
بسم الله الرحمن الرحيم

دانشگاه اصفهان

دانشکده مهندسی کامپیوتر

گروه مهندسی نرم‌افزار

گزارش پروژه کارشناسی

رشته مهندسی کامپیوتر گرایش نرم‌افزار

عنوان پژوهش

ارائه راهکاری برای مصورسازی الگوریتم‌های ساختار داده توسط زبان لاتک

استاد راهنما:

دکتر افسانه فاطمی

دکتر آرش شفیعی

پژوهشگران:

مهروسادات نوحی

یاسمین اکبری

شهریور ۱۴۰۳

دانشگاه اصفهان

دانشکده مهندسی کامپیوتر

گروه مهندسی نرم‌افزار

پروژه کارشناسی رشته‌ی مهندسی کامپیوتر گرایش نرم‌افزار

خانم‌ها مهروسادات نوحی و یاسمین اکبری

تحت عنوان

ارائه راهکاری برای مصورسازی الگوریتمهای ساختار داده توسط زبان لاتک

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

۱- استاد راهنمای پژوهش:

دکتر امضا

۲- استاد داور :

دکتر امضا

امضای مدیر گروه

تشکر و قدردانی

اکنون که به یاری خداوند این دوره را به پایان رساندیم، بر خود واجب میدانیم از اساتید راهنمای گرامی سرکار خانم دکتر فاطمی و جناب آقای دکتر آرش شفیعی به پاس زحمات بیشائبه‌شان در طی انجام این تحقیق سپاسگزاری نماییم.

همچنین از خانوادهایمان که در تمام دوران تحصیلی مارا حمایت کرده‌اند تقدیر و تشکر مینماییم.

تقدیم به

تمام کسانی که در این تاریکی

روشنایی‌بخش راه ما بودند.

چکیده:

در دنیای امروز، الگوریتم‌ها به عنوان قلب تپنده محاسبات و پردازش‌های کامپیوتری شناخته می‌شوند. از مرتب‌سازی داده‌ها و جست‌وجوی اطلاعات تا مدیریت منابع در سیستم‌های چندکاربره، الگوریتم‌ها نقشی کلیدی در بهینه‌سازی و اجرای وظایف مختلف دارند. به دلیل اهمیت بالای الگوریتم‌ها، درک عمیق و دقیق عملکرد آن‌ها برای دانشجویان، پژوهشگران، و برنامه‌نویسان امری ضروری است.

پروژهشی که در این گزارش مورد بررسی قرار گرفته شده است، با هدف پیاده‌سازی و مصورسازی طیف گسترده‌ای از الگوریتم‌های ساختار داده در زبان برنامه‌نویسی پایتون توسعه یافته است. این پروژهش نه تنها شامل پیاده‌سازی الگوریتم‌های مختلفی مانند الگوریتم‌های جست‌وجو، مرتب‌سازی، مدیریت لیست‌های پیوندی، انواع درخت، استک و صف است، بلکه با تولید خودکار فایل‌های لاتک، فرایند اجرای هر الگوریتم را به صورت بصری و گرافیکی نمایش می‌دهد.

برای درک و تعامل بهتر، امکان شخصی‌سازی تصاویر نیز برای کاربران فراهم شده است. کاربران می‌توانند باتوجه به نیاز و خروجی مورد نظر خود مشخص کنند که تصاویر خروجی با چه ابعاد، رنگ‌ها، و داده‌های اولیه‌ای ایجاد شوند.

یکی از چالش‌های اصلی در آموزش و یادگیری الگوریتم‌ها، فهم دقیق نحوه عملکرد هر الگوریتم در مراحل مختلف است. این پروژه با ارائه تصاویر و نمودارهای گام به گام، به مخاطبان کمک می‌کند تا به صورت تعاملی و بصری، نحوه عملکرد هر الگوریتم را مشاهده و درک کنند.

این گزارش به بررسی مراحل توسعه این پروژه، تکنیک‌های به کار گرفته شده برای پیاده‌سازی الگوریتم‌ها، و همچنین روش‌های مصورسازی آن‌ها می‌پردازد. امید است این پروژه به عنوان یک منبع آموزشی و تحقیقاتی مورد استفاده قرار گیرد و به بهبود درک مفاهیم پایه‌ای علوم ومهندسی کامپیوتر کمک نماید.

واژگان کلیدی: الگوریتم، مصورسازی، لاتک

فهرست مطالب

عنوان **صفحه**

فهرست مطالب

عنوان **صفحه**

فهرست مطالب

عنوان **صفحه**

فهرست مطالب

عنوان **صفحه**

فهرست مطالب

عنوان **صفحه**

فهرست مطالب

عنوان **صفحه**

فهرست شکل‌ها

عنوان **صفحه**

کوته‌نوشت‌ها:

LIFO	Last In, First Out
FIFO	First In, First Out
FCFS	First Come, First Served
CPU	Central Processing Unit
SJN	Shortest Job Next
SRT	Shortest Remaining Time
MLFQ	Multi-Level Feedback Queue
RR	Round Robin
AVL	Adelson-Velsky and Landis
PyPI	Python Package Index
Pip	Python Installs Packages
BFS	Breadth-First Search
DFS	Depth-First Search
BST	Binary Search Tree

فصل اول
مقدمه

۱-۱- هدف پژوهش

امروزه عملکرد الگوریتم‌ها نقش بسیار مهمی درزمینه‌های مختلف از جمله علوم کامپیوتر، مهندسی، علوم پایه و حتی زندگی روزمره ما ایفا می‌کنند. با این حال، درک مفاهیم مربوط به الگوریتم‌ها برای بسیاری از افراد چالش برانگیز است. هدف از این پژوهش، ارائه یک راهکار نوآورانه برای تسهیل فرآیند یادگیری الگوریتم‌ها از طریق تجسم بصری و تعامل کاربر است. علاوه بر این، کاربران می‌توانند با مشاهده عملکرد الگوریتم‌ها و نمایش گام به گام اجرای الگوریتم بانحوه کارکرد آن آشنا بشوند. استفاده از روش‌های بصری و تعاملی در این ابزار، یادگیری الگوریتم‌ها را جذاب‌تر می‌کند، به خصوص برای افرادی که تمایل ذاتی به مفاهیم انتزاعی ندارند. در این پژوهش، کاربران می‌توانند اطلاعات اولیه مربوط به الگوریتم مورد نظر خود که در فهرست الگوریتم‌های پیاده‌سازی شده وجود دارد را بدهند؛ سپس، سیستم به صورت گام به گام مراحل اجرای الگوریتم را به صورت تصویری با استفاده از زبان لاتک[1] مصورسازی کند.

۱-۲- کاربردهای پژوهش

از کاربردهای پژوهش، می‌توان درآموزش، یادگیری، پژوهش و مرجع اشاره نمود. دربخش آموزش، این ابزارمی‌تواند در کلاس‌های درس، کتاب‌های آموزشی، دوره‌های آنلاین و یا به صورت خودآموزی برای آموزش الگوریتم‌های کامپیوتری به کار گرفته شود. در بخش یادگیری، دانش‌آموزان و دانشجویان می‌توانند از این ابزار برای یادگیری مفاهیم الگوریتم‌ها به روشی بصری استفاده کنند. دربخش پژوهش، محققان می‌توانند از این ابزار برای بررسی عملکرد الگوریتم‌های مختلف استفاده کنند و همچنین دربخش مرجع، این ابزار می‌تواند به عنوان منبعی برای افرادی که در حال حاضر بر روی مقاله‌ای یا پژوهشی کار می‌کنند مؤثر واقع شود.

---

[1] Latex

۱-۳- ساختار پایان نامه

در این پایان‌نامه به ارائه راهکاری برای مصورسازی الگوریتم‌های ساختار داده توسط زبان لاتک پرداخته شده است.

ساختار پایان‌نامه به صورت زیراست:

- در فصل دوم مروری بر ادبیات پژوهش انجام خواهد شد.
- در فصل سوم پیشینه پژوهش و به بررسی کارهای مشابه پرداخته خواهد شد.
- در فصل چهارم شرح پژوهش و همچنین پیاده‌سازی کامل مصورسازی الگوریتم‌ها توسط لاتک بررسی خواهد شد.
- در فصل پنجم کتابخآن‌های برای مصورسازی الگوریتم‌های مختلف کامپیوتری ارائه خواهد شد.
- در فصل ششم نتیجه‌گیری، ارزیابی و پیشنهادات وهمچنین توضیح کارهای آینده جهت بهبود و ادامهٔ پژوهش بررسی می‌شود.

فصل دوم
ادبیات پژوهش

۲-۱- مقدمه

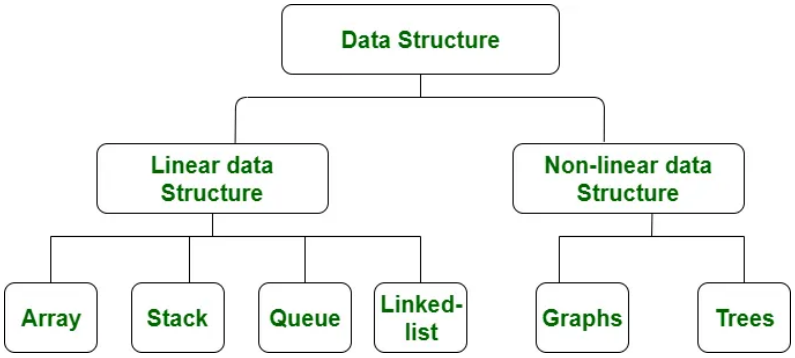
در این قسمت ادبیات پژوهش مورد بررسی قرار می‌گیرد. این قسمت شامل معرفی مفاهیم، ابزارها و کتابخانه‌هایی است که در انجام این پژوهش از آن‌ها استفاده شده است. در انجام این پژوهش الگوریتم‌هایی متعددی به کار گرفته شدند که در این فصل به توضیح مختصری از آن‌ها پرداخته می‌شود.

۲-۲- الگوریتم

به طورکلی الگوریتم مجموعه‌ای از دستورات یا قواعد مشخص و تعریف‌شده است که برای حل یک مسئله یا انجام یک وظیفه به ترتیب خاصی اجرا می‌شود. این دستورات باید به گونه‌ای باشند که در نهایت به یک نتیجه مشخص برسند[۱].

۲-۳- الگوریتم‌های ساختار داده

الگوریتم‌ها بسته به نوع کاربرد و ویژگی‌هایشان به دسته‌های مختلفی تقسیم می‌شوند[۲]. یکی از مهم‌ترین دسته‌ها، الگوریتم‌های ساختار داده هستند. الگوریتم‌های ساختار داده بلوک‌های اساسی برنامه‌نویسی کامپیوتری هستند. آن‌ها نحوه سازماندهی، ذخیره و دستکاری داده‌ها را در یک برنامه تعریف می‌کنند. الگوریتم‌های ساختار داده فقط برای سازماندهی داده‌ها استفاده نمی‌شود بلکه برای پردازش، بازیابی و ذخیره داده‌ها نیز به کار گرفته می‌شوند. الگوریتم‌های ساختار داده بسته به نوع ساختار داده‌ای که استفاده می‌کنند به صورت زیر تقسیم بندی می‌شوند.



شکل ۲-۱:تقسیم بندی الگوریتم‌های ساختارداده

ساختارداده‌هایی که در این پژوهش مورد استفاده قرار گرفتند:

الگوریتم‌های مرتبط با آرایه: برای این دسته از الگوریتم‌ها، ساختمان داده از نوع آرایه، به عنوان ورودی برای الگوریتم در نظر گرفته می‌شود.

الگوریتم‌های مرتبط با آرایه که در پژوهش به کار گرفته شده است، به شرح زیر می‌باشد:

۲-۳-۱- الگوریتم جستجوخطی[1]

الگوریتم جستجوی خطی یک روش ساده و ابتدایی برای جستجوی یک مقدار مشخص در یک آرایه است. در این الگوریتم، هر عنصر از آرایه به ترتیب بررسی می‌شود تا زمانی که عنصر مورد نظر پیدا شود یا تمام عناصر بررسی شوند.

۲-۳-۲- الگوریتم جستجودودویی[2]

الگوریتم جستجوی دودویی یک روش کارآمد برای جستجوی مقدار مشخصی در یک لیست مرتب‌شده است. این الگوریتم با استفاده از تقسیم مکرر لیست به دو بخش کوچکتر، به سرعت محل عنصر مورد نظر را پیدا می‌کند.

۲-۳-۳- الگوریتم مرتب‌سازی درجی[3]

الگوریتم مرتب‌سازی درجی یکی از الگوریتم‌های ساده و کارآمد برای مرتب‌سازی لیست‌ها و آرایه‌های کوچک است. این الگوریتم به این صورت عمل می‌کند که عناصر یک آرایه را یکی یکی انتخاب کرده و آن‌ها را در جای مناسب خود در لیست مرتب‌شده قرار می‌دهد.

۲-۳-۴- الگوریتم مرتب‌سازی انتخابی[4]

الگوریتم مرتب‌سازی انتخابی یکی از الگوریتم‌های مرتب‌سازی ساده و اولیه در علوم کامپیوتر است. این

---

[1] Linear Search Algorithm
[2] Binary Search Algorithm
[3] Insertion Sort
[4] Selection Sort

الگوریتم به این صورت عمل می‌کند که در هر مرحله کوچکترین (یا بزرگترین، بسته به نوع مرتب‌سازی) عنصر را از میان عناصر باقی‌مانده لیست پیدا کرده و آن را در جایگاه درست خود قرار می‌دهد.

۲-۳-۵- الگوریتم مرتب‌سازی حبابی[1]

الگوریتم مرتب‌سازی حبابی یکی از ساده‌ترین الگوریتم‌های مرتب‌سازی است که به صورت تکراری دو به دو عناصر مجاور را مقایسه کرده و در صورت نیاز آن‌ها را جابه‌جا می‌کند تا در نهایت لیست مرتب شود.

۲-۳-۶- الگوریتم مرتب‌سازی سریع[2]

الگوریتم مرتب‌سازی سریع الگوریتمی است که از روش تقسیم و حل[3] استفاده می‌کند. ابتدا یک عنصر به عنوان محور[4] انتخاب می‌شود، سپس لیست به دو زیرلیست تقسیم می‌شود: زیرلیست چپ شامل عناصری که کوچکتر یا مساوی محور هستند و زیرلیست راست شامل عناصری که بزرگتر از محور هستند. این فرآیند به صورت بازگشتی برای هر زیرلیست تکرار می‌شود تا تمام عناصر مرتب شوند.

۲-۳-۷- الگوریتم مرتب‌سازی ادغامی[5]

الگوریتم مرتب‌سازی ادغامی یک الگوریتم مرتب‌سازی کارآمد است که مانند الگوریتم مرتب‌سازی سریع از روش تقسیم و حل استفاده می‌کند. این الگوریتم به‌طور بازگشتی لیست را به دو نیمه تقسیم کرده، هر نیمه را به‌طور جداگانه مرتب می‌کند و سپس دو لیست مرتب‌شده را با هم ادغام می‌کند تا لیست نهایی مرتب‌شده حاصل شود.

الگوریتم‌های مرتبط با لیست پیوندی[6]: ساختمان داده این دسته از الگوریتم‌ها یک لیست پیوندی است. برای پیاده‌سازی لیست پیوندی یک طرفه و دو طرفه مورد استفاده قرار گرفت.

الگوریتم‌های مرتبط با لیست پیوندی یک طرفه[7] که در پژوهش به کار گرفته شده است، به شرح زیر می‌باشد:

۲-۳-۸- الگوریتم ساخت لیست پیوندی یک طرفه

لیست پیوندی یک‌طرفه یک ساختار داده‌ای پویا است که در آن مجموعه‌ای از عناصر به هم مرتبط هستند. هر عنصر در لیست یک گره[8] نامیده می‌شود و شامل دو بخش است: داده و اشاره‌گری[9] به گره بعدی. آخرین

---

[1] Bubble Sort
[2] Quick Sort
[3] Divide and Conquer
[4] Pivot
[5] Merge Sort
[6] Linked List
[7] Single Linked List
[8] Node
[9] Pointer

گره در لیست به گره‌ای اشاره نمی‌کند و اشاره‌گر آن به طور معمول مقدار null دارد. در این نوع لیست، امکان دسترسی به گره‌های قبل از گره جاری به طور مستقیم وجود ندارد و حرکت در لیست تنها به سمت جلو (یک‌طرفه) امکان‌پذیر است.

۲-۳-۹- الگوریتم اضافه شدن گره به لیست پیوندی یک طرفه

برای اضافه کردن یک گره جدید به لیست پیوندی یک طرفه، باید ابتدا یک گره جدید ایجاد شود و سپس آن را به موقعیت مناسب در لیست متصل گردد. گره جدید را می‌توان به ابتدای لیست، انتهای لیست یا یک موقعیت خاص اضافه نمود.

۲-۳-۱۰- الگوریتم حذف گره از لیست پیوندی یک طرفه

حذف گره از لیست پیوندی یک‌طرفه شامل سه مرحله اصلی است: ابتدا باید گره مورد نظر پیدا شود، سپس اشاره‌گر گره قبلی را به گره بعدی گره مورد نظر تنظیم کرد تا گره مورد نظر از زنجیره لیست حذف شود، و در نهایت، گره حذف شده از حافظه آزاد شود.

۲-۳-۱۱- الگوریتم جستجوگره در لیست پیوندی یک طرفه

الگوریتم جستجو در لیست پیوندی یک طرفه برای یافتن یک عنصر خاص در لیست استفاده می‌شود. در این الگوریتم، به طور خطی و با پیمایش از گره ابتدا شروع می‌شود و به ترتیب از گره‌ای به گره بعدی حرکت می‌کنیم تا به گره مورد نظر برسیم یا به انتهای لیست برسیم.

الگوریتم‌های مرتبط با لیست پیوندی دوطرفه که در پژوهش به کار گرفته شده است، به شرح زیر می‌باشد:

۲-۳-۱۲- الگوریتم ساخت لیست پیوندی دو طرفه[1]

لیست پیوندی دوطرفه ساختاری از داده‌ها است که در آن هر گره شامل دو اشاره‌گر است: یکی به گره قبلی و دیگری به گره بعدی. این ساختار این امکان را می‌دهد که هم به سمت جلو و هم به سمت عقب در لیست حرکت کرد. ساخت لیست پیوندی دوطرفه شامل ایجاد گره‌ها و تنظیم صحیح اشاره‌گرها برای هر گره است.

۲-۳-۱۳- الگوریتم اضافه شدن گره به لیست پیوندی دو طرفه

برای اضافه کردن یک گره به لیست پیوندی دو طرفه، ابتدا باید گره جدید ایجاد شود و حافظه مورد نیاز به آن اختصاص یابد. سپس، اگر لیست خالی بود، گره جدید به عنوان گره‌ی ابتدایی در لیست قرار می‌گیرد. اگر لیست خالی نبود، گره جدید می‌تواند به ابتدای لیست یا انتهای آن اضافه شود.

---

[1] Doubly Linked List

۲-۳-۱۴- الگوریتم حذف گره از لیست پیوندی دو طرفه

حذف گره از لیست پیوندی دوطرفه شامل سه مرحله اصلی است: ابتدا باید گره مورد نظر پیدا شود، سپس اشاره‌گرگره قبلی را به گره بعدی گره مورد نظر، تنظیم کرد همچنین اشاره‌گر گره بعدی را به گره ماقبل گره مورد نظر باید تنظیم نمود تا گره مورد نظر از زنجیره لیست حذف شود، و در نهایت، گره حذف شده را از حافظه آزاد شود.

۲-۳-۱۵- الگوریتم جستجو گره در لیست پیوندی دو طرفه

این الگوریتم همانند الگوریتم جستجو در لیست پیوندی یک طرفه برای یافتن یک عنصر خاص در لیست استفاده می‌شود. در این الگوریتم، به طور خطی و با پیمایش از گره ابتدا شروع می‌شود و به ترتیب از گره‌ای به گره بعدی حرکت می‌کنیم تا به گره مورد نظر برسیم یا به انتهای لیست برسیم.

الگوریتم‌های مرتبط با استک[1] و صف[2]: ساختمان داده این دسته از الگوریتم‌ها استک و صف و در صورت لزوم و بسته به نوع الگوریتم آرایه می‌باشد. الگوریتم‌های اضافه، حذف، پیاده‌سازی استک با صف و یا پیاده‌سازی صف با استک و پیاده‌سازی هر دو با آرایه می‌باشد.

الگوریتم‌های مرتبط با استک که در پژوهش به کار گرفته شده است، به شرح زیر می‌باشد:

۲-۳-۱۶- الگوریتم‌های مرتبط با عملیات اصلی روی استک

الگوریتم‌های مرتبط با عملیات اصلی مانند push و pop در استک پیاده‌سازی شده است. عملیات push یک عنصر جدید را به بالای استک اضافه می‌کند. عملیات pop عنصر موجود در بالای استک را حذف و بازمی‌گرداند.

۲-۳-۱۷- الگوریتم پیاده سازی استک با صف

پیاده‌سازی استک با استفاده از صف‌هارا می‌توان به دو روش اصلی انجام داد: استفاده از دو صف یا استفاده از یک صف. در هر دو روش، هدف این است که رفتار استک LIFO[3]را با استفاده از صف‌ها شبیه‌سازی کنیم.

۲-۳-۱۸- الگوریتم پیاده سازی صف با دو استک

برای پیاده‌سازی صف با دو استک، از دو استک stack1 و stack2 استفاده می‌شود. برای عملیات افزودن به صف[4]، عنصر جدید به stack1 اضافه می‌شود. برای عملیات حذف از صف[5]، اگر stack2 خالی است، تمام عناصر stack1 به stack2منتقل می‌شوند تا ترتیب عناصر معکوس شود و قدیمی‌ترین عنصر به بالای stack2

---

[1] Stack
[2] Queue
[3] Last In, First Out
[4] Enqueue
[5] Dequeue

منتقل شود، سپس عنصر بالای stack2 حذف و بازگردانده می‌شود. این روش با استفاده از دو استک، رفتار صف FIFO [1]را شبیه‌سازی می‌کند.

۲-۳-۱۹- الگوریتم‌های معروف سیستم عامل مرتبط با استک

- الگوریتم زمان‌بندی بر اساس اولویت[2]: یکی از الگوریتم‌های زمان‌بندی CPU[3] است که به هر فرآیند یک اولویت اختصاص می‌دهد و فرآیندها بر اساس اولویت‌هایشان زمان‌بندی می‌شوند. فرآیندی که دارای بالاترین اولویت است، ابتدا اجرا می‌شود. اگر دو فرآیند دارای اولویت یکسان باشند، از روش‌های مختلفی مانند FCFS [4] برای زمان‌بندی آن‌ها استفاده می‌شود.

- الگوریتم Shortest Job Next: این الگوریتم فرآیندها را بر اساس زمان مورد نیازشان برای تکمیل زمان‌بندی می‌کند، به طوری که فرآیند با کمترین زمان اجرا، ابتدا اجرا می‌شود. این الگوریتم به صورت غیر پیش‌دستی[5] عمل می‌کند، به این معنی که فرآیند جاری تا اتمام کامل آن اجرا می‌شود و نمی‌تواند متوقف شود تا فرآیند دیگری با زمان کمتر وارد شود. این الگوریتم به کاهش زمان انتظار متوسط کمک می‌کند.

- الگوریتم Shortest Remaining Time: یک نسخه پیش‌دستی[6] از الگوریتم Shortest Job Next است که در آن سیستم در هر لحظه فرآیندی را که کمترین زمان باقی‌مانده تا تکمیل را دارد، انتخاب و اجرا می‌کند. اگر یک فرآیند جدید با زمان باقی‌مانده کمتر وارد سیستم شود، فرآیند جاری متوقف شده و فرآیند جدید اجرا می‌شود. این الگوریتم بهینه‌سازی زمان پاسخ‌دهی را هدف قرار می‌دهد اما ممکن است به مشکل گرسنگی [7]برای فرآیندهای طولانی منجر شود.

الگوریتم‌های مرتبط با صف که در پژوهش به کار گرفته شده است، به شرح زیر می‌باشد:

۲-۳-۲۰- الگوریتم‌های معروف سیستم عامل مرتبط با صف

- الگوریتم FCFS: این الگوریتم به معنای ترتیبی ساده و منظم در مدیریت پردازش‌ها است. در این روش، پردازش‌ها به همان ترتیبی که وارد صف می‌شوند، به اجرا درمی‌آیند؛ به عبارتی، هر پردازش به نوبت پس از پردازش قبلی خود اجرا می‌شود.

---

[1] First In, First Out
[2] Priority Scheduling
[3] Central processing unit
[4] First-Come, First-Served
[5] non-preemptive
[6] preemptive
[7] Starvation

- الگوریتم MLFQ[1]: این الگوریتم یک روش پیشرفته برای زمان‌بندی پردازش‌ها است که با استفاده از چندین صف با اولویت‌های مختلف، به بهبود کارایی و کاهش زمان انتظار کمک می‌کند. این الگوریتم پردازش‌ها را به صف‌های مختلف با توجه به زمان مورد نیاز آن‌ها اختصاص می‌دهد و با جابجایی بین این صف‌ها بر اساس میزان استفاده ازCPU ، سعی در توازن بار و کاهش زمان انتظار پردازش‌های کوتاه‌تر دارد.

- الگوریتم RR[2]: این الگوریتم یک روش ساده و عادلانه برای زمان‌بندی پردازش‌ها است که به هر پردازش به طور مساوی و در دوره‌های مشخص به نام قطعه زمان [3] اختصاص می‌دهد. به این ترتیب، پردازش‌ها به صورت دوره‌ای و به ترتیب اجرا می‌شوند، و هر پردازش پس از پایان قطعه زمان خود به انتهای صف منتقل می‌شود تا در دور بعدی اجرا شود.

الگوریتم‌های مرتبط با درخت: ساختمان داده برای این دسته الگوریتم، درخت است. الگوریتم‌های مرتبط با درخت که در پژوهش به کار گرفته شده است، به شرح زیر می‌باشد:

۲-۳-۲۱- الگوریتم ساخت درخت دودویی[4]

درخت دودویی ساختاری است که در آن هر گره می‌تواند حداکثر دو فرزند داشته باشد: یکی به عنوان فرزند چپ و دیگری به عنوان فرزند راست. برای ساخت درخت دودویی، ابتدا یک گره ریشه ایجاد می‌شود و سپس به ترتیب، گره‌های جدید به درخت اضافه می‌شوند.

۲-۳-۲۲- الگوریتم اضافه کردن گره جدید به درخت دودویی

الگوریتم اضافه کردن گره جدید به درخت دودویی، به صورت سطح[5] گره جدید را در اولین موقعیت خالی در سطح پایین‌ترین سطح درخت قرار می‌دهد. ابتدا گره ریشه در صف قرار می‌گیرد، سپس گره‌ها یکی یکی از صف خارج شده و بررسی می‌شود که آیا فرزند چپ یا راست آن گره خالی است. اگر فرزند چپ خالی باشد، گره جدید به آن اختصاص داده می‌شود و الگوریتم خاتمه می‌یابد، و اگر هر دو فرزند پر باشند، گره‌ها به صف اضافه می‌شوند تا سطح بعدی بررسی شود.

۲-۳-۲۳- الگوریتم حذف گره از درخت دودویی

الگوریتم حذف گره از درخت دودویی شامل جستجوی گره مورد نظر و سه حالت اصلی است: اگر گره بدون فرزند باشد، به سادگی حذف می‌شود؛ اگر گره تنها یک فرزند داشته باشد، فرزند آن جایگزین گره حذف شده

---

[1] Multi-Level Feedback Queue
[2] Round Robin
[3] time quantum
[4] Binary Tree
[5] Level-Order

می‌شود؛ و اگر گره دو فرزند داشته باشد، باید گره جانشین پیدا کرد، معمولاً کوچک‌ترین گره بزرگ‌تر[1] از گره یا بزرگ‌ترین گره کوچک‌تر[2] از آن، و سپس مقدار این گره جانشین را به جای مقدار گره حذف شده قرار داده و گره جانشین را با یکی از روش‌های بالا حذف کرد.

