به نام خدا

تمرین ۲

درس بازیابی اطلاعات

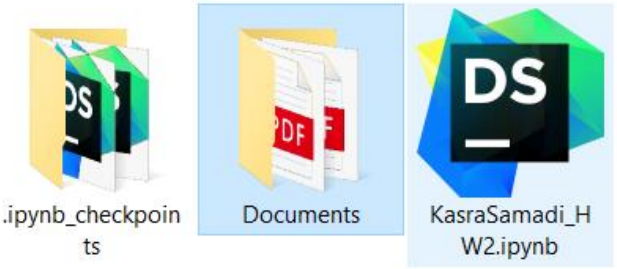
استاد : دکتر رضاپور

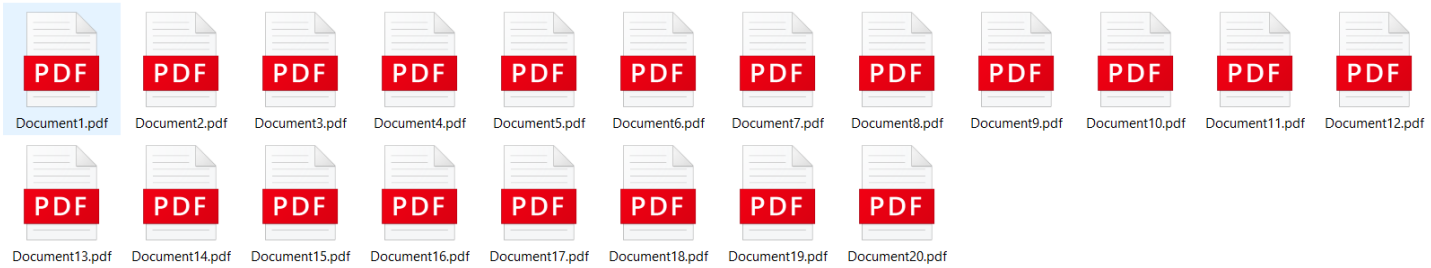
کسری صمدی<۹۹۳۶۲۳۰۳۰>

بهمن ۱۴۰۲

شما می‌بایست با استفاده از تکنیک فرانت کدینگ، نسخه فشرده شده‌ای از ایندکس تکلیف شماره ۱ ایجاد کرده و جایگزین کنید به شکلی که در پاسخ‌دهی به کوئری‌های بولینی کاربر هیچ تفاوتی ایجاد نشود. کدها می‌بایست با زبان پایتون نوشته شود. برنامه می‌بایست قابل تست باشد وگرنه نمره‌ای به تکلیف تعلق نمی‌گیرد. لذا توضیحات کافی به صورت تصویری در رابطه با برنامه توسعه داده شده در قالب یک فایل ورد پیوست تکلیف باشد.

همانطور که در این تمرین ۱ خواسته شده بود، ابتدا همان ۲۰ مقاله با فرمت pdf با نام‌های Document1 تا Document20 نام‌گذاری شده و در پوشه‌ای با نام Documents در کنار فایل اجرایی کد پایتون ذخیره شده است.





حال به سراغ بخش کد پایتون می‌رویم :

در این بخش ابتدا کتابخانه‌های موردنیاز را ایمپورت می‌کنیم.

```
In [1]: import nltk
        import PyPDF2
        import string
        from ordered_set import OrderedSet
```

کتابخانه nltk : برای پردازش زبان طبیعی استفاده می‌شود و در این تمرین، برای tokenize کردن جملات موجود در مقالات استفاده شده است.

کتابخانه pyPDF2 : برای استخراج متن مقالات کاربرد دارد.

کتابخانه string : برای کار با دیتاتایپ استرینگ استفاده می‌شود و در این تمرین، برای حذف علائم نگارشی (Remove punctuation) استفاده شده است.

