا نباید به ماژول‌های سطح پایین وابسته باشند، بلکه هر دو باید به

انتزاع‌ها (Interface) یا کلاس‌های انتزاعی وابسته باشند. همچنین انتزاع‌ها نباید به جزئیات وابسته باشند؛ جزئیات باید به انتزاع وابسته باشند. این اصل باعث افزایش انعطاف‌پذیری و کاهش وابستگی مستقیم بین اجزای سیستم می‌شود.

۲-۵- جمع‌بندی

در این فصل به معرفی و تبیین مفاهیم نظری و پایه‌ای موردنیاز پژوهش پرداخته شد. ابتدا مفهوم الگوریتم و ویژگی‌ها و دسته‌بندی‌های مختلف آن بررسی گردید تا اهمیت و جایگاه آن در علوم کامپیوتر روشن شود. سپس زبان‌ها و ابزارهای مورد استفاده در این پروژه شامل لاتک، انگولار و تایپ‌اسکریپت معرفی شدند که هرکدام نقش مهمی در تولید و نمایش خروجی‌های پژوهش ایفا می‌کنند. در ادامه، اصول و الگوهای معماری نرم‌افزار مطرح شد؛ ابتدا معماری چندلایه و مزایای آن شرح داده شد و سپس برخی از الگوهای طراحی مانند استراتژی و کارخانه که در طراحی سیستم‌های نرم‌افزاری پرکاربرد هستند، معرفی گردید. در نهایت اصول SOLID به عنوان یکی از مهم‌ترین رویکردها در مهندسی نرم‌افزار مورد بررسی قرار گرفتند.

آنچه در این فصل ارائه شد، مبنای نظری لازم برای درک بهتر مراحل آتی پژوهش را فراهم می‌آورد. در ادامه ، با تکیه بر این مفاهیم، به شرح پروژه و جزئیات مربوط به طراحی و پیاده‌سازی چارچوب پیشنهادی برای ضبط، قاب‌سازی و نمایش الگوریتم‌ها پرداخته خواهد شد.

فصل سوم
پژوهش‌های مشابه

۳-۱- مقدمه

در این فصل، به بررسی پروژه‌ها و ابزارهای مشابه در زمینه‌ی مصورسازی الگوریتم‌ها و ساختارهای داده پرداخته می‌شود. هدف از این بررسی، شناسایی نقاط قوت و ضعف راهکارهای موجود و مقایسه‌ی آن‌ها با پروژه حاضر است. مطالعه‌ی این پروژه‌ها به درک بهتر رویکردهای مختلف در طراحی ابزارهای آموزشی و پژوهشی کمک می‌کند و امکان ارائه‌ی ویژگی‌های منحصربه‌فرد و بهبود یافته در پروژه حاضر را فراهم می‌آورد. ابزارهای بررسی شده شامل پلتفرم‌های آنلاین تعاملی، کتابخانه‌های آموزشی و نرم‌افزارهای متن‌باز است که هرکدام با تمرکز خاصی بر نوع الگوریتم یا نوع مصورسازی طراحی شده‌اند.

۳-۲- نمونه‌های مشابه

در زمینه‌ی مصورسازی الگوریتم‌ها و ساختارهای داده، تاکنون ابزارها و پلتفرم‌های متنوعی توسعه یافته‌اند. این ابزارها اغلب با هدف آموزش، درک مفهومی بهتر الگوریتم‌ها، و کمک به دانشجویان و پژوهشگران ارائه شده‌اند. در ادامه، چند نمونه برجسته معرفی و تحلیل می‌شود.

۳-۲-۱- VisuAlgo

VisuAlgo یک پلتفرم تعاملی آموزشی است که الگوریتم‌ها و ساختارهای داده را به‌صورت بصری نمایش می‌دهد. این ابزار شامل طیف وسیعی از الگوریتم‌ها از جمله مرتب‌سازی، جست‌وجو، درخت‌ها و گراف‌هاست و هر الگوریتم با توضیحات متنی و کد نمونه همراه است. کاربر می‌تواند مراحل اجرای الگوریتم را به‌صورت پویانمایی مشاهده کند و درک بهتری از نحوه‌ی عملکرد الگوریتم‌ها به دست آورد [۱].

محدودیت‌ها در مقایسه با پروژه حاضر:

- خروجی‌ها صرفاً گرافیکی و در بستر وب هستند و امکان استفاده مستقیم در مقالات، کتاب‌ها و اسناد علمی وجود ندارد.
- شخصی‌سازی خروجی‌ها (مانند تغییر رنگ‌ها یا قالب نمایش) در سطح محدودی انجام می‌شود.
- تمرکز اصلی بر آموزش عمومی است، نه مستندسازی علمی و پژوهشی.

Algorithm Visualizer -۲-۲-۳

Algorithm Visualizer یک پلتفرم متن‌باز و تعاملی است که به‌صورت آنلاین در دسترس قرار دارد. این ابزار الگوریتم‌ها را مرحله‌به‌مرحله اجرا کرده و به شکل انیمیشن در محیط وب نمایش می‌دهد. کاربران می‌توانند ورودی‌های دلخواه خود را به الگوریتم بدهند و اجرای آن را به‌صورت بصری دنبال کنند. ساختار این ابزار بر پایه‌ی استفاده از Tracer و Renderer است. Tracer ها برای زبان‌های مختلف پیاده‌سازی شده‌اند و امکان ثبت گام‌های الگوریتم را فراهم می‌کنند؛ اما در بخش نمایش، تنها یک Renderer وبی وجود دارد و توسعه‌ی Renderer جدید به دلیل معماری دولایه و پیچیدگی کد، بسیار دشوار و زمان‌بر است [۲].

تفاوت با پروژه حاضر:

- خروجی محدود به وب: این ابزار تنها در بستر وب خروجی می‌دهد و قابلیت تبدیل نتایج به قالب‌های علمی مانند LaTeX یا PDF وجود ندارد؛ در مقابل، پروژه حاضر خروجی استاندارد و قابل استناد برای استفاده در مقالات و مستندات تولید می‌کند.
- تمرکز بر تعامل بصریAlgorithm Visualizer : بیشتر بر درک مفهومی و یادگیری تعاملی تمرکز دارد، در حالی‌که پروژه حاضر علاوه بر نمایش، قابلیت ثبت دقیق، بازسازی کامل و مستندسازی روند اجرا را نیز فراهم می‌کند.
- معماری پیچیده در توسعه: اضافه کردن Renderer جدید مستلزم نوشتن حجم زیادی کد به دلیل معماری دولایه است؛ در حالی‌که در پروژه حاضر سعی شده معماری به‌گونه‌ای طراحی شود که Renderer ها مینیمال بوده و توسعه‌ی آن‌ها ساده‌تر و کم‌هزینه‌تر انجام شود.
- محدودیت در توسعه پژوهشی: این ابزار بیشتر به‌عنوان یک پلتفرم آموزشی شناخته می‌شود و قابلیت انعطاف‌پذیری و توسعه در سطح پژوهشی را ندارد.

Sort Visualizer -۳-۲-۳

Sorting Visualizer دسته‌ای از ابزارهای آنلاین هستند که صرفاً بر نمایش و مصورسازی الگوریتم‌های

مرتب‌سازی تمرکز دارند. این ابزارها معمولاً با استفاده از کتابخانه‌های جاوااسکریپت توسعه یافته‌اند و الگوریتم‌هایی مانند Bubble Sort، Merge Sort، Quick Sort و موارد مشابه را به‌صورت انیمیشن و مرحله‌به‌مرحله نمایش می‌دهند. کاربران می‌توانند اجرای هر الگوریتم را مشاهده کرده و درک بهتری از روند مرتب‌سازی داده‌ها پیدا کنند. این ابزارها عموماً جنبه‌ی آموزشی دارند و برای یادگیری مفهومی مرتب‌سازی بسیار پرکاربرد هستند [۳].

محدودیت‌ها و تفاوت با پروژه حاضر:

- محدودیت در پوشش الگوریتم‌ها: این ابزارها تنها روی الگوریتم‌های مرتب‌سازی متمرکز هستند و الگوریتم‌های پیشرفته‌تر مانند الگوریتم‌های گرافی، درختی یا ساختارهای داده‌ی پیچیده‌تر را پشتیبانی نمی‌کنند. در مقابل، پروژه حاضر با رویکردی ماژولار، قابلیت پوشش طیف گسترده‌ای از الگوریتم‌ها و ساختارها را دارد.
- خروجی صرفاً نمایشی: در Sorting Visualizer خروجی‌ها به شکل انیمیشن وبی هستند و قابلیت استفاده در پژوهش‌ها و مستندسازی علمی را ندارند. اما پروژه حاضر خروجی استاندارد و قابل استناد مانند LaTeX تولید می‌کند که می‌تواند مستقیماً در مقالات و گزارش‌های علمی به‌کار رود.
- شخصی‌سازی محدود داده‌ها: در این ابزارها داده‌ها معمولاً به‌صورت پیش‌فرض تعریف شده‌اند و کاربران امکان تعریف ورودی دلخواه به‌طور کامل ندارند. در پروژه حاضر، کاربران می‌توانند داده‌ها و پارامترهای مختلف را به‌طور کامل شخصی‌سازی کرده و خروجی مناسب با نیاز پژوهشی خود دریافت کنند.
- وابستگی به فناوری‌های وب: این ابزارها به‌طور کامل بر پایه‌ی جاوااسکریپت و محیط وب پیاده‌سازی شده‌اند. این موضوع باعث می‌شود خروجی‌ها تنها در بستر مرورگر قابل استفاده باشند. اما در پروژه حاضر، خروجی‌ها به‌صورت فایل‌های قابل حمل و استاندارد تولید می‌شوند که محدودیت استفاده در بستر خاصی را ندارند.

Data Structure Visualizations –۴-۲-۳

این ابزار دانشگاهی توسط پروفسور David Galles از دانشگاه سن‌فرانسیسکو توسعه داده شده و به‌صورت رایگان و متن‌باز در اختیار کاربران قرار گرفته است. این ابزار مجموعه‌ی گسترده‌ای از الگوریتم‌ها و ساختمان‌های داده مانند درخت‌ها، گراف‌ها، صف‌ها و لیست‌های پیوندی را به‌صورت تعاملی و گرافیکی نمایش می‌دهد. کاربران می‌توانند ورودی‌ها را تغییر داده و اجرای الگوریتم را مرحله‌به‌مرحله مشاهده و بررسی کنند. دسترسی به این ابزار به‌صورت آنلاین و تحت وب فراهم است [۴].

تفاوت با پروژه حاضر:

- آنلاین بودن و محدودیت دسترسی: این ابزار صرفاً به‌صورت آنلاین در دسترس است و نیاز به اتصال اینترنت دارد. در شرایطی که اینترنت محدود یا غیرممکن باشد، استفاده از آن دشوار می‌شود؛ در حالی‌که پروژه‌ی حاضر پس از نصب، به‌صورت آفلاین عمل کرده و خروجی LaTeX و PDF تولید می‌کند که در هر شرایطی قابل استفاده است.
- عدم تمرکز بر مستندسازی علمی: ابزار صرفاً برای نمایش تعاملی ساخته شده و قابلیت تولید خروجی استاندارد و قابل استناد در پژوهش‌های علمی را ندارد؛ اما در پروژه‌ی حاضر امکان تولید خروجی‌های مستندشده همراه با توضیحات و ارجاع‌دهی علمی فراهم است.
- شخصی‌سازی خروجی‌ها: در ابزار Galles خروجی‌ها ثابت و غیرقابل تغییرند، ولی در پروژه‌ی حاضر می‌توان پارامترهایی مانند رنگ، ابعاد و سبک نمایش را شخصی‌سازی کرد.
- ماهیت آموزشی در برابر پژوهشی: ابزار بیشتر جنبه‌ی آموزشی دارد و برای نمایش ساده‌ی الگوریتم‌ها طراحی شده، در حالی‌که پروژه‌ی حاضر علاوه بر آموزش، بُعد پژوهشی و مستندسازی را نیز پشتیبانی می‌کند.
- قابلیت توسعه‌پذیری و متن‌باز بودن: ابزار Galles الگوریتم‌های از پیش تعریف‌شده را اجرا می‌کند و تغییر یا افزودن الگوریتم‌های جدید دشوار است؛ اما پروژه‌ی حاضر معماری ماژولار دارد که توسعه‌دهندگان می‌توانند الگوریتم‌های جدید را به سادگی به سیستم اضافه کنند.