۲-۳-۲۴- الگوریتم جستجو گره در درخت دودویی

برای جستجو گره در درخت دودویی باید ابتدا درخت را پیمایش نمود تا گره مورد نظر را پیدا کرد. درصورت وجود گره، گره مورد جستجو با موفقیت یافت می‌شود در غیر اینصورت گره مورد نظر در درخت وجود ندارد.

۲-۳-۲۵- الگوریتم پیمایش درخت دودویی

درخت دودویی را به دو روش اصلی می‌توان پیمایش کرد: پیمایش سطحی[3] و پیمایش عمقی[4].

۱. پیمایش سطحی: در این روش، گره‌های درخت به ترتیب سطح از بالا به پایین پیمایش می‌شوند. ابتدا ریشه، سپس گره‌های سطح دوم، و به همین ترتیب تا انتهای درخت پیمایش می‌شوند. این نوع پیمایش معمولاً با استفاده از صف انجام می‌شود.

۲. پیمایش عمقی: در این روش، درخت به عمق پیمایش می‌شود و خود به سه نوع تقسیم می‌شود:

- پیمایش Preorder Traversal: ابتدا گره جاری بازدید می‌شود، سپس زیر درخت چپ و در نهایت زیر درخت راست.
- پیمایش Inorder Traversal: ابتدا زیر درخت چپ بازدید می‌شود، سپس گره جاری و در نهایت زیر درخت راست.
- پیمایش Postorder Traversal : ابتدا زیر درخت چپ، سپس زیر درخت راست بازدید می‌شود و در نهایت گره جاری.

۲-۳-۲۶- الگوریتم ساخت درخت دودویی جستجو[5]

الگوریتم ساخت درخت دودویی جستجو با افزودن گره‌ها به ترتیب خاصی انجام می‌شود که مقادیر کوچکتر در سمت چپ و مقادیر بزرگتر در سمت راست قرار می‌گیرند. ابتدا با یک درخت خالی شروع می‌شود و اولین گره به عنوان ریشه قرار می‌گیرد. برای افزودن هر گره جدید، از ریشه شروع کرده و مقدار گره جدید با گره جاری مقایسه می‌شود؛ اگر مقدار گره جدید کمتر باشد، به زیر درخت چپ رفته و اگر بیشتر باشد، به زیر درخت راست می‌رود. این روند تا زمانی که به یک مکان خالی برسد ادامه می‌یابد و گره جدید در آنجا قرار داده می‌شود.

---

[1] Successor
[2] Predecessor
[3] Breadth-First Traversal
[4] Depth-First Traversal
[5] Binary Search Tree

۲-۳-۲۷- الگوریتم اضافه کردن گره جدید به درخت دودویی جستجو

الگوریتم اضافه کردن گره جدید به درخت دودویی جستجو به این صورت است که ابتدا از ریشه درخت شروع کرده و مقدار گره جدید را با گره جاری مقایسه می‌کنیم. اگر مقدار گره جدید کوچکتر از گره جاری باشد، به زیر درخت چپ می‌رویم؛ و اگر بزرگتر باشد، به زیر درخت راست می‌رویم. این روند را ادامه می‌دهیم تا به یک گره بدون فرزند در سمت مناسب برسیم. سپس، گره جدید را به عنوان فرزند چپ یا راست این گره قرار می‌دهیم.

۲-۳-۲۸- الگوریتم حذف گره از درخت دودویی جستجو

الگوریتم حذف گره از درخت دودویی جستجو شامل جستجوی گره مورد نظر و سه حالت اصلی است: اگر گره بدون فرزند باشد، به سادگی حذف می‌شود؛ اگر گره تنها یک فرزند داشته باشد، فرزند آن جایگزین گره حذف شده می‌شود؛ و اگر گره دو فرزند داشته باشد، باید گره جانشین پیدا کرد، معمولاً کوچک‌ترین گره بزرگ‌تر از گره یا بزرگ‌ترین گره کوچک‌تر از آن، و سپس مقدار این گره جانشین را به جای مقدار گره حذف شده قرار داده و گره جانشین را با یکی از روش‌های بالا حذف کرد.

۲-۳-۲۹- الگوریتم جستجو گره در درخت دودویی جستجو

الگوریتم جستجوی گره در درخت دودویی جستجو به این صورت عمل می‌کند که ابتدا جستجو از ریشه درخت شروع می‌شود و مقدار گره مورد نظر با مقدار گره جاری مقایسه می‌شود. اگر مقدار مورد نظر برابر با مقدار گره جاری باشد، جستجو موفقیت‌آمیز بوده و گره پیدا شده است. اگر مقدار مورد نظر کمتر از مقدار گره جاری باشد، جستجو به زیر درخت چپ منتقل می‌شود؛ و اگر بزرگتر باشد، جستجو به زیر درخت راست منتقل می‌شود. این فرآیند به صورت بازگشتی یا تکراری تا زمانی که گره مورد نظر پیدا شود یا به یک گره خالی برسیم (که نشان‌دهنده عدم وجود گره در درخت است) ادامه می‌یابد.

۲-۳-۳۰- الگوریتم پیمایش درخت دودویی جستجو

درخت دودویی جستجو را به دو روش اصلی می‌توان پیمایش کرد: پیمایش سطحی و پیمایش عمقی.

۱. پیمایش سطحی: در این روش، گره‌های درخت به ترتیب سطح از بالا به پایین پیمایش می‌شوند. ابتدا ریشه، سپس گره‌های سطح دوم، و به همین ترتیب تا انتهای درخت پیمایش می‌شوند. این نوع پیمایش معمولاً با استفاده از صف انجام می‌شود.

۲. پیمایش عمقی: در این روش، درخت به عمق پیمایش می‌شود و خود به سه نوع تقسیم می‌شود:

- پیمایش Preorder Traversal: ابتدا گره جاری بازدید می‌شود، سپس زیر درخت چپ و در نهایت زیر درخت راست.

- پیمایش Inorder Traversal: ابتدا زیر درخت چپ بازدید می‌شود، سپس گره جاری و در نهایت زیر درخت راست.
- پیمایش Postorder Traversal : ابتدا زیر درخت چپ، سپس زیر درخت راست بازدید می‌شود و در نهایت گره جاری.

۲-۳-۳۱- الگوریتم ساخت درخت AVL[1]

الگوریتم ساخت درخت AVL شامل درج گره‌ها در یک درخت دودویی جستجو به همراه اطمینان از متوازن بودن درخت است. درخت AVL یک درخت دودویی جستجو متوازن است که در آن تفاوت ارتفاع بین زیر درخت چپ و راست هر گره نباید بیشتر از ۱ باشد.

۲-۳-۳۲- الگوریتم اضافه کردن گره جدید به درخت AVL

الگوریتم اضافه کردن گره جدید به درخت AVL شامل مراحل زیر است:

۱. درج گره: گره جدید را به درخت به همان روش که در درخت دودویی جستجو انجام می‌شود، می‌توان اضافه نمود. یعنی با مقایسه مقدار گره جدید با گره‌های موجود و قرار دادن آن در مکان مناسب.
۲. به‌روزرسانی ارتفاع‌ها: پس از درج گره، ارتفاع هر گره در مسیر از گره جدید به ریشه باید به‌روزرسانی شود. ارتفاع هر گره برابر با ارتفاع بلندترین زیر درخت آن به‌علاوه ۱ است.
۳. محاسبه فاکتور توازن[2]: فاکتور توازن برای هر گره در مسیر به‌روزرسانی شده محاسبه می‌شود. فاکتور توازن برابر با اختلاف ارتفاع زیر درخت چپ و راست گره است.
۴. بررسی عدم توازن: اگر فاکتور توازن گره‌ها خارج از بازه مجاز (۱، ۰، ۱-) باشد، گره‌های نامتوازن شناسایی شده و باید چرخش‌های مناسب برای برقراری توازن انجام شود.
۵. انجام چرخش‌ها: بر اساس نوع عدم توازن، یکی از چهار نوع چرخش زیر انجام می‌شود:
   - **چرخش راست**[3]: برای تصحیح عدم توازن در زیر درخت چپ گره.
   - **چرخش چپ**[4]: برای تصحیح عدم توازن در زیر درخت راست گره.
   - **چرخش چپ ـ راست**[5]: برای تصحیح عدم توازن در زیر درخت چپ گره که گره جدید در زیر درخت راست آن قرار دارد.

---

[1] Adelson-Velsky and Landis
[2] Balance Factor
[3] Right Rotation
[4] Left Rotation
[5] Left-Right Rotation

- **چرخش راست_چپ**[1]: برای تصحیح عدم توازن در زیر درخت راست گره که گره جدید در زیر درخت چپ آن قرار دارد.

۲-۳-۳۳- الگوریتم حذف گره جدید از درخت AVL

الگوریتم حذف گره از درخت AVL شامل مراحل زیر است:

۱. یافتن و حذف گره: ابتدا باید گره مورد نظر برای حذف را مانند درخت دودویی جستجو پیداکرد و آن را حذف نمود. این شامل سه حالت است:
   - **حذف گره بدون فرزند:** گره را به سادگی می‌توان حذف کرد.
   - **حذف گره با یک فرزند:** گره حذف شده را می‌توان با فرزندش جایگزین کرد.
   - **حذف گره با دو فرزند:** کوچک‌ترین گره بزرگ‌تر یا بزرگ‌ترین گره کوچک‌تر را پیدا کرده و مقدار آن را جایگزین مقدار گره حذف شده باید نمود، سپس کوچک‌ترین گره بزرگ‌تر یا بزرگ‌ترین گره کوچک‌تر را باید از درخت حذف کرد.
۲. به‌روزرسانی ارتفاع‌ها: پس از حذف گره، ارتفاع هر گره در مسیر از گره حذف شده به ریشه باید به‌روزرسانی شود. ارتفاع هر گره برابر با ارتفاع بلندترین زیر درخت آن به‌علاوه ۱ است.
۳. محاسبه فاکتور توازن: فاکتور توازن برای هر گره در مسیر به‌روزرسانی شده محاسبه می‌شود. فاکتور توازن برابر با اختلاف ارتفاع زیر درخت چپ و راست گره است.
۴. بررسی عدم توازن: اگر فاکتور توازن گره‌ها خارج از بازه مجاز (۱، ۰، ۱-) باشد، گره‌های نامتوازن شناسایی شده و باید چرخش‌های مناسب برای برقراری توازن انجام شود.
۵. انجام چرخش‌ها: بر اساس نوع عدم توازن، یکی از چهار نوع چرخش زیر انجام می‌شود:
   - چرخش راست: برای تصحیح عدم توازن در زیر درخت چپ گره.
   - چرخش چپ: برای تصحیح عدم توازن در زیر درخت راست گره.
   - چرخش چپ ـ راست: برای تصحیح عدم توازن در زیر درخت چپ گره که گره جدید در زیر درخت راست آن قرار دارد.
   - چرخش راست_چپ : برای تصحیح عدم توازن در زیر درخت راست گره که گره جدید در زیر درخت چپ آن قرار دارد.

۲-۳-۳۴- الگوریتم ساخت درخت قرمز وسیاه[2]

الگوریتم ساخت درخت قرمز و سیاه شامل مراحل زیر است:

۱. درج گره: گره جدید مانند درخت دودویی جستجو به درخت اضافه می‌شود. گره جدید به‌طور موقت به رنگ قرمز رنگ‌آمیزی می‌شود. این مرحله شامل یافتن مکان مناسب و درج گره جدید در درخت است.

---

[1] Right-Left Rotation
[2] Red-Black Tree

۲. بررسی قوانین رنگ: پس از درج گره، باید مطمئن شد که درخت قوانین رنگ درخت قرمز و سیاه را رعایت می‌کند. قوانین اصلی به شرح زیر است:
   - هر گره باید یا قرمز یا سیاه باشد.
   - ریشه درخت باید سیاه باشد.
   - هر برگ (گره‌های تهی) باید سیاه باشد.
   - اگر گره‌ای قرمز است، هر دو فرزند آن باید سیاه باشند (هیچ دو گره قرمز پشت سر هم نباید باشند).
   - هر مسیر از گره به هر برگ باید تعداد برابر از گره‌های سیاه را داشته باشد.
۳. اصلاح عدم توازن: بعد از درج گره و بررسی قوانین رنگ، ممکن است نیاز به اصلاحات برای برقراری قوانین درخت قرمز و سیاه باشد. این اصلاحات شامل تغییر رنگ‌ها و چرخش‌ها است:
   - چرخش چپ: برای تصحیح عدم توازن زمانی که گره قرمز در زیر درخت راست قرار دارد.
   - چرخش راست : برای تصحیح عدم توازن زمانی که گره قرمز در زیر درخت چپ قرار دارد.
   - تعویض رنگ: برای تصحیح عدم توازن زمانی که هر دو فرزند گره قرمز هستند.

۲-۳-۳۵- الگوریتم اضافه کردن گره جدید به درخت قرمز وسیاه

الگوریتم اضافه کردن گره جدید به درخت قرمز و سیاه به شرح زیر است:

۱. درج گره: گره جدید به درخت به‌صورت مشابه درخت دودویی جستجو اضافه می‌شود و به‌طور موقت به رنگ قرمز رنگ‌آمیزی می‌شود. مکان مناسب برای گره جدید پیدا و گره در درخت درج می‌شود.
۲. بررسی و اصلاح قوانین رنگ: پس از درج گره، قوانین درخت قرمز و سیاه بررسی و در صورت لزوم اصلاحات انجام می‌شود. اگر گره جدید به عنوان فرزند یک گره قرمز قرار گیرد (یعنی گره جدید و والدش هر دو قرمز باشند)، گره‌های والد، عمو و دایی گره جدید بررسی و اصلاح می‌شود.
۳. اصلاح عدم توازن: برای حفظ قوانین درخت قرمز و سیاه، اصلاحات زیر انجام می‌شود:
   - چرخش چپ: زمانی که گره جدید به عنوان فرزند راست گره قرمز والدش قرار دارد و نیاز به تنظیم درخت است.
   - چرخش راست: زمانی که گره جدید به عنوان فرزند چپ گره قرمز والدش قرار دارد و نیاز به تنظیم درخت است.
   - تعویض رنگ: زمانی که گره جدید و گره‌های هم‌سطح والد و عمو هر دو قرمز هستند. در این حالت، رنگ گره والد و عمو به سیاه و رنگ گره پدربزرگ به قرمز تغییر می‌یابد.
۴. تنظیم رنگ ریشه: در پایان، رنگ ریشه همیشه به رنگ سیاه تنظیم می‌شود، زیرا این یکی از قوانین اصلی درخت قرمز و سیاه است.

۲-۳-۳۶- الگوریتم حذف گره جدید از درخت قرمز وسیاه

الگوریتم حذف گره از درخت قرمز و سیاه به شرح زیر است:

۱. حذف گره: گره مورد نظر از درخت مشابه درخت دودویی جستجو حذف می‌شود. برای حذف گره، ممکن است گره مورد نظر یک یا دو فرزند داشته باشد. اگر گره دو فرزند داشته باشد، باید جایگزین مناسبی پیدا شده و جایگزین شود.
۲. تبدیل گره به گره قابل حذف: اگر گره موردنظر دو فرزند داشته باشد، گره جانشین (جانشین جایگزین) پیدا و گره مورد نظر با گره جانشین جایگزین می‌شود. سپس گره جانشین حذف می‌شود که همیشه گره با صفر یا یک فرزند است.
۳. اصلاحات بعد از حذف: پس از حذف گره، ممکن است درخت قوانین درخت قرمز و سیاه را رعایت نکند. اصلاحات زیر برای بازگرداندن قوانین انجام می‌شود:
- تبدیل گره قرمز به سیاه: اگر گره حذف شده سیاه باشد و گره فرزندش قرمز نباشد، باید گره فرزند به سیاه تبدیل شود و گره‌های والد و دایی بررسی شوند.
- چرخش‌ها و تغییر رنگ‌ها: ممکن است برای حفظ قوانین درخت نیاز به چرخش چپ یا راست و تغییر رنگ گره‌ها باشد. این شامل چرخش‌ها برای تصحیح عدم توازن و تغییر رنگ گره‌ها برای اطمینان از رعایت قوانین درخت است.

٤. تنظیم رنگ ریشه: در پایان، اطمینان حاصل می‌شود که ریشه درخت همیشه به رنگ سیاه باقی بماند، زیرا این یکی از قوانین اصلی درخت قرمز و سیاه است.

۲-۴- فرآیند مصورسازی الگوریتم

مصورسازی الگوریتم به معنای تبدیل الگوریتم‌ها به نمایش‌های بصری است تا فهم و تحلیل آن‌ها را ساده‌تر شود. درواقع با هدف ارائه روشی جذاب و بصری برای آموزش و یادگیری الگوریتم‌های کامپیوتری است. این روش‌ها کمک می‌کنند تا فرآیندها، ساختارها و مراحل الگوریتم را بهتر درک نمود. این موضوع به کاربران کمک می‌کند تا نحوه عملکرد الگوریتم را به وضوح مشاهده و درک کنند.

۲-۵- لاتک

لاتک یک سیستم حروف‌چینی[1] است که برای ایجاد اسناد با کیفیت بالا، به خصوص اسناد علمی و ریاضی، استفاده می‌شود. این سیستم توسط لسلی لمپورت[2] در اوایل دهه ۱۹۸۰ بر اساس سیستم حروف‌چینی دیگری به نام TeX که توسط دونالد کنوت[3] توسعه یافته بود، ایجاد شد. لاتک به جای اینکه مانند نرم‌افزارهای واژه‌پرداز معمولی، به طور مستقیم بر روی ظاهر و قالب‌بندی متن توجه کند، به نویسنده اجازه می‌دهد تا بر روی محتوای

[1] typesetting
[2] Leslie Lamport
[3] Donald Knuth

علمی و ساختار کلی سند متمرکز شود ولاتک به طور خودکار قالب‌بندی مناسبی را اعمال می‌کند. این سیستم برای نوشتن مقالات علمی، پایان‌نامه‌ها، کتاب‌ها و ارائه‌های مبتنی بر متن بسیار محبوب است[۳].

۲-۶- تیکزد[1]

تیکزد یک بسته گرافیکی برای زبان برنامه‌نویسی LaTeX است که به کاربران امکان می‌دهد نمودارها، شکل‌ها و دیاگرام‌های پیچیده را به راحتی رسم کنند[۴]. این بسته معمولاً برای اهداف آکادمیک و علمی به کار می‌رود و امکان تولید گرافیک‌های با کیفیت بالا را فراهم می‌کند. در این پژوهش تمامی الگوریتم‌ها با استفاده از کتابخانه تیکزد مصورسازی شده است.

۲-۷- کتابخانه PyLateX

کتابخانه PyLaTeX برای ایجاد و مدیریت اسناد لاتک در زبان برنامه‌نویسی پایتون طراحی شده است. این کتابخانه امکان تولید اسناد لاتک را به صورت برنامه‌نویسی فراهم می‌آورد و نیاز به نوشتن کد لاتک را کاهش می‌دهد.

۲-۸- کتابخانه pdflatex

کتابخانه pdflatex یکی از ابزارهای رایج برای تولید فایل‌های PDF از کد لاتک است. این ابزار به صورت خط‌دستوری عمل می‌کند و به کاربران این امکان را می‌دهد تا فایل‌های لاتک خود را به مستندات PDF با کیفیت بالا تبدیل کنند. برخلاف سایر برنامه‌های تولید لاتک که ممکن است به چندین مرحله برای تولید فایل PDF نیاز داشته باشند، pdflatex قادر است به طور مستقیم فایل‌های PDF را تولید کند.

۲-۹- بسته xcolor

بسته xcolor در لاتک این امکان را می‌دهد که رنگ‌های مختلف را به متون، پس‌زمینه‌ها و حاشیه‌ها اضافه کرد. با استفاده از این بسته، می‌توان متن را به رنگ‌های مختلف رنگ‌آمیزی کرد یا پس‌زمینه‌ی متنی را تغییر داد. این ویژگی به ویژه در مستندات علمی و فنی برای تأکید بر نکات کلیدی یا زیباسازی متون مفید است. به سادگی با بارگذاری xcolor و استفاده از دستورات مانند \textcolor و \colorbox می‌توان این تغییرات را اعمال نمود.

۲-۱۰- بسته listings

بسته listings در لاتک برای نمایش و قالب‌بندی کدهای برنامه‌نویسی به کار می‌رود. این بسته این امکان را می‌دهد که کدهای منبع را با رنگ‌ها، فونت‌ها و فرمت‌های مختلف نمایش داد، که می‌تواند برای مستندات فنی، مقالات علمی و مستندات نرم‌افزاری بسیار مفید باشد.

---

[1] Tikz

۲-۱۱- بسته amsmath

بسته amsmath در لاتک یکی از بسته‌های کلیدی برای نوشتن فرمول‌های ریاضی پیشرفته است. این بسته محیط‌ها و دستورات متعددی را برای مدیریت معادلات و فرمول‌های چند خطی فراهم می‌کند، مانند محیط‌های align برای تراز کردن معادلات، gather برای نمایش چندین معادله در خطوط جداگانه و multline برای نوشتن معادلات طولانی که به خطوط متعدد شکسته می‌شوند. استفاده از amsmath به نویسندگان کمک می‌کند تا معادلات پیچیده را به شکلی منظم و قابل فهم ارائه دهند.

۲-۱۲- بسته geometry

بسته geometry در لاتک به کاربران این امکان را می‌دهد که به سادگی ابعاد صفحه، حاشیه‌ها، و سایر تنظیمات مربوط به قالب‌بندی سند را کنترل کنند. با استفاده از این بسته، می‌توان اندازه حاشیه‌ها را دقیقاً مشخص نمود، اندازه صفحه را تغییر داد، و حتی قالب‌های سفارشی برای اندازه‌های غیر استاندارد ایجاد کرد. این بسته برای تنظیمات دقیق و سفارشی‌سازی اسناد، به ویژه در اسناد علمی و فنی، بسیار مفید است.

۲-۱۳- برخی دستورات مهم لاتک

در لاتک برای هر عملیاتی دستور مربوطه تعریف شده است. در زیر برخی دستورات مهم لاتک آورده شده است:

۲-۱۳-۱- دستور title

دستورtitle در لاتک برای تنظیم عنوان سند استفاده می‌شود. این دستور معمولاً در ابتدای سند و قبل از دستور maketitle قرار می‌گیرد. دستور title به خودی خود عنوان را در سند نمایش نمی‌دهد، بلکه آن را ذخیره می‌کند تا زمانی که دستور maketitle فراخوانی شود، عنوان به همراه نام نویسنده (با دستور author و تاریخ با دستور date) در صفحه عنوان نمایش داده شود.

۲-۱۳-۲- دستور author

دستور author در لاتک برای مشخص کردن نام نویسنده یا نویسندگان یک سند استفاده می‌شود. این دستور اطلاعات نویسنده را ذخیره می‌کند تا با استفاده از دستور maketitle در صفحه عنوان نمایش داده شود.

۲-۱۳-۳- دستور date

دستور date در لاتک برای تعیین تاریخ سند استفاده می‌شود. این دستور معمولاً همراه با دستورات title و author در ابتدای سند قرار می‌گیرد. تاریخ تعیین‌شده با استفاده از دستور maketitle در صفحه عنوان نمایش داده می‌شود.

۲-۱۳-۴- دستور maketitle

دستور maketitle در لاتک برای ایجاد و نمایش صفحه عنوان سند استفاده می‌شود. این دستور اطلاعاتی که با دستورات title و author وdate تعیین شده‌اند، در یک قالب پیش‌فرض نمایش می‌دهد. دستور maketitle باید بعد از این دستورات و معمولاً درست پس از شروع سند با {begin{document قرار گیرد.

۲-۱۳-۵- دستور section

دستور section در لاتک برای ایجاد یک بخش جدید در سند استفاده می‌شود. با استفاده از این دستور، عنوانی برای بخش موردنظر ایجاد می‌شود و به‌طور خودکار شماره‌بندی انجام می‌گیرد. از subsection و subsubsection برای ایجاد بخش‌های فرعی استفاده می‌شود.

۲-۱۳-۶- دستور {begin{tikzpicture

دستور {begin{tikzpicture در لاتک برای ایجاد یک محیط جهت ترسیم تصاویر و نمودارها با استفاده از بسته‌ی TikZ به‌کار می‌رود. در این محیط، اشکال، خطوط، و متن‌های مختلف ترسیم می‌شوند. این محیط با استفاده از دستور {end{tikzpicture به پایان می‌رسد. درون این محیط، اشیاء و نمودارها با استفاده از دستورات خاص TikZ رسم می‌شوند.

۲-۱۳-۷- دستور {begin{document

دستور {begin{document در لاتک برای آغاز بخش اصلی سند استفاده می‌شود. تمام محتوای سند از جمله متن، عناوین، جداول، تصاویر و غیره، پس از این دستور نوشته می‌شود. این دستور باید بعد از تنظیمات ابتدایی مانند عنوان، نویسنده، تاریخ، و تنظیمات دیگر، و قبل از پایان سند با دستور {end{document قرار گیرد.

۲-۱۴- بسته PyPI[1]

یک مخزن بزرگ برای بسته‌های نرم‌افزاری پایتون است که به برنامه‌نویسان امکان می‌دهد کتابخانه‌ها و ابزارهای خود را به اشتراک بگذارند و از دیگران استفاده کنند. وقتی کتابخآن‌های را در PyPI منتشر می‌شود، افراد دیگر می‌توانند به‌راحتی با استفاده از ابزارهایی مانند pip[2] آن را نصب کنند.

[1] Python Package Index
[2] pip installs packages

۲-۱۵- جمع‌بندی

در این فصل ابتدا مفاهیم، روش‌ها، اصطلاحات و کتابخانه‌های مورد نیاز در این پژوهش که از آن‌ها استفاده شده است معرفی شدند. در ادامه به بررسی پیشینه پژوهش، شرح پژوهش، نحوه مصورسازی الگوریتم‌ها، ایجاد کتابخانه و در نهایت به نتایج آن‌ها پرداخته می‌شود.

فصل سوم
پژوهش‌های مشابه

۳-۱- مقدمه

در این فصل پیشینه پژوهش مورد بررسی قرار می‌گیرد. ابزارهایی وجود دارند که کارکرد نسبتا مشابهی با پژوهش مورد نظر ما دارند. در این پژوهش، به بررسی و مقایسه این ابزارها با پژوهش انجام شده پرداخته می‌شود و نشان داده خواهد شد که چگونه این پژوهش با تمرکز بر تولید خروجی لاتک ، می‌تواند به‌عنوان یک ابزار قدرتمند و کاربردی در حوزه‌های آموزشی و علمی به‌کار گرفته شود.

۳-۲- نمونه‌های مشابه

۱. سایت مصورسازی الگوریتم‌ها (VisuAlgo): یک پلتفرم تعاملی آنلاین است که به صورت گسترده برای آموزش و درک بهتر الگوریتم‌ها و ساختارهای داده استفاده می‌شود. این سایت شامل مجموعه‌ای از الگوریتم‌هاست که در دسته‌بندی‌های مختلف مانند مرتب‌سازی، جستجو، ساختارهای داده و گراف قرار می‌گیرند. هر الگوریتم با یک توضیح مختصر همراه است که مبانی و کارکرد آن را شرح می‌دهد. علاوه بر این، کدهای مربوط به الگوریتم‌ها به زبان‌های برنامه‌نویسی مختلف مانند پایتون، جاوا اسکریپت، و C++ ارائه شده‌اند. ویژگی کلیدی VisuAlgo، امکان مشاهده و تعامل با مصورسازی‌های تعاملی است که مراحل اجرای الگوریتم‌ها را به تصویر می‌کشد. کاربران می‌توانند به‌طور مستقیم با این مصورسازی‌ها تعامل داشته باشند، ورودی‌ها را تغییر دهند و نتایج متفاوت را مشاهده کنند[۵].

تفاوت با پروژه ما:

- پیاده‌سازی بر بستر اینترنت: VisuAlgo به صورت آنلاین و از طریق وب‌سایت عمل می‌کند و کاربر برای استفاده از آن نیاز به دسترسی به اینترنت دارد. در صورتی که کتابخانه پیاده‌سازی شده در این پژوهش پس از نصب، کاملاً به صورت آفلاین قابل استفاده است.

