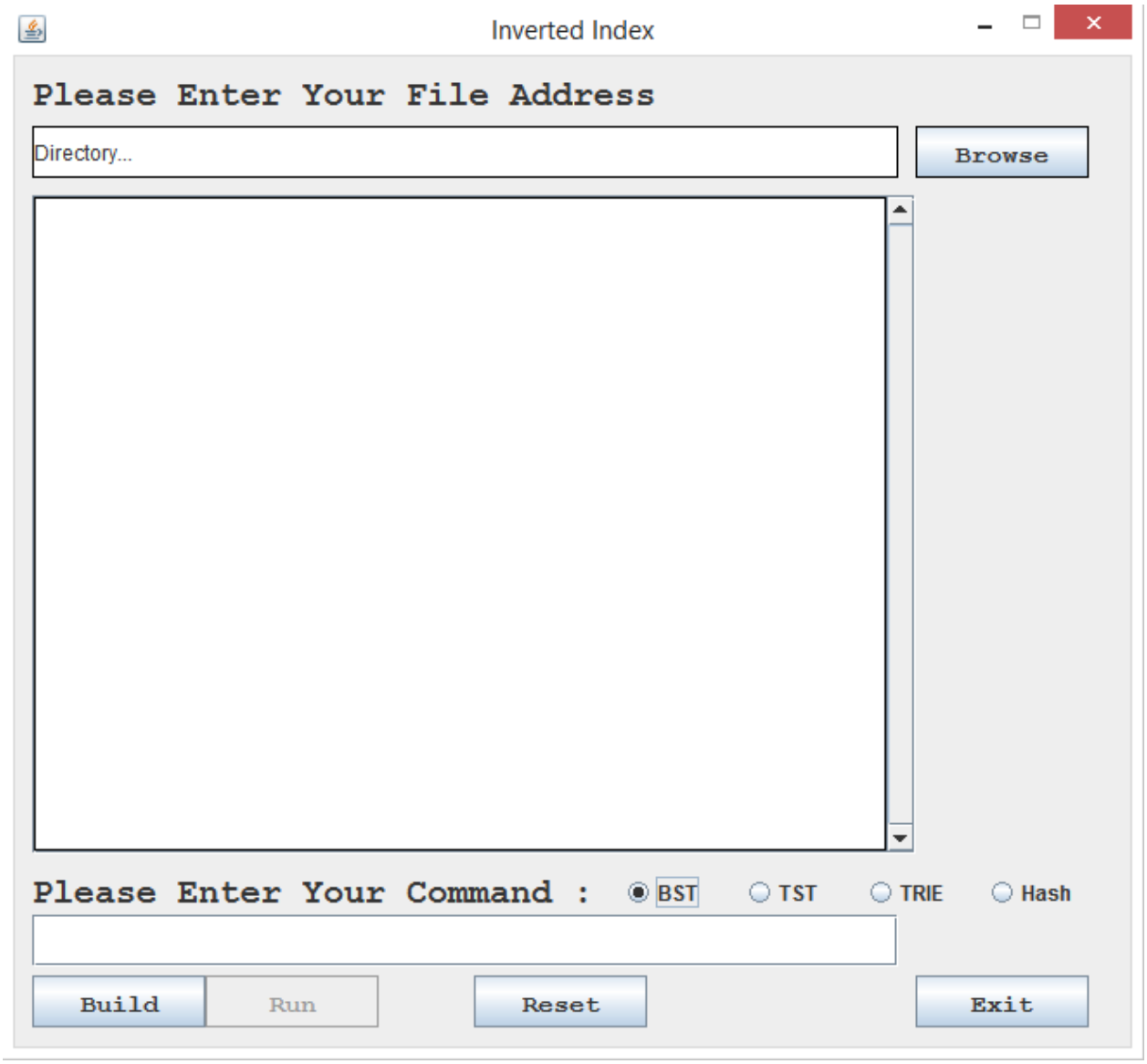
به نام خدا

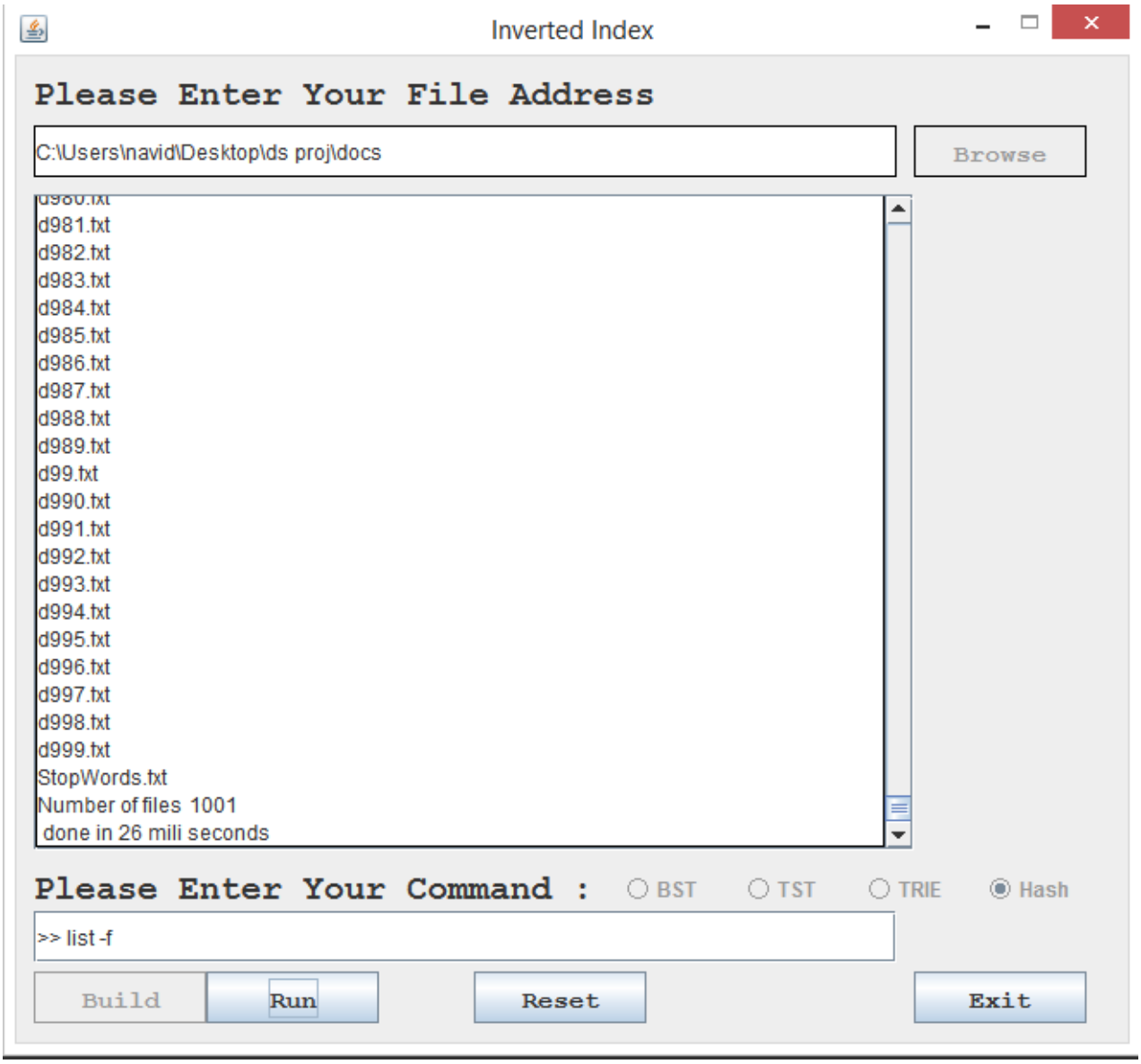
گزارش پروژه "Inverted Index"

• نگاه کلی

Inverted Index الگوریتمی برای جستجوی کلمات در یک فایل است. در این روش از درخت برای ذخیره کلمات استفاده می‌شود. تصویری از ظاهر برنامه را در زیر مشاهده می‌کنید:



در ابتدا کاربر دایرکتوری مورد نظر خود را وارد کرده از بین چهار ساختمان داده ی , BST TST, TRIE, Hash یکی را انتخاب می کند. با زدن دکمه Build برنامه ابتدا StopWords و با توجه به آن، ساختمان داده ای از کلمات موجود در فایل های درون دایرکتوری که درون StopWords قرار ندارد ، می سازد. پس از ساخت کامل درخت کاربر می تواند دستور موردنظر خود را وارد کرده و با زدن دکمه Run نتیجه را مشاهده نماید.



فهرستی از دستورات قابل اجرا توسط کاربر به شرح زیر است:

۱- >> add " file name " : فایل موردنظر را در درخت ساخته شده قرار می دهد.

۲ – " file name " del >> : فایل موردنظر را از درخت ساخته شده حذف می کند.

