

دانشکده مهندسی کامپیوتر

دانشگاه اصفهان

**تکلیف دوم بازیابی اطلاعات**

استاد: دکترمحمد مهدی رضاپور

مهروالسادات نوحی

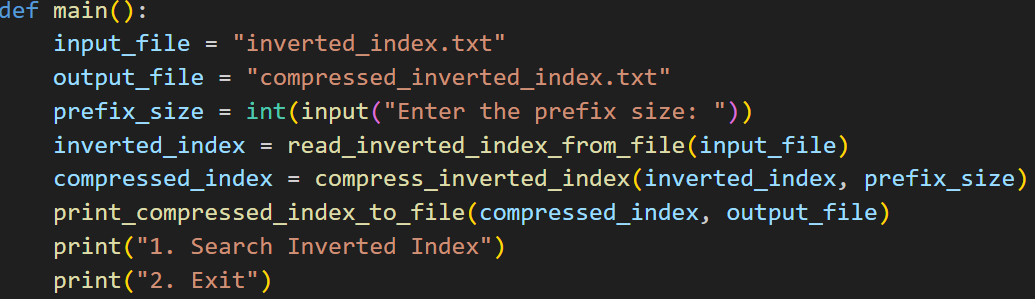
993613061

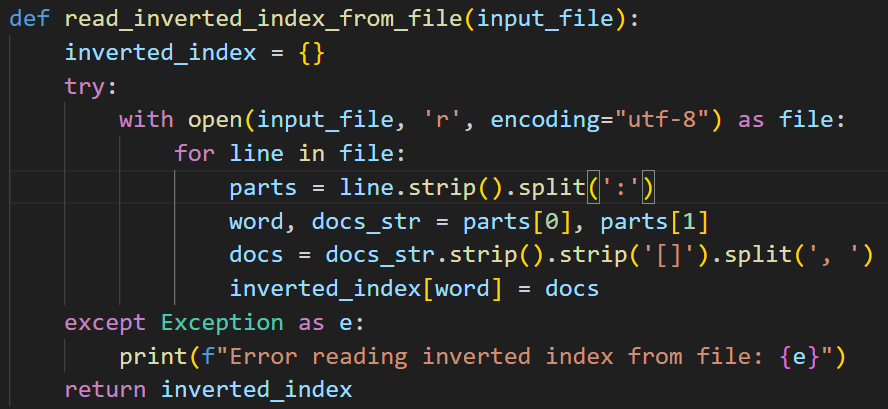
بهار 1403

صورت تمرین: شما می‌بایست با استفاده از تکنیک فرانت کدینگ، نسخه فشرده شده‌ای از ایندکس تکلیف شماره 1 ایجاد کرده و جایگزین کنید به شکلی که در پاسخ دهی به کوئری های بولینی کاربر هیچ تفاوتی ایجاد نشود. کدها می‌بایست با زبان پایتون نوشته شود. برنامه می‌بایست قابل تست باشد وگرنه نمره‌ای به تکلیف تعلق نمی‌گیرد.

گام اول:

در تکلیف قبل ما تونستیم جدول را ایجاد کنیم و سرچ بزنیم حال باز میاییم و شروع به خواندن جدول می‌کنیم و در یک دیکشنری می‌ریزیم که کلید آن کلمه یا توکن و مقدار آن لیست داکیومنت‌ها هست و حالا میخواییم front\_coding بزنیم. فعلا در این گام فقط از فایل جدول معکوس خوندیم و دیکشنری inverted\_index قرار دادیم و قرار است خروجی این الگوریتم front\_coding را output می‌ریزیم.





توضیحات:

ابتدا ببینیم الگوریتم فرانت کدینگ چجوری کار میکنه. الگوریتم برای فشرده کردن جدول معکوس از لحاظ سایز و ذخیره در حافظه است. به این صورت اگر مثلا داشته باشیم hello , hell در اینجا حروف hell مشترکه و برای هر دو کلمه حافظه داریم می‌گیریم و به جای این کار می‌توانیم کلمه hell که به آن prefix گفته می‌شود را یکبار ذخیره کنیم و برای بقیه حروف باقیمانده که به اصطلاح suffix گفته می‌شود به صورت عادی ذخیره می‌کنیم یک مثال ببینیم:

lea#f: ["'1'", "'13'"]

lea#ves: ["'1'", "'13'"]

lea#ding: ["'1'", "'4'", "'8'", "'13'", "'15'", "'17'"]

lea#d: ["'1'", "'2'", "'4'", "'7'", "'8'", "'10'", "'11'", "'13'", "'15'"]

الان lea 4 بار ذخیره شده است که اینجا اگر مثلا 10000 کلمه دارای lea باشد الکی حافظه گرفته میاییم و فقط lea را یکبار ذخیره می‌کنیم.