- نوع خروجی: VisuAlgo خروجی خود را به صورت گرافیکی و تعاملی از طریق وب‌سایت ارائه می‌دهد. این نوع خروجی برای یادگیری بصری بسیار مفید هستند، اما برای مستندسازی و چاپ در مقالات یا کتاب‌ها مناسب نیستند. در مقابل، پروژه انجام شده با تمرکز بر تولید خروجی به صورت کد لاتک امکان استفاده از آن را به‌راحتی در مقالات علمی، کتاب‌ها و اسناد آموزشی فراهم می‌کند؛ که این ویژگی کتابخانه پیاده‌سازی شده را برای کاربردهایی که نیاز به مستندسازی دقیق و چاپی دارند، مفید واقع می‌شود.

۲. پلتفرم آنلاین تعاملی برای مصورسازی الگوریتم‌های مرتب‌سازی (Sorting Algorithms Animations): این پلتفرم یک ابزار آنلاین است که به فقط بر روی مصورسازی الگوریتم‌های مرتب‌سازی تمرکز دارد. این پلتفرم از کتابخانه‌های جاوا اسکریپت استفاده می‌کند تا مراحل مختلف اجرای الگوریتم‌های مرتب‌سازی مانند مرتب‌سازی حبابی، مرتب‌سازی ادغامی، مرتب‌سازی سریع و دیگر الگوریتم‌ها را به صورت گرافیکی به تصویر می‌کشد[۶].

تفاوت با پروژه ما:

- آنلاین بودن و وابستگی به کتابخانه‌های جاوا اسکریپت: این پلتفرم به صورت آنلاین و تحت وب اجرا می‌شود، که به معنای نیاز به دسترسی دائمی به اینترنت است. علاوه بر این، از کتابخانه‌های جاوا اسکریپت برای تولید گرافیک‌ها و انیمیشن‌های تعاملی استفاده می‌کند. این وابستگی به تکنولوژی‌های وب باعث می‌شود که کاربران نتوانند به‌راحتی خروجی‌های تولید شده را در اسناد چاپی یا آفلاین مورد استفاده قرار دهند. اما در پروژه انجام شده، خروجی به زبان لاتک تولید می‌شود که قابلیت استفاده در محیط‌های آفلاین را دارد. این خروجی‌ها به‌صورت مستقیم در مقالات علمی و آموزشی قابل استفاده هستند و نیازی به تبدیل یا تغییرات اضافی ندارند.
- محدودیت در نوع الگوریتم‌ها و داده‌ها: این پلتفرم تنها به مصورسازی الگوریتم‌های مرتب‌سازی می‌پردازد و الگوریتم‌های دیگر مانند جستجو، درخت و ساختارهای داده را پوشش نمی‌دهد. از طرف دیگر، داده‌های مورد استفاده در این پلتفرم به صورت پیش‌فرض ارائه می‌شوند و کاربران نمی‌توانند داده‌های دلخواه خود را وارد کنند. در صورتی که پروژه پیاده‌سازی شده در این پژوهش از این نظر انعطاف بیشتری دارد؛ زیرا نه‌تنها می‌تواند انواع مختلف الگوریتم‌ها را پوشش دهد، بلکه کاربران می‌توانند داده‌های دلخواه خود را به الگوریتم‌ها بدهند و خروجی لاتک مربوط به آن‌ها را دریافت کنند.

۳. ابزار مصورسازی الگوریتم‌ها و ساختارهای داده توسط پروفسور دیوید گالس (Data Structure Visualizations): ابزار Data Structure Visualizations توسط پروفسور دیوید گالس از دانشگاه سن فرانسیسکو توسعه داده شده است. این ابزار به‌صورت رایگان و متن‌باز در دسترس است و طیف وسیعی از

الگوریتم‌ها و ساختارهای داده را به صورت تعاملی و گرافیکی به نمایش می‌گذارد. این ابزار شامل الگوریتم‌های مختلفی مانند درخت‌ها، گراف‌ها، لیست‌های پیوندی، و صف‌ها است که به صورت بصری و قابل فهم نمایش داده می‌شوند. کاربران می‌توانند مراحل مختلف اجرای الگوریتم‌ها را مشاهده کنند و با تغییر ورودی‌ها، نتایج مختلف را بررسی نمایند. این ابزار به صورت آنلاین و تحت وب ارائه می‌شود[۷].

تفاوت با پروژه ما:

- آنلاین بودن، محدودیت دسترسی و نوع خروجی: ابزار Data Structure Visualizations به صورت آنل‌ایین عمل می‌کند و کاربران برای دسترسی به آن نیازمند اتصال به اینترنت هستند. این امر ممکن است در شرایطی که دسترسی به اینترنت محدود یا غیرممکن است، مشکل‌ساز شود. در مقابل، در پروژه انجام شده پس از نصب کتابخانه، به صورت آفلاین عمل کرده و خروجی لاتک تولید می‌کند که می‌تواند در هر شرایطی مورد استفاده قرار گیرد.

از مزایا پروژه ما و تفاوت آن با دیگر پروژه ها:

قابلیت شخصی‌سازی تصاویر: در کتابخانه پیاده‌سازی شده، امکان تعیین پارامترهایی از جمله رنگ و ابعاد تصاویر تولیدی برای نمایش مراحل اجرای الگوریتم‌ها در خروجی فراهم شده است. که این امر می‌تواند باعث رضایت هرچه بیشتر کاربر از خروجی تولید شده، شود.

پشتیبانی از مستندسازی و ارجاع‌دهی: قابلیت افزودن توضیحات و ارجاع‌ها در خروجی‌های تولید شده در پژوهش انجام شده، از مزیت‌های دیگر آن است. این امر برای تهیه مستندات آموزشی یا علمی بسیار مهم است، زیرا نویسندگان می‌توانند به‌طور دقیق جزئیات اجرای الگوریتم‌ها را توضیح دهند و منابع مرتبط را در اسناد خود ارجاع دهند. این قابلیت به کاربران اجازه می‌دهد تا مستندات علمی کاملی را ایجاد کنند که شامل توضیحات دقیق، تصاویر مرحله به مرحله و ارجاع‌های مرتبط با پژوهش‌های دیگر باشد.

گسترش‌پذیری و متن‌باز بودن: پتانسیل توسعه این پژوهش می‌تواند از دیگر مزیت‌های آن است. به عنوان مثال، کاربران می‌توانند الگوریتم‌های جدیدی را به کتابخانه پیاده‌سازی شده اضافه کنند یا قالب‌های مختلف لاتک را برای تولید خروجی‌های سفارشی طراحی کنند.

۳-۳- جمع‌بندی

در این فصل ابتدا پژوهش‌های مشابه کار ما معرفی شده‌اند و در گام بعد در هر مرحله تفاوت پژوهش ما با سایر پژوهش‌ها ذکر شده است همچنین مزایا ویک مقایسه کوتاه نیز برای مثال آورده شده است. به طور کلی، پروژه ما می‌تواند به‌عنوان یک ابزار قدرتمند در آموزش و مستندسازی الگوریتم‌ها و ساختارهای داده به‌کار گرفته شود و نیازهای کاربران را در زمینه‌های مختلف علمی و آموزشی برآورده سازد. در فصل بعد به طور کامل به پیاده‌سازی و شرح پروژه پرداخته خواهد شد.

فصل چهارم
شرح پژوهش

۴-۱- مقدمه

در این فصل در مورد چگونگی پیاده‌سازی الگوریتم‌ها برای تولید خروجی به زبان لاتک که حاوی تصاویر گام به گام مراحل اجرای الگوریتم می‌باشد توضیح داده خواهد شد و در توضیح پیاده‌سازی هر الگوریتم به تفصیل در مورد تنظیمات ورودی‌ها و توابع اصلی به کار برده شده شرحی داده خواهد شد. در پایان نیز نمونه های از تصاویر خروجی چند الگوریتم را مشاهده خواهیم‌کرد.

۴-۲- الگوریتم جست‌وجوی دودویی

در این بخش از پژوهش، فرآیند الگوریتم جست‌وجوی دودویی به صورت گرافیکی و بصری مصورسازی شده است. برای این منظور، کدی به زبان پایتون نوشته شده است که با استفاده از کتابخانه‌های pylatex و TikZ، یک سند لاتک تولید می‌کند. این سند به تصویر کشیدن مراحل مختلف جست‌وجوی دودویی و نمایش وضعیت آرایه در هر مرحله می‌پردازد. هدف اصلی این کد، ارائه‌ی تصویری روشن و قابل درک از فرآیند جست‌وجوی دودویی است که به کاربران این امکان را می‌دهد تا مراحل مختلف الگوریتم را به صورت تعاملی مشاهده کنند. همچنین، کاربران می‌توانند آرایه ورودی، هدف جست‌وجو و ویژگی‌های گرافیکی مانند اندازه و و رنگ‌ها را به دلخواه تنظیم کنند.

کد نوشته شده شامل دو بخش اصلی است. اولین بخش، تابع create_visualization، مسئول ایجاد سند لاتک و تصویرسازی فرآیند جست‌وجوی دودویی است. این تابع ورودی‌هایی مانند آرایه ورودی و هدف جست‌وجو را دریافت کرده و همچنین اندازه سلول‌های نمایشی و رنگ‌های مختلف برای نمایش وضعیت‌های مختلف در آرایه را به عنوان پارامترهای دیگر تنظیم می‌کند. رنگ‌ها شامل رنگ‌های شاخص‌های چپ، راست، میانی، پس‌زمینه آرایه و رنگ عنصر هدف هستند. در نهایت، این تابع مراحل زیر را به ترتیب اجرا می‌کند: ایجاد سند لاتک با تنظیم حاشیه‌ها و تعریف رنگ‌ها، نمایش آرایه اولیه و مرتب شده با استفاده از تابع draw_array،

اجرای الگوریتم جست‌وجوی دودویی و نمایش گام به گام مراحل آن، و نمایش مرحله نهایی که در آن عنصر هدف در آرایه پیدا می‌شود.

بخش دوم شامل تابع draw_array است که مسئول رسم و نمایش آرایه با استفاده از رنگ‌ها و اندازه‌های مشخص‌شده است. این تابع با استفاده از شیء TikZ، آرایه را به نمایش در می‌آورد و هر عنصر را در یک سلول مجزا قرار می‌دهد. با توجه به موقعیت‌های مختلف (چپ، راست، میانی و هدف)، سلول‌ها با رنگ‌های مناسب پر می‌شوند.

کد نوشته شده به عنوان یک ابزار آموزشی بسیار مفید است و می‌تواند در تولید گزارش‌های تصویری از مراحل جست‌وجوی دودویی مورد استفاده قرار گیرد. این ابزار به کاربران، از جمله دانشجویان و پژوهشگران، کمک می‌کند تا درک بهتری از الگوریتم جست‌وجوی دودویی و نحوه عملکرد آن پیدا کنند. علاوه بر این، این کد می‌تواند برای تهیه مستندات آموزشی یا مقالات آکادمیک نیز به کار رود. به طور کلی، این کد با ترکیب قدرت پایتون و لاتک ، یک ابزار قدرتمند برای مصورسازی الگوریتم‌های جست‌وجو، به ویژه جست‌وجوی دودویی، ارائه می‌دهد. انعطاف‌پذیری در تنظیمات مربوط به رنگ‌ها و اندازه‌ها، این امکان را فراهم می‌آورد که نتایج سفارشی‌شده‌ای تولید کرد که فرآیند جست‌وجو را به شکلی شفاف و دقیق نمایش می‌دهند. این ویژگی‌ها باعث می‌شود تا این ابزار، گزینه‌ای ایده‌آل برای آموزش و تحقیق در زمینه الگوریتم‌ها باشد.

۴-۳- الگوریتم جست‌وجوی خطی

الگوریتم جست‌وجوی خطی یک روش ساده و ابتدایی برای یافتن یک عنصر خاص در یک آرایه است. این الگوریتم با بررسی هر عنصر از آرایه، یکی پس از دیگری، به دنبال عنصر مورد نظر می‌گردد و در صورتی که عنصر مورد نظر را پیدا کند، موقعیت آن را برمی‌گرداند. در غیر این صورت، اگر جست‌وجو به پایان برسد و عنصر پیدا نشود، مقدار ۱- برگردانده می‌شود که نشان‌دهنده عدم وجود عنصر در آرایه است.

تابع اصلی linear_search مسئول اجرای الگوریتم جست‌وجوی خطی است. این تابع یک آرایه از اعداد صحیح و همچنین عدد مورد جست‌وجو (هدف) را به‌عنوان ورودی دریافت می‌کند. سپس با شروع از اولین عنصر آرایه، هر عنصر با عدد هدف مقایسه می‌شود. اگر عنصر مورد نظر یافت شود، موقعیت آن به همراه یک لیست از تمامی مراحل جست‌وجو بازگردانده می‌شود. این لیست شامل موقعیت فعلی جست‌وجو و نتیجه مقایسه (یافتن یا نیافتن عنصر) است. در صورتی که جست‌وجو به پایان برسد و عنصر یافت نشود، مقدار ۱- و لیست مراحل جست‌وجو بازگردانده می‌شود.

تابع generate_latex مسئول تولید کد لاتک برای تجسم گرافیکی مراحل جست‌وجو است. این تابع آرایه، لیست مراحل جست‌وجو، عدد هدف و رنگ‌های مشخص‌شده برای نمایش عناصر یافت‌شده و نیافت‌شده را به‌عنوان ورودی دریافت می‌کند. سپس کد لاتک را تولید می‌کند که در آن هر مرحله از جست‌وجو به‌صورت

گرافیکی نمایش داده می‌شود. در این تجسم گرافیکی، از بسته‌های لاتک مانند tikz استفاده شده است تا هر عنصر آرایه به‌صورت یک مستطیل رنگی نمایش داده شود. مستطیل‌های مربوط به عناصری که با موفقیت یافت شده‌اند با رنگ مخصوص به خود و سایر عناصر با رنگ‌های مختلف نمایش داده می‌شوند تا فرآیند جست‌وجو به‌صورت بصری و گرافیکی قابل مشاهده باشد.

در این کد، کاربر می‌تواند اندازه آرایه و عناصر آن را وارد کند و همچنین رنگ‌های دلخواه برای عناصر یافت‌شده و یافت نشده را انتخاب کند. خروجی این کد یک فایل لاتک است که تمامی مراحل جست‌وجوی خطی را به‌صورت گام‌به‌گام نمایش می‌دهد. این پیاده‌سازی نه تنها الگوریتم جست‌وجوی خطی را به‌صورت کامل و دقیق اجرا می‌کند، بلکه با ارائه تجسم گرافیکی از مراحل مختلف، آن را به یک ابزار آموزشی قدرتمند تبدیل کرده است.

۴-۴- الگوریتم مرتب‌سازی ادغامی

الگوریتم مرتب‌سازی ادغامی یکی از روش‌های موثر و کارآمد برای دسته‌بندی داده‌ها است که بر مبنای اصل "تقسیم و غلبه" عمل می‌کند. کدی که برای پیاده‌سازی این الگوریتم در پژوهش نوشته شده است، به طور گرافیکی و با استفاده از لاتک، فرآیند تقسیم یک آرایه به زیرآرایه‌ها و ادغام آن‌ها به یک لیست مرتب‌شده را به تصویر می‌کشد.

یکی از بخش‌های کلیدی این کد، تابع draw_array است که وظیفه ترسیم آرایه‌ها و برجسته‌سازی عناصر مختلف آن‌ها در یک سند لاتک را بر عهده دارد. این تابع با استفاده از TikZ، اشکال گرافیکی لازم را ایجاد کرده و از ویژگی‌هایی نظیر رنگ‌بندی سفارشی برای نمایش مراحل مختلف مرتب‌سازی بهره می‌برد. این تابع شامل تنظیماتی برای اندازه و رنگ سلول‌های آرایه، نمایش مقادیر عناصر و اضافه کردن عناوین دلخواه به بالای هر آرایه است.

تابع دیگر، merge_sort_visualize، مسئولیت تجسم‌سازی مراحل مرتب‌سازی ادغامی را بر عهده دارد. در این تابع، مراحل مختلف تقسیم و ادغام آرایه به طور جداگانه ذخیره شده و سپس در یک سند لاتک نمایش داده می‌شوند. این فرآیند با تقسیم آرایه به دو قسمت و ادامه دادن این تقسیمات تا رسیدن به عناصر منفرد آغاز شده و سپس ادغام این بخش‌ها به ترتیب برای ایجاد یک لیست مرتب‌شده صورت می‌گیرد.

تابع create_visualization نیز به منظور مدیریت تمامی مراحل تجسم‌سازی در سند لاتک طراحی شده است. در این تابع، سند لاتک با استفاده از تنظیمات اولیه‌ای نظیر geometry و tikz ایجاد شده و رنگ‌های مورد نیاز برای نمایش مراحل مختلف تعریف می‌شوند. همچنین، یک بخش مقدمه در سند اضافه می‌شود که توضیحاتی درباره الگوریتم مرتب‌سازی ادغامی ارائه می‌دهد و در نهایت، مراحل تقسیم و ادغام به ترتیب در سند درج می‌شوند.

این کد به کاربر این امکان را می‌دهد که ورودی‌های مختلفی نظیر رنگ‌های مورد نظر برای هر مرحله از مرتب‌سازی، ابعاد سلول‌های آرایه و مقادیر آرایه ورودی را به صورت دلخواه تنظیم کند. این تنظیمات به عنوان پارامترهای ورودی به توابع انتقال داده شده و در فرآیند تجسم‌سازی استفاده می‌شوند.

۴-۵- الگوریتم مرتب‌سازی سریع

الگوریتم مرتب‌سازی سریع یکی از الگوریتم‌های کارآمد در دسته‌بندی داده‌ها است که بر اساس اصل "تقسیم و غلبه" عمل می‌کند. این کد با بهره‌گیری از لاتک به شکل گرافیکی فرآیند تقسیم یک آرایه به بخش‌های کوچک‌تر و مرتب‌سازی آن‌ها را به نمایش می‌گذارد.

یکی از بخش‌های اصلی این کد، تابع partition است که وظیفه پیدا کردن موقعیت تقسیم (Partition) را بر عهده دارد. در این تابع، عنصر محوری به عنوان آخرین عنصر در آرایه انتخاب می‌شود و آرایه به دو بخش تقسیم می‌شود: بخشی شامل عناصر کوچک‌تر یا مساوی با عنصر محوری و بخشی دیگر شامل عناصر بزرگ‌تر از آن. این تابع همچنین مراحل مرتب‌سازی را در لیست‌های steps و swaps ثبت می‌کند تا در مراحل بعدی به نمایش درآیند.

تابع دیگر این پژوهش، quicksort نام دارد که الگوریتم اصلی مرتب‌سازی سریع را اجرا می‌کند. این تابع به صورت بازگشتی عمل کرده و آرایه را به بخش‌های کوچک‌تر تقسیم کرده و سپس مرتب می‌کند. این فرآیند با تقسیم آرایه به دو قسمت آغاز می‌شود و در نهایت، بخش‌های تقسیم‌شده به صورت ادغامی مرتب می‌شوند تا یک لیست مرتب‌شده به دست آید.

برای نمایش گرافیکی مراحل مختلف مرتب‌سازی، تابع save_steps_as_tikz طراحی شده است. این تابع، سند لاتک را ایجاد کرده و مراحل ثبت‌شده در steps و swaps را به صورت تصویری و با استفاده از TikZ در یک فایل PDF به نمایش می‌گذارد. این تابع شامل تنظیماتی برای اندازه مربع‌های نمایش‌دهنده عناصر آرایه، رنگ‌های پایه برای نمایش عناصر و رنگ‌های ویژه برای نمایش عناصر جابه‌جا شده در هر مرحله است.

کاربران این کد قادر هستند تنظیمات ورودی مانند اندازه مربع‌ها و رنگ‌های مورد نظر برای نمایش مراحل مختلف مرتب‌سازی را تعیین کنند. این تنظیمات به عنوان ورودی‌های تابع در فرآیند تجسم‌سازی استفاده شده و در نهایت خروجی سفارشی‌شده‌ای بر اساس نیاز کاربر تولید می‌شود.

در نتیجه، این کد با ارائه روشی تعاملی و گرافیکی برای درک بهتر الگوریتم مرتب‌سازی سریع، ابزاری مفید برای کاربردهای آموزشی و مستندسازی در حوزه علوم کامپیوتر فراهم می‌کند. با قابلیت تنظیمات انعطاف‌پذیر در ابعاد و رنگ‌ها، نتایج به دست آمده به خوبی تمامی مراحل این الگوریتم را به تصویر می‌کشند.

۴-۶- الگوریتم مرتب‌سازی حبابی

الگوریتم مرتب‌سازی حبابی یکی از الگوریتم‌های ساده و کلاسیک در علم رایانه است که به‌منظور مرتب‌سازی لیست‌ها طراحی شده است. این الگوریتم به طور مکرر از میان لیست عبور کرده و عناصر مجاور را مقایسه و در صورت لزوم جابجا می‌کند تا لیست به ترتیب مرتب شود. هدف این پژوهش، مصورسازی مراحل مختلف الگوریتم مرتب‌سازی حبابی به‌صورت گرافیکی است. برای این منظور، از لاتک و TikZ برای ایجاد نمودارهایی استفاده می‌شود که فرآیند مرتب‌سازی را به طور بصری نمایش می‌دهند.

کد پیاده‌سازی این قسمت شامل دو تابع اصلی است. تابع get_user_input مسئول دریافت ورودی‌های مورد نیاز برای مصورسازی است. این ورودی‌ها شامل آرایه‌ای هستند که باید مرتب شود، رنگ‌های اولیه و ثانویه برای نمایش مقایسه عناصر و تعداد آرایه‌هایی که در هر صفحه از سند لاتک نمایش داده می‌شود. تابع دوم، generate_latex_code، کد لاتک را تولید می‌کند که شامل نمودارهایی است که مراحل مختلف مرتب‌سازی حبابی را نمایش می‌دهد. در این نمودارها، هر عنصر از آرایه با یک مستطیل نمایان می‌شود و رنگ‌های تعیین‌شده برای نمایش مقایسه و جابجایی عناصر به‌کار می‌رود.

ورودی‌های مورد نیاز برای این کد شامل آرایه‌ای از اعداد صحیح است که باید مرتب شوند، رنگ اولیه برای نمایش عنصر اول در حال مقایسه، رنگ ثانویه برای نمایش عنصر دوم در حال مقایسه، و تعداد آرایه‌های نمایش داده‌شده در هر صفحه است. نمودارهای تولیدشده با استفاده از این کد شامل نمایش گرافیکی مراحل مختلف الگوریتم مرتب‌سازی حبابی هستند. هر مرحله به‌طور مجزا، وضعیت آرایه را پس از پردازش هر جفت عنصر و در صورت جابجایی آن‌ها نمایش می‌دهد. در این نمودارها، عناصر آرایه با مستطیل‌هایی نمایش داده می‌شوند و رنگ‌های اولیه و ثانویه برای نمایش مقایسه دو عنصر استفاده می‌شود. در صورت جابجایی عناصر، رنگ‌ها و موقعیت مستطیل‌ها به‌روزرسانی می‌شود تا تغییرات به‌وضوح نمایش داده شوند.

۴-۷- الگوریتم مرتب‌سازی درجی

الگوریتم مرتب‌سازی درجی یکی از ساده‌ترین و در عین حال مؤثرترین الگوریتم‌های مرتب‌سازی است که به دلیل سادگی و کارایی در برخی از سناریوها، به‌ویژه برای آرایه‌های کوچک، مورد استفاده قرار می‌گیرد. در این پیاده‌سازی از زبان برنامه‌نویسی پایتون و ابزارهایی نظیر pylatex، Latex و Tikz، فرآیند مرتب‌سازی به‌طور دقیق و گام‌به‌گام دنبال می‌شود تا ترتیب عناصر در یک آرایه مشخص به حالت صعودی درآید.

پیاده‌سازی الگوریتم مرتب‌سازی درجی، از دو بخش اصلی شامل یک کلاس و دو تابع کلیدی تشکیل شده است. این اجزا به گونه‌ای طراحی شده‌اند که فرآیند مرتب‌سازی و تجسم گرافیکی آن به صورت دقیق و کارآمد انجام شود.

دو تابع کلیدی insertion_sort و generate_latex پیاده‌سازی شده‌اند که به ترتیب برای اجرای

الگوریتم مرتب‌سازی درجی و تولید کد لاتک به منظور تجسم گرافیکی فرآیند مرتب‌سازی استفاده می‌شوند.

تابع insertion_sort مسئول اجرای خود الگوریتم مرتب‌سازی درجی است. این تابع یک آرایه از اعداد صحیح را به‌عنوان ورودی دریافت می‌کند و با استفاده از الگوریتم مرتب‌سازی درجی، آرایه را به صورت صعودی مرتب می‌کند. در این الگوریتم، هر عنصر از آرایه (به جز اولین عنصر که به‌طور پیش‌فرض مرتب شده در نظر گرفته می‌شود) با عناصر قبلی خود مقایسه می‌شود تا در جایگاه صحیح خود قرار گیرد. در هر مرحله از این مقایسه و جابجایی، وضعیت فعلی آرایه به همراه موقعیت عنصر کلیدی (key) و عنصر مقایسه‌شده در یک لیست ذخیره می‌شود. این لیست شامل تمامی مراحل مرتب‌سازی است که بعداً برای ایجاد نمایش گرافیکی از آن استفاده می‌شود.

تابع generate_latex مسئول ایجاد کد لاتک برای تجسم گرافیکی مراحل مرتب‌سازی است. این تابع آرایه اولیه، لیست مراحل مرتب‌سازی، و رنگ‌های مشخص‌شده برای نمایش عناصر کلیدی و مقایسه‌شده را به‌عنوان ورودی دریافت می‌کند و سپس کد لاتک مربوطه را تولید می‌کند. در این کد، از بسته‌های لاتک مانند tikz برای رسم تصاویر استفاده می‌شود. در هر مرحله، عناصر آرایه به‌صورت مستطیل‌هایی با رنگ‌های مشخص نمایش داده می‌شوند که موقعیت‌های کلیدی و مقایسه‌شده را به‌طور واضح نشان می‌دهند. این تصاویر به‌صورت یک سند لاتک ذخیره می‌شوند که قابل مشاهده و چاپ است.

۴-۸- الگوریتم مرتب سازی انتخابی

الگوریتم مرتب‌سازی انتخابی یک روش ساده و مؤثر برای مرتب‌سازی آرایه‌ها است که در آن در هر مرحله، کوچکترین عنصر موجود در بخش مرتب‌نشده آرایه پیدا شده و با اولین عنصر آن بخش مبادله می‌شود. این فرآیند برای تمامی عناصر آرایه تکرار می‌شود تا کل آرایه به ترتیب صعودی مرتب شود.

در کد این قسمت تابع اصلی selection_sort وظیفه اجرای این الگوریتم را بر عهده دارد. این تابع یک آرایه از اعداد صحیح را به‌عنوان ورودی دریافت می‌کند و در هر مرحله از مرتب‌سازی، موقعیت عنصر فعلی (کلید)، کوچکترین عنصر یافت‌شده در بخش مرتب‌نشده (حداقل مقدار)، و عنصر در حال مقایسه را ثبت می‌کند. این اطلاعات به‌صورت مراحل مختلف در یک لیست ذخیره می‌شوند تا بعداً برای تجسم گرافیکی استفاده شوند. این رویکرد به کاربر اجازه می‌دهد تا تمامی مراحل مرتب‌سازی را به‌صورت دقیق مشاهده کند و به‌طور کامل از نحوه عملکرد الگوریتم آگاه شود.

تابع generate_latex که مسئول تولید کد لاتک برای تجسم گرافیکی مراحل مرتب‌سازی است، آرایه ورودی، لیست مراحل، و رنگ‌های مشخص‌شده برای نمایش عناصر مختلف (کلید، حداقل مقدار، و عنصر در حال مقایسه) را به‌عنوان ورودی دریافت می‌کند. این تابع با استفاده از بسته‌های لاتک مانند tikz و xcolor، یک سند لاتک تولید می‌کند که هر مرحله از مرتب‌سازی انتخابی را به‌صورت گرافیکی نمایش می‌دهد. در این

تجسم گرافیکی، از مستطیل‌های رنگی برای نمایش عناصر آرایه استفاده شده است؛ به‌طوری که هر عنصر بسته به نقش آن در آن مرحله (کلید، حداقل مقدار، یا عنصر در حال مقایسه) با رنگ مشخصی نمایش داده می‌شود.