کتابخانه ordered_set : یک داده ساختار ست (مجموعه) است که عناصر با ترتیب در آن ذخیره می‌شوند. همانطور که می‌دانید در مجموعه‌ها (Set)، عناصر تکراری ذخیره نمی‌شوند.

```
In [2]: documents_folder_name = "Documents"
        documents_name = "Document"
        documents_count = 20
        num_common_letters = 2
        compressed_inverted_index = {}
```

حال در این قسمت، متغیرهای ثابت را تعریف می‌کنیم که به ترتیب شامل نام فولدر ذخیره کننده مقالات (Documents)، نام ثابت برای هر مقاله (Document) و تعداد مقالات (۲۰) است.

متغیر num_common_letters تعداد حروف مشترک و یکسان کلمات را مشخص می‌کند.

همچنین یک دیکشنری (compressed_inverted_index) که ایندکس‌های معکوس به صورت فشرده‌شده را در خود ذخیره می‌کند، تعریف شده است. کلیدهای این دیکشنری، حروف مشترک و یکسان کلمات استخراج شده از مقالات به طول num_common_letters است و مقادیر (value) این دیکشنری، خود یک دیکشنری دیگر است که کلیدهایش، بقیه‌ی حروف کلمات است و مقادیر (value) آن، مجموعه‌ای (set) از نام‌های مقالاتی است که کلمه موردنظر در آن مقالات وجود دارد.

در این تمرین چون مقدار num_common_letters برابر ۲ تعریف شده است، کلیدهای موجود در دیکشنری compressed_inverted_index همگی دو حرفی هستند و کلماتی که دو حرف اول یکسانی داشته باشند، در یک سطر از دیکشنری compressed_inverted_index قرار می‌گیرند و سپس با توجه به سایر حروف غیر مشترکشان که به عنوان کلیدهای دیکشنری دوم (دیکشنری که به عنوان مقدار (value) دیکشنری compressed_inverted_index تعریف شده است)، درنظر گرفته شده‌اند، کلمات دسته‌بندی شده و در value دیکشنری دوم (داخلی) که یک OrderedSet است، نام مقاله‌ی شامل کلمه‌ی موردنظر ذخیره می‌شود.

```
In [3]: for i in range(documents_count):
            doc_name = f"{documents_name}{i + 1}"
            doc_address = fr"{documents_folder_name}\{doc_name}.pdf"

            with open(doc_address, "rb") as pdf_file:
                read_pdf = PyPDF2.PdfReader(pdf_file)
                number_of_pages = len(read_pdf.pages)
```

در این قسمت، نام‌های مقالات (Document1 تا Document20) همچنین آدرس آن‌ها (\Documents\Document1..20) به ازای هر مقاله ساخته می‌شود و با استفاده از کتابخانه PyPDF2 مقاله موردنظر خوانده شده و تعداد صفحات آن بدست می‌آید.

```
        number_of_pages = len(read_pdf.pages)
        for page_num in range(number_of_pages):
            page = read_pdf.pages[page_num]
            page_content = page.extract_text()

            lower_text = page_content.lower()
            text_without_punctuation_marks = lower_text.translate(str.maketrans('', '', string.punctuation))

            tokens = nltk.word_tokenize(text_without_punctuation_marks)
```

حال با استفاده از تعداد صفحات بدست آمده برای مقاله موردنظر، متن هر صفحه از آن استخراج می‌شود. سپس متن بدست آمده به حروف کوچک (lowercase) تبدیل شده و علائم نگارشی (punctuation marks) از آن حذف می‌شود.