۳-۲-۵- پژوهش‌های گروه نرم‌افزاری

در برخی پژوهش‌های دانشگاهی و پروژه‌های نرم‌افزاری، ابزارهایی برای مصورسازی الگوریتم‌ها ارائه شده است که تمرکز اصلی آن‌ها بر تولید خروجی‌های قابل چاپ و علمی مانند LaTeX [۵] و PDF بوده است. این ابزارها معمولاً الگوریتم‌ها را به‌صورت کدهای از پیش‌نوشته‌شده و اختصاصی پیاده‌سازی کرده‌اند؛ به این معنا که برای هر الگوریتم، کد ویژه‌ای تعبیه شده و در نتیجه امکان گسترش یا افزودن الگوریتم‌های جدید به‌سادگی وجود ندارد [۶].

محدودیت‌ها:

- نیاز به پیاده‌سازی دستی برای هر الگوریتم.
- انعطاف‌پذیری پایین در افزودن قابلیت‌های جدید.
- عدم وجود معماری عمومی برای ثبت، بازسازی و نمایش الگوریتم‌ها.

۳-۳- جمع‌بندی

در این فصل، تعدادی از پروژه‌ها و ابزارهای مشابه در زمینه‌ی مصورسازی الگوریتم‌ها و ساختارهای داده بررسی شد. هر یک از این پروژه‌ها مزایا و محدودیت‌های خاص خود را دارند؛ برخی تمرکز اصلی خود را بر آموزش تعاملی و مشاهده‌ی مرحله‌به‌مرحله الگوریتم‌ها قرار داده‌اند، برخی دیگر امکان شخصی‌سازی محدود و خروجی‌های غیرقابل چاپ دارند، و بعضی صرفاً روی یک دسته خاص از الگوریتم‌ها مانند مرتب‌سازی متمرکز هستند. مقایسه‌ی این پروژه‌ها با پروژه‌ی حاضر نشان می‌دهد که ابزار ما علاوه بر فراهم کردن محیط تعاملی، قابلیت تولید خروجی استاندارد و قابل چاپ با استفاده از زبان لاتک، پشتیبانی از انواع مختلف الگوریتم‌ها و انعطاف در شخصی‌سازی تصاویر را نیز ارائه می‌دهد. این ویژگی‌ها، پروژه حاضر را از نظر آموزشی و پژوهشی متمایز می‌کند و امکان استفاده‌ی گسترده در مستندسازی علمی و آموزش تعاملی را فراهم می‌سازد.

فصل چهارم
شرح پروژه

۴-۱- مقدمه

در این فصل به بررسی جزئیات طراحی و پیاده‌سازی سیستم مصورسازی الگوریتم‌ها می‌پردازد. در این فصل، ابتدا معماری کلی سیستم و ساختار اجزای مختلف آن معرفی می‌شود تا دید جامعی از نحوه تعامل بخش‌ها و جریان داده‌ها ارائه گردد. سپس جزئیات پیاده‌سازی هر بخش شامل ضبط‌کننده‌های رویداد، قاب‌سازها و نمایشگرها توضیح داده می‌شود. هدف از این فصل، روشن کردن چگونگی پیاده‌سازی معماری و اجزای سیستم، نحوه ثبت و پردازش رویدادها، و ارائه خروجی‌های تصویری و تعاملی است، به گونه‌ای که خواننده بتواند عملکرد سیستم و منطق پشت آن را به‌طور کامل درک کند.

۴-۲- طراحی

سیستم طراحی شده به گونه‌ای است که تمامی رخدادهای الگوریتم‌ها در طول اجرا ثبت شده و به مرحله‌ای تبدیل می‌شوند که بتوان آن‌ها را به‌صورت تصویری و تعاملی نمایش داد. هدف از این طراحی، ایجاد چارچوبی منعطف، قابل توسعه و در عین حال ساده برای پیاده‌سازی انواع الگوریتم‌هاست.

در این سیستم، هر ماژول مسئولیت مشخصی دارد و بر اساس یک معماری چندلایه توسعه یافته است که امکان نگهداری و توسعه مستقل هر بخش را فراهم می‌کند. این معماری باعث می‌شود که ثبت رخدادها،

ساخت قاب‌های تصویری و نمایش نهایی کاملاً از هم تفکیک شده و هر ماژول بتواند بدون وابستگی شدید به دیگر بخش‌ها، تغییر یا بهبود یابد. طراحی سیستم همچنین شامل تعریف قراردادها و واسطهایی است که نحوه تعامل بین اجزا را استاندارد می‌کند؛ به‌گونه‌ای که داده‌ها از مرحله ثبت رویداد تا تولید خروجی نهایی با کمترین احتمال خطا و بیشترین انعطاف انتقال یابند.

یکی از ویژگی‌های مهم طراحی این سیستم، قابلیت تولید خروجی‌های چندگانه است: هم قالب LaTeX برای استفاده در مقالات و مستندسازی دقیق و هم قالب وب تعاملی برای مشاهده مرحله‌به‌مرحله الگوریتم‌ها. این قابلیت مستلزم طراحی دقیق جریان داده و مدیریت وضعیت‌هاست، به‌طوری که سیستم بتواند بدون تداخل یا از دست رفتن اطلاعات، داده‌های ثبت‌شده را به انواع مختلف خروجی تبدیل کند.

همچنین، طراحی سیستم به‌گونه‌ای انجام شده است که توسعه و افزودن الگوریتم‌های جدید یا تغییر سبک نمایش یه حتی استفاده از زبانی جدید برای نوشتن الگوریتم‌های خود در آن‌ها ساده باشد. چارچوب طراحی شده شامل الگوهایی است که امکان گسترش و سفارشی‌سازی بخش‌های مختلف را بدون نیاز به بازنویسی کامل فراهم می‌کند و در عین حال، به دلیل سادگی پیاده‌سازی، از پیچیدگی غیرضروری اجتناب شده است.

در مجموع، طراحی سیستم با تمرکز بر انعطاف‌پذیری، قابلیت توسعه، مدیریت دقیق رخدادها و تولید خروجی‌های قابل استفاده در محیط‌های مختلف، پایه‌ای مستحکم برای پیاده‌سازی سیستم مصورسازی الگوریتم‌ها فراهم کرده است و مسیر روشن و قابل پیگیری برای مراحل بعدی پیاده‌سازی ارائه می‌دهد.

۴-۲-۱- معماری کلی سیستم

در ابتدا معماری بر پایه ساختمان‌های داده ردیابی‌شده (Tracked Data Structures) شکل گرفت. در این مدل، کاربران به جای آرایه‌ها یا لیست‌های معمولی از نسخه‌های ردیابی شده مانند آرایه ردیابی‌شده (TrackedArray) استفاده می‌کردند. سپس یک لایه انتزاعی (Abstraction Layer) تعریف می‌شد که تصویرهای لحظه‌ای (Snapshot) داده‌ها را دریافت کرده و آن‌ها را به شکل گراف یا ساختار داده انتزاعی مناسب ارائه می‌کرد. این لایه به الگوریتم اجازه می‌داد تا ماهیت گرافی یا درختی داشته باشد، در حالی که داده‌های داخلی به صورت آرایه یا لیست ذخیره می‌شدند.

مشکلات ایده اولیه:

۱. انعطاف پایین در ذخیره‌سازی انتزاع‌ها: هر نوع انتزاع مانند گراف، آرایه یا سایر ساختمان‌های داده می‌تواند به روش‌های مختلفی ذخیره شود. برای هر روش، لایه انتزاعی (Abstraction Layer) باید

سفارشی شود تا تصویر لحظه‌ای مناسب (Snapshot) بسازد. این باعث محدودیت برای کاربران یا پیچیدگی بالا در پیاده‌سازی می‌شد.

۲. الگوریتم‌های بدون تغییر داده: الگوریتم‌هایی مانند تلگوریتم‌های پیمایش و جستجو که فقط داده‌ها را می‌خوانند و تغییر نمی‌دهند، هیچ چیزی را ردیابی نمی‌کردند و اجرای آن‌ها با این مدل امکان‌پذیر نبود.

در معماری نهایی همچنان چندلایه بود، اما لایه انتزاعی (Abstraction Layer) به گونه‌ای طراحی شد که دیگر نیازی به محدود کردن کاربر در نحوه ذخیره‌سازی داده‌ها نبود. کاربران می‌توانند گراف یا هر داده انتزاعی دیگری را به هر نحوی ذخیره کنند، ولی برای ضبط آن داده‌ها، یک ضبط‌کننده (Recorder) اختصاصی در اختیار دارند.

- ضبط‌کننده (Recorder) : برای هر نوع انتزاع (مثلاً گراف، آرایه، صف) تعریف شده‌اند و عمل‌های (Action) قابل انجام آن نوع داده را می‌شناسند. هنگام وقوع هر عمل، یک دستور (Command) تولید و به موتور ضبط‌کننده (Recorder Engine) ارسال می‌شود.
- موتور ضبط‌کننده (Recorder Engine) : تمامی دستورها (Command) را از ضبط‌کننده‌های مختلف جمع‌آوری و یک ضبط کامل (Recording) ایجاد می‌کند. ضبط یک قرارداد مشخص دارد که در آن تمام عمل‌ها و پارامترهایشان تعریف شده‌اند.

این معماری مشابه بخش ردیاب (Tracer) در Algorithm Visualizer است، اما با تفاوت کلیدی بدین صورت که در Algorithm Visualize، خروجی ردیاب (Tracer) مستقیماً به نمایش‌گر (Renderer) داده می‌شود. این باعث می‌شد نمایش‌گرها پیچیده و سخت‌گسترش‌پذیر باشند، به ویژه برای اضافه کردن نمایش‌گرهای جدید مانند LaTeX یا وب.

بهبود روش کنونی:

در این سیستم، این جریان شکسته شده و به دو بخش جدا تبدیل شده:

- موتور قاب‌سازی (Framer Engine)
- نمایش‌گر (Renderer)

موتور قاب‌سازی، ضبط (Recording) را می‌گیرد و آن را به انیمیشن (Animation) تبدیل می‌کند. انیمیشن شامل وضعیت هر انتزاع (Frame State) در هر فریم (Frame) است، بنابراین نمایش‌گر تنها کافی است این

انیمیشن را بخواند و نمایش دهد. در این طراحی، هر ضبط‌کننده (Recorder) متناظر با یک قاب‌ساز (Framer) است که دستورات (Command) آن را می‌فهمد و به وضعیت هر انتزاع (Frame State) تبدیل می‌کند. سپس برای هر Framer نیز یک Renderer وجود دارد که خروجی آن وضعیت‌ها را تفسیر کرده و نمایش می‌دهد.

مزایا:

- پیاده‌سازی نمایش‌گر سبک و مینیمال می‌شود.
- یکبار اجرای الگوریتم و ضبط آن کافی است تا بتوان چندین نمایش‌گر مختلف داشت یا نمایش تعاملی وب ارائه کرد.
- معماری بی‌وابستگی به زبان (Language-Agnostic) است: ضبط‌کننده‌ها می‌توانند در زبان‌های مختلف پیاده شوند، زیرا خروجی JSON ضبط (Recording) یکسان است.