برای تولید این سند لاتک ، از چندین پکیج استفاده شده است که هرکدام نقش ویژه‌ای در ایجاد و نمایش مراحل مختلف مرتب‌سازی انتخابی دارند. پکیج geometry برای تنظیم حاشیه‌های صفحه به‌کار رفته است، به‌طوری که حاشیه‌ها به ۰.۵ اینچ تنظیم شده‌اند تا فضای کافی برای نمایش گرافیکی فراهم شود. پکیج xcolor برای تعریف و استفاده از رنگ‌های دلخواه به‌کار گرفته شده تا هر عنصر بسته به نقش خود در مرتب‌سازی به‌خوبی تمایز داده شود. پکیج قدرتمند tikz برای رسم اشکال هندسی و گرافیک‌های برداری به کار رفته که امکانات زیادی برای تجسم داده‌ها و الگوریتم‌ها در لاتک فراهم می‌کند.

استفاده از این پکیج‌ها و ترکیب اجرای الگوریتم مرتب‌سازی انتخابی و تجسم گرافیکی مراحل مختلف آن، منجر به ایجاد یک سند لاتک کامل و قابل تجسم از مراحل الگوریتم مرتب‌سازی انتخابی شده است.

۴-۹- الگوریتم پیاده‌سازی صف با استفاده از دو پشته

در این بخش از پژوهش، پیاده‌سازی صف با استفاده از دو پشته به صورت گرافیکی و بصری مصورسازی شده است. صف، که به عنوان یک ساختار داده‌ای با الگوی FIFO شناخته می‌شود، در این کد با استفاده از دو پشته، که از الگوی LIFO پیروی می‌کنند، پیاده‌سازی شده است. این روش، به عنوان یک راه‌حل کلاسیک در علم کامپیوتر، برای نمایش نحوه عملکرد صف از پشته‌ها استفاده می‌کند. برای تصویرسازی مراحل مختلف این پیاده‌سازی، کدی به زبان پایتون نوشته شده که از کتابخانه‌های pylatex و TikZ برای ایجاد یک سند لاتک بهره می‌برد. این سند به طور جامع نحوه عملکرد صف با دو پشته را به تصویر می‌کشد.

هسته اصلی این پیاده‌سازی، کلاس QueueWithStacks است که تمامی عملیات‌های مربوط به صف را با استفاده از دو پشته مدیریت می‌کند. در این کلاس، دو پشته با نام‌های stack1 و stack2 تعریف شده‌اند که وظیفه عملیات‌های اضافه کردن به صف (enqueue) و برداشتن از صف (dequeue) را بر عهده دارند. همچنین، این کلاس شامل متدهایی برای مدیریت سند لاتک و ایجاد نمایش‌های گرافیکی از مراحل مختلف عملیات صف است. مهم‌ترین متدهای این کلاس شامل __init__ برای مقداردهی اولیه و تنظیمات سند لاتک ، create_document برای ایجاد بخش‌های ابتدایی سند و set_cell_dimensions برای تنظیم ابعاد سلول‌های گرافیکی پشته‌ها هستند. متدهای enqueue و dequeue به ترتیب برای اضافه کردن عنصر جدید به stack1 و برداشتن از صف با جابجایی عناصر بین دو پشته پیاده‌سازی شده‌اند. متد add_step برای هر مرحله از عملیات صف، بخش‌هایی به سند اضافه کرده و وضعیت فعلی پشته‌ها را رسم می‌کند. همچنین، متد draw_stack به رسم پشته‌ها در سند لاتک و متد generate_latex به تولید و ذخیره سند نهایی اختصاص دارد.

در ابتدای اجرای این کد، کاربر باید تنظیمات ورودی شامل رنگ‌های سلول‌های پشته، ابعاد سلول‌ها، مقادیر مربوط به عملیات enqueue و تعداد عناصری که باید از صف برداشته شود را وارد کند. این تنظیمات به کلاس QueueWithStacks هدایت شده و در طول اجرای کد برای تولید خروجی‌های گرافیکی مورد استفاده قرار می‌گیرند. هر مرحله از عملیات‌های enqueue و dequeue به صورت گرافیکی نمایش داده می‌شود، با استفاده از کتابخانه TikZ در لاتک که ابزاری قدرتمند برای رسم گرافیک‌ها در اسناد لاتک است. این نمایش شامل رسم پشته‌ها با اندازه و رنگ‌های تعیین شده و نمایش عناصر پشته‌ها به همراه نام آن‌ها است.

این کد به عنوان یک ابزار آموزشی بسیار مفید می‌تواند برای درک بهتر مفهوم پیاده‌سازی صف با دو پشته مورد استفاده قرار گیرد.

۴-۱۰- پیاده‌سازی پشته با استفاده از دو صف

پشته، که یک ساختار داده‌ای با ویژگی (LIFO) است، در این پیاده‌سازی به وسیله‌ی دو صف که به صورت (FIFO) عمل می‌کنند، شبیه‌سازی می‌شود. هدف این کد، ارائه گرافیکی مراحل مختلف عملیات‌های push و pop روی پشته از طریق تولید نمودارهایی در سند لاتک است.

کلاس Queue بخش ابتدایی این پیاده‌سازی را تشکیل می‌دهد و نماینده یک صف است که عملیات‌های پایه‌ای مانند اضافه کردن و حذف کردن آیتم‌ها را پیاده‌سازی می‌کند. این کلاس شامل ویژگی‌هایی نظیر items برای نگهداری آیتم‌های صف و متدهایی همچون __init__ برای ایجاد یک شیء جدید صف با لیستی خالی، is_empty برای بررسی خالی بودن صف، enqueue(data) برای اضافه کردن آیتم به صف و dequeue برای حذف و بازگرداندن اولین آیتم از صف است.

کلاس Stack، که با استفاده از دو صف پیاده‌سازی شده است، نماینده پشته در این پیاده‌سازی است. این کلاس شامل ویژگی‌هایی نظیر queue1 و queue2 برای نگهداری دو صف مورد استفاده، doc به عنوان شیء LaTeX Document برای تولید فایل خروجی، queue_capacity برای تعیین ظرفیت صف‌ها، width و height برای ابعاد مستطیل‌ها، push_color_name برای نام رنگ عملیات push، element_colors برای ذخیره رنگ هر عنصر push شده و font_size برای اندازه فونت متون است. متدهای این کلاس شامل is_empty برای بررسی خالی بودن پشته، push(data) برای اضافه کردن آیتم به پشته و به‌روزرسانی نمودارها، pop برای حذف و بازگرداندن آخرین آیتم از پشته و به‌روزرسانی نمودار و draw_queues(title) برای ترسیم وضعیت‌های مختلف صف‌ها با استفاده از TikZ می‌باشند.

تابع process_operations مسئول پردازش عملیات‌های push و pop و به‌روزرسانی سند لاتک با استفاده از کلاس Stack است. این تابع شامل دریافت عملیات‌ها از ورودی کاربر و اجرای این عملیات‌ها بر روی پشته به همراه به‌روزرسانی وضعیت‌های پشته است.

تابع main وظیفه دریافت ورودی‌های کاربر، ایجاد سند لاتک و تولید PDF نهایی را بر عهده دارد. این تابع شامل مراحل دریافت ورودی‌های مربوط به عملیات‌ها، ابعاد مستطیل‌ها، و رنگ عملیات‌های push، ایجاد سند لاتک و اضافه کردن توضیحات مربوط به پیاده‌سازی پشته با استفاده از دو صف، پردازش عملیات‌ها و تولید PDF نهایی است.

در تنظیمات ورودی، ورودی‌های مربوط به عملیات‌ها شامل رشته‌های نمایانگر عملیات‌های push و pop، ابعاد مستطیل‌ها که شامل عرض و ارتفاع هر مستطیل برای نمایش در نمودارها و رنگ عملیات‌های push که برای نمایش آیتم‌های push شده استفاده می‌شود، ارائه شده است. مراحل تصویری شامل رسم وضعیت صف‌ها در هر مرحله از عملیات‌های push و pop و ترسیم تصاویری که وضعیت فعلی صف‌ها را نشان می‌دهند با استفاده از رنگ‌های تعیین‌شده، اضافه کردن جدول نهایی به سند لاتک و نمایش ترتیب اضافه کردن و حذف کردن آیتم‌ها به صورت گرافیکی است.

۴-۱۱- الگوریتم عملیات‌های پشته

پشته، که یکی از ساختارهای داده‌ای کلیدی در علوم کامپیوتر است، به روش (LIFO) عمل می‌کند. هدف این کد، نمایش گرافیکی مراحل اضافه کردن و حذف کردن آیتم‌ها از پشته از طریق تولید نمودارهایی در سند لاتک است.

کلاس Stack به عنوان بخش اصلی کد، نماینده یک پشته است و عملیات‌های پایه‌ای مانند اضافه کردن (push) و حذف کردن (pop) آیتم‌ها را پیاده‌سازی می‌کند. این کلاس شامل ویژگی‌های مهمی همچون items برای نگهداری لیست آیتم‌های پشته و متدهایی از جمله __init__ برای ایجاد شیء جدید پشته، is_empty() برای بررسی خالی بودن پشته، push(item) برای اضافه کردن آیتم به پشته و pop برای حذف و بازگرداندن آخرین آیتم از پشته است.

تابع draw_stack مسئول ترسیم وضعیت پشته در هر مرحله و نمایش آن با استفاده از TikZ است. این تابع شامل ترسیم مستطیل‌ها برای نمایش آیتم‌ها با رنگ‌های مختلف، نمایش متن مربوط به هر آیتم در وسط مستطیل‌ها، و رسم مرزهای پشته و خطوط جداساز بین آیتم‌ها می‌باشد. به این ترتیب، وضعیت‌های مختلف پشته به صورت بصری و دقیق نمایش داده می‌شود.

تابع create_latex_document وظیفه ایجاد سند لاتک را بر عهده دارد که شامل نمودارهای مراحل مختلف اضافه کردن و حذف کردن آیتم‌ها از پشته است. این تابع شامل مراحل مهمی است از جمله دریافت ورودی‌های کاربر مانند آیتم‌های پشته، ابعاد مستطیل‌ها، رنگ‌ها و تعداد تصاویر در هر ردیف، ترسیم نمودارهای مربوط به عملیات‌ها با استفاده از تابع draw_stack، و ایجاد جداول نهایی برای نمایش وضعیت‌های مختلف پشته در مراحل مختلف.

در مراحل تصویری، آیتم‌های پشته در هر مرحله از اضافه کردن یا حذف کردن به صورت نمودارهایی با رنگ‌های تعیین‌شده توسط کاربر نمایش داده می‌شوند. جدول نهایی حاوی وضعیت‌های مختلف پشته به سند لاتک اضافه می‌شود و ترتیب اضافه کردن و حذف کردن آیتم‌ها به صورت گرافیکی نمایش داده می‌شود.

۴-۱۲- الگوریتم زمان‌بندی زمانبندی اولویت‌ها

الگوریتم زمانبندی اولویت‌ها یکی از روش‌های مهم در مدیریت زمان اجرای فرآیندها در سیستم‌های کامپیوتری است که بر اساس اولویت‌های اختصاص داده شده به هر فرآیند عمل می‌کند. در این الگوریتم، فرآیندهایی که دارای اولویت بالاتری هستند، در صف اجرا نسبت به فرآیندهای با اولویت پایین‌تر در اولویت قرار می‌گیرند. هدف این پژوهش، نمایش بصری و دقیق این فرآیند زمانبندی است.

یکی از اجزای کلیدی این قسمت، کلاس Process است که برای ایجاد و مدیریت اطلاعات مرتبط با هر فرآیند طراحی شده است. این کلاس ویژگی‌هایی مانند نام فرآیند، زمان اجرای آن، زمان ورود، اولویت و رنگ مربوط به هر فرآیند را تعریف و نگهداری می‌کند. این اطلاعات برای زمانبندی و نمایش گرافیکی فرآیندها مورد استفاده قرار می‌گیرند.

تابع priority_scheduling_latex بخش دیگری از کد است که وظیفه اصلی آن ایجاد یک سند لاتک برای نمایش مراحل زمانبندی فرآیندها است. این تابع با استفاده از الگوریتم زمانبندی اولویت‌ها، فرآیندها را بر اساس اولویت و زمان ورودشان مرتب کرده و نتایج حاصل از هر مرحله را به صورت گرافیکی در سند لاتک نمایش می‌دهد. در نهایت، سند نهایی شامل تمامی مراحل و یک جدول نهایی از فرآیندها تولید شده و به صورت PDF ذخیره می‌شود.

تابع draw_process به منظور رسم هر مرحله از زمانبندی فرآیندها در سند لاتک طراحی شده است. این تابع با استفاده از کتابخانه TikZ، هر فرآیند را با رنگ و ابعاد مشخص شده توسط کاربر در سلول‌های جداگانه‌ای رسم می‌کند. علاوه بر این، زمان شروع و پایان هر فرآیند به همراه نام آن در نمودارها نمایش داده می‌شود که به درک بهتر فرآیند زمانبندی کمک می‌کند.

تابع draw_execution_order نیز ترتیب اجرای فرآیندها را پس از اتمام زمانبندی به صورت یک نمودار کلی رسم می‌کند. در این نمودار، هر فرآیند با رنگ مربوطه و زمان‌های شروع و پایانش نمایش داده می‌شود تا یک نمای کلی از ترتیب اجرای فرآیندها ارائه شود.

۴-۱۳- الگوریتم زمان‌بندی کوتاه‌ترین زمان باقی‌مانده (SRT)

الگوریتم SRT یکی از تکنیک‌های برنامه‌ریزی فرآیند در سیستم‌های عامل است که به انتخاب و اجرای فرآیند با کمترین زمان باقی‌مانده اختصاص دارد. این کد به طور گرافیکی مراحل اجرای فرآیندها را در قالب یک سند لاتک ترسیم می‌کند و به درک بهتر نحوه عملکرد این الگوریتم کمک می‌نماید.

یکی از اجزای کلیدی این کد، کلاس Process است که به منظور نمایندگی و مدیریت ویژگی‌های مرتبط با هر فرآیند طراحی شده است. این کلاس شامل ویژگی‌هایی از جمله نام فرآیند، زمان لازم برای اجرای آن، زمان ورود، زمان باقی‌مانده، زمان پایان، زمان شروع سرویس‌دهی، و رنگ نماینده فرآیند در نمودار می‌باشد. این اطلاعات برای زمانبندی و نمایش گرافیکی فرآیندها مورد استفاده قرار می‌گیرند.

تابع srt_scheduling_latex مسئول محاسبه و ترسیم زمان‌بندی فرآیندها به روش SRT است و سند LaTeX را تولید می‌کند. در این تابع، رنگ‌های مربوط به هر فرآیند با استفاده از definecolor\ تعیین می‌شوند و مقیاس فونت بر اساس ابعاد سلول‌ها محاسبه می‌شود. سپس، با اجرای حلقه زمان‌بندی، الگوریتم SRT شبیه‌سازی شده و مراحل مختلف به همراه جداول نهایی به صورت گرافیکی در سند لاتک نمایش داده می‌شود.

تابع draw_process به منظور ترسیم فرآیندها و نمودارهای مرتبط در هر مرحله از زمان‌بندی استفاده می‌شود. این تابع به کمک TikZ مستطیل‌های نماینده فرآیندها را رسم کرده و زمان‌های شروع و پایان هر فرآیند را به نمایش می‌گذارد. کیفیت و دقت گرافیک‌های تولید شده با استفاده از این تابع بسیار بالا است.

تابع draw_execution_order برای ترسیم ترتیب اجرای فرآیندها در طول زمان طراحی شده است. مشابه تابع draw_process، این تابع نمودارهایی را که ترتیب و زمان‌های اجرای فرآیندها را نمایش می‌دهند، ایجاد می‌کند.

۴-۱۴- الگوریتم زمان‌بندی کوتاه‌ترین کار (SJN)

الگوریتم SJN یکی از روش‌های مهم در زمان‌بندی فرآیندها در سیستم‌های عامل است که به منظور کاهش زمان انتظار کلی سیستم، فرآیند با کمترین زمان لازم برای اجرا را انتخاب و اجرا می‌کند. کد ارائه‌شده، مراحل مختلف اجرای فرآیندها را به صورت گرافیکی در قالب یک سند لاتک ترسیم کرده و نتایج نهایی را به صورت جدول و نمودار نمایش می‌دهد.

کلاس Process به عنوان یکی از اجزای اصلی کد، نماینده یک فرآیند است و شامل ویژگی‌های کلیدی نظیر نام فرآیند، زمان لازم برای اجرا، زمان ورود به سیستم، زمان باقی‌مانده، زمان پایان، زمان شروع سرویس‌دهی و رنگ نماینده فرآیند در نمودار می‌باشد. این اطلاعات به طور مؤثری در شبیه‌سازی و نمایش الگوریتم SJN استفاده می‌شود.

تابع sjn_scheduling_latex مسئول محاسبه و ترسیم زمان‌بندی فرآیندها به روش SJN و تولید سند لاتک است. این تابع شامل مراحل مهمی است از جمله تعریف رنگ‌های مربوط به هر فرآیند با استفاده از definecolor\، محاسبه مقیاس برای اندازه فونت بر اساس ابعاد سلول‌ها، و اجرای حلقه زمان‌بندی برای شبیه‌سازی الگوریتم SJN. این مراحل به ترسیم دقیق مراحل مختلف و جداول نهایی کمک می‌کند.

تابع draw_process برای ترسیم فرآیندها و نمودارهای مرتبط در هر مرحله از زمان‌بندی طراحی شده است. این تابع با استفاده از TikZ، مستطیل‌های نماینده فرآیندها را رسم کرده و زمان‌های شروع و پایان هر فرآیند را به نمایش می‌گذارد، که به ایجاد گرافیک‌های دقیق و با کیفیت کمک می‌کند. همچنین، تابع draw_execution_order برای ترسیم ترتیب اجرای فرآیندها در طول زمان استفاده می‌شود و نمودارهای مربوط به ترتیب و زمان‌های اجرای فرآیندها را تولید می‌کند.

در مراحل تصویری این کد، ابتدا فرآیندها در هر مرحله از زمان‌بندی با استفاده از رنگ‌های تعیین‌شده ترسیم می‌شوند. سپس، جدول نهایی شامل زمان‌های ورود، زمان‌های لازم برای اجرا و زمان‌های شروع سرویس‌دهی فرآیندها به سند لاتک اضافه می‌شود. در نهایت، ترتیب اجرای فرآیندها به صورت گرافیکی نمایش داده می‌شود.

۴-۱۵- الگوریتم زمان‌بندی اجرا به ترتیب ورود(FCFS)

این الگوریتم یکی از ساده‌ترین و ابتدایی‌ترین روش‌ها برای مدیریت صف فرآیندها در سیستم‌های عامل است که بر مبنای زمان ورود فرآیندها عمل می‌کند. به عبارت دیگر، هر فرآیند بر اساس زمانی که وارد صف می‌شود، به ترتیب سرویس‌دهی می‌شود و این ترتیب تا پایان اجرای تمامی فرآیندها ادامه می‌یابد.

در این پیاده‌سازی، لیستی از فرآیندها که شامل زمان ورود و زمان اجرای هر فرآیند است، به‌عنوان ورودی دریافت می‌شود. سپس، فرآیندها بر اساس زمان ورود مرتب شده و به‌ترتیب سرویس‌دهی می‌شوند. کد لاتک تولید شده، شامل دو بخش اصلی است: اولین بخش مربوط به نمایش لحظه ورود هر فرآیند به صف و دومین بخش مربوط به خروج فرآیندها از صف پس از اجرای آن‌هاست. در هر یک از این مراحل، وضعیت صف به‌صورت گرافیکی و با استفاده از مستطیل‌های رنگی که نمایانگر هر فرآیند هستند، نمایش داده می‌شود.

یکی از ویژگی‌های قابل توجه این پیاده‌سازی، امکان انتخاب و نمایش رنگ‌های مختلف برای فرآیندها است که این امر باعث تمایز و شناسایی راحت‌تر فرآیندها در تجسم گرافیکی می‌شود. در این کد، از پکیج‌های قدرتمندی نظیر tikz برای رسم اشکال گرافیکی، xcolor برای استفاده از رنگ‌ها، و amsmath برای نگارش معادلات ریاضیاتی استفاده شده است. همچنین، قابلیت استفاده از لیست‌های مرتب‌شده و ساختارهای شرطی در طول فرآیند زمان‌بندی به‌خوبی پیاده‌سازی شده است.

این پیاده‌سازی با ارائه یک فایل لاتک که تمامی مراحل صف‌بندی و سرویس‌دهی به فرآیندها را به‌صورت گام‌به‌گام نمایش می‌دهد، یک ابزار مفید برای تحلیل و آموزش الگوریتم FCFS فراهم می‌کند. این فایل لاتک به‌ویژه برای دانشجویان و پژوهشگرانی که در زمینه سیستم‌های عامل و مدیریت صف‌ها فعالیت می‌کنند، می‌تواند به‌عنوان یک ابزار آموزشی مؤثر مورد استفاده قرار گیرد. تجسم گرافیکی مراحل مختلف این الگوریتم، نه تنها به درک بهتر عملکرد FCFS کمک می‌کند، بلکه تحلیل و ارزیابی آن را نیز تسهیل می‌نماید.

۴-۱۶- الگوریتم زمان‌بندی نوبت گردشی (Round Robin)

الگوریتم زمان‌بندی نوبت گردشی یکی از مهم‌ترین و پرکاربردترین الگوریتم‌های زمان‌بندی در سیستم‌های عامل است که به‌طور گسترده برای مدیریت و تخصیص منابع به فرآیندهای مختلف استفاده می‌شود. در این پیاده‌سازی، فرآیندها به‌طور دایره‌ای و به ترتیب در صف قرار می‌گیرند و هر فرآیند به میزان زمان مشخصی که به آن "زمان چرخش" یا "کوانتوم زمانی" می‌گویند، اجازه اجرای خود را دارد. اگر زمان اجرای یک فرآیند از کوانتوم زمانی تعیین شده بیشتر باشد، اجرای آن متوقف شده و به انتهای صف منتقل می‌شود تا دوباره نوبتش برسد. این فرایند تا زمان اتمام اجرای تمامی فرآیندها ادامه می‌یابد.

کد پیاده‌سازی این قسمت، مراحل پیاده‌سازی و تجسم گرافیکی الگوریتم نوبت گردشی را با استفاده از لاتک نشان می‌دهد. در ابتدا، لیستی از فرآیندها که شامل زمان ورود و زمان اجرای هر فرآیند است، به‌عنوان ورودی دریافت می‌شود. سپس، بر اساس زمان ورود، فرآیندها مرتب شده و در یک صف آماده قرار می‌گیرند. در هر مرحله از اجرای فرآیندها، وضعیت صف به‌طور گرافیکی نمایش داده می‌شود، به‌طوری که هر فرآیند با استفاده از یک رنگ منحصر به فرد نمایش داده می‌شود که تشخیص و تمایز فرآیندها را آسان‌تر می‌کند.

کد پیاده‌سازی شده از چندین بسته و ابزار در لاتک استفاده می‌کند تا بتواند فرآیندهای مختلف را به صورت گرافیکی و متنی نمایش دهد؛ بسته amsmath یکی از پرکاربردترین بسته‌ها در لاتک است که برای فرمول‌نویسی‌های ریاضیاتی به کار می‌رود. همچنین بسته tikz یکی از قدرتمندترین ابزارها برای ترسیم گرافیکی در لاتک است نیز مورد استفاده قرار گرفته شده.

بسته longtable که برای ایجاد جداولی که طول آن‌ها از یک صفحه بیشتر است، استفاده می‌شود و بسته xcolor که برای مدیریت و استفاده از رنگ‌ها در لاتک به کار می‌رود نیز از دیگر بسته های مورد استفاده در این پیاده سازی هستند.

در نهایت، این کد به تولید یک فایل لاتک منجر می‌شود که تمامی مراحل اجرای الگوریتم راند رابین را به‌صورت گرافیکی و متنی مستند می‌کند.

۴-۱۷- الگوریتم زمان‌بندی صف‌های چند سطحی (MLFQ)

این الگوریتم یکی از پیچیده‌ترین و در عین حال موثرترین الگوریتم‌های زمان‌بندی است که برای تخصیص منابع در سیستم‌های چندکاربره و چندوظیفه‌ای استفاده می‌شود. این الگوریتم با استفاده از چندین صف با سطوح اولویت مختلف، به هر فرآیند در سیستم اجازه می‌دهد تا بسته به میزان استفاده از پردازنده و رفتار خود، به صورت پویا بین صف‌ها جابه‌جا شود. در این الگوریتم، فرآیندهایی که زمان پردازش کمتری دارند، در صف‌های با اولویت بالاتر قرار می‌گیرند و فرآیندهایی که نیاز به زمان بیشتری دارند، به تدریج به صف‌های با اولویت پایین‌تر منتقل می‌شوند. این مکانیزم باعث می‌شود که فرآیندهای تعاملی که نیاز به پاسخگویی سریع دارند،

سریع‌تر اجرا شوند و در عین حال، از فرآیندهای طولانی‌تر به صورت منصفآن‌های پردازش شود.

کد پیاده‌سازی شده برای الگوریتم MLFQ در زبان لاتک طراحی شده است تا فرآیندهای مختلف را در صف‌های اولویت‌بندی شده نمایش دهد و مراحل مختلف اجرای این الگوریتم را به‌صورت گرافیکی و متنی مستند کند. این کد ابتدا فرآیندها را بر اساس زمان ورود آن‌ها مرتب کرده و سپس هر فرآیند را بر اساس سطح اولویت آن در صف مربوطه قرار می‌دهد. در هر گام زمانی، وضعیت صف‌ها نمایش داده می‌شود و فرآیندی که باید اجرا شود، از صف با اولویت بالاتر انتخاب می‌شود. اگر این فرآیند نتواند در زمان تعیین شده خود (که به آن کوانتوم زمانی می‌گویند) به اتمام برسد، به صف پایین‌تر منتقل می‌شود. این فرایند تا زمانی ادامه می‌یابد که یا همه فرآیندها به اتمام برسند یا تمام صف‌ها خالی شوند.

در کد پیاده‌سازی، از پکیج‌های مختلف لاتک از جمله amsmath، tikz، longtable و xcolor استفاده شده است تا نه تنها گرافیک صف‌ها و نمودارهای را به درستی نمایش داده شود، بلکه خوانایی و کیفیت مستندات نهایی نیز افزایش یابد. پکیج amsmath برای مدیریت و فرمول‌بندی محاسبات ریاضیاتی در لاتک استفاده می‌شود و پکیج tikz برای رسم اشکال هندسی و نمودارها، به خصوص نمایش صف‌های فرآیندها کاربرد دارد.

این پیاده‌سازی، به‌طور دقیق مراحل مختلف اجرای الگوریتم زمان‌بندی صف‌های چند سطحی را به تصویر می‌کشد و با هر تغییر در صف‌ها و اجرای فرآیندها، وضعیت به‌روزشده‌ای را نمایش می‌دهد. این نمایش گرافیکی به تحلیل و ارزیابی نحوه اجرای فرآیندها کمک کرده و به کاربران اجازه می‌دهد تا به‌صورت بصری وضعیت صف‌ها و تخصیص منابع را مشاهده کنند.

۴-۱۸- عملیات مختلف در لیست پیوندی یک طرفه

لیست پیوندی یک طرفه یکی از ساختارهای داده‌ای پایه است که در آن هر گره شامل یک داده و یک اشاره‌گر به گره بعدی می‌باشد. این ساختار داده‌ای برای مدیریت و سازمان‌دهی داده‌ها در برنامه‌های مختلف کاربرد دارد. در این گزارش، به تحلیل و مصورسازی عملیات‌های اصلی شامل ایجاد، حذف، درج، و جست‌وجو در لیست پیوندی یک طرفه پرداخته می‌شود و توضیحات مربوط به بخش‌های اصلی کدهای مربوط به این عملیات‌ها نیز ارائه خواهد شد.