بعد از آن، با استفاده از کتابخانه nltk، متن موردنظر tokenize (کلمه کلمه) می‌شود و کلمات به دست آمده از متن هر صفحه، در آرایه tokens ذخیره می‌شود.

```
tokens = nltk.word_tokenize(text_without_punctuation_marks)

for word in tokens:
    if len(word) > num_common_letters:
        common_letters = word[0:num_common_letters]
        uncommon_letters = word[num_common_letters:]

        if common_letters not in compressed_inverted_index:
            compressed_inverted_index[common_letters] = {}
            compressed_inverted_index[common_letters][uncommon_letters] = OrderedSet()
            compressed_inverted_index[common_letters][uncommon_letters].add(doc_name)

        elif common_letters in compressed_inverted_index:
            if uncommon_letters not in compressed_inverted_index[common_letters]:
                compressed_inverted_index[common_letters][uncommon_letters] = OrderedSet()
                compressed_inverted_index[common_letters][uncommon_letters].add(doc_name)

            elif uncommon_letters in compressed_inverted_index[common_letters]:
                compressed_inverted_index[common_letters][uncommon_letters].add(doc_name)

    else:
        if word not in compressed_inverted_index:
            compressed_inverted_index[word] = {}
            compressed_inverted_index[word]["NONE"] = OrderedSet()
            compressed_inverted_index[word]["NONE"].add(doc_name)

        elif word in compressed_inverted_index:
            try:
                compressed_inverted_index[word]["NONE"].add(doc_name)

            except KeyError:
                compressed_inverted_index[word]["NONE"] = OrderedSet()
                compressed_inverted_index[word]["NONE"].add(doc_name)
```

حال به ازای هر کلمه موجود در آرایه tokens ، با استفاده از طول num_common_letters ، حروف مشترک از ابتدای هر کلمه در common_letters ذخیره می‌شود و سایر حروف یعنی از ایندکس num_common_letters تا آخر، در uncommon_letters ذخیره می‌شود. حال به توجه به حالات مختلفی که ممکن است رخ دهد، دیکشنری compressed_inverted_index (کلیدهای آن common_letters و مقادیر آن یک دیکشنری دیگری است که کلیدهای آن uncommon_letters و مقادیر آن یک OrderedSet است که لیست نام‌های مقالات شامل کلمه‌ی مورد نظر در آن ذخیره می‌شود) مقداردهی و ساخته می‌شود.

نکته ۱ : از مجموعه یا Set به این علت استفاده شده است که نام مقالاتی که کلماتی تکراری در آن‌ها وجود دارند، فقط یک بار به مجموعه اضافه شود. برای مثال در Document1 ، ۲۰ با کلمه word1 به کار رفته است و با این کار در inverted_index با کلید word1 یک OrderedSet با یک عضو Document1 ذخیره می‌شود و مقاله Document1 فقط یکبار در OrderedSet مربوط به کلید word1 ذخیره می‌شود. نکته ۲ : از OrderedSet به این علت استفاده شده است که نام‌های مقالات به ترتیب در مجموعه ذخیره شوند.

```
[4]: for term, documents in compressed_inverted_index.items():
         print(term, "->", ", ".join(documents))
```

حال compressed_inverted_index ساخته شده را با قطعه کد بالا نمایش می‌دهیم که بخشی از خروجی آن مطابق تصویر زیر است:

```
ef -> ficient, fectively, ficiently, fective, fi, fectiveness, fect, ficiency, ficiencyperformance, ficiency-, cacy, cient
ly, cient, ficacy, ros, fcient, forts, fort, fects, f, NONE, ficiencyadjusted, iciaries, t, fectivefield, tdescription, ts
vi -> deo, a, deos, deoframe, sion, deomatch, sual, NONE, deosmean, jayanarasimhan, ew, sualizes, sualize, s, sualizations, su
omotor, kas, tal, d, or, ews, eira, rtual, sed, sible, ability, tbased, ncent, des, cky, ra, lleurbanne, olet, rtue, egi, ewed
, sa, dual, sibly, sibility, siting, sit, able, rtues, ding, ctory, olate, ating, ded, olates, sas, rtually, gorous, ewpoint,
sited, sioned, duals, de, tali, to, ana, rginia, jay, ce, o, rtualphoton, scous, scosity, olation, sh21, rtualities, rtuality,
olated, gdor, scosities
ob -> ject, jects, serve, jectagnostic, jectspecific, jective, tained, sta, stacles, stacle, tain, tains, NONE, jectlevel, tai
ning, served, serving, servations, jectives, jective8, jec, tai, vious, ligations, lique, servatoire, onyo, jectives10, jectiv
ely, servation, jectivedriven, jectives-, jections, jection, ligation, jected, c, li, aidli, i, servables, servable, eys
se -> gmentation, cond, g, quence, conds, mi, tting, n, veral, e, nsitivity, nsory, misupervised, parating, gment, lfpropagati
ons, lfpropagation, t, quences, lfattention, ries, tup, NONE, ttings, gments, ction, ts, condly, gmenting, vere, gmen, mantic,
amseg, ams, ong, lection, quencetosequence, gmentations, arch, rvers, minal, es, ntences14, ntence, rve, ntations, lfdriving,
nsor, ction32followedbyadiffusionmodelsection33wedetailthesimulation, lecting, verely, ttings1, ctions, rves, lect, ang, rvati
ons, nsors, rvice, quentially, lected, o, iberlich, rpent, pehri, pehriusceduzalan, lective, lectivity, parate, rpentb, rpentl
, rpenth, idenari, gmenta, ssment, archers, gmentat, quenc, gmentati, ara, daei, gme, gdiff, eding, ed, eded, vered, nse, ems,
en, minars, bastian, ctors, tuparxiv240317639v1, nting, am, condorder, lls, ek, llin, lffinancing, ll, mimartingale, mivariatio
n, c, condor, rved, pahvand, rvices, ller, llers, yed, nses, fidmazgi, ch, ctor, eks, izing, eking, curing, eker, curity, ven,
```