مثال دوم:

entries = [

("prefix", "fix1", "doc1"),

("prefix", "fix2", "doc2"),

("prefix", "fix1", "doc3"),

("dummy", "suffix", "doc4"),

("prefix", "fix3", "doc5")

]

الان pre عضو مشترک هست مجدد در واقع همان prefix من هست که 4بار تکرار شده و من دوست دارم اینجوری بشه.

pre:

fix1: doc1

fix2: doc2

fix3: doc5

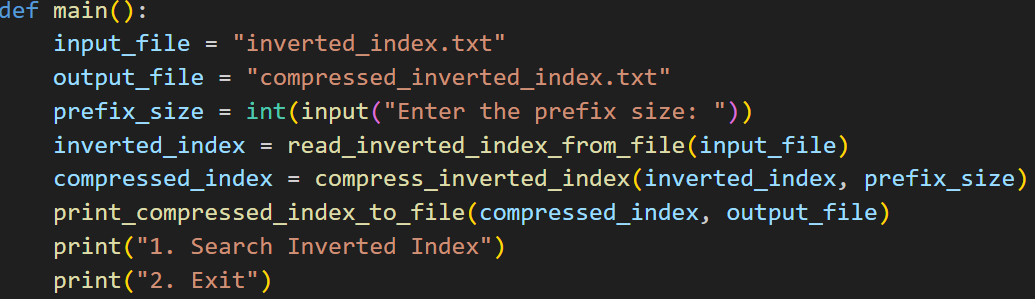
dummy:

suffix: doc4

خب حالا الگوریتم چیه؟

من گفتم بیاییم یک دیکشنری تعریف کنیم که کلید آن کلمه مثلا pre در این مثال هست و درمقدار آن یک دیکشنری دیگر که کلید آن suffix من باشد و مقدار دیکشنری دوم لیستی از داکیومنت‌ها باشد. خب بریم پیاده سازی کنیم دیگه!

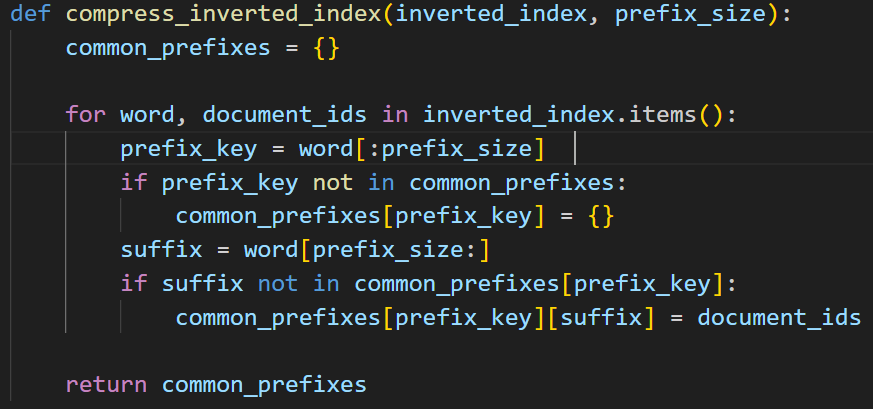
نکته: باید از کاربر بپرسیم مثلا چند کارکتر حروف مشترک داشته باشد یعنی اون بلاک چندباشد.برای همین از کاربر میخوام بگه چند کارکتر مد نظرشه.



گام دوم:

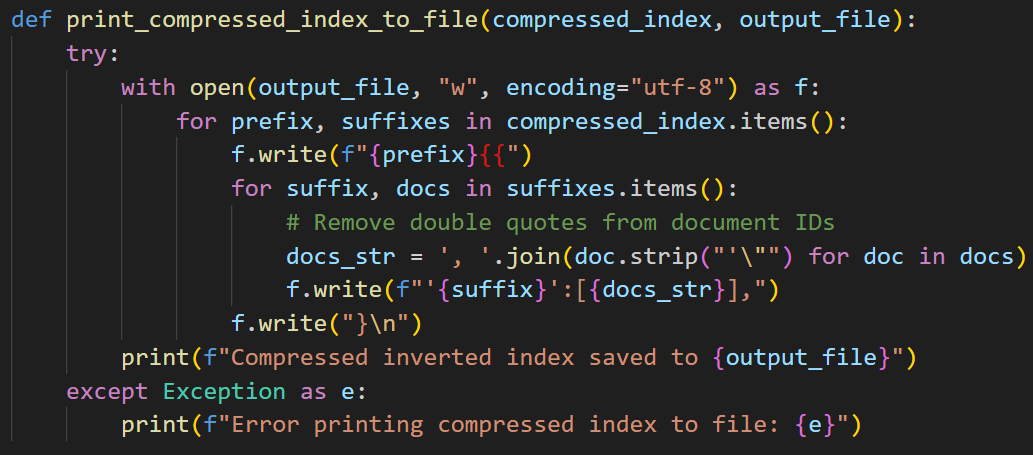
بریم تابع فشرده سازی جدول معکوس را ببینیم:

در ابتدا به اندازه prefix\_size از کلمه به عنوان prefix\_key در نظر می‌گیریم و اگر در دیکشنری common\_prefixes نبود اضافه می‌کنیم و برای این prefix یک دیکشنری خالی دیگر در نظر می‌گیریم و میریم سراغ بخش Suffix اگر در common\_prefixes نبود به عنوان کلید دیکشنری داخلی اضافه می‌کنیم و برای مقدار لیست داکیومنت قرار می‌دهیم. و فشرده سازی انجام شده است و خروجی را compress می‌ریزیم.



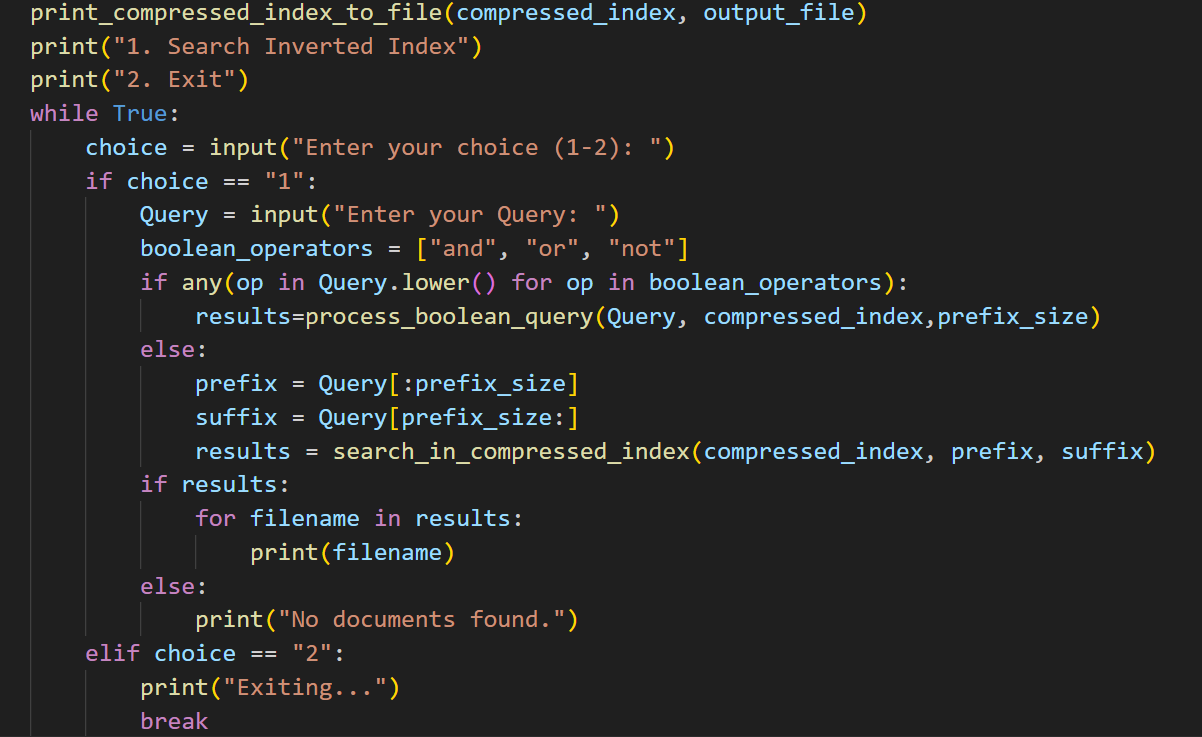
گام سوم:

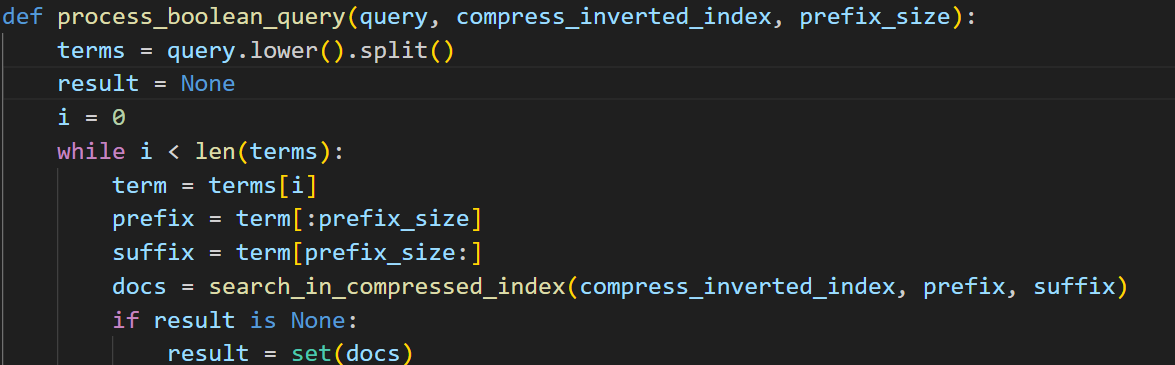
حالا این را در فایل خروجی می‌ریزیم.

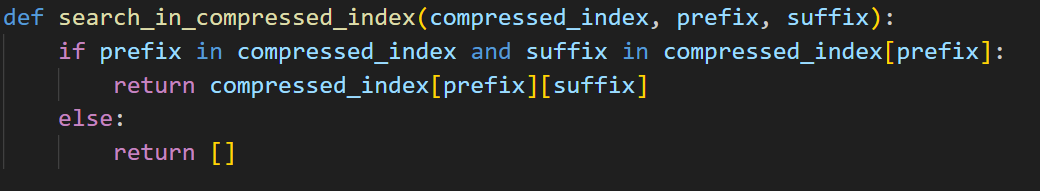


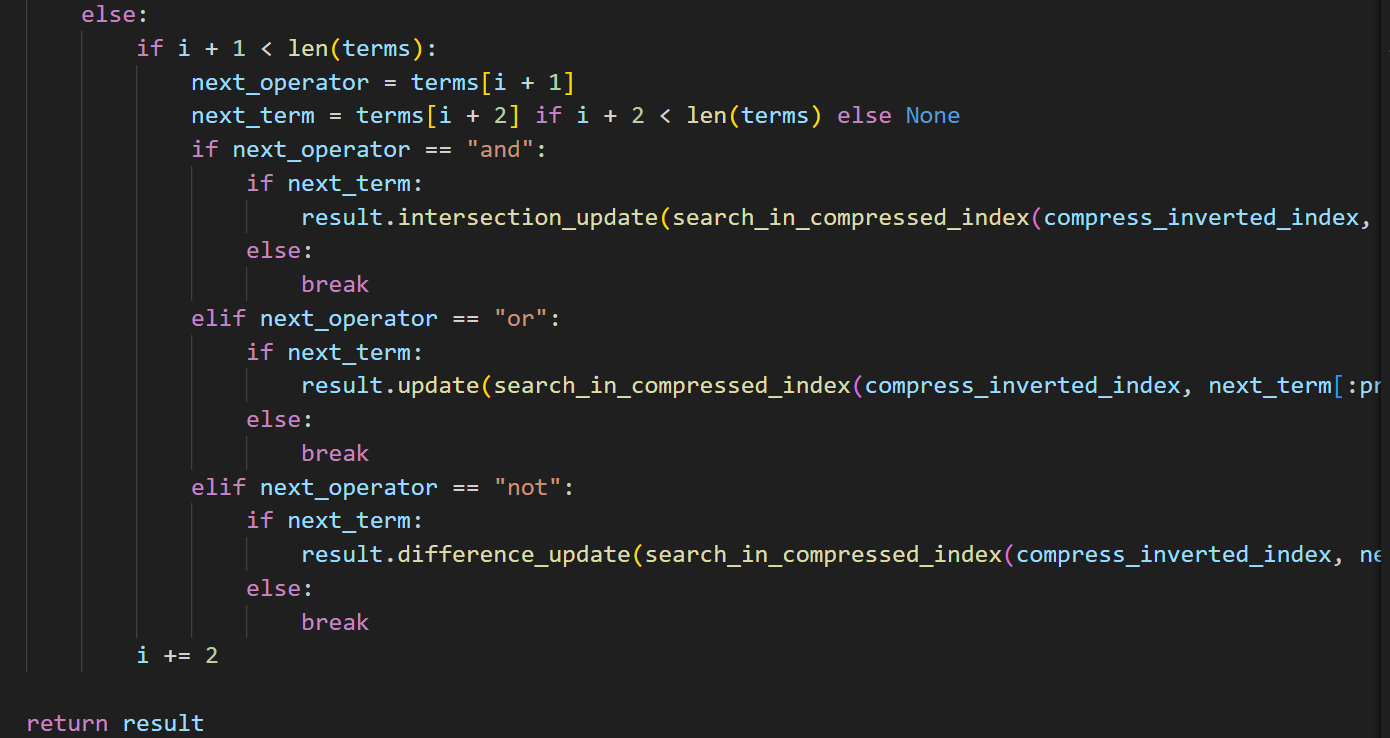
گام چهارم:

حالا بریم سراغ سرچ کاربر. من اینجوری فکر کردم که دو دسته کوعری از سمت کاربر داریم یکی کوعری ساده تک کلمه مثلا hello و دسته دوم کوعری‌های بولینی . اول کوعری کاربر را سرچ کنیم ببینیم اپراتور داره یا کلمه ساده است. اگر کلمه ساده باشه سرچ ساده است و اگر کوعری بولینی باشه تابع بولین فراخوانی می‌شود.

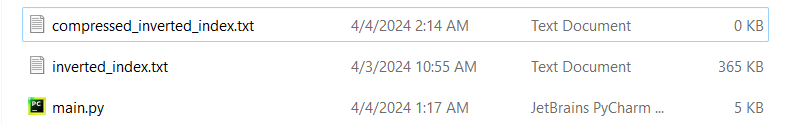




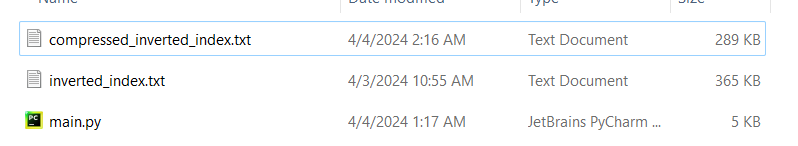




الان compressed\_inverted\_index قبل و بعد اجرای برنامه ببینیم. قبل از اجرا فایل را پاک کردم.

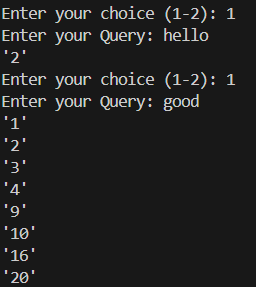


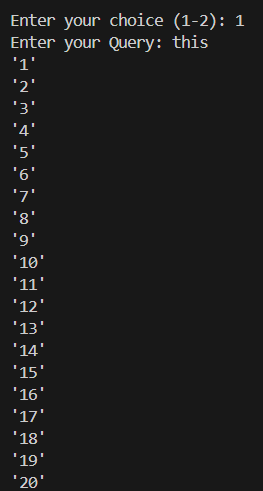
مشخص است که سایز از 365 به 289 کاهش یافت البته این به شرط این است که کاربر برای ورودی 3 کارکتر مشترک در نظر گرفته باشد.

تست برنامه:

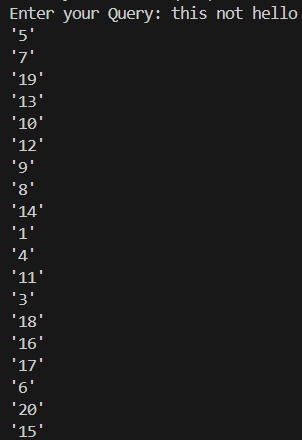
الان باید خروجی سرچ کاربر و کوعری هیچ تغییری نداشته باشیم صرفا سایز و حافظه کمتر داشته باشیم.

نکته: الان کلمه hello فقط در داکیومنت 2 است و کلمه this در هر 20 داکیومنت موجود است حال کوعری بولینی داشته باشیم و نباید در خروجی داکیومنت 2 باشد اگر اپراتور مثلا not باشد.





مشخص است که داکیومنت 2 نیست پس کوعری بولینی نیز درست است.



حالا همین را با and برویم : خروجی مورد انتظار ما 2 می‌باشد.