۴-۱۸-۱- عملیات ایجاد لیست پیوندی یک طرفه

لیست پیوندی یک طرفه از جمله ساختارهای داده‌ای بنیادی در علوم کامپیوتر است که به دلیل طراحی ساده و قابلیت‌های مؤثر در مدیریت داده‌ها، در بسیاری از پیاده‌سازی‌های نرم‌افزاری استفاده می‌شود. این ساختار داده‌ای شامل مجموعه‌ای از گره‌ها است که هر گره شامل داده و اشاره‌گری به گره بعدی در لیست است. هدف این بخش از پژوهش، پیاده‌سازی و مصورسازی فرآیند ایجاد لیست پیوندی یک طرفه با استفاده از کدهای لاتک می‌باشد. این فرآیند شامل افزودن گره‌ها به لیست و نمایش گرافیکی مراحل مختلف آن است.

در این قسمت، دو کلاس اصلی برای پیاده‌سازی لیست پیوندی یک طرفه تعریف شده‌اند. کلاس Node نماینده هر گره در لیست است و شامل دو ویژگی کلیدی است value که مقدار داده گره را نشان می‌دهد و next که اشاره‌گر به گره بعدی را نگه‌داری می‌کند. کلاس SingleLinkedList وظیفه مدیریت لیست را بر عهده دارد و متدهای کلیدی آن شامل __init__ برای ایجاد لیست خالی، append(value) برای افزودن یک گره جدید به انتهای لیست، و generate_latex_for_state() برای تولید کد لاتک به منظور نمایش حالت فعلی لیست می‌باشد. همچنین، متد generate_combined_latex(steps) برای ترکیب کدهای لاتک مراحل مختلف و تولید یک سند لاتک کامل به کار می‌رود.

در این کد، کاربر با وارد کردن مقادیر عددی به عنوان ورودی، گره‌های جدیدی به لیست اضافه می‌کند. زمانی که کاربر عبارت done را وارد می‌کند، عملیات ورودی پایان می‌یابد و کد لاتک برای نمایش مراحل مختلف لیست تولید می‌شود.

برای نمایش گرافیکی لیست پیوندی یک طرفه، هر گره در لیست با استفاده از دستور node\ در لاتک به صورت بلوک مستطیلی نمایش داده می‌شود که شامل مقدار گره است. اتصال گره‌ها با استفاده از فلش‌ها به یکدیگر به نمایش درمی‌آید که نشان‌دهنده اشاره‌گر next از گره جاری به گره بعدی است. در هر مرحله از افزودن گره‌ها، یک تصویر جداگانه ایجاد می‌شود که وضعیت فعلی لیست را نمایش می‌دهد.

این کد به طور ویژه برای مصورسازی فرآیند ایجاد لیست پیوندی یک طرفه طراحی شده است و به درک بهتر ساختار داده‌ها و الگوریتم‌ها کمک می‌کند. با استفاده از لاتک برای تولید نمایش‌های گرافیکی، این کد امکان تحلیل و نمایش بصری مراحل مختلف لیست پیوندی را فراهم می‌آورد و می‌تواند به عنوان ابزاری آموزشی و تحقیقاتی در زمینه ساختار داده‌ها مورد استفاده قرار گیرد.

۴-۱۸-۲- عملیات حذف در لیست پیوندی یک طرفه

عملیات حذف یکی از عملیات‌های کلیدی و کاربردی در ساختار داده‌ای لیست پیوندی یک طرفه به شمار می‌آید. هدف از پیاده‌سازی این قسمت، مصورسازی فرآیند حذف در لیست پیوندی یک طرفه با استفاده از لاتک است تا به‌طور گرافیکی مراحل مختلف این عملیات را نمایش دهد.

برای پیاده‌سازی لیست پیوندی یک طرفه، دو کلاس اصلی تعریف شده‌اند. کلاس Node نماینده هر گره در لیست است و شامل دو ویژگی کلیدی value برای نگهداری مقدار گره و next برای اشاره به گره بعدی می‌باشد. کلاس SingleLinkedList وظیفه مدیریت کل لیست را بر عهده دارد و شامل متدهایی است که امکان انجام عملیات مختلف حذف را فراهم می‌آورد. این متدها عبارتند از __init__، برای مقداردهی اولیه لیست به صورت خالی، append(value) برای افزودن یک گره جدید به انتهای لیست، delete_at_index(index) برای حذف گره‌ای در موقعیت مشخص، delete_by_value(value) برای حذف

گره‌ای با مقدار مشخص، () generate_latex_for_state برای تولید کد لاتک به منظور نمایش وضعیت فعلی لیست و generate_combined_latex(before_code, after_code) برای ترکیب کدهای لاتک قبل و بعد از حذف و ایجاد یک سند لاتک نهایی.

در این کد، کاربر می‌تواند یک گره را بر اساس موقعیت یا مقدار از لیست حذف کند. عملیات حذف با وارد کردن مقدار delete_index برای حذف بر اساس موقعیت و delete_value برای حذف بر اساس مقدار آغاز می‌شود. پس از تعیین نوع عملیات، کاربر موقعیت(ایندکس) یا مقدار مورد نظر برای حذف را وارد می‌کند. برای نمایش گرافیکی عملیات حذف، مراحل نام برده‌شده به ترتیب انجام می‌شود: نمایش گره‌ها قبل از حذف به‌صورت مستطیل‌هایی که شامل مقدار گره هستند، نمایش اتصال گره‌ها با استفاده از فلش‌هایی که نشان‌دهنده اشاره‌گر next است، نمایش گره‌ها بعد از حذف و به‌روزرسانی وضعیت جدید لیست به‌صورت تصویری، و مقایسه قبل و بعد از حذف با ایجاد کد نهایی لاتک که شامل دو بخش برای نمایش لیست قبل و بعد از حذف است و مقایسه‌ای جامع بین وضعیت‌های مختلف لیست را فراهم می‌آورد.

۴-۱۸-۳- عملیات درج در لیست پیوندی یک طرفه

در این قسمت، هدف اصلی مصورسازی عملیات درج یک گره جدید در لیست پیوندی یک طرفه با استفاده از لاتک است. لیست پیوندی یک طرفه ساختاری داده‌ای است که در آن هر گره به گره بعدی اشاره می‌کند و عملیات درج به ما این امکان را می‌دهد که گره‌ها را در موقعیت‌های مختلفی از لیست قرار دهیم. این مصورسازی به طور گرافیکی کمک می‌کند تا فرآیند درج در لیست پیوندی به وضوح نمایش داده شود.

عملیات درج در لیست پیوندی یک طرفه شامل دو کلاس اصلی است که برای پیاده‌سازی و مصورسازی لیست پیوندی یک طرفه طراحی شده‌اند. کلاس Node نماینده هر گره در لیست است و شامل دو ویژگی اصلی value برای نگهداری مقدار گره و next برای اشاره به گره بعدی می‌باشد. کلاس SingleLinkedList مدیریت کلی لیست را بر عهده دارد و شامل متدهایی است که عملیات درج را تسهیل می‌کند. این متدها شامل __init__ برای مقداردهی اولیه لیست به صورت خالی، append(value) برای افزودن گره جدید به انتهای لیست، insert_at_index(index,value) برای درج گره جدید در موقعیت مشخص، generate_latex_for_state برای تولید کد لاتک به منظور نمایش وضعیت فعلی لیست و generate_combined_latex(before_code, after_code) برای ترکیب کدهای لاتک قبل و بعد از درج و تولید یک سند لاتک نهایی هستند.

فرآیند درج گره جدید در لیست پیوندی شامل چند مرحله اصلی است. ابتدا، کاربر مقدار و موقعیت ایندکس گره جدید را وارد می‌کند. سپس، گره جدید در موقعیت مشخص شده به لیست اضافه می‌شود. مراحل بعدی شامل مصورسازی وضعیت لیست قبل و بعد از درج به‌صورت گرافیکی است. در سند لاتک برای نمایش

گرافیکی لیست پیوندی، هر گره به‌صورت مستطیل‌هایی که مقدار گره را نشان می‌دهند، نمایش داده می‌شود و گره‌ها با استفاده از فلش‌هایی که نمایانگر اشاره‌گر next هستند، به یکدیگر متصل می‌شوند. کد نهایی لاتک شامل دو بخش است؛ یکی برای نمایش لیست قبل از درج و دیگری برای نمایش لیست بعد از درج، که مقایسه‌ای دقیق بین وضعیت‌های مختلف لیست فراهم می‌آورد.

خروجی نهایی این کد در فایل Single_linked_list_combined.tex ذخیره می‌شود و با اجرای این فایل، مستند گرافیکی حاصل از عملیات درج قابل مشاهده است. این کد به‌ویژه در مستندات آموزشی و علمی برای مصورسازی عملیات درج در لیست پیوندی یک طرفه کاربرد دارد و به دانشجویان کمک می‌کند تا فرآیند درج گره‌ها در ساختار داده‌ای لیست پیوندی یک طرفه را به‌خوبی درک کنند و تغییرات در لیست را به وضوح مشاهده نمایند.

۴-۱۸-۴- عملیات جست‌وجو در لیست پیوندی یک طرفه

این قسمت به عملیات جست‌وجو در لیست پیوندی یک طرفه اختصاص دارد و با استفاده از لاتک، هر مرحله از این عملیات به‌طور گرافیکی نمایش داده می‌شود. هدف از این مصورسازی، ارائه فرآیند جست‌وجو به‌صورت بصری است تا کاربران بتوانند به‌راحتی مراحل مختلف جست‌وجو را درک کنند. در این نمایش، گره‌هایی که در حال جست‌وجو هستند با رنگ قرمز و گره‌ای که مقدار مورد نظر در آن یافت می‌شود با رنگ سبز مشخص می‌شود.

کد این قسمت شامل دو کلاس اصلی است که برای پیاده‌سازی و مصورسازی لیست پیوندی یک طرفه طراحی شده‌اند. کلاس Node نماینده یک گره در لیست پیوندی است و دارای دو بخش value برای نگهداری مقدار گره و next برای اشاره به گره بعدی می‌باشد. کلاس SingleLinkedList مسئول مدیریت کلی لیست و عملیات جست‌وجو است و شامل متدهایی است که فرآیند جست‌وجو را تسهیل می‌کند. این متدها شامل __init__ برای مقداردهی اولیه لیست به صورت خالی، append(value) برای افزودن یک گره جدید به انتهای لیست، generate_latex_for_state(highlight=None, highlight_color="red") برای تولید کد لاتک به منظور نمایش وضعیت فعلی لیست با برجسته کردن گره‌های مورد نظر، generate_combined_latex(steps) برای ترکیب تمام مراحل جست‌وجو و تولید یک سند لاتک نهایی، و search(value) برای جست‌وجوی یک مقدار خاص در لیست و تولید مراحل جست‌وجو برای مصورسازی هستند.

فرآیند جست‌وجو در لیست پیوندی شامل چند مرحله اصلی است. ابتدا، کاربر مقدار مورد نظر برای جست‌وجو در لیست را وارد می‌کند. سپس، عملیات جست‌وجو به‌صورت گره به گره انجام می‌شود تا زمانی که گره‌ای با مقدار مورد نظر پیدا شود یا لیست به پایان برسد. در هر مرحله از جست‌وجو، گره‌ای که در حال بررسی

است با رنگ قرمز برجسته می‌شود و اگر گره‌ای با مقدار مورد نظر پیدا شود، آن گره با رنگ سبز نمایش داده می‌شود. نهایتاً، تمام مراحل جست‌وجو در یک سند لاتک ترکیب شده و خروجی نهایی ساخته می‌شود.

خروجی نهایی کد در فایل Single_linked_list_steps.tex ذخیره می‌شود و با اجرای این فایل، کاربر می‌تواند مراحل جست‌وجو در لیست پیوندی یک طرفه را به‌صورت گرافیکی مشاهده کند. این بخش به درک بهتر فرآیند جست‌وجو و نحوه تعامل با ساختار داده‌ای لیست پیوندی کمک می‌کند.

۴-۱۹- عملیات‌های مختلف در لیست پیوندی دو طرفه

لیست پیوندی دو طرفه یکی از ساختارهای داده‌ای پیچیده است که به هر گره این امکان را می‌دهد که به گره قبلی و گره بعدی اشاره کند. این ویژگی باعث می‌شود که عملیات‌های مختلف بر روی این ساختار داده‌ای مانند ایجاد، حذف، درج و جست‌وجو پیچیده‌تر و متنوع‌تر از لیست‌های پیوندی یک طرفه باشند. پژوهش حاضر به بررسی و مصورسازی این عملیات‌ها با استفاده از لاتک می‌پردازد تا درک بهتری از نحوه عملکرد لیست‌های پیوندی دو طرفه فراهم کند.

۴-۱۹-۱- ایجاد لیست پیوندی دو طرفه

لیست پیوندی دو طرفه یکی از ساختارهای داده‌ای قدرتمند و انعطاف‌پذیر است که در آن هر گره علاوه بر اشاره‌گر به گره بعدی، به گره قبلی نیز اشاره می‌کند. این ویژگی دو طرفه، امکان حرکت و دسترسی به گره‌ها را از هر دو سمت جلو و عقب فراهم می‌آورد و کاربردهای وسیعی را در مدیریت داده‌ها به وجود می‌آورد. هدف این پژوهش، پیاده‌سازی یک لیست پیوندی دو طرفه و نمایش تصویری مراحل مختلف ایجاد این ساختار داده است تا کاربران بتوانند فرآیند ساخت و تغییرات آن را به‌خوبی درک کنند.

در این قسمت، دو کلاس اصلی تعریف شده است. کلاس Node که نمایانگر یک گره در لیست پیوندی دو طرفه است و شامل سه ویژگی اصلی می‌باشد: مقدار (value)، اشاره‌گر به گره بعدی (next) و اشاره‌گر به گره قبلی (prev). این طراحی اجازه می‌دهد تا هر گره به‌طور همزمان به گره قبلی و گره بعدی در لیست اشاره کند. کلاس DoublyLinkedList نیز مسئول مدیریت کل لیست پیوندی دو طرفه است. این کلاس شامل متدهایی برای افزودن گره‌های جدید به انتهای لیست (append) و تولید کد لاتک برای نمایش وضعیت فعلی لیست (generate_latex_for_state) می‌باشد.

در این قسمت، کاربر می‌تواند مقادیر دلخواه را برای افزودن به لیست وارد کند. هر مقدار جدید به انتهای لیست پیوندی دو طرفه افزوده می‌شود و با وارد کردن دستور done، فرایند ورود مقادیر به پایان می‌رسد. این تعامل به کاربر امکان می‌دهد تا به‌راحتی لیست را با داده‌های مورد نظر خود پر کند و فرآیند ایجاد لیست را تحت نظر داشته باشد.

برای نمایش تصویری مراحل ایجاد لیست، کد لاتک تولید می‌شود که با استفاده از پکیج tikz، گره‌ها و

اتصالات آن‌ها به‌صورت گرافیکی ترسیم می‌شود. هر مرحله از افزودن یک گره به لیست با تولید کد لاتک جدید مستند می‌شود و در نهایت، یک فایل لاتک نهایی ایجاد می‌شود که شامل تمامی مراحل افزوده شدن گره‌ها به لیست پیوندی دو طرفه است.

این کد به‌ویژه برای آموزش و درک ساختار داده‌های پیوندی دو طرفه طراحی شده است. با استفاده از این ابزار، کاربران می‌توانند فرآیند ایجاد یک لیست پیوندی دو طرفه را به‌صورت بصری مشاهده کنند، که به یادگیری بهتر و عمق درک مفاهیم ساختار داده‌ها کمک می‌کند. این پژوهش می‌تواند به‌عنوان یک ابزار آموزشی موثر برای تدریس مفاهیم ساختار داده‌ها و الگوریتم‌ها در دانشگاه‌ها و دوره‌های آموزشی کاربرد داشته باشد.

۴-۱۹-۲- حذف در لیست پیوندی دو طرفه

لیست پیوندی دو طرفه یکی از ساختارهای داده‌ای پیشرفته است که به هر گره اجازه می‌دهد تا به گره قبلی و بعدی خود اشاره کند. این ویژگی دو طرفه، امکان جابجایی و حرکت در هر دو جهت لیست را فراهم می‌آورد و به مدیریت داده‌ها انعطاف بیشتری می‌بخشد. عملیات حذف در لیست پیوندی دو طرفه شامل حذف یک گره خاص بر اساس اندیس یا مقدار آن است. پژوهش حاضر به‌منظور مصورسازی فرآیند حذف گره‌ها از این لیست طراحی شده تا کاربران بتوانند به‌طور بصری مراحل و تغییرات ناشی از عملیات حذف را مشاهده کنند.

در این پژوهش، دو کلاس اصلی وجود دارد. کلاس Node که نمایانگر یک گره در لیست پیوندی دو طرفه است و شامل سه بخش اصلی است: مقدار (value)، اشاره‌گر به گره بعدی (next) و اشاره‌گر به گره قبلی (prev). کلاس DoublyLinkedList برای مدیریت عملیات مختلف روی لیست پیوندی دو طرفه به‌کار می‌رود و متدهایی برای افزودن گره‌ها (append) و حذف آن‌ها (delete_at_index و delete_by_value) در خود دارد. علاوه بر این، متد generate_latex_for_state برای تولید کد لاتک جهت نمایش وضعیت فعلی لیست و متد generate_combined_latex برای تولید کد لاتک که وضعیت لیست قبل و بعد از عملیات حذف را مقایسه می‌کند، تعریف شده‌اند.

کاربر می‌تواند یکی از دو نوع عملیات حذف را انتخاب کند؛ نوع اول، حذف بر اساس اندیس که در این حالت، کاربر اندیس گره‌ای که می‌خواهد حذف کند را وارد می‌کند و نوع دوم حذف بر اساس مقدار است؛ در این حالت، کاربر مقدار گره‌ای که قصد حذف آن را دارد وارد می‌کند.

در این پیاده‌سازی، وضعیت لیست پیوندی دو طرفه قبل و بعد از انجام عملیات حذف به‌صورت تصویری نمایش داده می‌شود. این تصاویر با استفاده از کد لاتک و کتابخانه tikz تولید می‌شوند. ابتدا وضعیت لیست قبل از حذف گره ترسیم می‌شود و سپس پس از اجرای عملیات حذف، وضعیت جدید لیست نمایش داده می‌شود. این دو وضعیت در یک فایل لاتک ترکیب می‌شوند تا تغییرات ایجاد شده در لیست به‌وضوح قابل

مشاهده باشد.

پیاده‌سازی حذف در لیست پیوندی دو طرفه و نمایش تصویری آن، به کاربران این امکان را می‌دهد تا به‌طور تعاملی عملیات حذف گره‌ها را انجام دهند و تأثیر این عملیات را بر ساختار لیست مشاهده کنند.

۴-۱۹-۳- درج در لیست پیوندی دو طرفه

لیست پیوندی دو طرفه یکی از ساختارهای داده‌ای است که در آن هر گره به دو گره دیگر، یعنی گره قبلی و گره بعدی، اشاره می‌کند. این ویژگی به کاربران اجازه می‌دهد تا به راحتی در هر دو جهت لیست حرکت کنند و عملیات‌های مختلفی را به‌طور مؤثر انجام دهند. در این پژوهش، هدف بررسی و مصورسازی عملیات درج گره‌ها در لیست پیوندی دو طرفه است. این فرآیند به کاربران کمک می‌کند تا نحوه افزودن گره‌های جدید به لیست و تغییرات ناشی از این عملیات را به‌طور واضح مشاهده کنند.

در این پژوهش، دو کلاس اصلی مورد استفاده قرار گرفته است. کلاس Node نمایانگر یک گره در لیست پیوندی دو طرفه است که شامل مقدار (value)، اشاره‌گر به گره بعدی (next) و اشاره‌گر به گره قبلی (prev) می‌باشد. کلاس DoublyLinkedList مسئول مدیریت عملیات‌های مختلف روی لیست پیوندی دو طرفه است و متدهایی برای افزودن گره‌ها (append) و درج گره‌ها در اندیس مشخص (insert_at_index) فراهم می‌کند. برای مصورسازی وضعیت لیست پیوندی دو طرفه، متد generate_latex_for_state استفاده می‌شود که وضعیت فعلی لیست را به کد لاتک تبدیل می‌کند. همچنین، متد generate_combined_latex کد لاتک را برای نمایش وضعیت لیست قبل و بعد از عملیات درج ایجاد می‌کند.

کاربر می‌تواند با وارد کردن مقدار گره جدید و اندیس مورد نظر برای درج، عملیات درج را انجام دهد. این اطلاعات به برنامه اجازه می‌دهد تا گره جدید را در موقعیت مشخص شده در لیست پیوندی دو طرفه اضافه کند. پس از انجام عملیات درج، وضعیت لیست پیوندی دو طرفه قبل و بعد از این عملیات به‌صورت تصویری نمایش داده می‌شود.

پژوهش به‌منظور مصورسازی تغییرات لیست پیوندی دو طرفه از کد لاتک و کتابخانه tikz استفاده می‌کند. ابتدا وضعیت لیست پیوندی دو طرفه قبل از درج گره ترسیم می‌شود و سپس پس از انجام عملیات درج، وضعیت جدید لیست نمایش داده می‌شود. این دو وضعیت در یک فایل لاتک ترکیب می‌شوند تا تغییرات به‌طور واضح و قابل فهم ارائه شود.

این کد به‌منظور مصورسازی و درک بهتر عملیات درج در لیست‌های پیوندی دو طرفه طراحی شده است. کاربران می‌توانند به‌صورت تعاملی عملیات درج گره جدید را انجام داده و تأثیر آن بر ساختار لیست را مشاهده کنند. این پژوهش می‌تواند ابزار مفیدی در آموزش ساختار داده‌ها و الگوریتم‌ها باشد و به دانشجویان کمک کند تا مفاهیم مرتبط با لیست‌های پیوندی دو طرفه را به‌طور عملی تجربه کنند.

۴-۱۹-۴- جست‌وجو در لیست پیوندی دو طرفه

در این بخش، پیاده‌سازی جست‌وجو در لیست پیوندی دو طرفه و مصورسازی این عملیات به صورت مرحله‌به‌مرحله بررسی می‌شود. با استفاده از لاتک و کتابخانه tikz، عملیات جست‌وجو به‌طور تصویری نمایش داده می‌شود تا فرآیند جست‌وجو به‌وضوح قابل درک باشد.

دو کلاس اصلی برای پیاده‌سازی و مصورسازی لیست پیوندی دو طرفه استفاده شده است. کلاس Node نمایانگر یک گره در لیست پیوندی دو طرفه است و شامل ویژگی‌های value، next و prev می‌باشد. کلاس DoublyLinkedList مسئول مدیریت عملیات مختلف مانند افزودن و جست‌وجو در لیست پیوندی دو طرفه است. از جمله متدهای کلیدی این کلاس می‌توان به generate_latex_for_state اشاره کرد که وضعیت فعلی لیست را به کد لاتک تبدیل می‌کند و generate_combined_latex که کد لاتک برای نمایش وضعیت لیست در هر مرحله از جست‌وجو را تولید می‌کند.

فرآیند جست‌وجو به‌صورت مرحله‌به‌مرحله و به‌صورت تصویری نمایش داده می‌شود. در هر مرحله، گره‌ای که مورد بررسی قرار می‌گیرد با رنگ خاصی مشخص می‌شود و تمام مراحل جست‌وجو در یک فایل لاتک ترکیب می‌شود تا فرآیند به‌طور کامل و گرافیکی نمایش داده شود. این کد به کاربران این امکان را می‌دهد که عملیات جست‌وجو در لیست پیوندی دو طرفه را به‌طور تصویری مشاهده کنند و نحوه عملکرد جست‌وجو را به‌طور عملی درک کنند.

۴-۲۰- درخت دودویی

در علوم رایانه، یک درخت دودویی یک ساختمان داده‌ی درخت است که در آن هر گره حداکثر دو گره فرزند دارد که فرزندان راست و چپ نامیده می‌شوند. در درخت دودویی، درجه هر گره حداکثر می‌تواند دو باشد. هدف این بخش از پژوهش، مصورسازی عملیات ایجاد، حذف و درج در درخت دودویی است.

۴-۲۰-۱- عملیات ایجاد درخت دودویی

عملیات ایجاد درخت دودویی با هدف ایجاد و نمایش گرافیکی درخت‌های دودویی با استفاده از لاتک و بسته TikZ طراحی شده است. هدف اصلی از اجرای این پژوهش، ساخت یک درخت دودویی از طریق ورودی‌های کاربر و تولید مستندات لاتک برای نمایش بصری این درخت است. این روش به کاربران کمک می‌کند تا به‌راحتی ساختار درخت دودویی را درک کرده و تغییرات آن را به‌صورت مرحله‌ای مشاهده کنند.

در این قسمت، چندین بخش کلیدی وجود دارد که به‌طور جامع به شرح آن‌ها پرداخته می‌شود. اولین بخش، کلاس BinaryTreeNode است که نمایانگر یک گره در درخت دودویی است. هر گره شامل نام یا مقدار و اشاره‌گرهایی به فرزندان چپ و راست خود است و یک تابع سازنده برای ایجاد و مقداردهی اولیه به این گره‌ها استفاده می‌شود. این ساختار پایه‌ای، اساس ایجاد و مدیریت درخت دودویی را فراهم می‌کند.

بخش دوم پژوهش، تابع insert_into_bst است که برای درج مقادیر در یک درخت جست‌وجوی دودویی (BST) طراحی شده است. این تابع در شرایطی که درخت خالی باشد، یک گره جدید به‌عنوان ریشه درخت اضافه می‌کند و مقادیر جدید بر اساس مقایسه با گره‌های موجود، در سمت چپ یا راست قرار می‌گیرند. همچنین، تابع add_node برای افزودن گره‌های جدید به درخت دودویی موجود، طراحی شده است. این تابع به کاربر این امکان را می‌دهد که با مشخص کردن مقدار والد، مقدار گره جدید و موقعیت آن (چپ یا راست)، به سادگی درخت دودویی خود را توسعه دهد.

یکی از مهم‌ترین بخش‌های این پژوهش، تابع generate_latex_binary_tree است که وظیفه تولید کد لاتک برای رسم درخت دودویی را بر عهده دارد. این تابع به‌صورت بازگشتی ساختار درخت را به فرمت TikZ تبدیل می‌کند و نمایش گرافیکی دقیقی از آن ارائه می‌دهد. علاوه بر این، تابع generate_latex_document برای تولید مستندات لاتک که شامل درخت دودویی نهایی و توضیحات مربوط به آن است، طراحی شده است. این مستندات به کاربران اجازه می‌دهد تا درخت دودویی خود را به‌صورت کامل مشاهده و تحلیل کنند.

در ادامه، تابع generate_latex_document_steps نیز برای تولید مستنداتی طراحی شده است که روند ساخت درخت دودویی را به‌صورت مرحله‌ای نشان می‌دهد. این مستندات شامل تصاویر مرحله‌ای از تغییرات درخت است و به کاربران کمک می‌کند تا هر مرحله از فرآیند ساخت درخت را به‌صورت بصری دنبال کنند و از تغییرات آن آگاه شوند.

۴-۲۰-۲- عملیات حذف در درخت دودویی

در این بخش از پژوهش، هدف اصلی نمایش و تحلیل فرآیند حذف گره‌ها در درخت دودویی با استفاده از لاتک و بسته TikZ است. این بخش به گونه‌ای طراحی شده که الگوریتم حذف گره از درخت دودویی را به صورت بصری و گرافیکی نمایش دهد. استفاده از این روش به کاربران امکان می‌دهد تا مراحل مختلف حذف گره را به طور واضح و دقیق مشاهده کرده و درک عمیق‌تری از فرآیند حذف در درخت‌های دودویی پیدا کنند.

این قسمت شامل چندین بخش کلیدی است که هر یک نقش مهمی در نمایش و تحلیل فرآیند حذف گره دارند. ابتدا، کلاس Node که نمایانگر یک گره در درخت دودویی است، معرفی می‌شود. هر گره شامل داده‌هایی است که در آن ذخیره می‌شود و اشاره‌گرهایی به فرزندان چپ و راست خود دارد. تابع سازنده این کلاس، برای ایجاد و مقداردهی اولیه به گره‌ها استفاده می‌شود و پایه‌گذار ساختار درخت دودویی است.