با توجه به مقدار num_common_letters کلیدهای این دیکشنری (compressed_inverted_index) همگی دو حرفی هستند و مقادیر آن شامل سایر حروف کلمات است که به عنوان کلید در دیکشنری دوم ذخیره شده است. برای مثال دو حرف ef اولین کلید دیکشنری compressed_inverted_index است که مقدار آن یک دیکشنری دیگری است که اولین کلید آن، ficient است و مقدار (value) آن که در تصویر بالا نمایش داده نشده است، لیستی (OrderedSet) از نام مقالاتی است که شامل کلمه‌ی efficient است. نام مقالات با ترتیب و به صورت صعودی در این لیست ذخیره شده‌اند و همچنین نام تکراری در آن‌ها وجود ندارد.

```
def find_word(word_str):
    if len(word_str) > num_common_letters:
        commonLetters = word_str[0:num_common_letters]
        uncommonLetters = word_str[num_common_letters:]
        if commonLetters in compressed_inverted_index:
            if uncommonLetters in compressed_inverted_index[commonLetters]:
                res = compressed_inverted_index[commonLetters][uncommonLetters]
                res_dict = dict({word_str: res})
                return res_dict
            else: # not found
                return None
        else: # not found
            return None
    else:
        if word_str in compressed_inverted_index:
            res = compressed_inverted_index[word_str]["NONE"]
            res_dict = dict({word_str: res})
            return res_dict

        else: # not found
            return None
```

در تصویر بالا، تابع find_word برای جستوجوی کلمه‌ی ورودی (word_str) تعریف شده است. این تابع در صورت یافتن کلمه word_str ، کلمه را همراه با لیست نام‌های مقالاتی که شامل آن کلمه هستند، به صورت یک دیکشنری dict({word_str: res}) باز می‌گرداند و در صورتی که کلمه word_str در compressed_inverted_index یافت نشود، مقدار None ، ریترن می‌شود.

```
ans1 = int(input("(1) Single word Search\n(2) Combined search\n "))
if ans1 == 1:  # Single word Search
    word = input("Write your word to search : (Input must be in lowercase letters) ")
    res_dict = find_word(word)
    if res_dict is not None:
        for term, documents in res_dict.items():
            print(term, "->", ", ".join(documents))

    else:
        print(f"\'{word}\' NOT FOUND")
```

حال نوبت ورودی گرفتن از کاربر و تست برنامه است. در اینجا کاربر می‌تواند دو نوع جستوجو را انجام دهد:

۱- جستوجو تک کلمه‌ای

۲- جستوجو ترکیبی

کاربر با وارد کردن شماره‌ی جستوجوی موردنظر خود، می‌تواند جستوجو را انجام دهد.

۱- جستوجو تک کلمه‌ای

در جستوجو تک کلمه‌ای کاربر فقط یک تک کلمه را وارد می‌کند و مقالاتی که شامل آن کلمه هستند به او نمایش داده می‌شود و در صورت یافت نشدن کلمه‌ی موردنظر در compressed_inverted_index، پیغام "'word' NOT FOUND" نمایش داده می‌شود.