محدودیت‌ها:

- هم اکنون به دلیل مرحله به مرحله بودن این معماری ابتدا باید اجرای هر مرحله به صورت کامل پایان‌پذیرد و خروجی آن تولید شود سپس خروجی مرحله قبلی به عنوان ورودی لایه بعدی مورد استفاده قرار گیرد بنابر این الگوریتم‌های پایان‌ناپذیر نمی‌توانند ضبط و نمایش داده شوند، مگر اینکه در آینده امکان جریان‌سازی (Streaming) اضافه شود.

۴-۲-۲- قراردادها و واسط‌ها

قراردادها (Contracts) نقش کلیدی در معماری دارند و باعث کمینه شدن جفت‌شدگی میان پیاده‌سازی ماژول‌های لایه‌های مختلف می‌شود.

هر ضبط‌کننده (Recorder) قرارداد خود را دارد که شامل عمل‌های قابل انجام و پارامترهای هر عمل است. این قراردادها به موتور قاب‌سازی (Framer Engine) امکان می‌دهد بدون نیاز به دانستن جزئیات داخلی ضبط‌کننده، ضبط کامل را به انیمیشن (Animation) تبدیل کند.

همچنین قرارداد بین موتور قاب‌سازی و نمایش‌گر تعریف شده است بدین صورت که انیمیشن (Animation) به شکل JSON استاندارد حاوی وضعیت (State) هر انتزاع در هر فریم است. نمایش‌گرها موظف هستند فقط این JSON را بخوانند و نمایش دهند.

مزایای قراردادها:

- اضافه کردن نمایش‌گر جدید بدون تغییر در ضبط‌کننده یا موتور قاب‌سازی امکان‌پذیر است.

- پیاده‌سازی نمایش‌گر ساده و سبک می‌شود.
- اجرای الگوریتم و تولید انیمیشن مستقل از زبان برنامه‌نویسی است.

۴-۲-۳- تعامل بین اجزا

تعامل اجزای سیستم بر اساس جریان داده‌ها و قراردادهای مشخص تعریف شده است. هر جزء تنها وظیفه‌ی خود را انجام می‌دهد و خروجی آن، ورودی جزء بعدی خواهد بود. این رویکرد باعث استقلال، سادگی در توسعه، و قابلیت جایگزینی یا افزودن اجزای جدید می‌شود.

جریان تعامل به صورت زیر است:

۱. ضبط‌کننده (Recorder)

- درون الگوریتم فراخوانی می‌شود و عمل‌های (Action) مربوط به یک انتزاع خاص (مانند درج، حذف یا تغییر در آرایه یا یال‌های گراف) را ثبت می‌کند.
- هر عمل به یک دستور (Command) استاندارد تبدیل می‌شود. این دستور شامل نوع عمل و پارامترهای موردنیاز آن است.
- برای هر انتزاع (Array, Graph, Log, …) یک ضبط‌کننده اختصاصی وجود دارد.

۲. موتور ضبط‌کننده (Recorder Engine)

- دستورات تولید شده از تمام ضبط‌کننده‌ها را جمع‌آوری می‌کند.
- بر اساس یک قرارداد استاندارد، یک ضبط (Recording) ایجاد می‌شود. این ضبط در قالب JSON ذخیره می‌گردد و مستقل از زبان برنامه‌نویسی است.

۳. موتور قاب‌سازی(Framer Engine)

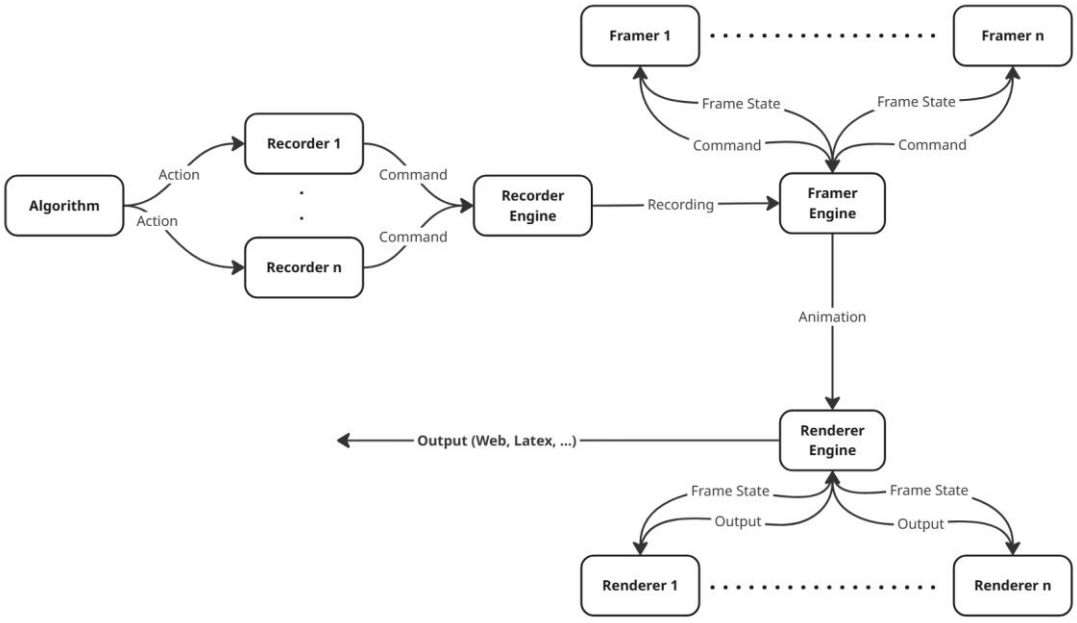
- هر ضبط‌کننده (Recorder) یک قاب‌ساز (Framer) متناظر دارد. قاب‌ساز وظیفه دارد دستورات همان ضبط‌کننده را تفسیر کرده و به وضعیت‌های متوالی (Frame State) تبدیل کند.
- بدین ترتیب هر Command خام، به یک Frameتبدیل می‌شود که بیانگر وضعیت کامل داده‌ها پس از آن عمل است.
- قاب‌سازی به انیمیشن (Animation) منجر می‌شود که دنباله‌ای ازFrame های مرتب‌شده در زمان است.

٤. نمایش‌گر (Renderer)

- هر قاب‌ساز (Framer) یک نمایش‌گر (Renderer) متناظر دارد که وضعیت‌های تولیدشده را

تفسیر کرده و در خروجی نمایش می‌دهد.

- به دلیل این طراحی، نمایش‌گر بسیار ساده است و کافی است تنها Frame State را بفهمد. به همین دلیل، افزودن نمایش‌گرهای جدید مانند LaTeX یا وب ساده و کم‌هزینه خواهد بود.



شکل ۴-۱: معماری چارچوب مصورسازی

۴-۳- پیاده‌سازی

در این بخش توضیح داده می‌شود که معماری معرفی‌شده چگونه در عمل پیاده‌سازی شده است. هر لایه شامل اجزای اختصاصی برای انتزاع‌های مختلف (گراف، آرایه دوبعدی، نمودار، لاگ) است. پیاده‌سازی بر اساس زبان TypeScript برای ضبط‌کننده‌ها و قاب‌سازها، و در دو مسیر نمایش‌گر جداگانه در LaTeX و وب (Angular) انجام شده است [۷].

۴-۳-۱- ضبط کننده‌ها

در وضعیت کنونی ضبط کننده‌ها و موتور ضبط تنها برای TypeScript پیاده شده است اما از آنجا که

ماژول ضبط فقط باید از قرارداد ضبط پیروی کند پیاده‌سازی آن در زبان‌های دیگر کار دشواری نخواهد بود.

۴-۳-۱-۱- پیاده‌سازی ضبط‌کننده‌ها در TypeScript

برای پیاده‌سازی بخش ضبط‌کننده‌ها (Recorder) در زبان TypeScript، یک ساختار کلی و یکنواخت طراحی شد که تمامی تغییرات و عملیات الگوریتم را ثبت می‌کند. این ساختار از سه جزء اصلی تشکیل شده است:

- دستور(Command) : هر عملیات الگوریتم به صورت یک دستور مستقل ذخیره می‌شود. این دستور شامل شناسه، نوع داده، عمل (Action) و پارامترهای مربوطه است. این طراحی باعث می‌شود تمامی عملیات‌ها در قالبی یکسان قابل ذخیره و پردازش باشند.
- گروه دستور (Command Group) : مجموعه‌ای از چند دستور که با هم یک مرحله اتمیک از الگوریتم را تشکیل می‌دهند. به این ترتیب چند تغییر همزمان در قالب یک مرحله واحد ثبت می‌شوند.
- ضبط کامل (Recording) : دنباله‌ای از گروه‌های دستور که در نهایت تاریخچه کامل اجرای الگوریتم را تشکیل می‌دهد.

برای مدیریت این ساختار، یک موتور مرکزی به نام Recorder Engine پیاده‌سازی شده است. این موتور وظایف زیر را بر عهده دارد:

- نگهداری لیست ضبط‌ها (Recording)
- افزودن دستورهای جدید
- گروه‌بندی دستورها در قالب مراحل اتمیک
- آماده‌سازی خروجی برای بخش قاب‌ساز (Framer)

الگوریتم‌ها مستقیماً با این موتور کار نمی‌کنند، بلکه از طریق ضبط‌کننده‌های اختصاصی دستورات خود را ثبت می‌کنند. هر ضبط‌کننده واسطی است بین عملیات الگوریتم و موتور ضبط و الگوریتم با صدا زدن رویه‌های موجود برای هر ضبط کننده اتفاقی که در الگوریتم رخ داده را در ضبط کننده متناظر ضبط می‌کند تا در مراحل بعدی مصور سازی مورد استفاده قرار گیرد.

۴-۳-۱-۱-۱- ضبط‌کننده گراف (Graph Recorder)

ضبط‌کننده گراف برای ثبت تمامی تغییرات و عملیات روی ساختار گراف طراحی شده است. این

ضبط‌کننده به ما اجازه می‌دهد هر عملی که روی گره‌ها و یال‌ها انجام می‌شود را ردیابی کنیم، مانند افزودن یا حذف گره‌ها، افزودن یا حذف یال‌ها و همچنین اعمال تغییرات بصری مانند برجسته کردن یا پاک کردن برجسته‌سازی گره‌ها و یال‌ها.

به عنوان مثال، در یک الگوریتم جستجوی عمق‌اول، وقتی الگوریتم گره‌ای را بازدید می‌کند، دستور مربوط به برجسته‌سازی آن گره ثبت می‌شود تا در مراحل بعدی نمایش، وضعیت گراف قابل پیگیری باشد. این ضبط‌کننده به صورت دقیق تغییرات را به دستورها (Command) تبدیل می‌کند و به موتور ضبط‌کننده ارسال می‌کند تا در نهایت یک ضبط کامل ایجاد شود.

GraphAction

```
export enum GraphAction {
  ADD_NODE = 'AddNode',
  REMOVE_NODE = 'RemoveNode',
  ADD_EDGE = 'AddEdge',
  REMOVE_EDGE = 'RemoveEdge',
  REMOVE_EDGES = 'RemoveEdges',
  SET_NODE_HIGHLIGHT = 'SetNodeHighlight',
  CLEAR_NODE_HIGHLIGHT = 'ClearNodeHighlight',
  CLEAR_ALL_NODES_HIGHLIGHT = 'ClearAllNodesHighlight',
  SET_EDGE_HIGHLIGHT = 'SetEdgeHighlight',
  SET_EDGES_HIGHLIGHT = 'SetEdgesHighlight',
  CLEAR_EDGE_HIGHLIGHT = 'ClearEdgeHighlight',
  CLEAR_EDGES_HIGHLIGHT = 'ClearEdgesHighlight',
  CLEAR_ALL_EDGES_HIGHLIGHT = 'ClearAllEdgesHighlight',
}
```

شکل ۴-۲: اعمال ضبط کننده گراف

۴-۳-۱-۱-۲- ضبط‌کننده آرایه دوبعدی (Array۲D Recorder)

این ضبط‌کننده مسئول ثبت تغییرات روی آرایه‌های دوبعدی است که در بسیاری از الگوریتم‌ها کاربرد دارد. هر عملی که روی سلول‌ها یا سطرها انجام شود، به صورت یک دستور ثبت می‌شود. این ضبط‌کننده به

ویژه برای الگوریتم‌هایی مانند مرتب‌سازی، الگوریتم‌های ماتریسی یا بازی‌های مبتنی بر جدول کاربرد دارد.