یکی دیگر از بخش‌های اصلی پژوهش، تابع inorder است که برای دریافت ترتیب میانی (inorder) درخت استفاده می‌شود. این تابع به صورت بازگشتی عمل می‌کند و داده‌های درخت را در ترتیب میانی جمع‌آوری می‌کند که این ترتیب یکی از پایه‌های تحلیل ساختار درخت‌های دودویی است.

تابع generate_latex_binary_tree برای تولید کد لاتک طراحی شده که با استفاده از بسته TikZ، درخت دودویی را به صورت گرافیکی رسم می‌کند. این تابع به صورت بازگشتی عمل می‌کند تا کل ساختار درخت را به قالب TikZ تبدیل کند و به نمایش بگذارد. نمایش گرافیکی دقیق درخت در این مرحله، به فهم بهتر ساختار درخت کمک شایانی می‌کند.

تابع generate_latex_document_steps مستندات لاتک را تولید می‌کند که مراحل مختلف حذف گره از درخت دودویی را به تصویر می‌کشد. این مستندات شامل تصاویر مرحله‌ای از تغییرات درخت و توضیحات مربوط به الگوریتم حذف است، که به کاربر این امکان را می‌دهد تا هر مرحله از فرآیند حذف را به صورت گرافیکی و همراه با توضیحات مشاهده کند.

تابع delete_deepest وظیفه حذف عمیق‌ترین گره در درخت دودویی را بر عهده دارد. این تابع به دنبال گره‌ای که در پایین‌ترین سطح درخت قرار دارد، می‌گردد و آن را حذف می‌کند. سپس، تابع deletion برای حذف یک گره خاص از درخت دودویی استفاده می‌شود. این تابع ابتدا گره مورد نظر را پیدا کرده و داده‌های آن را با داده‌های عمیق‌ترین گره جایگزین می‌کند و سپس عمیق‌ترین گره را حذف می‌کند. این فرآیند باعث می‌شود که ساختار درخت پس از حذف گره، همچنان معتبر باقی بماند.

در نهایت، تابع main که تابع اصلی پژوهش است، به ایجاد درخت دودویی نمونه، دریافت ورودی از کاربر برای حذف گره‌ها، و تولید مستندات لاتک با تصاویر مرحله‌ای از حذف گره‌ها می‌پردازد. این تابع همچنین فایل لاتک نهایی را تولید و ذخیره می‌کند تا کاربران بتوانند نتایج را به صورت مستند و گرافیکی مشاهده کنند.

مستندات تولید شده شامل توضیحات دقیق در مورد الگوریتم حذف و تصاویر مرحله‌ای از تغییرات درخت است، که این امر به کاربران کمک می‌کند تا فرآیند حذف گره‌ها را به راحتی مشاهده و تحلیل کرده و درک بهتری از عملکرد درخت‌های دودویی به دست آورند.

۴-۲۰-۳- عملیات درج در درخت دودویی

در این قسمت، فرآیند درج گره‌های جدید به روش سطحی (level-order) در درخت دودویی به صورت بصری و گرافیکی نمایش داده شده است. هدف اصلی از طراحی این پژوهش، ایجاد ابزاری است که به کاربران امکان دهد تغییرات درخت دودویی را در هر مرحله از درج گره‌ها مشاهده و تحلیل کنند. این ابزار با استفاده از کدهای لاتک و بسته TikZ، ساختار درخت را به صورت تدریجی و مرحله‌به‌مرحله نمایش می‌دهد و به کاربران کمک می‌کند تا مفاهیم مرتبط با درج گره‌ها را به صورت بصری درک کنند.

یکی از بخش‌های اصلی پژوهش، کلاس BinaryTreeNode است که نمایانگر یک گره در درخت دودویی می‌باشد. هر گره در این کلاس شامل داده‌های خود و اشاره‌گرهایی به فرزندان چپ و راست خود است. متد __init__ این کلاس برای ایجاد و مقداردهی اولیه به گره‌ها استفاده می‌شود، که اساس ساختار درخت را شکل

می‌دهد.

تابع insert_level_order، وظیفه درج گره جدید به روش سطحی در درخت دودویی را بر عهده دارد. این تابع به جست‌وجوی مکان مناسب برای درج گره جدید پرداخته و پس از یافتن موقعیت مناسب، گره جدید را به درخت اضافه می‌کند. بعد از هر عملیات درج، وضعیت درخت به صورت کد لاتک ذخیره می‌شود که می‌توان آن را برای نمایش گرافیکی درخت استفاده کرد.

در ادامه، تابع generate_latex_binary_tree برای تولید کد لاتک به منظور رسم درخت دودویی طراحی شده است. این تابع به صورت بازگشتی عمل کرده و درخت را به قالب TikZ تبدیل می‌کند تا ساختار آن به صورت گرافیکی نمایش داده شود. این بخش از پژوهش به کاربران امکان می‌دهد تا به صورت تصویری تغییرات ایجاد شده در ساختار درخت را مشاهده کنند.

تابع generate_latex_document_steps وظیفه تولید مستندات لاتک را دارد که مراحل مختلف درج گره‌ها در درخت دودویی را نشان می‌دهد. این مستندات شامل تصاویر مرحله‌ای از تغییرات درخت و توضیحات مربوط به الگوریتم درج است که به کاربران کمک می‌کند تا هر مرحله از فرآیند را به صورت بصری دنبال کنند.

در نهایت، تابع main که به عنوان تابع اصلی پژوهش عمل می‌کند، به ایجاد درخت دودویی اولیه، دریافت ورودی از کاربر برای درج گره‌های جدید و تولید مستندات لاتک می‌پردازد. این تابع همچنین فایل لاتک نهایی را تولید و ذخیره می‌کند که شامل تصاویر مرحله‌ای و توضیحات مرتبط با فرآیند درج گره‌ها است.

این ابزار با دریافت ورودی‌های لازم از کاربر، امکان درج گره‌های جدید را فراهم می‌کند. پس از هر عملیات درج، وضعیت جدید درخت به صورت گرافیکی نمایش داده می‌شود و هر مرحله از این فرآیند در یک تصویر جداگانه ثبت می‌گردد. این تصاویر به همراه توضیحات مربوطه، در مستندات نهایی لاتک ذخیره می‌شوند که می‌تواند برای تحلیل و ارائه‌ی تغییرات درخت مفید باشد.

۴-۲۰-۴- عملیات جست‌وجو در درخت دودویی

در این قسمت، فرآیند جست‌وجو در درخت دودویی به صورت بصری و با استفاده از ابزارهای لاتک و TikZ نمایش داده شده است. درخت دودویی یکی از ساختارهای داده‌ای پرکاربرد در علم رایانه است و الگوریتم‌های جست‌وجو در این درخت‌ها به دنبال یافتن یک مقدار مشخص در میان گره‌ها هستند. هدف از این پژوهش، ایجاد یک ابزار آموزشی و تحلیلی برای نمایش گرافیکی فرآیند جست‌وجو در درخت دودویی است تا کاربران بتوانند تغییرات و نحوه عملکرد الگوریتم را در هر مرحله مشاهده و تحلیل کنند.

این پژوهش شامل چندین بخش اصلی است که هر کدام نقش مهمی در اجرای الگوریتم جست‌وجو و نمایش نتایج آن به صورت گرافیکی دارند. یکی از این بخش‌ها، کلاس BinaryTreeNode است که نمایانگر یک گره در درخت دودویی می‌باشد. این کلاس شامل مقدار (value) و اشاره‌گرهایی به گره‌های چپ و راست

خود است و با استفاده از متد __init__، گره‌ها ایجاد و مقداردهی اولیه می‌شوند.

تابع search_binary_tree مسئولیت جست‌وجوی یک مقدار مشخص در درخت دودویی را بر عهده دارد. این تابع به صورت بازگشتی عمل می‌کند و هر مرحله از جست‌وجو را در قالب کد لاتک ذخیره می‌کند تا وضعیت درخت در هر مرحله از جست‌وجو به خوبی قابل مشاهده باشد. این تابع با بررسی گره فعلی و مقایسه آن با مقدار هدف، به جست‌وجوی مقدار مورد نظر در زیرشاخه‌های چپ یا راست ادامه می‌دهد.

برای نمایش گرافیکی درخت و وضعیت آن در طول فرآیند جست‌وجو، تابع generate_latex_binary_tree طراحی شده است. این تابع با استفاده از لاتک و بسته TikZ، درخت دودویی را رسم می‌کند و گره‌هایی را که در فرآیند جست‌وجو مورد بررسی قرار گرفته‌اند، با رنگی خاص مشخص می‌کند. این ویژگی به کاربران امکان می‌دهد تا به صورت بصری تغییرات درخت را در طول جست‌وجو مشاهده کنند و عملکرد الگوریتم را بهتر درک کنند.

تابع generate_latex_document_steps نیز کد لاتک را تولید می‌کند که شامل تصاویر مختلف از درخت در مراحل مختلف جست‌وجو است. این تصاویر به صورت گرافیکی تغییرات درخت را در هر مرحله از جست‌وجو نمایش می‌دهند و به همراه توضیحات مرتبط، در مستندات نهایی لاتک ذخیره می‌شوند. این بخش از پژوهش به ویژه برای تحلیل و ارائه نتایج جست‌وجو در قالب مستندات علمی و آموزشی بسیار مفید است.

در نهایت، تابع main به عنوان تابع اصلی پژوهش عمل می‌کند. این تابع با ایجاد درخت دودویی اولیه و دریافت ورودی از کاربر برای جست‌وجوی مقدار خاص، فرآیند جست‌وجو را اجرا کرده و نتایج آن را به صورت مستندات لاتک با تصاویر مرحله‌ای تولید و ذخیره می‌کند. کاربران می‌توانند مقدار هدف خود را وارد کرده و هر مرحله از جست‌وجو را به صورت تصویری مشاهده کنند. نمایش گرافیکی هر مرحله از جست‌وجو به کاربران کمک می‌کند تا عملکرد الگوریتم را بهتر درک کنند و بتوانند تحلیل‌های دقیقی از تغییرات درخت ارائه دهند.

۴-۲۰-۵- عملیات پیمایش عمقی در درخت دودویی

هدف اصلی این قسمت، مصورسازی فرآیند پیمایش عمقی در درخت دودویی با استفاده از زبان پایتون و تولید خروجی‌های گرافیکی به صورت کد لاتک بود. این پژوهش با تمرکز بر سه نوع پیمایش عمقی یعنی پیش‌سفری، میان‌سفری و پس‌سفری، توانسته است مراحل مختلف این پیمایش‌ها را به صورت تصویری نمایش دهد و درک بهتری از عملکرد این الگوریتم‌ها را فراهم کند. الگوریتم‌های پیمایش درخت دودویی یکی از مفاهیم بنیادی در علوم کامپیوتر محسوب می‌شوند و درک صحیح آن‌ها نقش مهمی در یادگیری و تحلیل داده‌های ساختاریافته دارد.

یکی از اجزای کلیدی این پژوهش، کلاس BinaryTreeNode است که ساختار گره‌های درخت دودویی را تعریف می‌کند. این کلاس شامل سه ویژگی اصلی است value برای نگهداری مقدار گره، left برای اشاره

به فرزند چپ و right برای اشاره به فرزند راست. با استفاده از این کلاس، ساختار درخت دودویی به خوبی مدل‌سازی شده و آماده برای پیمایش در مراحل مختلف است.

در ادامه، سه تابع اصلی برای اجرای انواع مختلف پیمایش عمقی درخت دودویی تعریف شده‌اندpre_order_traversal که گره‌ها را به ترتیب پیش‌سفری پیمایش می‌کند، in_order_traversal که ترتیب میان‌سفری را دنبال می‌کند و post_order_traversal که گره‌ها را به ترتیب پس‌سفری پیمایش می‌کند. این توابع با پیمایش گره‌های درخت، اطلاعات لازم را برای تولید خروجی گرافیکی فراهم می‌کنند.

برای نمایش گرافیکی درخت و مراحل مختلف پیمایش، تابع generate_latex_binary_tree مورد استفاده قرار گرفته است. این تابع با استفاده از کد لاتک و بسته TikZ، درخت دودویی را به صورت گرافیکی رسم می‌کند. در هر مرحله از پیمایش، گره‌هایی که در حال پردازش هستند، با رنگ آبی هایلایت می‌شوند تا به وضوح مشخص شود که کدام گره در آن لحظه مورد بررسی قرار گرفته است. این قابلیت به کاربران اجازه می‌دهد تا به صورت بصری مراحل پیچیده پیمایش درخت را دنبال کنند.

تابع generate_latex_document_steps نیز تمام مراحل پیمایش را به صورت یک سند لاتک سازماندهی می‌کند. این سند شامل تصاویر مختلفی از درخت در هر مرحله از پیمایش است که هر کدام با توضیحات مرتبط، به ترتیب در سند نهایی قرار می‌گیرند.

خروجی نهایی، الگوریتم‌های پیمایش عمقی درخت دودویی را به صورت بصری و قابل فهم نمایش می‌دهد، درنتیجه ابزاری قدرتمند برای آموزش و تحلیل این الگوریتم‌ها می‌باشد و می‌توانند در تسهیل درک مفاهیم پیچیده و مراحل اجرای این الگوریتم‌ها مؤثر باشد.

۴-۲۰-۶- عملیات پیمایش سطحی در درخت دودویی

هدف اصلی این قسمت، مصورسازی فرآیند پیمایش سطحی (BFS) در درخت دودویی و تولید خروجی‌های گرافیکی به صورت سند لاتک می‌باشد. الگوریتم پیمایش سطحی یکی از روش‌های اساسی در پردازش داده‌های ساختاریافته در قالب درخت است که گره‌ها را به ترتیب از بالا به پایین و از چپ به راست پیمایش می‌کند. این پژوهش توانسته است مراحل مختلف این پیمایش را به صورت بصری و گرافیکی نمایش دهد و به این ترتیب درک عمیق‌تری از این الگوریتم برای کاربران فراهم کند.

یکی از اجزای کلیدی این پژوهش، کلاس BinaryTreeNode است که ساختار گره‌های درخت دودویی را تعریف می‌کند. هر گره در این ساختار شامل یک مقدار (value) و دو اشاره‌گر به فرزندان چپ (left) و راست (right) است. با استفاده از این کلاس، کاربر می‌تواند یک درخت دودویی را به صورت دستی تعریف کند و سپس فرآیند پیمایش سطحی را روی آن اعمال کند.

تابع breadth_first_traversal به عنوان هسته اصلی پیمایش سطحی درخت عمل می‌کند. این تابع

گره‌ها را به ترتیب سطحی (از ریشه به پایین و از چپ به راست) پیمایش کرده و در هر مرحله گره فعلی را هایلایت می‌کند. این مرحله‌ها به صورت جداگانه ذخیره شده و در نهایت در خروجی لاتک نمایش داده می‌شوند.

برای تولید نمایش گرافیکی درخت در هر مرحله، از تابع generate_latex_binary_tree استفاده می‌شود. این تابع با تولید کد لاتک و استفاده از بسته TikZ، درخت دودویی را در هر مرحله از پیمایش رسم می‌کند. در این نمایش، گره‌ای که در حال پردازش است، با رنگ صورتی هایلایت می‌شود تا به وضوح نشان داده شود که کدام گره در آن لحظه در حال بررسی است.

تابع generate_latex_document_steps تمامی مراحل پیمایش را به صورت یک سند کامل لاتک سازماندهی کرده و خروجی نهایی را تولید می‌کند. این سند شامل تصاویر جداگآن‌های است که هر کدام نمایش‌دهنده یک مرحله از پیمایش سطحی درخت هستند. این خروجی نهایی به شکل یک فایل لاتک با نام binary_tree_bfs_steps.tex ذخیره می‌شود. در نتیجه، این پژوهش با موفقیت الگوریتم پیمایش سطحی (BFS) را در درخت دودویی مصورسازی کرده و خروجی گرافیکی آن را به صورت سندی دقیق و منظم ارائه داده است.

۴-۲۱- درخت جست‌وجوی دودویی

درخت دودویی جست‌وجو یک ساختار داده‌ای کلیدی در علوم کامپیوتر است که به منظور نگهداری و جست‌وجوی مؤثر مقادیر استفاده می‌شود. این درخت به گونه‌ای طراحی شده است که هر گره می‌تواند حداکثر دو فرزند داشته باشد و مقادیر در هر زیر درخت به ترتیب صعودی یا نزولی مرتب شده‌اند. هدف از این بخش، پیاده‌سازی عملیات های ایجاد درخت دودویی جست‌وجو، حذف، درج، جست‌وجو و پیمایش های عمقی و سطحی در درخت دودویی و ارائه نمایش گرافیکی آن‌ها با استفاده از لاتک و کتابخانه TikZ است تا فرآیند ساخت درخت و ساختار نهایی آن به صورت بصری و واضح نمایش داده شود.

۴-۲۱-۱- عملیات ایجاد درخت جست‌وجوی دودویی

قسمت پیاده‌سازی عملیات ایجاد درخت جست‌وجوی دودویی شامل بخش‌های اصلی زیر است. کلاس BinaryTreeNode برای تعریف گره‌های درخت جست‌وجوی دودویی طراحی شده و شامل ویژگی‌های name برای مقدار یا نام گره، left و right برای اشاره به فرزندان چپ و راست است. تابع insert_into_bst مسئول اضافه کردن گره‌های جدید به درخت دودویی جست‌وجو است. اگر درخت خالی باشد، این تابع یک گره جدید به عنوان ریشه ایجاد می‌کند؛ در غیر این صورت، به صورت بازگشتی عمل کرده و گره جدید را در موقعیت مناسب قرار می‌دهد. تابع add_node مشابه insert_into_bst عمل کرده و به صورت بازگشتی گره‌های جدید را در مکان مناسب خود در درخت قرار می‌دهد.

برای تولید نمایش گرافیکی درخت، تابع generate_latex_binary_tree استفاده می‌شود. این تابع با

استفاده از دستورات TikZ و به صورت بازگشتی درخت را پیمایش کرده و گرافیک مربوط به هر گره و ارتباطات آن با گره‌های فرزند را تولید می‌کند. همچنین، تابع generate_latex_document کل سند لاتک شامل درخت دودویی جست‌وجوی نهایی را ایجاد کرده و درخت نهایی را به صورت گرافیکی در سند قرار می‌دهد.

کاربر برای اجرای کد باید ورودی‌های لازم را ارائه دهد، از جمله انتخاب نوع نمایش (نهایی یا مراحل گام به گام) و ورود مقادیر عددی برای افزودن به درخت. مقادیر باید به صورت عددی وارد شوند و با تایپ done پایان یابند. در مرحله نهایی، سند لاتک شامل تصاویر گرافیکی از درخت دودویی جست‌وجو تولید می‌شود، در حالی که در حالت مراحل گام به گام، سند شامل تصاویری از درخت در هر مرحله از افزودن گره‌ها است.

نمایش بصری درخت‌ها کمک می‌کند تا نحوه ساخت و سازمان‌دهی درخت‌های دودویی جست‌وجو بهتر درک شود و ایجاد مستندات با تصاویر گرافیکی از درخت‌ها برای اهداف آموزشی و پژوهشی بسیار مفید است.

۴-۲۱-۲- عملیات حذف در درخت جست‌وجوی دودویی

هدف این بخش از پژوهش پیاده‌سازی درخت دودویی جست‌وجو و عملیات حذف گره‌ها از آن، به همراه ارائه نمایش گرافیکی از فرآیندها در قالب سند لاتک می‌باشد.

کد این قسمت شامل بخش‌های مختلفی است. کلاس BinaryTreeNode به عنوان نماینده یک گره در درخت دودویی جست‌وجو عمل می‌کند و دارای ویژگی‌هایی از جمله name برای ذخیره مقدار گره و left و right برای اشاره‌گر به فرزندان چپ و راست است. تابع add_node مسئول افزودن گره‌های جدید به درخت است و به گونه‌ای عمل می‌کند که ویژگی‌های BST در درخت حفظ شود.

تابع find_min برای یافتن کوچک‌ترین مقدار در زیر درخت سمت راست یک گره خاص به کار می‌رود و در فرآیند حذف گره‌ها مورد استفاده قرار می‌گیرد. تابع delete_node برای حذف یک گره با مقدار مشخص از درخت دودویی جست‌وجو طراحی شده و در صورت نیاز از تابع find_min برای جایگزینی گره حذف‌شده استفاده می‌کند تا ساختار درخت به‌گونه‌ای به‌روز شود که ویژگی‌های BST همچنان حفظ گردد.

برای تولید نمایش گرافیکی درخت دودویی جست‌وجو، تابع generate_latex_binary_tree استفاده می‌شود که به صورت بازگشتی درخت را پیمایش کرده و با استفاده از دستورات TikZ گرافیک مربوط به هر گره و ارتباطات آن با گره‌های فرزند را تولید می‌کند. تابع generate_latex_documen t نیز برای تولید سند نهایی لاتک شامل درخت دودویی جست‌وجو پس از انجام عملیات حذف به کار می‌رود، در حالی که تابع generate_latex_document_steps سندی را تولید می‌کند که شامل مراحل گام‌به‌گام حذف گره‌ها از درخت است و هر مرحله شامل نمایش درخت قبل و بعد از حذف گره می‌باشد.

تنظیمات ورودی پژوهش شامل مقادیر عددی برای ساخت درخت دودویی جست‌وجو و مقادیر گره‌هایی است که باید حذف شوند. کاربر می‌تواند انتخاب کند که سند نهایی شامل نمایش مراحل گام‌به‌گام حذف گره‌ها

باشد یا فقط نمایش نهایی درخت را مشاهده کند. در مرحله نهایی، درخت دودویی جست‌وجو به صورت گرافیکی در سند لاتک نمایش داده می‌شود و در حالت مراحل گام‌به‌گام حذف، سند شامل نمودارهایی از فرآیند حذف گره‌ها به صورت گام‌به‌گام است.

۴-۲۱-۳- عملیات درج در درخت جست‌وجوی دودویی

این بخش به پیاده‌سازی الگوریتم درج در درخت جست‌وجوی دودویی اختصاص دارد. هدف این کد، ساخت درخت جست‌وجوی دودویی از مجموعه‌ای از اعداد و نمایش فرآیند ساخت درخت به صورت گام به گام در قالب اشکال لاتک است.

در این پژوهش، کلاس BinaryTreeNode برای ایجاد گره‌های درخت جست‌وجوی دودویی طراحی شده است. هر گره شامل یک مقدار (name) و دو اشاره‌گر به گره‌های فرزند چپ (left) و راست (right) می‌باشد. تابع insert_into_bst مسئول درج مقادیر جدید در محل مناسب درخت است و به صورت بازگشتی عمل می‌کند تا گره جدید را در زیرشاخه چپ یا راست گره فعلی قرار دهد. برای تولید نمایش گرافیکی درخت جست‌وجوی دودویی، تابع generate_latex_binary_tree استفاده می‌شود که کد LaTeX را به صورت نمودارهای TikZ تولید می‌کند و وضعیت درخت را پس از هر مرحله از درج نمایش می‌دهد. تابع generate_latex_document یک سند لاتک کامل برای نمایش نهایی درخت پس از انجام تمام درج‌ها ایجاد می‌کند. همچنین، تابع generate_latex_document_steps سندی تولید می‌کند که فرآیند ساخت درخت را به صورت گام به گام نمایش داده و هر مرحله از درج را با نمایش گره جدید به رنگ صورتی مشخص می‌کند.

ورودی‌های این کد شامل مجموعه‌ای از اعداد صحیح است که توسط کاربر وارد می‌شود و به ترتیب به درخت جست‌وجوی دودویی اضافه می‌شود تا ساختار نهایی درخت شکل گیرد. نمودارهای تولید شده در هر مرحله، وضعیت درخت را پس از درج هر عدد نمایش می‌دهند. این نمودارها با استفاده از TikZ در لاتک تولید شده و گره‌های جدید با رنگ صورتی به تصویر کشیده می‌شوند.

۴-۲۱-۴- عملیات جست‌وجو در درخت جست‌وجوی دودویی

در این قسمت به پیاده‌سازی الگوریتم جست‌وجو در این ساختار داده‌ای اختصاص دارد و هدف آن جست‌وجوی یک مقدار مشخص در درخت و نمایش فرآیند جست‌وجو به صورت تصویری و گام به گام است.

در این پژوهش، کلاس BinarySearchTreeNode برای ایجاد گره‌های درخت جست‌وجوی دودویی طراحی شده است. هر گره شامل یک مقدار (value) و دو اشاره‌گر به فرزندهای چپ و راست می‌باشد. تابع search_bst به عنوان هسته اصلی جست‌وجو عمل کرده و مقدار مشخص را به صورت بازگشتی در درخت جست‌وجوی دودویی جست‌وجو می‌کند. در هر مرحله، وضعیت درخت در قالب نمودار لاتک تولید می‌شود و مراحل جست‌وجو در یک لیست ذخیره می‌گردند. تابع generate_latex_binary_tree برای تولید کد لاتک

جهت نمایش درخت و رنگ‌آمیزی گره‌هایی که با مقدار مورد جست‌وجو مطابقت دارند، به کار می‌رود. در این تابع، گره‌هایی که مقدار آن‌ها با مقدار جست‌وجو شده تطابق دارد، به صورت رنگی نمایش داده می‌شوند. همچنین، تابع generate_latex_document_steps سندی کامل از مراحل مختلف جست‌وجو را تولید می‌کند که شامل نمایش گرافیکی هر مرحله از مقایسه گره فعلی با مقدار مورد جست‌وجو است.

ورودی این کد مقداری است که کاربر قصد جست‌وجوی آن را دارد و این مقدار در درخت جست‌وجوی دودویی جست‌وجو می‌شود. در هر مرحله از جست‌وجو، درخت به صورت نمودار لاتک نمایش داده می‌شود و گره‌ای که با مقدار جست‌وجو مقایسه می‌شود، به رنگ سرخابی مشخص می‌شود.

۴-۲۱-۵- عملیات پیمایش عمقی در درخت جست‌وجوی دودویی

در این قسمت به بررسی و پیاده‌سازی دو الگوریتم کلیدی در زمینه درخت‌های دودویی پرداخته شده است: جست‌وجو در درخت دودویی و پیمایش عمق. هدف اصلی این کدها، اجرای جست‌وجوی یک گره خاص در درخت دودویی و نمایش مراحل مختلف پیمایش درخت به روش‌های پیش‌سفری، درون‌سفری و پس‌سفری است.

کدهای پژوهش شامل دو کلاس اصلی به نام‌های BinarySearchTreeNode و BinaryTreeNode هستند که نماینده گره‌های درخت جست‌وجوی دودویی هستند. هر کدام از این کلاس‌ها شامل مقدار گره و اشاره‌گرهایی به فرزندهای چپ و راست می‌باشند. برای جست‌وجو، تابع search_bst طراحی شده است که به دنبال یک مقدار خاص در درخت می‌گردد و مراحل جست‌وجو را در لیستی ذخیره می‌کند تا برای تولید خروجی تصویری استفاده شود. تابع generate_latex_binary_tree نیز مسئول تولید کد لاتک برای نمایش گرافیکی درخت دودویی است که ارتباطات بین گره‌ها را به صورت بصری به نمایش می‌گذارد.

برای پیمایش درخت، سه تابع اصلی تعریف شده‌اند: pre_order_traversal، که در آن ابتدا گره ریشه، سپس زیر درخت چپ و بعد زیر درخت راست پیمایش می‌شود؛ in_order_traversal، که ابتدا زیر درخت چپ، سپس گره ریشه و در نهایت زیر درخت راست پیمایش می‌شود و post_order_traversal، که ابتدا زیر درخت چپ، سپس زیر درخت راست و در نهایت گره ریشه پیمایش می‌شود. تابع generate_latex_document_steps برای تولید سند لاتک استفاده می‌شود که شامل مراحل مختلف جست‌وجو و پیمایش در درخت است.