نمونه ورودی‌های این بخش در زیر آمده است :

```
(1) Single word Search
(2) Combined search
1
```

```
(1) Single word Search
(2) Combined search
 1

Write your word to search : (Input must be in lowercase letters)
and
```

```
Write your word to search : (Input must be in lowercase letters) and
and -> Document1, Document2, Document3, Document4, Document5, Document6, Docume
nt7, Document8, Document9, Document10, Document11, Document12, Document13, Docu
ment14, Document15, Document16, Document17, Document18, Document19, Document20
```

کلمه‌ی and در compressed_inverted_index یافت شد و لیستی از نام مقالاتی که در آن‌ها، این کلمه وجود دارد، نمایش داده شده است.

```
(1) Single word Search
(2) Combined search
 1
Write your word to search : (Input must be in lowercase letters) laptop
'laptop' NOT FOUND
```

کلمه laptop در compressed_inverted_index یافت نشد و پیغام ‘laptop’ NOT FOUND چاپ شده است.

۲- جستوجو ترکیبی

```
elif ans1 == 2 : # Combined search
    print("(1) AND")
    print("(2) OR")
    op = int(input("Enter your Operator : "))
    word1, word2 = input("Write your Two words to search (Separated by space) :
    res_dict1 = find_word(word1)
    res_dict2 = find_word(word2)
```

در جستوجوی ترکیبی، کاربر می‌تواند دو نوع جستوجو را انجام دهد:

۱- جستوجو ترکیبی AND

۲- جستوجو ترکیبی OR

کاربر با وارد کردن شماره‌ی جستوجوی موردنظر خود، می‌تواند جستوجو را انجام دهد.

بعد از مشخص شدن نوع جستوجو، کاربر باید دو کلمه‌ای را که می‌خواهد به صورت AND یا OR جستوجو کند، با فاصله (space) از هم وارد کند.

۱- جستوجو ترکیبی AND

```
if op == 1:  # AND
    if res_dict1 is not None and res_dict2 is not None:
        value_list1 = res_dict1[word1]
        value_list2 = res_dict2[word2]
        and_result = value_list1.intersection(value_list2)
        if len(and_result) > 0:
            and_result_dict = dict({f"{word1} & {word2}": and_result})
            for term, documents in and_result_dict.items():
                print(term, "->", ", ".join(documents))
        else:
            print(f"There is No document intersection between \'{word1}\' and \'{word2}\'")
    else :
        print(f"\'{word1}\' or \'{word2}\' NOT FOUND")
```

کاربر در این نوع جستوجو، دو کلمه‌ای را که می‌خواهد به صورت AND جستوجو شوند، با فاصله از هم وارد می‌کند و سپس لیستی از مقالاتی که شامل توامان آن دو کلمه هستند، نمایش داده می‌شود. اگر کلمه ۱ یا کلمه ۲ در دیکشنری compressed_inverted_index نباشند، پیغام NOT FOUND چاپ می‌شود و اگر دو کلمه‌ی موردنظر کاربر، اشتراکی از لحاظ نام مقالاتی که شامل آن‌ها هستند، نداشته باشند، پیغام

"'\There is No document intersection between \'{word1}\' and \'{word2}" نمایش داده می‌شود.

نمونه ورودی‌های این بخش در زیر آمده است :

```
(1) Single word Search
(2) Combined search
 2
(1) AND
(2) OR

Enter your Operator : 1
```

```
Write your Two words to search (Separated by space) :
object video
```

```
Write your Two words to search (Separated by space) : object video
object & video -> Document1, Document3, Document6
```

به عنوان ورودی، دو کلمه‌ی video و object وارد شد و به عنوان خروجی، لیستی از نام‌های مقالاتی که شامل این دو کلمه به صورت and هستند، نمایش داده شده است.

