به نام خدا

پروژه اول درس ساختمان داده‌ها

پیاده‌سازی جستجوگر متنی ساده با استفاده از الگوریتم Inverted Index

این الگوریتم روی تعدادی فایل متنی اجرا می‌شود و سپس به ازای جستجوی یک کلمه، فایل‌هایی را که کلمه مورد نظر در آن‌ها آمده است برمی‌گرداند. این الگوریتم که در موتورهای جستجو نیز استفاده می‌شود، دارای یک درخت جستجو است که کلمه‌هایی که در فایل‌ها آمده است در آن قرار می‌گیرند. هر عنصر از این درخت خود به یک لیست اشاره می‌کند که در آن لیست، فایل‌هایی که کلمه در آنها وجود داشته است، ذخیره شده‌اند. در حالت کلی، نیازی به نگه‌داشتن تعداد و محل دقیق وقوع یک کلمه در این لیست ندارید اما برای انجام قسمت امتیازی پروژه باید این اطلاعات هم در درخت ذخیره شوند. برای آشنایی بیشتر با این الگوریتم می‌توانید به اینجا مراجعه کنید. درخت جستجوی مطرح‌شده می‌تواند به صورت درخت جستجوی سه‌تایی (TST) متوازن یا درخت پیشوندی (Trie) یا درخت جستجوی دودویی (BST) متوازن باشد.

پروژه

در این پروژه باید الگوریتم Inverted Index با توجه به موارد زیر پیاده‌سازی شود:

۱. همه‌ی ساختمان داده‌های مورد نیاز با توجه به تعریف پروژه باید پیاده‌سازی شوند. استفاده از ساختمان داده‌های آماده مجاز نیست. فقط استفاده از آرایه (ساده، ArrayList یا Vector) مجاز است. صف، پشته، لیست پیوندی، درخت یا هر ساختمان داده دیگر مورد استفاده باید پیاده‌سازی شود. (به عنوان درس توجه کنید!)

۲. در قسمت لیست‌کردن کلمات، بعضی از کلمات پرتکرار که تقریباً در همه جملات تکرار می‌شوند و تأثیر خاصی در معنای جمله ندارند مثل this، is و what که به آن‌ها stop word گفته می‌شود، در نظر گرفته نمی‌شوند. فهرستی از این کلمات در فایل ضمیمه پروژه به همین نام آمده است.

۳. اعداد، نشانه‌ها و کاراکترهای اضافه نیز در لیست‌کردن کلمات نباید در نظر گرفته شوند.

۴. این برنامه باید یک رابط کاربری داشته باشد که دارای بخشی برای گرفتن آدرس پوشه‌ای شامل متن‌های ورودی، یک دکمه برای ساختن درخت، یک قسمت برای نوشتن پرس‌وجو[1]های متنی و نمایش نتیجه آن‌ها، و یک قسمت برای انتخاب نوع ساختمان داده‌ی درخت جستجو داشته باشد. این موارد برای مشخص شدن کلیات رابط کاربری بیان شده و جزئیات پیاده‌سازی آن بر عهده شماست و اجباری در یکسان‌بودن همه رابط‌های کاربری نیست. همچنین در قسمت نتیجه‌ی پرس‌وجوهای متنی باید امکان مشاهده‌ی خروجی پرس‌وجوهایی که پیشتر وارد شده‌اند، نیز وجود داشته باشد. نمونه‌ای از رابط کاربری در پایان همین تعریف پروژه وجود دارد.

۵. برای انتخاب فایل‌های متنی که می‌خواهیم الگوریتم را روی آن‌ها اجرا کنیم، آدرس پوشه شامل آن‌ها را در قسمت Folder وارد می‌کنیم.

[1] Query

این قسمت می‌تواند به صورت گرفتن آدرس یا بازشدن یک پنجره و رفتن به محل پوشه مورد نظر انجام گیرد. (شبیه قسمت browse که در برنامه‌های مختلف موجود است).

۶. پس از مشخص‌کردن آدرس پوشه و زدن دکمه‌ی Build، الگوریتم باید روی تمام فایل‌های متنی موجود درآن پوشه اجرا شود و آماده‌ی گرفتن پرس‌وجوهای متنی باشد. در انتهای فرآیند Build، خلاصه‌ای از نتیجه‌ی کار نمایش داده شود. مانند: تعداد فایل‌های فهرست شده و تعداد کلمات موجود در درخت.