کاربر می‌تواند نوع پیمایش را از میان گزینه‌های پیش‌سفری، درون‌سفری یا پس‌سفری انتخاب کند و برای جست‌وجو در درخت، مقدار مورد نظر را وارد نماید. در هر مرحله از اجرای الگوریتم، گره‌های درخت به صورت بصری و با استفاده از کد لاتک نمایش داده می‌شوند. گره‌هایی که هدف جست‌وجو هستند، با رنگ خاصی برجسته می‌شوند.

این کدها به منظور نمایش بصری مراحل جست‌وجو و پیمایش در درخت‌های دودویی طراحی شده‌اند و می‌توانند به عنوان ابزاری آموزشی برای فهم بهتر الگوریتم‌های درخت دودویی استفاده شوند. با استفاده از این کدها، فرآیندهای پیچیده جست‌وجو و پیمایش به صورت گام‌به‌گام و قابل فهم‌تر به نمایش گذاشته شده و تحلیل آن‌ها آسان‌تر می‌شود.

۴-۲۱-۶- عملیات پیمایش سطحی در درخت جست‌وجوی دودویی

در ان قسمت به بررسی و مصورسازی عملیات پیمایش سطحی در درخت جست‌وجوی دودویی پرداخته شده‌است. الگوریتم پیمایش سطحی، تمامی گره‌های موجود در یک سطح از درخت را قبل از پردازش گره‌های سطح بعدی پیمایش می‌کند. هدف اصلی این کد، نمایش گرافیکی مراحل مختلف این نوع پیمایش و نمایش وضعیت درخت در هر مرحله با استفاده از لاتک و TikZ است.

در این کد، کلاس BinaryTreeNode برای تعریف ساختار یک گره از درخت جست‌وجوی دودویی استفاده شده است. هر گره شامل یک مقدار و ارجاع‌هایی به گره‌های فرزند چپ و راست خود می‌باشد. تابع breadth_first_traversal عملیات پیمایش سطحی را با استفاده از یک صف (queue) از گره‌ها انجام می‌دهد. در این تابع، گره‌ها یکی یکی از صف خارج شده و وضعیت فعلی درخت برای هر گره در حال پردازش، در لیستی به نام steps ذخیره می‌شود. تابع generate_latex_binary_tree نیز مسئول تولید کد لاتک برای رسم درخت به صورت گرافیکی است و گره هدف در حال پردازش به صورت ویژه با رنگ متمایز نمایش داده می‌شود. همچنین، تابع generate_latex_document_steps مراحل مختلف پیمایش سطحی را به صورت تصاویر و به ترتیب زمانی در قالب یک سند لاتک تنظیم می‌کند.

ورودی اصلی این کد یک درخت جست‌وجوی دودویی است که به صورت دستی در کد تعریف شده است. این درخت شامل گره‌هایی با مقادیر مشخص است: ریشه درخت با مقدار ۱۰، گره‌های فرزند چپ و راست ریشه با مقادیر ۵ و ۲۰، و گره‌های فرزند چپ و راست زیر درخت چپ و راست با مقادیر ۳، ۷، ۱۵ و ۲۵. کد تولید شده، مراحل پیمایش سطحی درخت را در قالب نمودارهایی که با استفاده از TikZ و لاتک رسم شده‌اند، نمایش می‌دهد. هر مرحله از پیمایش، وضعیت درخت را پس از پردازش یک گره به صورت بصری نمایش می‌دهد و گره در حال پردازش با رنگی متمایز مشخص می‌شود.

۴-۲۲- درخت قرمز- سیاه

درخت قرمز- سیاه یک نوع درخت جست‌وجوی دودویی خودمتعادل است که هر گره آن با یکی از دو رنگ قرمز یا سیاه مشخص می‌شود. این درخت به دلیل حفظ ویژگی‌هایی مانند عدم وجود دو گره قرمز متوالی و تعداد یکسان گره‌های سیاه در مسیرهای مختلف از ریشه تا برگ، تعادل خود را حفظ می‌کند. هدف این بخش از پژوهش، مصورسازی عملیات ایجاد، حذف و درج در درخت قرمز- سیاه است تا فرآیندهای پیچیده‌ی مربوط

به متعادل‌سازی آن به‌صورت بصری نمایش داده شوند.

۴-۲۲-۱- عملیات ایجاد درخت قرمز-سیاه

این قسمت از پژوهش شامل چندین بخش اصلی است که در آن، کلاس RedBlackNode برای تعریف یک گره در درخت قرمز- سیاه استفاده می‌شود. هر گره شامل اطلاعاتی مانند مقدار، رنگ (قرمز یا سیاه)، و اشاره‌گرهایی به فرزندان چپ و راست و والدین است. کلاس RedBlackTree ساختار درخت قرمز- سیاه را پیاده‌سازی می‌کند و شامل روش‌هایی برای درج گره‌ها، چرخش‌های چپ و راست، و متعادل‌سازی درخت پس از هر درج است. روش‌های left_rotate و right_rotate عملیات چرخش را انجام می‌دهند که برای متعادل‌سازی درخت پس از هر درج ضروری است. روش insert مسئولیت درج یک گره جدید در درخت را بر عهده دارد و در صورت نیاز، عملیات متعادل‌سازی را نیز فراخوانی می‌کند. همچنین، روش fix_insert پس از درج یک گره، با استفاده از چرخش‌ها و تغییر رنگ‌ها، درخت را متعادل می‌کند تا ویژگی‌های خاص درخت قرمز- سیاه حفظ شوند. تولید کد لاتک نیز برای نمایش گرافیکی درخت قرمز- سیاه در هر مرحله از عملیات درج و متعادل‌سازی بر عهده دارد.

کاربر می‌تواند مجموعه‌ای از اعداد را به عنوان ورودی وارد کند که به ترتیب به عنوان گره‌های جدید در درخت قرمز- سیاه درج می‌شوند. پس از ورود تمامی مقادیر، با وارد کردن done، فرآیند درج خاتمه می‌یابد و عملیات‌های مرتبط به پایان می‌رسد. در این پژوهش، هر مرحله از عملیات درج و متعادل‌سازی درخت قرمز- سیاه به‌صورت تصویری و با استفاده از لاتک نمایش داده می‌شود. در هر تصویر ایجاد شده، وضعیت فعلی درخت به همراه گره‌هایی که تغییراتی در آن‌ها رخ داده است، به وضوح نمایش داده می‌شود. این نوع نمایش، به کاربران کمک می‌کند تا بهتر با نحوه عملکرد درخت‌های قرمز- سیاه و تغییرات ایجاد شده در هر مرحله آشنا شوند.

۴-۲۲-۲- عملیات حذف در درخت قرمز-سیاه

در این بخش به بررسی و پیاده‌سازی عملیات حذف در درخت‌های قرمز- سیاه پرداخته خواهد شد و همچنین چگونگی مصورسازی فرآیند حذف به‌صورت گرافیکی و مرحله‌به‌مرحله شرح داده می‌شود.

در قسمت عملیات حذف در درخت قرمز-سیاه، کدهای ارائه شده شامل دو کلاس اصلی هستند: کلاس RedBlackNode که مسئولیت تعریف گره‌های درخت را بر عهده دارد. هر گره شامل مقدار، رنگ (قرمز یا سیاه)، و اشاره‌گرهایی به فرزندان و والد خود است. کلاس دوم RedBlackTree است که شامل تمامی عملیات‌های مرتبط با درخت، از جمله چرخش‌های چپ و راست، درج گره‌ها، و عملیات حذف به همراه تنظیمات لازم برای حفظ ویژگی‌های درخت قرمز- سیاه می‌باشد. هر یک از این کلاس‌ها و توابع مرتبط با آن‌ها به‌گونه‌ای طراحی شده‌اند که تمامی ویژگی‌های اصلی درخت قرمز- سیاه را به دقت رعایت کنند.

برای شروع عملیات حذف، کاربر مقدار مورد نظر خود را به تابع delete ارائه می‌دهد. این تابع با توجه به

مقدار وارد شده، فرآیند حذف گره را آغاز می‌کند. در طول عملیات حذف، تغییرات مختلفی در ساختار درخت اتفاق می‌افتد که برای حفظ تعادل درخت ضروری هستند. این تغییرات شامل چرخش‌های چپ و راست و تغییر رنگ گره‌ها است که در هر مرحله به صورت دقیق و منظم انجام می‌شوند. برای نمایش بهتر این تغییرات، از توابعی استفاده شده است که با بهره‌گیری از کد لاتک ، تصاویری گرافیکی از وضعیت درخت در هر مرحله ایجاد می‌کنند.

در این تصاویر، هر گره با رنگ مشخص خود (قرمز یا سیاه) نمایش داده می‌شود و خطوط بین گره‌ها نشان‌دهنده رابطه والد - فرزندی هستند. به دلیل اهمیت موضوع، تغییرات هر مرحله از عملیات حذف به تفکیک و با توضیحات مربوطه نمایش داده می‌شود. این نمایش گرافیکی به کاربر کمک می‌کند تا به سادگی نحوه عملکرد درخت‌های قرمز- سیاه را با وجود چالش‌های مرتبط با حفظ تعادل در این درخت‌ها، درک کند و تأثیر هر عملیات بر روی ساختار درخت را به‌وضوح ببیند.

۴-۲۲-۳- عملیات درج در درخت قرمز- سیاه

قسمت مربوط به عملیات درج در درخت قرمز- سیاه شامل چندین بخش کلیدی است که به‌طور جداگانه به شرح آن‌ها می‌پردازیم. ابتدا، کلاس RedBlackNode تعریف شده که نمایانگر گره‌های درخت قرمز- سیاه است. این گره‌ها دارای ویژگی‌هایی نظیر مقدار، رنگ، اشاره‌گر به گره‌های چپ و راست، و همچنین اشاره‌گر به والد خود هستند. کلاس دوم، RedBlackTree، که خود درخت قرمز- سیاه را مدل می‌کند، شامل عملیات اصلی بر روی درخت از جمله چرخش‌های چپ و راست، و همچنین درج گره‌های جدید است. در این کلاس، توابع مهمی مانند left_rotate و right_rotate وجود دارند که به منظور حفظ تعادل درخت پس از هر عملیات، مورد استفاده قرار می‌گیرند. تابع insert وظیفه درج یک گره جدید را بر عهده دارد و با فراخوانی تابع fix_insert پس از هر درج، اطمینان حاصل می‌کند که تمامی قوانین درخت قرمز- سیاه رعایت شده‌اند و تعادل درخت حفظ می‌شود.

برای استفاده از این کد، کاربر مقدار مورد نظر خود را که قصد دارد در درخت قرمز- سیاه درج کند، به تابع insert وارد می‌کند. این مقدار می‌تواند هر عدد صحیحی باشد. پس از درج مقدار، کد به‌طور خودکار عملیات لازم برای حفظ تعادل درخت را انجام می‌دهد و در صورت بروز هرگونه نقض در قوانین درخت، اصلاحات لازم را اعمال می‌کند.

یکی از ویژگی‌های بارز این پژوهش، قابلیت مصورسازی تغییرات درخت پس از هر مرحله از عملیات درج است. برای این منظور، از یک تابع کمکی به نام generate_latex_red_black_tree استفاده می‌شود که در هر مرحله از درج، نمایشی از درخت قرمز- سیاه به فرمت لاتکتولید می‌کند. این نمایش‌ها شامل اطلاعاتی همچون رنگ گره‌ها و ارتباطات بین گره‌ها هستند و به‌طور بصری تغییراتی که در ساختار درخت پس از هر

عمل درج ایجاد می‌شود، به کاربر نمایش داده می‌شود.

۴-۲۳- درخت AVL

درخت AVL یک نوع درخت جستجوی دودویی متوازن است که به‌طور خودکار تعادل خود را حفظ می‌کند تا عملیات جستجو، درج، و حذف بهینه انجام شود. این پیاده‌سازی با هدف مصورسازی ساختار درخت و عملیات مختلف بر روی آن، طراحی شده است تا کاربران بتوانند به‌صورت بصری با عملکرد این نوع درخت آشنا شوند.

۴-۲۳-۱- عملیات ایجاد درخت AVL

در این بخش از پژوهش، پیاده‌سازی درخت AVL با استفاده از زبان برنامه‌نویسی پایتون و ابزارهای لاتکو TikZ بررسی می‌شود که هدف از پیاده‌سازی آن مصورسازی چگونگی فرایند ایجاد درخت AVL می‌باشد.

در کد این بخش، کلاس BinaryTreeNode نقش اصلی در پیاده‌سازی درخت AVL را ایفا می‌کند. این کلاس نمایانگر یک گره در درخت AVL است و شامل اطلاعات مربوط به نام گره، اشاره‌گرهای فرزندان چپ و راست، و ارتفاع گره می‌باشد. ویژگی‌های این کلاس شامل name برای نگهداری مقدار گره، left و right برای اشاره به فرزندان چپ و راست، و height برای محاسبه ارتفاع گره است. این کلاس برای ساخت گره‌ها در درخت استفاده می‌شود.

برای به‌روزرسانی و حفظ تعادل درخت، از چندین تابع کمکی استفاده می‌شود. تابع get_height ارتفاع یک گره را بازمی‌گرداند و در صورت تهی بودن گره، ارتفاع صفر را بازمی‌گرداند. تابع update_height ارتفاع گره را بر اساس ارتفاع فرزندانش به‌روزرسانی می‌کند. تابع get_balance تعادل گره را با تفریق ارتفاع فرزند راست از فرزند چپ محاسبه می‌کند. برای برقراری تعادل درخت، دو تابع rotate_right و rotate_left به ترتیب چرخش راست و چپ را بر روی گره‌های مورد نیاز انجام می‌دهند.

تابع insert_into_avl وظیفه درج یک مقدار جدید در درخت AVL را برعهده دارد. این تابع ابتدا بررسی می‌کند که آیا ریشه درخت خالی است یا نه و در صورت خالی بودن، یک گره جدید با مقدار داده‌شده ایجاد می‌کند. سپس مقدار جدید با مقدار گره ریشه مقایسه شده و به سمت چپ یا راست حرکت می‌کند. پس از درج، ارتفاع و تعادل گره‌ها به‌روزرسانی می‌شود و در صورت نیاز، چرخش‌های لازم برای برقراری تعادل درخت انجام می‌شود.

برای مصورسازی درخت و عملیات انجام‌شده بر روی آن، تابع generate_latex_binary_tree کد لاتکمربوط به نمایش درخت را تولید می‌کند. این تابع برای هر گره، کدی برای نمایش نام و تعادل آن تولید کرده و فرزندان چپ و راست را به‌صورت بازگشتی پردازش می‌کند. در صورت نیاز به برجسته‌سازی گره‌ای خاص، آن گره با رنگ متفاوت نمایش داده می‌شود. سپس، تابع generate_latex_document یک سند

لاتکتولید می‌کند که شامل توضیحات مربوط به درخت AVL و نمودار درخت با تمامی تغییرات و جزئیات است.

این پیاده‌سازی به کاربران امکان می‌دهد تا فرآیند درج مقادیر جدید در درخت AVL و نحوه برقراری تعادل را به‌صورت تصویری مشاهده کنند. با هر بار درج یک گره جدید، وضعیت درخت نمایش داده می‌شود و در صورت بروز عدم تعادل، گره‌های مربوطه با رنگ‌های متفاوت نمایش داده می‌شوند. در نهایت، وضعیت نهایی درخت پس از اعمال چرخش‌ها نمایش داده می‌شود.

۴-۲۳-۲- عملیات حذف گره از درخت AVL

در این بخش به چگونگی پیاده‌سازی عملیات حذف گره از درخت AVL پرداخته شده است. این پیاده‌سازی با استفاده از زبان برنامه‌نویسی پایتونو ابزارهای لاتکو TikZ طراحی شده تا ساختار درخت و عملیات مختلفی که بر روی آن انجام می‌شود، به‌صورت تصویری نمایش داده شوند.

در پیاده‌سازی این درخت، کلاس BinaryTreeNode نقش اصلی را ایفا می‌کند. این کلاس برای نمایش یک گره در درخت AVL طراحی شده و شامل اطلاعاتی نظیر نام گره، اشاره‌گرهای فرزندان چپ و راست و ارتفاع گره می‌باشد. ویژگی‌های این کلاس شامل name برای نگهداری مقدار گره، left و right برای اشاره به فرزندان چپ و راست و height برای محاسبه ارتفاع گره است. این کلاس به عنوان پایه‌ای برای ساخت گره‌ها در درخت مورد استفاده قرار می‌گیرد.

برای انجام عملیات حذف و حفظ تعادل درخت، از چندین تابع کمکی استفاده می‌شود. تابع get_height برای بازگرداندن ارتفاع یک گره به کار می‌رود و در صورت تهی بودن گره، ارتفاع صفر را بازمی‌گرداند. تابع update_height نیز ارتفاع گره را بر اساس ارتفاع فرزندان آن به‌روزرسانی می‌کند. تابع get_balance برای محاسبه تعادل گره از طریق تفریق ارتفاع فرزند راست از فرزند چپ به کار می‌رود. در صورت نیاز به حفظ تعادل، از توابع rotate_right و rotate_left برای چرخش گره‌ها به ترتیب به سمت راست و چپ استفاده می‌شود.

تابع delete_from_avl برای حذف یک گره با مقدار مشخص از درخت AVL طراحی شده است. این تابع به صورت بازگشتی به جست‌وجوی گره‌ای با مقدار مورد نظر ادامه می‌دهد و در صورت یافتن آن، عملیات حذف را انجام داده و گره‌های فرزند مربوطه را به‌روزرسانی می‌کند. پس از حذف گره، ارتفاع و تعادل گره‌ها مجدداً محاسبه می‌شود و در صورت نیاز، چرخش‌های لازم برای بازگرداندن تعادل به درخت انجام می‌شود.

برای نمایش گرافیکی عملیات انجام‌شده بر روی درخت، از تابع generate_latex_binary_tree استفاده می‌شود که کد لاتک مربوط به نمایش درخت را تولید می‌کند. این تابع برای هر گره کدی تولید می‌کند که نام و تعادل آن را نمایش می‌دهد و فرزندان چپ و راست را به‌صورت بازگشتی پردازش می‌کند. همچنین، فرآیند

حذف و متوازن‌سازی در قالب مراحل مختلف و با استفاده از تابع generate_latex_document_steps نمایش داده می‌شود که سندی لاتک تولید می‌کند و شامل نمودار درخت و توضیحات مربوط به آن است.

این پیاده‌سازی به کاربران امکان می‌دهد تا فرآیند حذف گره از درخت AVL و نحوه برقراری تعادل در آن را به‌صورت تصویری مشاهده کنند. پس از هر بار حذف یک گره، وضعیت جدید درخت نمایش داده می‌شود و در صورت بروز عدم تعادل، گره‌های مربوطه با رنگ‌های متفاوت نشان داده می‌شوند.

۴-۲۳-۳- عملیات ایجاد گره در درخت AVL

در این بخش به بررسی و پیاده‌سازی عملیات ایجاد گره در درخت AVL و نمایش چگونگی افزودن یک گره به درخت و متعادل‌سازی خودکار پس از هر عملیات درج با استفاده از زبان برنامه‌نویسی پایتون پرداخته خواهدشد .

در این پیاده سازی ابتدا، یک کلاس به نام BinaryTreeNode تعریف می‌شود که نماینده‌ی گره‌ها در درخت AVL است. این کلاس شامل اطلاعاتی نظیر نام گره، اشاره‌گرهای فرزندان چپ و راست، و ارتفاع گره می‌باشد. ارتفاع گره به عنوان طولانی‌ترین مسیر از آن گره تا یک برگ محاسبه می‌شود و در عملیات‌های متعادل‌سازی نقشی کلیدی دارد. به منظور بررسی وضعیت تعادل گره‌ها، تابعی به نام get_balance پیاده‌سازی شده که تفاوت ارتفاع زیردرخت‌های چپ و راست یک گره را محاسبه می‌کند.

یکی از بخش‌های مهم این پیاده‌سازی، عملیات چرخش است که برای بازگرداندن تعادل به درخت پس از درج یک گره جدید مورد استفاده قرار می‌گیرد. دو نوع چرخش اصلی شامل چرخش راست و چرخش چپ در اینجا پیاده‌سازی شده‌اند که هر کدام در شرایط خاصی از عدم تعادل مورد استفاده قرار می‌گیرند. در این پیاده‌سازی، تابع insert_into_avl مسئولیت درج گره‌های جدید به درخت AVL را بر عهده دارد. این تابع پس از درج گره جدید، وضعیت تعادل درخت را بررسی کرده و در صورت لزوم چرخش‌های لازم را اعمال می‌کند تا تعادل درخت حفظ شود.

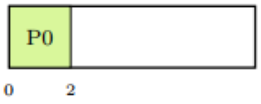
در نهایت، به منظور مصورسازی ساختار درخت و نمایش چگونگی تغییرات ایجادشده پس از هر عملیات درج، توابعی برای تولید کد لاتک طراحی شده‌اند. این توابع به صورت بازگشتی گره‌ها و ساختار درخت را به کد لاتک تبدیل کرده و امکان نمایش گرافیکی درخت AVL را فراهم می‌آورند. چنین نمایشی می‌تواند به درک بهتر مفهوم تعادل در درخت‌های AVL و چگونگی اعمال چرخش‌ها کمک کند. این پیاده‌سازی یک ابزار مفید برای تجسم فرآیندهای درخت AVL بوده و می‌تواند به‌عنوان یک ابزار آموزشی مؤثر در تدریس مفاهیم مربوط به درخت‌های متوازن مورد استفاده قرار گیرد.

۴-۲۴- نمونه هایی از تصاویر خروجی

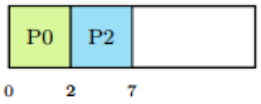
تمامی خروجی ها برای صفحات A4 تنظیم شده‌اند. با قرار دادن فایل با خروجی tex. در TeXstudio و اجرای آن می‌توان خروجی pdf آن را نیز مشاهده کرد[۸].

۴-۲۴-۱- الگوریتم زمان‌بندی کوتاه‌ترین کار

1.1 Time 0: Process P0

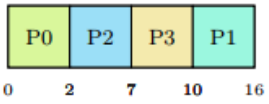


1.2 Time 2: Process P2



1.3 Time 7: Process P3

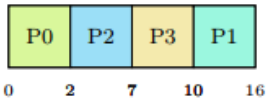
1.4 Time 10: Process P1



1.5 Final Process Table

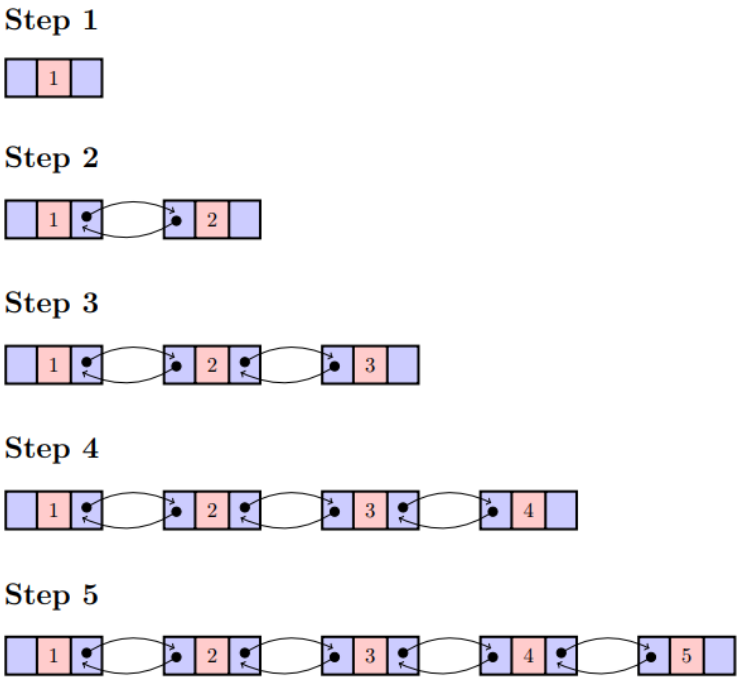
Process	Arrival Time	Burst Time	Service Time
P0	0	2	0
P1	1	6	10
P2	2	5	2
P3	3	3	7

1.6 Execution Order



شکل۴-۱: الگوریتم زمان‌بندی کوتاه‌ترین کار برای چهار پردازش مشخص شده

۴-۲۴-۲- الگوریتم ایجاد لیست پیوندی دوطرفه



شکل۴-۲: الگوریتم ایجاد لیست پیوندی دوطرفه

۴-۲۴-۲- الگوریتم ایجاد درخت قرمز- سیاه

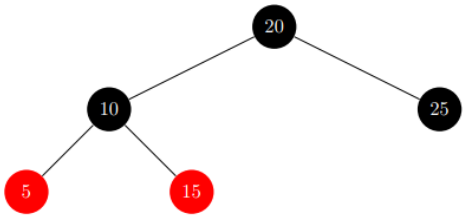


Figure 1: Initial Tree

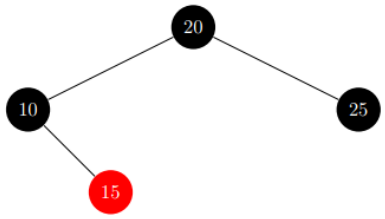
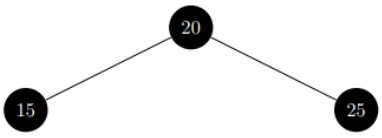


Figure 2: Step 2



شکل۴-۳: الگوریتم ایجاد درخت قرمز- سیاه

۴-۲۵- جمع‌بندی

در این فصل، به تفصیل نحوه‌ی پیاده‌سازی الگوریتم‌های مختلف در پژوهش توسعه یافته بررسی شد. این پیاده‌سازی‌ها با هدف تولید فایل‌های لاتک که به صورت بصری مراحل اجرای هر الگوریتم را به تصویر می‌کشند، انجام شده‌اند. الگوریتم‌های مورد بررسی در این پژوهش شامل الگوریتم‌های کلاسیکی همچون الگوریتم‌های جست‌وجو، مرتب‌سازی، مدیریت لیست‌های پیوندی، انواع درخت، استک و صف بوده‌اند که هر یک به صورت گام‌به‌گام تجسم یافته‌اند.

در این فصل، نحوه استفاده از کتابخانه‌های مختلف پایتون و لاتک مانند pylatex و TikZ برای تولید تصاویر گرافیکی و مستندات آموزشی توضیح داده شد. در هر پیاده‌سازی، تلاش شده است تا با ایجاد انعطاف‌پذیری در تنظیمات مربوط به ورودی‌ها و ویژگی‌های گرافیکی، کاربران قادر باشند تا خروجی‌های دلخواه و سفارشی‌سازی شده‌ای برای مقاصد آموزشی و تحقیقاتی خود ایجاد کنند.

این فصل نشان داد که چگونه می‌توان با استفاده از ترکیب قدرتمند پایتون و لاتک ، ابزاری کارآمد برای مصورسازی الگوریتم‌ها ایجاد کرد. نتایج حاصل از پیاده‌سازی‌ها نشان می‌دهد که این ابزار می‌تواند به عنوان یک منبع آموزشی ارزشمند در کلاس‌های درس، مقالات آکادمیک و مطالعات پژوهشی مورد استفاده قرار گیرد. در نهایت، این فصل با ارائه پیاده‌سازی‌های موفق، بستر مناسبی را برای ادامه کارهای آموزشی و پژوهشی فراهم کرده است.

فصل پنجم
توسعه و پیاده‌سازی کتابخانه‌ای جامع

۵-۱- مقدمه

در این فصل، به طراحی یک کتابخانه‌ای جامع برای مصورسازی الگوریتم‌ها می‌پردازیم. هدف از این کتابخانه، فراهم کردن ابزاری قدرتمند برای نمایش گرافیکی و تعاملی انواع مختلف الگوریتم‌ها است. در فصل قبل، به مصورسازی الگوریتم‌ها با استفاده از لاتک پرداخته شد. با این حال، برای ایجاد یک ابزار جامع و قابل توسعه، نیازمند یک کتابخانه‌ی مستقل هستیم که بتواند طیف گسترده‌ای از الگوریتم‌ها را پوشش دهد.

۵-۲- بررسی چند فایل ضروری در توسعه کتابخانه

در ابتدا یک توضیحات کلی از بعضی از فایل‌های موردنیاز داده می‌شود و در ادامه ساختار کتابخانه جهت استفاده توضیح داده خواهد شد.