به عنوان مثال، در الگوریتم مرتب‌سازی حبابی، هر بار که دو مقدار در یک ردیف جابجا می‌شوند، دستور تغییر سلول‌ها ثبت می‌شود تا در نمایش مراحل، این جابجایی قابل مشاهده باشد. علاوه بر تغییر مقادیر، این ضبط‌کننده امکان برجسته‌سازی محدوده‌ای از سلول‌ها یا سطرها و پاک کردن برجسته‌سازی‌ها را نیز فراهم می‌کند.

Array2D

```
export enum Array2DAction {
  SET_CELLS = 'SetCells',
  INSERT_CELLS = 'InsertCells',
  REMOVE_CELLS = 'RemoveCells',
  PUSH_CELLS = 'PushCells',
  POP_CELLS = 'PopCells',
  SHIFT_CELLS = 'ShiftCells',
  UNSHIFT_CELLS = 'UnshiftCells',
  INSERT_ROWS = 'InsertRows',
  PUSH_ROWS = 'PushRows',
  POP_ROWS = 'PopRows',
  SHIFT_ROWS = 'ShiftRows',
  UNSHIFT_ROWS = 'UnshiftRows',
  SET_CELLS_HIGHLIGHT = 'SetCellsHighlight',
  CLEAR_CELLS_HIGHLIGHT = 'ClearCellsHighlight',
  CLEAR_ALL_CELLS_HIGHLIGHT = 'ClearAllCellsHighlight',
  CLEAR_ALL_ROWS_HIGHLIGHT = 'ClearAllRowsHighlight',
}
```

شکل ۴-۳: اعمال ضبط کننده آرایه دو بعدی

۴-۳-۱-۱-۳- ضبط‌کننده نمودار (Chart Recorder)

این ضبط‌کننده تغییرات روی لیست‌های عددی یا داده‌های نموداری مانند نمودارهای میله‌ای را ثبت می‌کند. هر تغییر روی داده‌ها یا برجسته‌سازی محدوده‌ای از داده‌ها به صورت دستور ثبت می‌شود. این ضبط‌کننده برای الگوریتم‌هایی که با داده‌های عددی سروکار دارند مانند مرتب‌سازی، جستجو و الگوریتم‌های

آماری بسیار مفید است.

به عنوان مثال، در الگوریتم مرتب‌سازی انتخابی، هنگام مقایسه عناصر یک بازه از داده‌ها، عملیات برجسته‌سازی برای آن بازه ثبت می‌شود تا در مرحله نمایش، داده‌های مقایسه شده قابل مشاهده باشند. همچنین تغییر مقدار سلول‌ها یا اضافه و حذف سلول‌ها نیز ثبت می‌شود تا تمام مراحل الگوریتم قابل پیگیری باشند.

Chart

```
export enum ChartAction {
  SET_CELLS = 'SetCells',
  REMOVE_CELLS = 'RemoveCells',
  INSERT_CELLS = 'InsertCells',
  PUSH_CELLS = 'PushCells',
  POP_CELLS = 'PopCells',
  SHIFT_CELLS = 'ShiftCells',
  UNSHIFT_CELLS = 'UnshiftCells',
  SET_CELLS_HIGHLIGHT = 'SetCellsHighlight',
  CLEAR_CELLS_HIGHLIGHT = 'ClearCellsHighlight'
}
```

شکل ۴-۴: اعمال ضبط کننده نمودار

۴-۳-۱-۱-۴- ضبط‌کننده پیام (Log Recorder)

این ضبط‌کننده برای ثبت پیام‌های متنی الگوریتم طراحی شده است. پیام‌ها می‌توانند شامل توضیحات مرحله‌ای، اعلان‌های وضعیت یا اطلاعات کمکی برای کاربر باشند. هر پیام هنگام وقوع به دستور تبدیل می‌شود و در ضبط نهایی ثبت می‌شود.

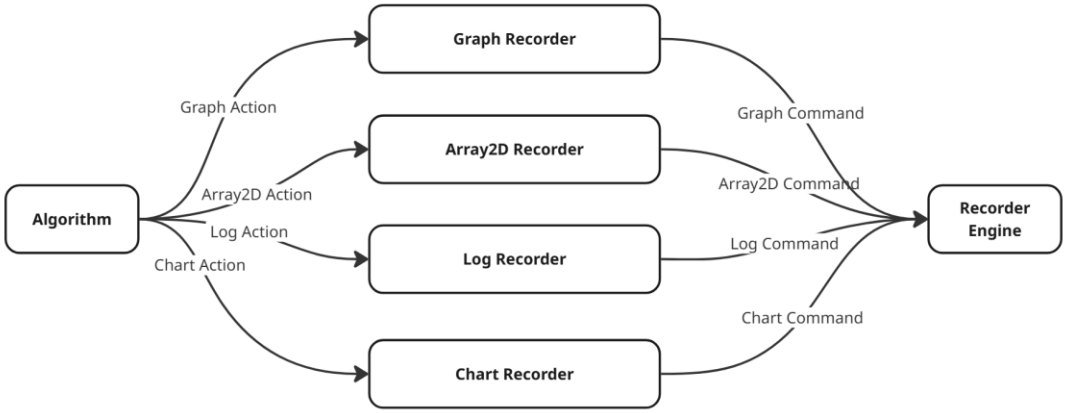
به عنوان مثال، در الگوریتم دایکسترا، بعد از به‌روزرسانی فاصله یک گره، پیام «فاصله گره X به مقدار Y کاهش یافت» توسط دستور مربوطه ثبت می‌شود تا در نمایش مراحل، توضیح مرحله‌ای الگوریتم نشان داده شود.

Log

```
export enum LogAction {
  SET_MESSAGE = 'SetMessage',
  CLEAR_MESSAGE = 'ClearMessage',
}
```

شکل ۴-۵: اعمال ضبط کننده پیام

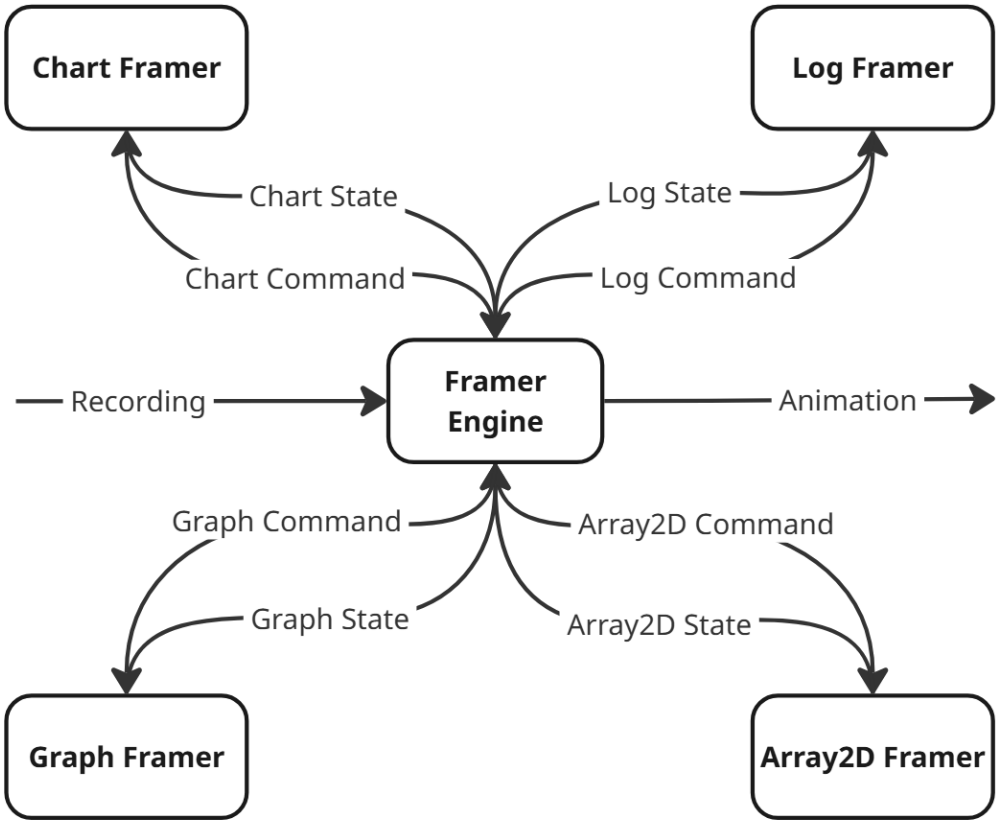
این طراحی ماژولار باعث می‌شود الگوریتم‌ها تنها با متدهای ضبط‌کننده‌ها درگیر باشند و نیازی به دانستن جزئیات ذخیره‌سازی یا ساختار دستورات نداشته باشند. در نتیجه، بخش ضبط‌کننده‌ها یک رابط ساده و در عین حال قدرتمند برای ثبت تمامی رخدادهای الگوریتم فراهم می‌آورد.



شکل ۴-۶: ضبط کننده در عمل

۴-۳-۲- قاب‌سازها

قاب‌سازها (Framer) در معماری سیستم وظیفه دارند دستورهای ثبت شده توسط ضبط‌کننده‌ها (Recorder) را دریافت کرده و آن‌ها را به وضعیت‌های لحظه‌ای (Frame State) تبدیل کنند. این وضعیت‌ها نمایانگر تصویر هر انتزاع در یک فریم مشخص هستند و به نمایش‌گر (Renderer) داده می‌شوند. در واقع، Framer ها میان ضبط (Recording) و نمایش (Rendering) قرار دارند و نقش مترجم بین دستورات الگوریتم و نمایش گرافیکی آن را ایفا می‌کنند. این طراحی باعث می‌شود نمایش‌گرها سبک، مستقل از الگوریتم و ساده‌تر برای توسعه و اضافه کردن نوع جدید داده یا Renderer باشند.



شکل ۴-۷: ضبط کننده در عمل

۴-۳-۲-۱- پیاده‌سازی قاب‌سازها در TypeScript

در پیاده‌سازی فعلی، برای هر نوع داده انتزاعی یک Framer اختصاصی تعریف شده است. هر Framer دستورات مربوط به خود را درک کرده و وضعیت لحظه‌ای متناظر تولید می‌کند. خروجی هر Framer یک Frame State است که نمایش‌گر با استفاده از آن می‌تواند تغییرات الگوریتم را مرحله به مرحله نمایش دهد.

در این پیاده‌سازی جهت استفاده از قاب‌ساز مناسب برای تبدیل هر دستور به وضعیت لحظه‌ای از الگو طراحی استراتژی و برای ساخت نمونه‌ای از قاب‌ساز مناسب از الگوی طراحی کارخانه استفاده شده است.

۴-۳-۲-۱-۱- قاب‌ساز گراف (Graph Framer)

قاب‌ساز (Framer) گراف مسئول تبدیل دستورات مرتبط با گراف‌ها به وضعیت لحظه‌ای گره‌ها و یال‌ها و همچنین مدیریت برجسته‌سازی آن‌ها است. دستورها شامل افزودن/حذف گره‌ها، افزودن/حذف یال‌ها و تغییر برجسته‌سازی گره‌ها و یال‌ها می‌باشند.