۷. شما می‌بایست درخت جستجوی خود را به هر سه روش Trie، BST متوازن و TST متوازن پیاده‌سازی نمایید. در هنگام خروج از برنامه می‌بایست ارتفاع درخت و زمان صرف‌شده (برحسب میلی‌ثانیه) را نیز چاپ کنید که به این ترتیب بتوان حافظه مصرفی و زمان مورد نیاز این سه ساختمان داده را با یکدیگر مقایسه کرد. تحلیل این‌که کدام‌یک از ساختمان داده‌های مطرح‌شده برای این کار بهتر است، بخشی از پروژه است. پیشنهاد استفاده از ساختمان داده‌های دیگر به عنوان بخش امتیازی در نظر گرفته می‌شود.

۸. دستورات متنی که برنامه باید اجرا کند:

- *اضافه‌کردن فایل*: برای این دستور، نام یک فایل را می‌دهیم و اطلاعات آن باید در ساختمان داده‌های موجود اضافه شود. این فایل از همان پوشه‌ای که آدرس آن قبلاً وارد شده است، در برنامه اضافه می‌شود.

```
>> add <document_name_in_current_folder>
>> add d1
d1 successfully added.
>> add d2
err: document not found.
>> add d3
 err: already exists, you may want to update.
```

- *حذف‌کردن فایل*: اسم فایلی که در برنامه قرار دارد را می‌دهیم و اطلاعات آن باید از ساختمان داده‌های موجود در برنامه حذف شود.

```
>> del <document_name>
>> del d1
d1 successfully removed from lists.
>> del d2
err: document not found.
```

- *به‌روزرسانی اطلاعات یک فایل*: اسم فایل مورد نظر را می‌دهیم و اطلاعات مربوط به آن در ساختمان داده‌های موجود در برنامه به‌روز می‌شود.

```
>> update <document_name>
>> update d1
d1 successfully updated.
>> update d2
err: document not found.
```

- *لیست گرفتن:*
  - ❖ لیست تمام کلمات موجود در برنامه

```
>> list -w
|brown -> d1, d2, d6
|fox -> d1, d5, d6
|bird -> d3
…
Number of words = 52
```

- ❖ لیست تمام فایل‌هایی که در برنامه موجود هستند و فهرست شده‌اند

```
>> list -l
d1, d2, d3, …
Number of listed docs = 8
```

- ❖ لیست تمام فایل‌های موجود در پوشه فعلی

```
>> list -f
d1, d2, d3, d4, …
Number of all docs = 10
```

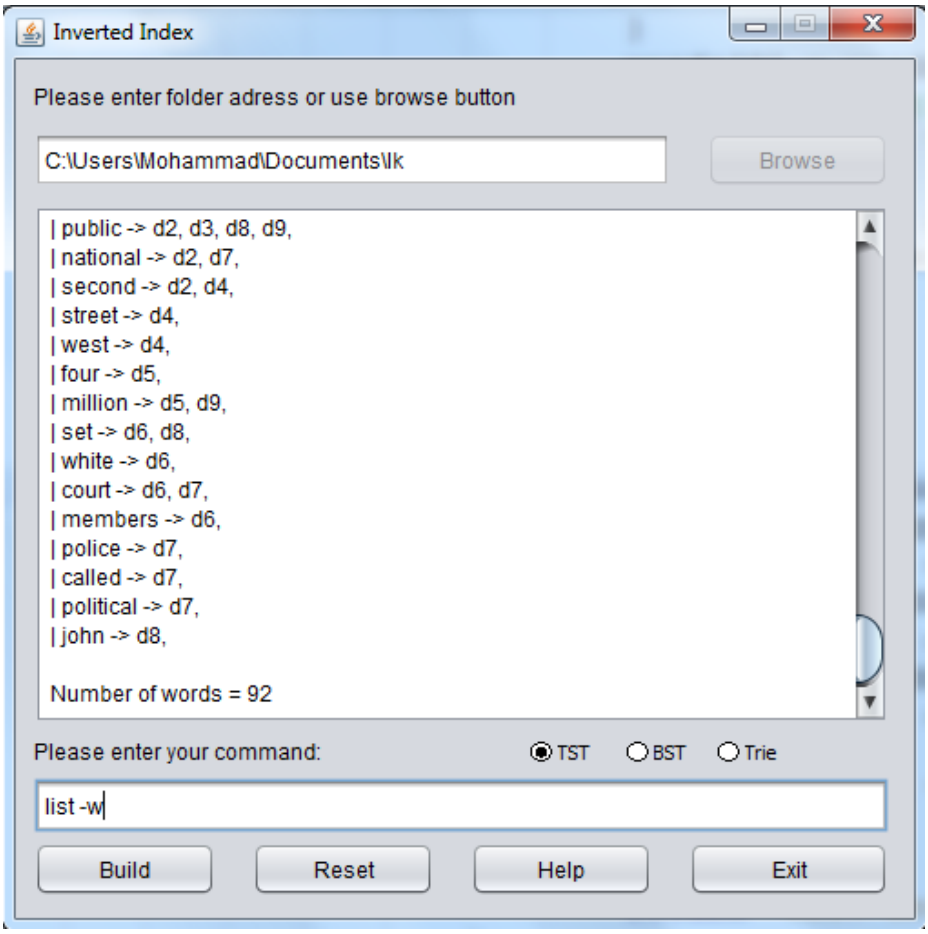
- *جستجوی عبارت*: stop wordها در عبارت مورد جستجو هم نادیده گرفته می‌شوند. در مثال زیر نتیجه حاصل از اشتراک لیست مربوط به brown و fox به دست آمده است. مشخص‌کردن محل دقیق وقوع کلمات در فایل (قسمت‌های داخل پرانتز در مثال زیر) امتیازی بوده و خلاصه‌ای از نتیجه جستجو در متن فایل است. در d1 فاصله دو عبارت brown و fox زیاد است و به همین خاطر خلاصه به چند بخش تقسیم شده است اما در d6 محل پیدا شدن دو عبارت به هم نزدیک بوده و خلاصه در یک قسمت نمایش داده شده است. می‌توانید این خلاصه را به صورت دیگری نیز نمایش دهید ولی قسمت‌هایی از متن که آورده می‌شوند باید مشابه مثال زیر باشد.

```
>> search -s "what is brown fox"
Appears in:
d1
|-> (… and brown bird gets a worm …) (… fox cached blue jay …)
d6
|-> (… here we have a brown fox in the document …)
```