۳- " file name " update >> : اطلاعات موجود در فایل را در درخت ساخته شده ، به روز رسانی می کند.

۴- list –w >> : کلمات موجود در درخت را می دهد.

۵- list –f >> : لیستی از تمام فایل های موجود در دایرکتوری موردنظر را می دهد.

۶- list –l >> : لیستی از تمام فایل های موجود در درخت ساخته شده را می دهد.

۷- search –w "word" >> : فهرستی از تمام فایل هایی را که شامل word است ، می دهد.

۸- search –s "string" >> : فهرستی از فایل هایی که کلمات درون string در آن ها قرار دارد را می دهد.

۹- مشابه دستور command در ویندوز، با زدن دکمه های Up & Down Arrow Key می توان به دستوراتی که قبلا انجام شده است ، دسترسی داشت.

• ساختمان داده های استفاده شده در پروژه

۱- درخت (Tree) : این ساختمان داده جهت ذخیره کلمات استفاده می‌شود. مقایسه بین انواع درخت های موجود در برنامه (BST,TST,TRIE) در پایان گزارش انجام گرفته است.

در ضمن، تمامی توابع موردنیاز به صورت بازگشتی پیاده سازی شده است.

۲- Hash : نوع دیگری از ساختمان داده که در ذخیره کلمات از آن استفاده شده است که در آن آرایه ای به طول تعداد حروف انگلیسی (۲۶) انتخاب شده است. هر کلمه بر اساس حرف

اولش در یکی از خانه های این آرایه قرار می گیرد. در هر یک از این خانه ها از درخت BST AVL برای ذخیره ی کلمه در خانه مورد نظر استفاده شده است.

۳- پشته (Stack) : برای ذخیره دستورات انجام شده توسط کاربر برای نمایش تاریخچه دستورات مورد استفاده قرار گرفته است.

۴- لیست پیوندی(Link List) : برای ذخیره فایل هایی که هر کلمه در آن ها قرار دارد، استفاده شده است.

۵- وکتور (Vector) : این ساختمان داده هم در درخت TRIE برای ذخیره فرزندان هر گره و هم در پیاده سازی دستور پیدا کردن عبارت برای نگه داشتن فایل های مشترک بین کلمات استفاده شده است. از مزایای این ساختمان داده در مقایسه با Array نامحدود بودن طول آن هم به جهت اطمینان از عدم کمبود حافظه اختصاص داده شده است.

• مقایسه بین ساختمان داده های پیاده سازی شده

پس از انجام چند بار آزمایش ، نتایج به دست آمده در جدول زیر ثبت شده است :

*زمان ها بر حسب میلی ثانیه

	height	Create	add	del	update	list w	list f	list l	search w	search s
BST	۱۳	۵۵۰۲	۱۲	۳۹۰۰	۲۳۵۹	۳۷۶	۱۰	۷	۲	۳۳
BST (AVL)	۸	۴۵۵۰	۲۰	۸	۱۰	۴۴۲	۹	۷	۲	۱۵
TST	۱۶	۶۳۰۷	۶	۳۵۳۰	۱۹۴۰	۳۲۴	۱۴	۸	۱	۳۶
TST Balance	۱۵	۷۰۸۲	۱۲	۶	۱۷	۳۶۴	۱۲	۷	۲	۲۰
TRIE	۱۱	۴۹۰۰	۱۲	۳۷۲۰	۱۸۵۰	۳۵۰	۱۴	۷	۱	۲۱
Hash	-	۵۹۲۲	۱۱	۶	۹	۲۶۸	۱۰	۷	۲	۱۳

- با توجه به این که در دایرکتوری مربوطه ۱۰۰۰ فایل موجود است و بر اساس نتایج بدست آمده ، می توان به طور شهودی بیان کرد که BST AVL و Hash از میان ساختمان داده های موجود در استفاده برای تعداد فایل های زیاد عملکرد بهتری دارند. با این که ساختن Hash زمان بیشتری برده است اما در بقیه بخش ها نتایج بدست آمده بهتر از سایر ساختمان داده هاست.

- چالش ها

حذف گره های اضافی از درخت پس از حذف یک فایل در درخت های BST,TRIE

استفاده از یک object برای تمامی درخت ها برای کوتاه شدن کد در حالی که هر درخت یک کلاس مجزا داشته، توابع یکسان دارند اما argument آن ها متفاوت است: ساختن یک interface و implement کردن تمامی درخت ها از آن و فراخوانی توابع هر کلاس در تابع override مربوط به آن تابع.

بالانس کردن درخت های BST,TRIE