Graph

```
export type GraphState = {
  name: string;
  nodes: GraphNodeState[];
  edges: GraphEdgeState[];
  isDirected: boolean;
};

export type GraphNodeState = {
  id: string;
  label: string;
  highlightTags: string[];
};

export type GraphEdgeState = {
  id: string;
  source: string;
  target: string;
  label: string;
  highlightTags: string[];
};
```

شکل ۴-۸: مدل‌های وضعیت لحظه‌ای گراف

هر گره شامل شناسه (id)، برچسب (label) و تگ‌های برجسته‌سازی است و هر یال شامل شناسه (id)، مبدأ و مقصد (source, target)، برچسب و تگ‌های برجسته‌سازی می‌باشد. قاب‌ساز گراف تمامی تغییرات گراف، از جمله افزودن/حذف گره و یال، و اعمال برجسته‌سازی‌ها را مرحله به مرحله تبدیل به وضعیت لحظه‌ای می‌کند تا نمایش‌گر بتواند تغییرات گراف را به صورت دقیق و تعاملی نمایش دهد.

۴-۳-۲-۱-۲- قاب‌ساز آرایه دوبعدی (Array²D Framer)

قاب‌ساز (Framer) آرایه دوبعدی مسئول تبدیل تمامی دستورهای مرتبط با آرایه دوبعدی (مانند افزودن، حذف، جایگزینی یا جابه‌جایی سلول‌ها و ردیف‌ها، و همچنین مدیریت برجسته‌سازی) به وضعیت لحظه‌ای سلول‌ها و ردیف‌ها است.

Array2D

```
export type Array2dState = {
  name: string;
  values: Array2dCellState[][];
};

export type Array2dCellState = {
  value: string;
  highlightTags: string[];
};
```

شکل ۴-۹: مدل‌های وضعیت لحظه‌ای آرایه دوبعدی

هر سلول شامل مقدار (value) و لیست تگ‌های برجسته‌سازی (highlightTags) است. قاب‌ساز (Framer) تغییرات سلول‌ها و ردیف‌ها و اعمال برجسته‌سازی را مرحله به مرحله ثبت می‌کند تا نمایش‌گر بتواند هر فریم از وضعیت آرایه را دقیقاً نشان دهد. این روش اجازه می‌دهد تغییرات هم‌زمان چندین ردیف و سلول به صورت مجزا یا گروهی مشاهده شود.

۴-۳-۲-۱-۳- قاب‌ساز نمودار (Chart Framer)

قاب‌ساز (Framer) نمودار مسئول تبدیل دستورات مرتبط با نمودارهای میله‌ای (Chart) به وضعیت لحظه‌ای میله‌ها و برجسته‌سازی آن‌ها است. عملیات شامل افزودن، حذف، جایگزینی، جابه‌جایی یا تغییر مقدار میله‌ها و همچنین مدیریت برجسته‌سازی‌ها می‌باشد.

Chart

```
export type ChartState = {
  name: string;
  bars: ChartBarState[];
};

export type ChartBarState = {
  value: number;
  label: string;
  highlightTags: string[];
};
```

شکل ۴-۱۰: مدل‌های وضعیت لحظه‌ای نمودار

هر میله نمودار شامل مقدار (value)، برچسب (label) و لیست تگ‌های برجسته‌سازی (highlightTags) است. قاب‌ساز (Framer) تغییرات داده‌ها و برجسته‌سازی‌ها را به ترتیب دریافت کرده و وضعیت لحظه‌ای هر میله را تولید می‌کند. این ساختار امکان نمایش دقیق تغییرات الگوریتمی روی نمودارها، از جمله تغییر هم‌زمان چند میله، را فراهم می‌کند.

۴-۳-۲-۱-۴- قاب‌ساز پیام (Log Framer)

قاب‌ساز (Framer) پیام‌ها مسئول تبدیل دستورهای پیام (Log) به وضعیت لحظه‌ای متن‌ها است. عملیات شامل ثبت پیام جدید، پاک‌کردن پیام‌ها یا به‌روزرسانی پیام‌ها می‌باشد.

Log

```
export type Log = {
  name: string;
  message: string;
};
```

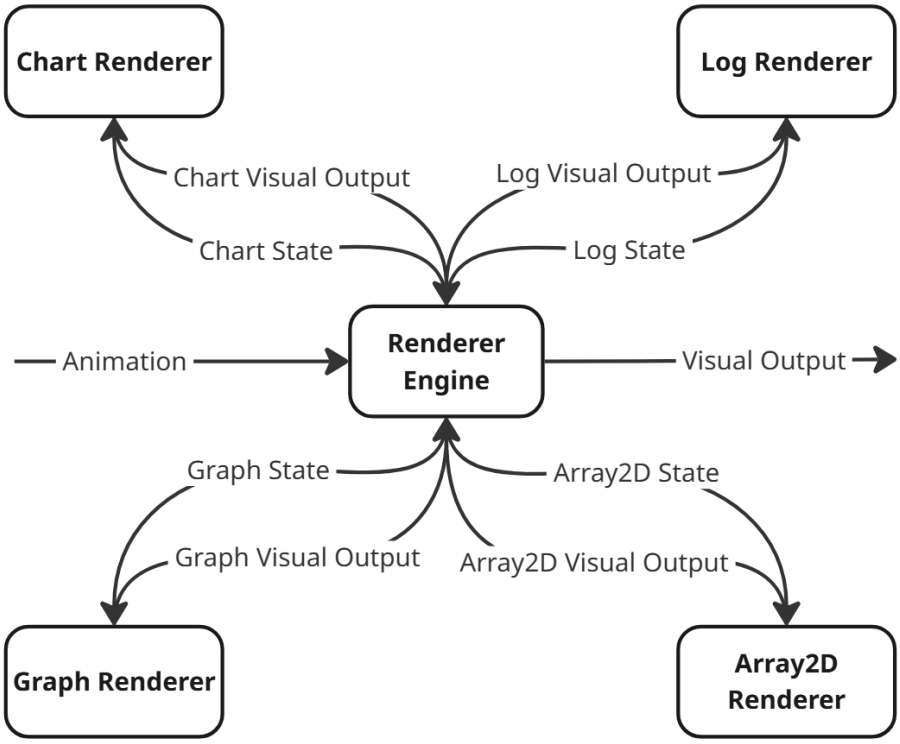
شکل ۴-۱۱: مدل‌های وضعیت لحظه‌ای پیام

هر پیام شامل نام (name) و متن پیام (message) است. قاب‌ساز (Framer) پیام‌ها تغییرات متن‌ها را مرحله به مرحله تبدیل به وضعیت لحظه‌ای می‌کند تا نمایش‌گر بتواند پیام‌ها را در هر مرحله الگوریتم به صورت دقیق نشان دهد. این روش امکان بررسی مرحله به مرحله الگوریتم و تعامل کاربر با جریان داده‌ها را فراهم می‌کند.

۴-۳-۳- نمایش‌گرها

نمایش‌گرها (Renderere) در معماری سیستم وظیفه دارند وضعیت‌های لحظه‌ای (Frame State) تولید شده توسط قاب‌ساز (Framer) را دریافت کرده و آن‌ها را به صورت بصری نمایش دهند. به عبارت دیگر، نمایش‌گرها (Renderere) مسئول تبدیل داده‌های داخلی به تصویر قابل مشاهده برای کاربر هستند.

هر نمایش‌گرها (Renderere) باید بتواند Animation که شامل مجموعه‌ای از وضعیت‌های لحظه‌ای (Frame State) است را دریافت کند و آن را به شکل قابل نمایش تبدیل کند. این رویکرد باعث می‌شود نمایش‌گرها مستقل از الگوریتم باشند و قابلیت استفاده مجدد برای انواع داده و الگوریتم‌ها را داشته باشند.



شکل ۴-۱۲: نمایش‌گر در عمل

۴-۳-۳-۱- پیاده‌سازی نمایش‌گرهای Latex

نمایش‌گر LATEX مسئول تبدیل وضعیت‌های لحظه‌ای (Frame State) تولید شده توسط قاب‌سازها (Framer) به قالب LATEX/TikZ [۸] است، به‌طوری که بتوان مستندات الگوریتم‌ها را در قالب PDF تولید و منتشر کرد. این رویکرد امکان ارائه مستندات دقیق، استاندارد و قابل چاپ را فراهم می‌آورد.

روش کار:

- ورودی این نمایش‌گر (Renderere) یک Animation است که شامل مجموعه‌ای از وضعیت‌های لحظه‌ای (Frame State) انواع داده انتزاعی می‌باشد.
- نمایش‌گر (Renderere) با پردازش هر وضعیت‌های لحظه‌ای (Frame State) ، عناصر بصری مرتبط با آن داده را به قالب LATEX/TikZ تبدیل می‌کند، از جمله:
  - آرایه‌های دوبعدی و برجسته‌سازی سلول‌ها
  - نمودارها و مقادیر هر میله با برجسته‌سازی‌ها
  - گراف‌ها شامل گره‌ها، یال‌ها و هایلایت‌ها
  - پیام‌های متنی وLog ها
- پس از تولید کدهای LATEX، امکان تولید PDF فراهم می‌شود که نمایشی دقیق از وضعیت الگوریتم در هر مرحله ارائه می‌دهد.

مزایا:

- امکان تهیه مستندات چاپی و استاندارد از الگوریتم‌ها
- نمایش دقیق و مرحله به مرحله وضعیت داده‌ها در قالبی قابل اشتراک‌گذاری
- استقلال از زبان برنامه‌نویسی الگوریتم و قابلیت استفاده مجدد برای انواع داده و الگوریتم‌ها

۴-۳-۳-۲- پیاده‌سازی نمایش‌گرهای Web

نمایش‌گر وب با استفاده از Angular طراحی شده است تا امکان ارائه الگوریتم‌های تعاملی و قابل مرور در مرورگر فراهم شود. این نمایش‌گر (Renderere) با استفاده از دو کامپوننت اصلی به نام‌های web-renderer و web-player وضعیت‌های لحظه‌ای (Frame State) تولید شده توسط قاب‌سازها (Framer) را دریافت و بصری‌سازی می‌کند.

روش کار:

- ورودی این نمایش‌گر (Renderere)، مشابه نمایش‌گر LATEX (Renderere)، یک Animation شامل مجموعه‌ای از وضعیت‌های لحظه‌ای (Frame State) انواع داده انتزاعی است.
- کامپوننت Web Renderer مسئول ترسیم وضعیت‌های لحظه‌ای داده‌ها است و عناصر بصری مانند گراف‌ها، آرایه‌ها، نمودارها و پیام‌ها را در قالب HTML و CSS رندر می‌کند.
- کامپوننت Web Player قابلیت کنترل پخش انیمیشن‌ها را فراهم می‌کند، شامل جلو، عقب، توقف و انتخاب فریم خاص.
- با این طراحی، یک اجرای واحد از الگوریتم و تولید Animation کافی است تا چندین کاربر و کامپوننت مختلف بتوانند داده‌ها را به شکل تعاملی مشاهده کنند.