- *جستجوی کلمات*: یک کلمه به شما داده می‌شود و شما باید آن را در درخت جستجوی خود بیابید و لیست فایل‌هایی که این کلمه در آن‌ها آمده است را به عنوان خروجی برگردانید.

```
>> search -w "Hello"
d1, d6, d3, …
```

۹. مشابه عملکرد cmd در ویندوز یا ترمینال در لینوکس، با زدن دکمه بالا (Arrow Key - Up) باید تاریخچه دستورات انجام‌شده توسط کاربر نمایش داده شود (دستوری که آخرین بار وارد شده است، اولین بار برگردانده می‌شود[2]) و کاربر بتواند از آن‌ها استفاده کند.



شکل ۱ – نمای ساده نمونه رابط کاربری

[2] LIFO

نکات پیاده‌سازی

۱. توجه کنید که خوانا بودن کد (مانند نام گذاری مناسب متغیرها و توابع، ماژول‌بندی منطقی، دندانه‌داربودن[3] کد، کامنت گذاری) نیز مورد ارزیابی قرار خواهد گرفت و قسمتی از نمره شما را تشکیل خواهد داد.

۲. پیاده‌سازی باید به صورت تک نفره باشد و محدودیتی برای زبان پیاده‌سازی وجود ندارد و می‌توان از هر زبانی استفاده کرد اما دقت کنید که استفاده از ساختمان داده‌های آماده و به صورت کتابخانه مجاز نیست.

۳. بحث و بررسی برای فهم الگوریتم‌ها بین دانشجویان آزاد است اما هر دانشجو موظف است به تنهایی پروژه را انجام دهد. همچنین با مواردی که تقلب و کپی‌کردن تشخیص داده شوند، برخورد خواهد شد (برای تشخیص درصد شباهت کدها از سامانه‌ی Moss استفاده می‌شود).

۵. در زمان تحویل حضوری، برنامه‌ی شما باید برای فایل‌هایی که مصحح آن‌ها را به برنامه می‌دهد و پرس‌وجوهای واردشده جواب صحیح بدهد. همچنین دانشجو باید به تمام جزئیات پیاده‌سازی کد کاملاً مسلط باشد. در مورد قسمت‌هایی از کد و نحوه عملکرد برنامه نیز از دانشجو سوال خواهد شد.

۶. برای پرسش و پاسخ درباره پروژه از این فروم استفاده کنید.

۷. موعد تحویل این پروژه تا ساعت ۲۳:۵۵ روز سه‌شنبه ۱۶ آذر ۱۳۹۵ خواهد بود. پوشه مربوط به کد پروژه را همراه با یک فایل pdf حاوی شرح انجام پروژه، نحوه اجرای برنامه و گزارش مربوط به تحلیل ساختمان داده‌های مورد استفاده را در قالب یک فایل zip به شکل زیر بارگذاری کنید. زمان و چگونگی نحوه تحویل حضوری متعاقباً اعلام می‌شود.

StudentNumber-FirstName-LastName-Proj1.zip

e.g. 9431555-Ali-Ahmadi-Proj1.zip

[3] Indentation