۵-۲-۱- فایل نیازمندی‌ها

در این کتابخانه در پوشه اصلی یک فایل به نام requirements.txt وجود دارد که تمام نیازمندی‌های لازم جهت نصب کتابخانه و اجرای الگوریتم‌ها می‌باشد. باید قبل از استفاده از کتابخانه حتما این نیازمندی‌ها به درستی نصب شده باشند در غیر صورت ممکن است کتابخانه به درستی پاسخگو نباشد یا خطایی رخ دهد. ابتدا، اگر نیاز به استفاده از این کتابخانه دارید، باید این دو وابستگی را که در فایل نوشته شده است نصب کنید:

- Setuptools =72.1.0
- PyLaTeX =1.4.2

۵-۲-۲- فایل مجوز[1]

در این کتابخانه در پوشه اصلی یک فایل به نام LICENSE وجود دارد، از آنجایی که پروژه به صورت

[1] LICENSE

متن‌باز است از گواهی GPL[1] استفاده شده است. در ابتدای همه فایل‌های برنامه گواهی آزاد به صورت زیر قرار داده شده است[۹].

```
"""
    <Python library for visualizing data structure algorithms by generating latex output.>
    Copyright (C) 2024  Yasamin Akbari and Mahroo Noohi

    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program.  If not, see <https://www.gnu.org/licenses/>.
"""
```

شکل۵-۱: گواهی GPL

۵-۲-۳- فایل README

در این فایل تمام توضیحات کامل جهت تسهیل کار با کتابخانه و همچنین دستورهای لازم جهت نصب به همراه ذکر مثال آورده شده است.

۵-۲-۴- فایل Setup

فایل setup در پایتون، مسئول توزیع و بسته‌بندی یک پروژه است. این فایل شامل اطلاعاتی درباره‌ی نام، نسخه، نویسنده، و وابستگی‌های پروژه می‌شود و به کاربران امکان می‌دهد که پروژه را به راحتی نصب و استفاده کنند. همچنین، با استفاده از این فایل، پروژه به‌گونه‌ای سازماندهی می‌شود که بتوان آن را به‌عنوان یک بسته‌ی قابل نصب در پلتفرم‌هایی مانند PyPI منتشر کرد. تنظیمات داخل این فایل مشخص می‌کنند که چه پکیج‌ها و فایل‌هایی در توزیع نهایی قرار بگیرند و چگونه نصب شوند.

[1] GENERAL PUBLIC LICENSE

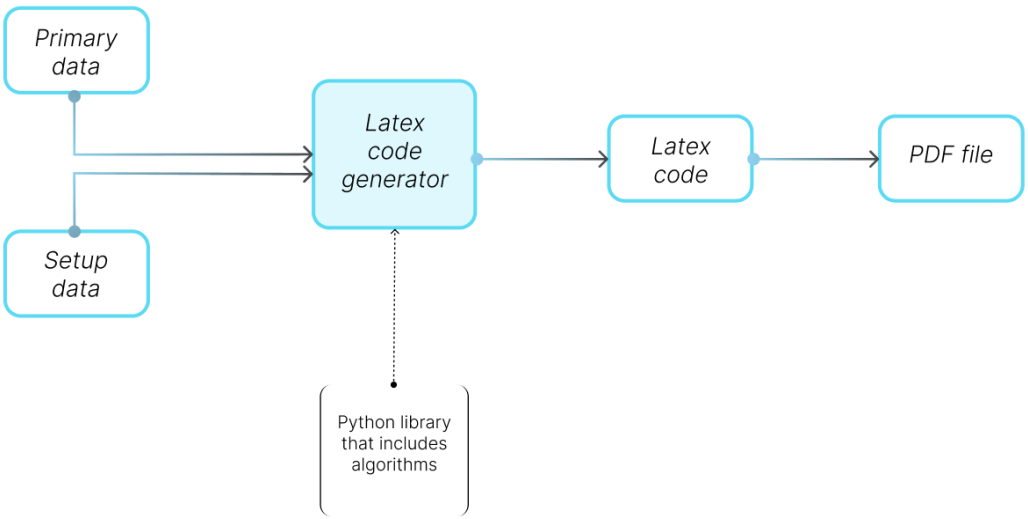
```python
from setuptools import setup, find_packages

setup(
    name="algorithm_visualization_library",
    version="0.1",
    license="GPL-3.0",
    packages=find_packages(),
    include_package_data=True,
    description="Providing a solution for visualizing data structure algorithms by LaTeX language",
    long_description=open('README.md').read(),
    long_description_content_type='text/markdown',
    author="Yasamin Akbari and Mahroo Noohi",
    author_email="mahroonoohi@gmail.com,yasamin.a.7250@gmail.com"
)
```

شکل۵-۲: محتویات فایل setup

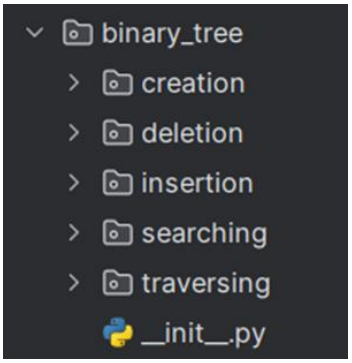
۵-۳- ساختار کتابخانه

به طور کلی ساختار اصلی به شکل زیر است:



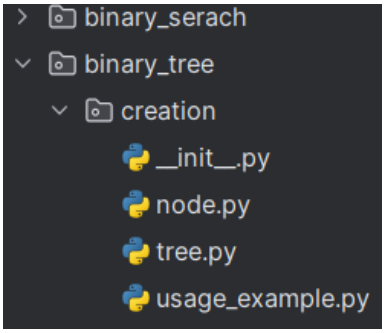
شکل۵-۳: ساختار کتابخانه

در این کتابخانه، تمامی الگوریتم‌ها در پوشه algorithm_visualization_library قرار دارند. الگوریتم های ساختار داده خود در داخل پوشه های جداگانه سازماندهی شده اند. برخی از الگوریتم‌ها دارای پوشه‌های داخلی هستند که نشان می‌دهد آن‌ها حاوی الگوریتم‌های فرعی هستند. به عنوان مثال، در پوشه درخت، پنج عملیات اصلی وجود دارد. ایجاد یک درخت، حذف یک گره داده شده، درج یک گره، جستجوی یک گره معین، و پیمایش درخت.



شکل۵-۴: ساختار قرارگیری عملیات اصلی الگوریتم درخت دودویی در کتابخانه

در زیر چندالگوریتم برای مثال آورده شده است:



شکل۵-۵: ساختار قرارگیری فایل‌های ایجاد درخت دودویی در کتابخانه

در هر پوشه، دو فایل کلیدی وجود دارد که برای همه الگوریتم ها قابل دسترسی است:

- فایل __init__.py: این فایل نقشی کلیدی در پکیج‌های پایتون دارد. این فایل وقتی پکیج وارد می‌شود، اجرا شده و به دایرکتوری می‌گوید که به‌عنوان یک پکیج شناخته شود. همچنین، تعیین می‌کند که چه مواردی برای کاربران در دسترس باشند و چگونه پیکربندی‌ها در سطح پکیج مدیریت شوند. به عبارت دیگر، این فایل مشخص می‌کند که پکیج هنگام وارد شدن چه چیزهایی را ارائه دهد و چگونه سازماندهی شده باشد. در هر پوشه از کتابخانه، یک فایل __init__ وجود دارد که مسئول مدیریت ماژول‌ها یا پکیج‌های سطح پایین‌تر خود است. این ساختار لایه‌ای به این صورت عمل می‌کند که هر فایل __init__ وظایف ماژول‌های داخل دایرکتوری مربوطه را تعریف می‌کند. با حرکت به سمت بالا در درخت دایرکتوری، این فایل‌ها با هم همکاری می‌کنند تا به تدریج مجموعه‌ی کامل قابلیت‌های کتابخانه را تشکیل دهند. در بالاترین سطح، فایل اصلی فایل __init__ مجموعه‌ای از تمام قابلیت‌های کتابخانه را یکجا جمع‌آوری و در دسترس قرار می‌دهد. این فایل فقط زمانی که قرار است کتابخانه توسعه داده شود دستخوش تغییرات گردد.

- فایل usage_example.py: این فایل معمولاً به‌عنوان یک اسکریپت در مخزن یک کتابخانه قرار می‌گیرد تا نحوه‌ی استفاده از آن کتابخانه را نشان دهد. این فایل به‌عنوان یک راهنمای عملی برای کاربرانی که می‌خواهند کتابخانه را در عمل ببینند، بدون اینکه نیاز باشد تمام مستندات را مطالعه کنند، عمل می‌کند. در واقع برای هر الگوریتمی که قرار است مورد استفاده قرار گیرد بعد از فراخوانی باید حتما از محتوای این فایل یک کپی گرفته و در کد موردنظر استفاده گردد.

نحوه استفاده از هر الگوریتم به صورت عملی به شرح زیر می‌باشد:

در گام اول باید بسته به اینکه از چه الگوریتمی قرار است مورد استفاده باشد باید فراخوانی لازم با توجه اسم الگوریتم طبق الگوی زیر انجام شود:

```
from algorithm_visualization_library. algorithm_name_file/algorithm_name_folder  import *
```

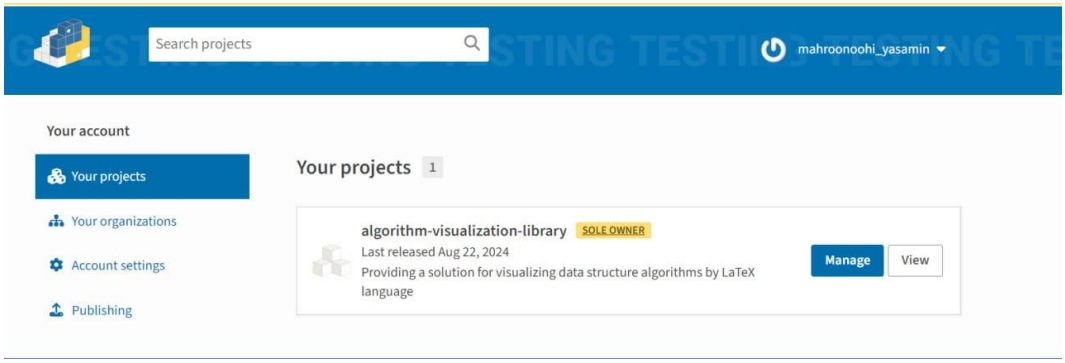
یک مثال برای فراخوانی الگوریتم‌های merge_sort و avl_tree به شرح زیر است:

```
from algorithm_visualization_library. merge_sort import *
from algorithm_visualization_library.avl_tree. creation import *
```

نکته: برای آنکه بررسی شود الگوریتم چه فایل‌هایی دارد، همه فایل‌ها را وارد باید شود. از usage_example به هیچ عنوان فراخوانی نشود.

بعد از اینکه فراخوانی‌ها انجام شد اکنون به سراغ فایل usage_example رفته کل محتوای این فایل کپی و در قسمت فایل خود اضافه گردد و در صورت لزوم و بسته به نیاز کاربر میتوان این فایل را تغییر داد. این فایل راهنمای کامل استفاده از توابع و کلاس‌ها است و حتما از این فایل استفاده شود تا خطایی دریافت نگردد. برای مثال‌های بیشتر حتما به فایل README.md مراجعه شود.

همچنین کتابخانه برای دسترسی همگانی در PyPI قرار داده شد[۱۰]:



شکل۵-۶ : قرارگیری در PyPI

به دو روش می‌توان این کتابخانه را نصب نمود:

۱. از آنجایی که کتابخانه در PyPI قرار داده شده است، کافی است دستور زیر در ترمینال وارد گردد و نیاز به انجام کار خاصی نیست:

```
pip install -i https://test.pypi.org/simple/ algorithm-visualization-library
```

۲. اگر راه اول به هر دلیلی مشکل داشت در این گام ابتدا باید با دستور زیر از گیت‌هاب به منبع پروژه دسترسی داشت:

https://github.com/ui-ce/algorithm-visualizer

اگر خواستید داخل همین کتابخانه فایل خود را قرار دهید داخل همین پوشه کتابخانه شده و با دستور زیر کتابخانه نصب شود:

```
pip install .
```

و اگر در فایل دیگری در یک بخش دیگر از سیستم‌عامل قرار است کدنویسی انجام شود از دستور زیر استفاده گردد:

```
pip install <libary address path>
```

نکته: libary address path آدرس جایی است که کتابخانه در حال حاضر روی آن قرار دارد.

نکته: حتما قبل از استفاده از کتابخانه دستور زیر را در ترمینال اجرا کرده تا خطایی دریافت نگردد:

```
pip install -r requirements.txt
```

۵-۴- جمع‌بندی

در این فصل ابتدا به بررسی چند فایل ضروری در توسعه کتابخانه پرداخته شد. در گام بعد با ساختار کتابخانه با جزئیات کامل و همچنین نحوه استفاده از این کتابخانه و نصب آن به صورت کامل شامل تمامی دستورهای مورد نیاز شرح داده شده است. در فصل بعد به ارزیابی و نتیجه‌گیری، پیشنهادات و کارهای آتی پرداخته خواهد شد.

فصل ششم
ارزیابی، نتیجه‌گیری و پیشنهادهایی برای ادامه پژوهش

۶-۱- ارزیابی

در این بخش، عملکرد و ویژگی‌های پروژه مورد ارزیابی قرار می‌گیرد. پروژه به طور کامل به پیاده‌سازی موفقیت‌آمیز الگوریتم‌ها، ارائه قابلیت‌های تنظیمات سفارشی و معرفی ویژگی‌های منحصر به فرد خود پرداخته است.

تمامی الگوریتم‌های مدنظر در پروژه با دقت و موفقیت پیاده‌سازی شده‌اند. اجرای الگوریتم‌ها بدون هیچ گونه محدودیتی انجام شده و نتایج حاصل از آن‌ها به طور دقیق با پیش‌بینی‌های نظری هماهنگ است. در طول آزمایش‌ها، عملکرد هر الگوریتم مطابق با انتظارات طراحی شده و ورودی‌های مختلف به درستی پردازش شده است.

یکی از ویژگی‌های کلیدی پروژه، قابلیت دریافت داده‌ها و تنظیمات مختلف بر اساس نیاز کاربران است. کاربران می‌توانند داده‌ها، ابعاد، رنگ‌ها و ویژگی‌های بصری را به دلخواه خود تنظیم کنند. این ویژگی به آن‌ها این امکان را می‌دهد که نتایج را به صورت متناسب با نیازهای خاص خود مشاهده کرده و فرآیندهای الگوریتمی را به شکل قابل فهم و شخصی‌سازی شده دریافت نمایند.

بررسی‌های انجام شده نشان می‌دهد که هیچ پروژه مشابهی با تمام ویژگی‌های نام برده شده وجود ندارد. این پروژه با تمرکز بر تولید مستندات بصری و تعاملی از الگوریتم‌ها و همچنین قابلیت‌های سفارشی‌سازی، به طور منحصر به فرد و متفاوت از دیگر پروژه‌ها طراحی شده است. ترکیب زبان برنامه‌نویسی پایتون با LaTeX برای مصورسازی الگوریتم‌ها و تولید مستندات بصری، ویژگی‌های خاص این پروژه را به خوبی نمایش می‌دهد.

۶-۲- نتیجه‌گیری

این پروژه با هدف پیاده‌سازی و مصورسازی الگوریتم‌های مختلف، توانسته است ابزار مفیدی برای تحلیل و بررسی الگوریتم‌ها ارائه دهد. با استفاده از ترکیب زبان برنامه‌نویسی پایتون و لاتک، پروژه موفق شده است تصاویری شفاف و قابل درک از فرآیندهای الگوریتمی ارائه دهد که می‌تواند به طور مؤثری در زمینه‌های مختلف

آموزشی و تحقیقاتی مورد استفاده قرار گیرد.

در کاربردهای آموزشی، این پروژه می‌تواند به عنوان یک ابزار قدرتمند برای توضیح و تدریس الگوریتم‌ها مورد استفاده قرار گیرد. با ارائه تصاویری گرافیکی از مراحل مختلف الگوریتم‌ها، کاربران می‌توانند درک بهتری از عملکرد الگوریتم‌ها و نحوه تغییرات در هر مرحله به دست آورند. این ویژگی به ویژه برای دانش‌آموزان و دانشجویان در فهم بهتر مباحث پیچیده الگوریتمی مفید است.

در زمینه تحقیقاتی، این پروژه می‌تواند به تحلیل دقیق‌تر الگوریتم‌ها کمک کند. همچنین محققان و پژوهشگران می‌توانند از ابزارهای این پروژه برای تجزیه و تحلیل الگوریتم‌ها، ارزیابی عملکرد آن‌ها و مستندسازی نتایج استفاده کنند. به‌ویژه، توانایی سفارشی‌سازی و تولید مستندات بصری، به تحلیل‌های دقیق و نتایج شفاف‌تر کمک می‌کند.

در نهایت، می‌توان گفت این پروژه با تمرکز بر تولید مستندات بصری و تعاملی با امکان شخصی‌سازی تصاویر مطابق با نیاز و سلیقه کاربر و استفاده از ابزارهای پیشرفته برای مصورسازی الگوریتم‌ها، به طور منحصر به فردی از سایر پروژه‌های مشابه، متمایز می‌شود.

۶-۳- پیشنهادهایی برای ادامه پژوهش

با توجه به ظرفیت‌های گسترده، جذابیت وکارایی این پروژه، امکانات وسیعی برای توسعه و گسترش آن وجود دارد. این پروژه با ارائه راهکاری قدرتمند برای پیاده‌سازی و مصورسازی الگوریتم‌ها، نقطه‌ی شروعی برای ارائه قابلیت‌های جدید و بهبودهای بیشتر است. به منظور ارتقاء این پروژه و بهره‌برداری بهتر از توانمندی‌های آن، در ادامه پیشنهادهایی ارائه خواهدشد که می‌تواند به گسترش دامنه کاربرد، افزایش کارایی و بهبود تجربه کاربری پروژه کمک کنند.

- افزودن الگوریتم‌های مربوط به گراف‌ها: با اضافه کردن الگوریتم‌های مربوط به گراف‌ها به پروژه، می‌توان دامنه کاربردهای آن را به طور قابل توجهی گسترش داد. الگوریتم‌های کوتاه‌ترین مسیر مانند دیکسترا و الگوریتم‌های مربوط به درخت‌های پوشا مانند الگوریتم کرسکال و پرایم، می‌توانند به کتابخانه اضافه شوند. این افزودنی‌ها به کاربران این امکان را می‌دهند که تحلیل‌های گسترده‌تری از ساختار گراف‌ها انجام دهند و مسائل پیچیده‌تر را مدل‌سازی کنند.
- امکان دریافت ورودی به صورت فایل: افزودن قابلیت دریافت ورودی به صورت فایل می‌تواند فرآیند استفاده از پروژه را تسهیل کند. با این ویژگی، کاربران قادر خواهند بود تا داده‌ها و پارامترهای مورد نظر خود را از فایل‌های متنی بارگذاری کنند و نتایج تحلیل‌های الگوریتمی را به راحتی مشاهده کنند. این قابلیت می‌تواند به ویژه برای کاربرانی که با داده‌های بزرگ و پیچیده سر و کار دارند، بسیار مفید باشد.

- تولید خروجی تصاویر متحرک: علاوه بر تولید خروجی لاتک، امکان تولید تصاویر متحرک از مراحل مختلف الگوریتم‌ها می‌تواند به درک بهتر و جذاب‌تر نتایج کمک کند. تصاویر متحرک می‌توانند فرآیندهای الگوریتمی را به صورت متوالی و در حال حرکت نمایش دهند، که به ویژه برای آموزش و ارائه‌های بصری بسیار مفید است.
- ایجاد امکان پیاده‌سازی مشترک الگوریتم‌ها: به منظور افزایش انعطاف‌پذیری و کارایی، می‌توان امکان فراخوانی تعدادی از الگوریتم‌ها به صورت مشترک از طریق تعریف یک تابع جدید را فراهم کرد. این ویژگی می‌تواند به کاربران این امکان را بدهد که چندین الگوریتم را به طور همزمان پیاده‌سازی کنند و نتایج را به صورت یکپارچه مشاهده کنند. برای مثال، ترکیب الگوریتم‌های مرتب‌سازی با الگوریتم‌های جست‌وجو یا ترکیب الگوریتم‌های گرافی با الگوریتم‌های پردازش داده، می‌تواند کاربردهای جدیدی را ایجاد کند.
- امکان هماهنگی با انواع نوع و سایز صفحات: برای افزایش رضایت کاربران افزودن امکان هماهنگ‌سازی تصاویر خروجی با صفحات A5 و Beamer نیز میتواند مفید باشد.

با پیاده‌سازی این پیشنهادات، پروژه می‌تواند به ابزاری جامع‌تر و قدرتمندتر تبدیل شود که به نیازهای متنوع‌تری از کاربران پاسخ دهد و قابلیت‌های تحلیل و مصورسازی الگوریتمی را به سطح بالاتری ارتقا دهد.

پیوست ۱: نمونه‌ای از کد پیاده‌سازی

کد پیاده‌سازی الگوریتم زمان‌بندی کوتاه‌ترین کار[۱۱]:

```
"""
   <Python library for visualizing data structure algorithms by generating latex output.>
   Copyright (C) 2024  Yasamin Akbari and Mahroo Noohi

   This program is free software: you can redistribute it and/or modify
   it under the terms of the GNU General Public License as published by
   the Free Software Foundation, either version 3 of the License, or
   (at your option) any later version.

   This program is distributed in the hope that it will be useful,
   but WITHOUT ANY WARRANTY; without even the implied warranty of
   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
   GNU General Public License for more details.

   You should have received a copy of the GNU General Public License
   along with this program.  If not, see <https://www.gnu.org/licenses/>.
"""
from pylatex import Document, Section, Subsection, Tabular, Figure, NoEscape, Package

processes = []



class Process:
   def __init__(self, name, burst_time, arrival_time, color):
      self.name = name
      self.burst_time = burst_time
      self.arrival_time = arrival_time
      self.remaining_time = burst_time
      self.completion_time = 0
      self.service_start_time = 0
      self.color = color



def sjn_scheduling_latex(processes, cell_width, cell_height):
   doc = Document()
   doc.packages.append(Package('xcolor'))
   doc.packages.append(Package('tikz'))
   doc.packages.append(Package('geometry', options='left=0.5in, right=0.5in, top=1in, bottom=1in'))

   for process in processes:
      color_code = process.color.lstrip('#')
      doc.preamble.append(NoEscape(r'\definecolor{%s}{HTML}{%s}' % (process.name,
```

```
color_code)))

        font_scale = min(cell_width, cell_height)

        with doc.create(Section('SJN Scheduling')):
            doc.append(NoEscape(r"""
            \section*{Introduction}
            The "Shortest-Job-Next" (SJN) algorithm is one of the scheduling algorithms in operating systems designed to manage processes in a multitasking system. In this algorithm, processes that have a shorter execution time are executed earlier than other processes. In other words, SJN tries to prioritize the shortest process to minimize the waiting time of the whole system.
            """))

            time = 0
            completed_processes = []
            execution_order = []

            while len(completed_processes) < len(processes):
                available_processes = [p for p in processes if p.arrival_time <= time and p.remaining_time > 0]

                if not available_processes:
                    time += 1
                    continue

                current_process = min(available_processes, key=lambda x: x.remaining_time)
                execution_order.append((current_process.name, time, time + current_process.remaining_time))
                current_process.service_start_time = time

                with doc.create(Subsection(f'Time {time}: Process {current_process.name}')):
                    draw_process(doc, processes, execution_order, current_process, time, cell_width, cell_height,
                                 font_scale)

                time += current_process.remaining_time
                current_process.remaining_time = 0
                current_process.completion_time = time
                completed_processes.append(current_process)

            with doc.create(Subsection('Final Process Table')):
                with doc.create(Tabular('|c|c|c|c|')) as table:
                    table.add_hline()
                    table.add_row((NoEscape(r'\textbf{Process}'), NoEscape(r'\textbf{Arrival Time}'),
                                   NoEscape(r'\textbf{Burst Time}'), NoEscape(r'\textbf{Service Time}')))
                    table.add_hline()
```

```
                for process in processes:
                    table.add_row((process.name, process.arrival_time, process.burst_time,
process.service_start_time))
                    table.add_hline()

            with doc.create(Subsection('Execution Order')):
                draw_execution_order(doc, execution_order, len(processes), cell_width, cell_height,
font_scale)

        doc.generate_pdf('sjn_scheduling', clean_tex=False)



    def draw_process(doc, processes, execution_order, current_process, time, cell_width, cell_height,
font_scale,
                     completed=False):
        with doc.create(Figure(position='h!')) as fig:
            fig.append(NoEscape(r'\centering'))
            fig.append(NoEscape(r'\begin{tikzpicture}'))
            width = cell_width * len(processes)
            fig.append(NoEscape(r'\draw[thick] (0,0) rectangle (%f, %f);' % (width, cell_height)))
            x = 0
            for name, start_time, end_time in execution_order:
                process = next(p for p in processes if p.name == name)
                fig.append(NoEscape(
                    r'\node[draw, minimum width=%fcm, minimum height=%fcm, text centered, fill=%s,
font=\fontsize{%d}{%d}\selectfont] at (%f, %f) {%s};' % (
                        cell_width, cell_height, process.name, int(10 * font_scale), int(12 * font_scale),
                        x + cell_width / 2, cell_height / 2, name)))
                fig.append(NoEscape(r'\node[font=\fontsize{%d}{%d}\selectfont] at (%f, -0.3) {%d};' % (
                int(8 * font_scale), int(10 * font_scale), x, start_time)))
                x += cell_width
                fig.append(NoEscape(r'\node[font=\fontsize{%d}{%d}\selectfont] at (%f, -0.3) {%d};' % (
                int(8 * font_scale), int(10 * font_scale), x, end_time)))
            fig.append(NoEscape(r'\end{tikzpicture}'))



    def draw_execution_order(doc, execution_order, num_processes, cell_width, cell_height,
font_scale):
        with doc.create(Figure(position='h!')) as fig:
            fig.append(NoEscape(r'\centering'))
            fig.append(NoEscape(r'\begin{tikzpicture}'))
            width = cell_width * num_processes
            fig.append(NoEscape(r'\draw[thick] (0,0) rectangle (%f, %f);' % (width, cell_height)))
            x = 0
            for name, start_time, end_time in execution_order:
                process = next(p for p in processes if p.name == name)
                fig.append(NoEscape(
```

```
                r'\node[draw, minimum width=%fcm, minimum height=%fcm, text centered, font=\fontsize{%d}{%d}\selectfont, fill=%s] at (%f, %f) {%s};' % (
                    cell_width, cell_height, int(10 * font_scale), int(12 * font_scale), process.name,
                    x + cell_width / 2, cell_height / 2, name)))
            fig.append(NoEscape(r'\node[font=\fontsize{%d}{%d}\selectfont] at (%f, -0.3) {%d};' % (
            int(8 * font_scale), int(10 * font_scale), x, start_time)))
            x += cell_width
            fig.append(NoEscape(r'\node[font=\fontsize{%d}{%d}\selectfont] at (%f, -0.3) {%d};' % (
            int(8 * font_scale), int(10 * font_scale), x, end_time)))
        fig.append(NoEscape(r'\end{tikzpicture}'))
```

منابع

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. MIT press, 2022.

[2] M. T. Goodrich, R. Tamassia, and M. H. Goldwasser, *Data structures and algorithms in Java*. John wiley & sons, 2014.

[3] L. E. Sébastien, Basic tutorial to LATEX programming, 2020.

[4] T. Tantau, Tik Z and pgf The Tik Z and PGF Packages, 2007. [Online]. Available: http://sourceforge.net/projects/pgf

[5] https://visualgo.net/en.

[6] https://www.toptal.com/developers/sorting-algorithms.

[7] David Galles, https://www.cs.usfca.edu/~galles/visualization/Algorithms.html.

[8] https://www.texstudio.org/.

[9] https://www.gnu.org/licenses/gpl-3.0.en.html.

[10] https://test.pypi.org/project/algorithm-visualization-library/0.1/.

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