مزایا:

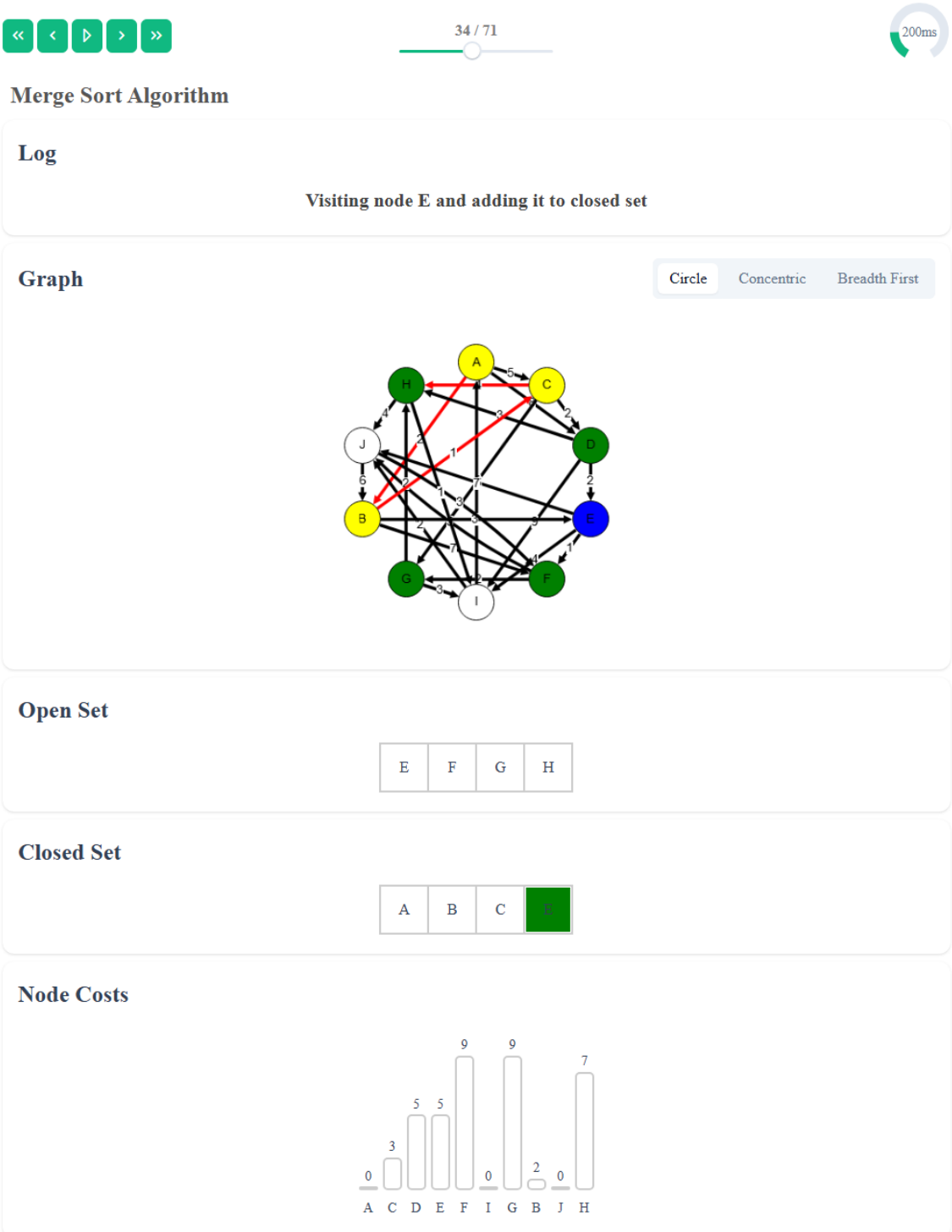
- امکان مصورسازی تعاملی و زنده الگوریتم‌ها در مرورگر
- قابلیت استفاده همزمان از یک Animation برای نمایش‌های مختلف و کنترل تعاملی
- استقلال از زبان برنامه‌نویسی الگوریتم و امکان استفاده مجدد برای هر نوع داده انتزاعی
- امکان توسعه آسان با اضافه کردن کامپوننت‌ها یا افکت‌های بصری جدید

Angular

```
<web-player
  [animationLength]="getAnimationLength()"
  (frameIndexChange)="onFrameIndexChange($event)"
></web-player>
<web-renderer
  [frameIndex]="frameIndex"
  [animation]="animation"
  [rendererMetadata]="rendererMetadata1"
  [hasPlayer]="false"
></web-renderer>
<web-renderer
  [frameIndex]="frameIndex"
  [animation]="animation2"
  [rendererMetadata]="rendererMetadata2"
  [hasPlayer]="false"
></web-renderer>
<web-renderer
  [frameIndex]="frameIndex"
  [animation]="animation3"
  [rendererMetadata]="rendererMetadata2"
  [hasPlayer]="false"
></web-renderer>
```

شکل ۴-۱۳: نمایش‌گر در عمل

در ادامه می‌توانید نمونه‌ای از نمایش‌گر Web برای مصورسازی الگوریتم A* که شامل انواع نمایش‌گرها از جمله پیام، گراف، آرایه دوبعدی و نمودار می‌شود مشاهده کنید.



شکل ۴-۱۴: نمایش‌گر در عمل

۴-۴- جمع‌بندی

در این فصل، طراحی و پیاده‌سازی کامل سیستم مصورسازی الگوریتم‌ها بر اساس معماری چندلایه تشریح شد. ابتدا معماری کلی شامل ضبط‌کننده‌ها (Recorder) موتور قاب‌سازی (Framer Engine) و نمایش‌گرها (Renderer) معرفی شد و چالش‌های مدل اولیه بر پایه داده‌های ردیابی‌شده و لایه انتزاعی بررسی گردید. سپس نحوه پیاده‌سازی هر بخش در TypeScript تشریح شد؛ از جمله ضبط‌کننده‌های مختص انواع داده‌ها و قاب‌سازهای (Framer) متناظر که وضعیت هر انتزاع (Frame State) را تولید می‌کنند. در نهایت، پیاده‌سازی نمایش‌گرها شامل نمایش‌گر Latex (Renderer) برای تولید خروجی PDF و نمایش‌گر Web (Renderer) برای مصورسازی تعاملی با Angular بررسی شد. این ساختار، با جداسازی وظایف، انعطاف‌پذیری بالا و استقلال از زبان برنامه‌نویسی، امکان توسعه آسان، استفاده مجدد از انیمیشن‌ها و افزودن داده‌ها و الگوریتم‌های جدید را فراهم می‌آورد و پایه‌ای قوی برای توسعه سیستم مصورسازی الگوریتم‌ها ایجاد کرده است.

فصل پنجم
نتایج

۵-۱- مقدمه

در این فصل به منظور ارزیابی و نمایش توانمندی‌های چارچوب طراحی‌شده، یک مطالعه‌ی موردی کامل ارائه می‌شود. در این مطالعه، یک الگوریتم نمونه انتخاب شده و کل فرآیند از نوشتن الگوریتم خام، اضافه کردن ضبط‌کننده‌ها (Recorder) به آن، تولید خروجی ضبط‌شده و انتقال آن به قاب‌ساز (Framer) و در نهایت نمایش وضعیت‌های به‌دست‌آمده توسط نمایشگرها (Renderer) مورد بررسی قرار می‌گیرد. هدف از این بخش آن است که نشان داده شود معماری سه‌لایه‌ای پیشنهادی (ضبط، قاب‌سازی و نمایش) چگونه در عمل قابل استفاده بوده و چگونه می‌تواند روند فهم و تحلیل الگوریتم‌ها را برای کاربر تسهیل کند.

۵-۲- اضافه کردن ضبط کننده‌ها

برای نمایش عملی نحوه استفاده از معماری طراحی‌شده، الگوریتم جستجوی دودویی (Binary Search) به عنوان نمونه انتخاب شده است. دلیل این انتخاب آن است که این الگوریتم هم شامل عملیات پیمایش (حرکت روی بازه‌های آرایه) و هم شامل شرط‌گذاری و تصمیم‌گیری است؛ بنابراین می‌تواند به‌خوبی توانایی ضبط‌کننده‌ها(Renderer) و نمایش‌گرها (Renderer) را نشان دهد.

در این پیاده‌سازی، الگوریتم روی یک آرایه‌ی مرتب از اعداد اجرا می‌شود (انتزاع داده‌ای :نمودار یک‌بعدی .(Chart – برای ثبت مراحل اجرا، دو ضبط‌کننده به الگوریتم افزوده شده‌اند:

- ضبط‌کننده‌ی لاگ (Log Recorder) : ثبت پیام‌های متنی الگوریتم در هر مرحله (مانند محدوده‌ی جستجو یا مقایسه با مقدار میانی).
- ضبط‌کننده‌ی نمودار (Chart Recorder) : نمایش وضعیت آرایه و مشخص‌کردن بخش‌های در حال بررسی با استفاده از هایلایت.

ابتدا ضبط‌کننده‌ها به موتور ضبط‌کننده متصل می‌شوند:

Initialization

```
const recorderEngine = new RecorderEngine();

recorderEngine.beginGroup();
const logRecorder = new LogRecorder(recorderEngine, { name: 'Log' });
const chartRecorder = new ChartRecorder(recorderEngine, {
  name: 'Array',
  values: array.map(item => ({ value: item }))
});
recorderEngine.endGroup();
```

شکل ۵-۱: تعریف و مقداردهی اولیه ضبط کننده‌ها

پس از آن، اجرای الگوریتم همراه با فراخوانی ضبط‌کننده‌ها انجام می‌شود. به‌عنوان مثال در هر مرحله بازه‌ی در حال جستجو و عنصر میانی مشخص و در ضبط‌کننده‌ها ثبت می‌گردد:

Recording

```
while (left <= right) {
  const mid = Math.floor((left + right) / 2);

  recorderEngine.beginGroup();
  logRecorder.setMessage({ message: `Searching from index ${left} to ${right}` });
  chartRecorder.clearCellsHighlight({ startIndex: 0, endIndex: array.length - 1 });
  chartRecorder.setCellsHighlight({ startIndex: left, endIndex: right, highlightTags:
['section'] });
  recorderEngine.endGroup();

  recorderEngine.beginGroup();
  logRecorder.setMessage({ message: `Comparing middle cell with ${target}` });
  chartRecorder.setCellsHighlight({ startIndex: mid, endIndex: mid, highlightTags: ['section',
'middle'] });
  recorderEngine.endGroup();

  if (array[mid] === target) {
    recorderEngine.beginGroup();
    logRecorder.setMessage({ message: `Found ${target} at index ${mid}` });
    chartRecorder.clearCellsHighlight({ startIndex: 0, endIndex: array.length - 1 });
    chartRecorder.setCellsHighlight({ startIndex: mid, endIndex: mid, highlightTags:
['target'] });
    recorderEngine.endGroup();
    break;
  } else if (array[mid] < target) {
    left = mid + 1;
  } else {
    right = mid - 1;
  }
}
```

شکل ۵-۲: ضبط الگوریتم با استفاده از ضبط کننده‌های تعریف شده

در پایان، با فراخوانی متد getRecording بر روی Recorder Engine خروجی ضبط تولید می‌شود و با توجه به قرارداد Recording بخشی از خروجی به شکل زیر خواهد بود:

Output

```
return recorderEngine.getRecording();
```

شکل ۵-۳: گرفتن خروجی ضبط

Output


```
[
    {
      "id": "51647ec0-2abd-489c-9ec9-bda2b2f2e4a4",
      "type": "Log",
      "action": "SetMessage",
      "params": {
        "message": "Searching from index 0 to 9"
      }
    },
    {
      "id": "e0d86b39-e70f-4dd3-8ec0-08c990fc962d",
      "type": "Chart",
      "action": "ClearCellsHighlight",
      "params": {
        "startIndex": 0,
        "endIndex": 9
      }
    },
    {
      "id": "e0d86b39-e70f-4dd3-8ec0-08c990fc962d",
      "type": "Chart",
      "action": "SetCellsHighlight",
      "params": {
        "startIndex": 0,
        "endIndex": 9,
        "highlightTags": [
          "section"
        ]
      }
    }
  ],
  [
    {
      "id": "51647ec0-2abd-489c-9ec9-bda2b2f2e4a4",
      "type": "Log",
      "action": "SetMessage",
      "params": {
        "message": "Comparing current section middle cell value with 11"
      }
    },
    {
      "id": "e0d86b39-e70f-4dd3-8ec0-08c990fc962d",
      "type": "Chart",
      "action": "SetCellsHighlight",
      "params": {
        "startIndex": 4,
        "endIndex": 4,
        "highlightTags": [
          "section",
          "middle"
        ]
      }
    }
  ],
```


شکل ۵-۴: خروجی موتور ضبط کننده

۵-۳- تبدیل Recording به Animation

خروجی مرحله‌ی قبل یک Recording است که شامل لیستی از دستورات (Command) مربوط به همه‌ی ضبط‌کننده‌ها می‌باشد. این خروجی هنوز برای نمایش آماده نیست، زیرا صرفاً مجموعه‌ای از عملیات ثبت‌شده است. به همین دلیل در این بخش از موتور قاب‌ساز (Framer Engine) استفاده می‌شود.

موتور قاب‌ساز وظیفه دارد که ضبط (Recording) را به مجموعه‌ای از فریم‌ها (Frame) تبدیل کند. هر فریم نمایانگر وضعیت کامل انتزاع‌ها (مانند آرایه یا لاگ) در یک لحظه از اجرای الگوریتم است. حاصل این فرایند یک انیمیشن (Animation) است که در آن توالی وضعیت‌ها به ترتیب وقوعشان ذخیره شده‌اند.

در کد زیر، ابتدا ضبط تولید شده از الگوریتم به موتور قاب‌ساز داده می‌شود و خروجی آن در قالب انیمیشن استخراج می‌گردد:

Converting Recording into Animation

```
const recording = RecordBinarySearch(array, target);

const framer = new FramerEngine();
const animation = framer.getAnimation(recording);
```

شکل ۵-۵: تبدیل Recording به Animation

در اینجا:

- متد getAnimation مسئول تبدیل دستورات هر ضبط‌کننده (Recorder) به وضعیت‌های متناظر آن (Frame States) است.
- برای هر نوع داده (Graph, Chart, Log, Array۲D, …) یک قاب‌ساز (Framer) متناظر وجود دارد که دستورات مربوط به همان نوع داده را پردازش کرده و وضعیت نهایی را می‌سازد.
- در نهایت، خروجی این بخش یک انیمیشن واحد است که شامل همه‌ی فریم‌های ثبت‌شده از الگوریتم می‌باشد.

به این ترتیب، داده‌ی خام ضبط‌شده در قالبی منسجم و قابل‌فهم برای نمایش‌گرها (Renderer) در می‌آید.

بخشی از خروجی موتور قاب‌ساز (Framer Engine) برای الگوریتم جستجوی دودویی آورده شده است:

Output


```
[
  {
    "id": "030a03a4-ff90-4ee1-8746-de1e929cac45",
    "type": "Log",
    "state": {
      "name": "Log",
      "message": "Comparing current section middle cell value with 11"}
  },
  {
    "id": "3691ddd3-8ec0-4016-b552-3be9e6f2d1b4",
    "type": "Chart",
    "state": {
      "name": "Array",
      "bars": [
        {"value": 1, "label": null, "highlightTags": ["section"]},
        {"value": 3, "label": null, "highlightTags": ["section"]},
        {"value": 5, "label": null, "highlightTags": ["section"]},
        {"value": 7, "label": null, "highlightTags": ["section"]},
        {"value": 9, "label": null, "highlightTags": ["section", "middle"]},
        {"value": 11, "label": null, "highlightTags": ["section"]},
        {"value": 13, "label": null, "highlightTags": ["section"]},
        {"value": 15, "label": null, "highlightTags": ["section"]},
        {"value": 17, "label": null, "highlightTags": ["section"]},
        {"value": 20, "label": null, "highlightTags": ["section"]},
        {"value": 13, "label": null, "highlightTags": ["section"]}
      ]
    }
  }
]
```


شکل ۵-۶: خروجی موتور قاب‌ساز

۵-۴- تبدیل Animation به خروجی

پس از تولید انیمیشن در مرحله قبل، نوبت به نمایش‌گرها (Renderer) می‌رسد. نمایش‌گر وظیفه دارد انیمیشن (Animation) را دریافت کرده و وضعیت‌های مختلف آن را در قالب خروجی قابل مشاهده تولید کند.

هر نمایش‌گر می‌تواند پیاده‌سازی متفاوتی داشته باشد اما همگی از یک قرارداد مشترک پیروی می‌کنند:

- ورودی: یک انیمیشن شامل مجموعه‌ای از فریم‌ها.

- خروجی: فایل یا محتوای قابل مشاهده مانند Latex یا یک وب‌کامپوننت تعاملی.

از آنجا که هر نوع داده ممکن است نیازمند تنظیمات خاص خود باشد، برای هر Renderer امکان تعریف فراداده‌ها (Metadata) فراهم شده است. این فراداده‌ها مشخص می‌کنند که هر انتزاع (مانند نمودار یا لاگ) چگونه نمایش داده شود. به عنوان مثال، رنگ‌گذاری هایلایت‌ها یا نحوه چینش متن‌ها توسط فراداده تعیین می‌شوند.

کد زیر نمونه‌ای از استفاده از نمایش‌گر Latex (Renderer) برای تولید محتوای Latex را نشان می‌دهد :

Convertin Animation into Latex String

```
const latexString = renderer.render(animation, {
  documentName: "Binary Search",
  objectMetaData: [
    {
      type: 'Log',
      metadata: {
        alignName: 'left',
      }
    },
    {
      type: 'Chart',
      metadata: {
        alignName: 'left',
        highlightTags: [
          { tag: 'section', color: 'blue!50' },
          { tag: 'middle', color: 'brown!50' },
          { tag: 'target', color: 'green!50' }
        ]
      }
    }
  ]
});
```

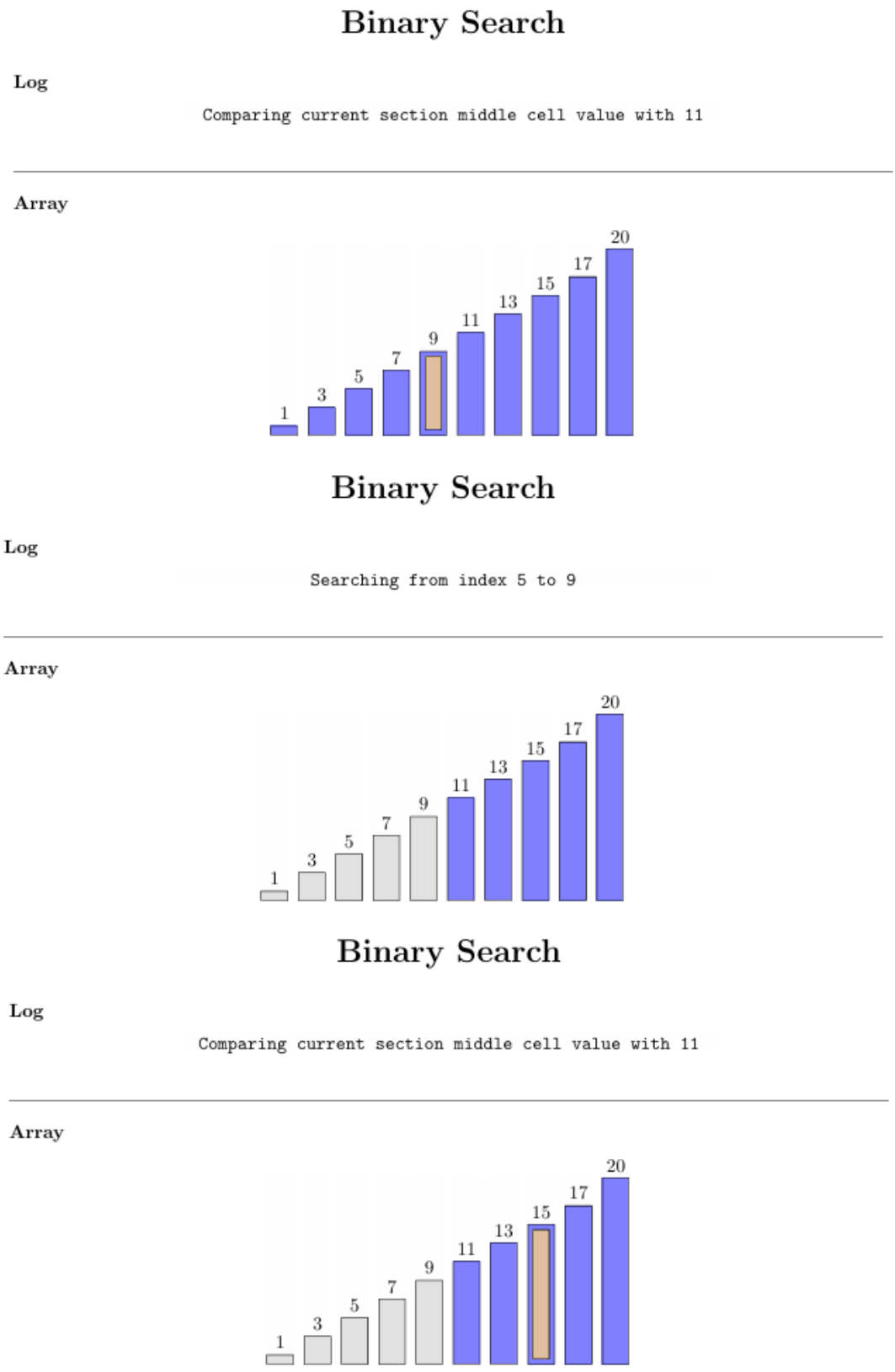
شکل ۵-۷: تبدیل Animation به محتوای Latex

در اینجا:

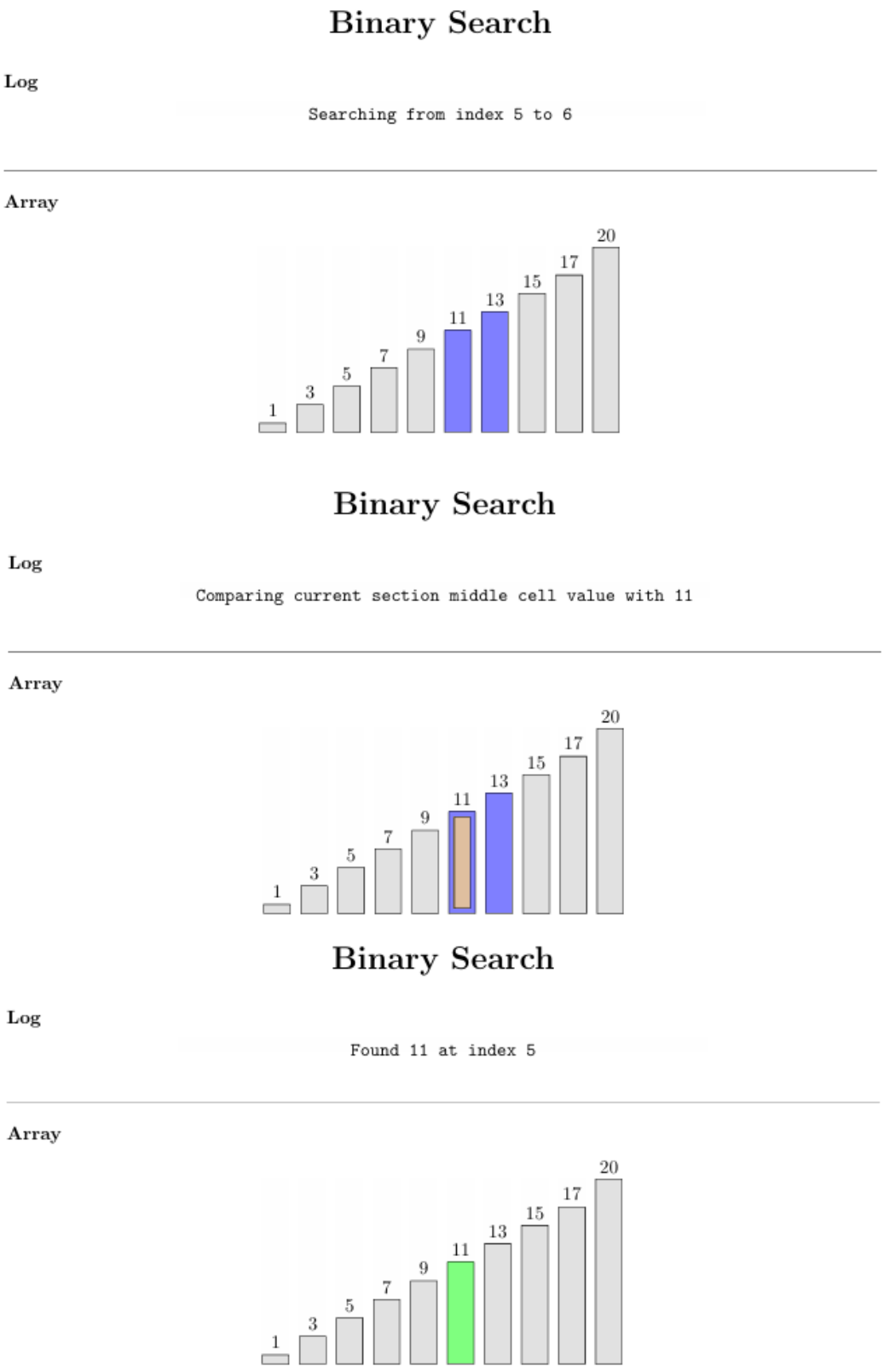
- برای شیء Log مشخص شده که متن‌ها در سمت چپ قرار گیرند.
- برای شیء Chart علاوه بر تنظیم مکان، رنگ‌های هایلایت برای بخش‌های مختلف (محدوده جستجو، خانه میانی، عنصر هدف) تعیین شده‌اند.
- خروجی نهایی یک محتوای LaTeX است که با اجرای دستور pdflatex به صورت خودکار به PDF تبدیل می‌شود.

به این ترتیب، با اضافه کردن فراداده و اجرای نمایش‌گر (Renderer) می‌توان برای یک انیمیشن واحد، نمایش‌های مختلفی تولید کرد.

شکل ۵-۸: خروجی تبدیل شده به PDF



شکل ۵-۹: خروجی تبدیل شده به PDF



شکل ۵-۱۰: خروجی تبدیل شده به PDF

۵-۵- جمع‌بندی

در این فصل روند کامل اجرای سیستم از ابتدا تا انتها نشان داده شد. ابتدا یک الگوریتم نمونه (جستجوی دودویی) انتخاب گردید و با اضافه کردن ضبط‌کننده‌ها، رخدادهای آن ثبت شد. سپس این داده‌ها توسط موتور قاب‌ساز (Framer Engine) به انیمیشن شامل فریم‌های متوالی تبدیل شدند. در نهایت، با استفاده از موتور نمایش‌گر (Renderer Engine) خروجی قابل مشاهده تولید گردید. همچنین بیان شد که هر نمایش‌گر می‌تواند بسته به نیاز، فراداده‌های اختصاصی برای تنظیم جزئیات نمایش دریافت کند. به این ترتیب، روشن شد که معماری سه‌لایه ضبط، قاب‌سازی و نمایش، فرآیندی یکپارچه و قابل توسعه برای تولید انواع خروجی‌های متنی و تصویری فراهم می‌سازد.

فصل ششم
نتیجه‌گیری و پیشنهادات

۶-۱- نتیجه‌گیری

پروژه حاضر با هدف طراحی یک چارچوب کامل برای ضبط، قاب‌سازی و نمایش الگوریتم‌ها با قابلیت خروجی چندگانه پیش رفت. در طول اجرای پروژه، ابتدا معماری مبتنی بر ضبط‌کننده‌ها (Recorder) تعریف شد تا تمامی رخدادهای الگوریتم‌ها ثبت شوند. سپس موتور قاب‌ساز (Framer Engine) این داده‌های ضبط‌شده را به توالی فریم‌های کامل (Frame State) تبدیل کرد و در نهایت نمایش‌گرها (Renderer) قادر شدند این فریم‌ها را به خروجی‌های متنی (LaTeX/TikZ) و تعاملی وب ارائه دهند.

این طراحی با جداسازی سه‌لایه ضبط، قاب‌سازی و نمایش، موجب شد که توسعه و افزودن الگوریتم‌ها و انتزاع‌های جدید به کاژول‌ها بسیار ساده باشد. همچنین معماری بی‌وابستگی به زبان برنامه‌نویسی (Language-Agnostic) امکان استفاده از سیستم در محیط‌ها و پروژه‌های مختلف را فراهم کرد. در حال حاضر، چارچوب قادر است آرایه‌ها، نمودارها، گراف‌ها و داده‌های متنی را به‌طور کامل پشتیبانی کند و خروجی‌ها را هم به شکلLatex و هم در قالب کامپوننت وب تعاملی تولید نماید. با این حال، محدودیت‌هایی مانند پردازش مرحله‌به‌مرحله و عدم پشتیبانی از الگوریتم‌های پایان‌ناپذیر، همچنان به‌عنوان نقاط قابل بهبود باقی مانده‌اند.

۶-۲- پیشنهادات

با توجه به ساختار فعلی و تجربیات کسب‌شده، چند مسیر توسعه و بهبود سیستم پیشنهاد می‌شود:

- اضافه کردن ورژنینگ (Versioning) به قرارداد Recording و Animation

پیاده‌سازی ورژنینگ امکان مدیریت تغییرات بین نسخه‌های مختلف ضبط یا فریم‌ها را فراهم می‌کند و سازگاری نسخه‌ها حفظ می‌شود. این ویژگی باعث می‌شود توسعه‌دهندگان بدون نگرانی از

ناسازگاری، ماژول‌های جدید اضافه کنند یا تغییرات را اعمال نمایند و تعامل بین اجزا بهبود یابد.

- پیاده‌سازی حالت جریان‌محور (Streaming)

قابلیت جریان‌محور (Streaming) اجازه می‌دهد داده‌های ضبط به‌صورت پیوسته و هم‌زمان به قاب‌ساز و نمایش‌گر ارسال شوند، بدون نیاز به تولید کامل هر مرحله. این ویژگی امکان نمایش الگوریتم‌های بی‌پایان یا جریان‌های داده طولانی را فراهم کرده و باعث افزایش کارایی و انعطاف سیستم می‌شود.

- اضافه کردن ضبط کننده، قاب‌ساز و نمایش‌گر برای انتزاع‌های جدید

توسعه سیستم برای پشتیبانی از انتزاع‌های دیگری مانند درخت‌ها امکان مصورسازی الگوریتم‌های پیچیده‌تر را فراهم می‌کند. این کار با توسعه ضبط‌کننده‌ها، قاب‌سازها و نمایش‌گرهای متناظر صورت می‌گیرد.

- توسعه نمایش‌گرهای جدید

علاوه بر خروجی‌های فعلی، اضافه کردن نمایش‌گرهای جدید مانند خروجی GIF یا ویدیو تجربه بصری و قابلیت اشتراک‌گذاری نتایج را بهبود می‌بخشد. این کار می‌تواند شامل تولید انیمیشن‌های قابل دانلود یا ایجاد پخش‌کننده داخلی برای مرور فریم‌ها باشد.

- افزودن قابلیت تعامل کاربر در حین اجرای الگوریتم

با فراهم کردن امکان ارسال داده یا پارامتر از سوی کاربر در طول اجرای الگوریتم، سیستم می‌تواند برای آموزش، بازی‌سازی یا نمایش زنده الگوریتم‌ها کاربردی‌تر شود. این ویژگی امکان آزمایش سناریوهای مختلف و بررسی رفتار الگوریتم‌ها را به‌صورت تعاملی فراهم می‌سازد. برای مثال با این توسعه می‌توان واکنش الگوریتمی مانند A* را پس از افرودن یک یال به گراف در حین اجرا برسی کرد.

- دریافت ورودی به صورت تعاملی

با فراهم کردن بستری که با استفاده از آن بتوان به صورت تعاملی ورودی‌های مورد نیاز را برای الگوریتم مورد نظر فراهم کردم می‌توان از ثابت بودن ورودی‌ها داخل کد جلوگیری کرده و در هر اجرا ورودی دل‌خواه خود را برای الگوریتن فراهم کنیم. برای مثال بستری تحت وب داشته باشیم که با استفاده از آن بتوان آرایه اولیه جهت الگوریتم مرتب‌سازی حبابی را فراهم کرد.

- ادغام ضبط کننده‌ها با داده‌ساختارها

اگر بتوان با تغییراتی در معماری کنونی روشی ارائه کرد که ضبط کننده مربوط به هر انتزاع را داخل داده‌ساختار همان انتزاع ادغام کند، میزان کد مورد نیاز جهت ضبط الگوریتم در حال اجرا به صورت قابل توجهی ماهش می‌یابد. برای مثال وقتی به داده‌ساختار آرایه یک مقدار اضافه کردیم به صورت

خودکار این عمل در ضبط کننده مربوط به آرایه ضبط شود.

- بهینه‌سازی حافظه و ذخیره‌سازی فریم‌ها

با استفاده از تکنیک‌های فشرده‌سازی یا ذخیره‌سازی تفاضلی (Delta Storage) می‌توان حجم داده‌های ضبط و فریم‌ها را کاهش داد، بدون از دست رفتن اطلاعات مهم. این کار برای الگوریتم‌ها و داده‌های حجیم کاربردی خواهد بود.

- توسعه کامپایلر برای تبدیل شبه کد به کد مصورسازی

کامپایلری توسعه داده شود که یک زبان شبه کد دارای تگ‌هایی برای مصورسازی را دریافت کند و به کد تایپ اسکریپت با توابع مورد نیاز جهت ضبط الگوریتم تبدیل کند و بدین صورت فرایند مصورسازی الگوریتم‌ها را ساده‌تر کند.

- توسعه مدل زبانی جهت تبدیل شبه کد به کد مصورسازی

مدل زبانی آموزش داده شود که هر شبه کدی با زبان طبیعی را دریافت کند و با درک الگوریتم، توابع مورد نیاز جهت ضبط الگوریتم برای مصورسازی را به آن بیافزاید و بدین صورت دانش اولیه مورد نیاز جهت مصورسازی الگوریتم‌ها توسط این چارچوب را کمینه کند.

با اعمال این پیشنهادات، چارچوب می‌تواند قابلیت‌های فعلی خود را تقویت کرده و به سمت پشتیبانی از الگوریتم‌های پیچیده‌تر و نمایش‌های تعاملی‌تر پیش رود، به طوری که هم برای پژوهش‌های علمی و هم برای آموزش و ارائه الگوریتم‌ها، یک ابزار کامل و قابل اعتماد باشد.

منابع:

[١] https://visualgo.net/en

[٢] https://algorithm-visualizer.org/
[٣] https://sortvisualizer.com/
[٤] David Galles, https://www.cs.usfca.edu/~galles/visualization/Algorithms.html

[٥] https://www.texstudio.org/

[٦] https://github.com/ui-ce/algo-visualizer

[٧] https://github.com/ui-ce/algorithm-visualizer.

[٨] T. Tantau, Tik Z and pgf The Tik Z and PGF Packages, ٢٠٠٧. [Online]. Available: http://sourceforge.net/projects/pgf