۲- جستوجو ترکیبی OR

```
    elif op == 2:  # OR
        if res_dict1 is not None or res_dict2 is not None:
            if not res_dict1:  # Check if value_list1 is empty
                or_result = res_dict2[word2]
                or_result_dict = res_dict2
            elif not res_dict2:  # Check if value_list2 is empty
                or_result = res_dict1[word1]
                or_result_dict = res_dict1
            else:
                value_list1 = res_dict1[word1]
                value_list2 = res_dict2[word2]
                or_result = value_list1.union(value_list2)
                or_result_dict = dict({f"{word1} | {word2}": or_result})

            for term, documents in or_result_dict.items():
                print(term, "->", ", ".join(documents))

        else :
            print(f"\'{word1}\' and \'{word2}\' NOT FOUND")

    else:
        print("Invalid Input")
else:
    print("Invalid Input")
```

کاربر در این نوع جستوجو، دو کلمه‌ای را که می‌خواهد به صورت OR جستوجو شوند، با فاصله از هم وارد می‌کند و سپس لیستی از مقالاتی که شامل آن دو کلمه به صورت OR هستند، نمایش داده می‌شود. اگر کلمه ۱ و کلمه ۲ در دیکشنری compressed_inverted_index نباشند، پیغام NOT FOUND چاپ می‌شود. اگر فقط کلمه ۱ در compressed_inverted_index نباشد، به عنوان نتیجه، لیست نام‌های مقالاتی که کلمه ۲ در آن‌ها وجود دارد، چاپ می‌شود و همچنین اگر فقط کلمه ۲ در compressed_inverted_index نباشد، به عنوان نتیجه، لیست نام‌های مقالاتی که کلمه ۱ در آن‌ها وجود دارد، چاپ می‌شود و اگر هر دو کلمه در compressed_inverted_index باشد، اجتماع یا OR لیست مقالاتی که شامل آن دو کلمه هستند، چاپ می‌شود.

نمونه ورودی‌های این بخش در زیر آمده است :

```
(1) Single word Search
(2) Combined search
 2
(1) AND
(2) OR

Enter your Operator : 2
```

```
Write your Two words to search (Separated by space) : object video
object | video -> Document1, Document3, Document6, Document8, Document9, Document11, Document2
```

به عنوان ورودی، دو کلمه‌ی video و object وارد شد و به عنوان خروجی، لیستی از نام‌های مقالاتی که شامل این دو کلمه به صورت or هستند، نمایش داده شده است.

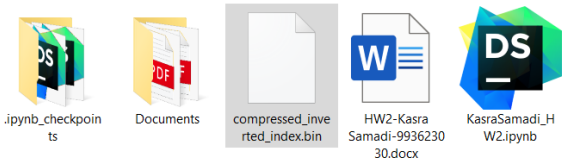
```
(1) Single word Search
(2) Combined search
 2
(1) AND
(2) OR
Enter your Operator : 2

Write your Two words to search (Separated by space) :
kasra object
```

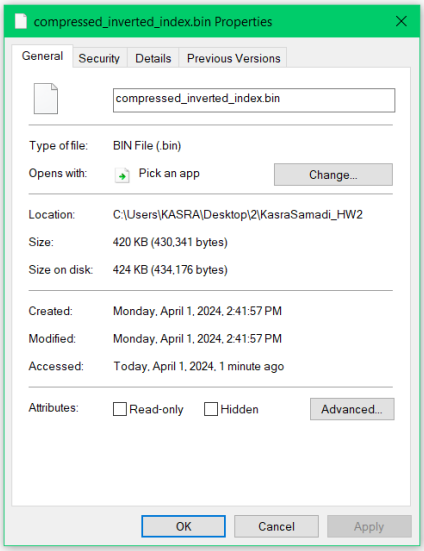
```
Write your Two words to search (Separated by space) : kasra object
object -> Document1, Document3, Document6, Document8, Document9, Document11
```

چون کلمه kasra در compressed_inverted_index وجود ندارد، فقط لیستی از مقالاتی که کلمه object در آن‌ها وجود دارد نمایش داده شده است.

```
In [7]: import pickle
file_path = 'compressed_inverted_index.bin'
with open(file_path, 'wb') as file:
    pickle.dump(compressed_inverted_index, file)
```

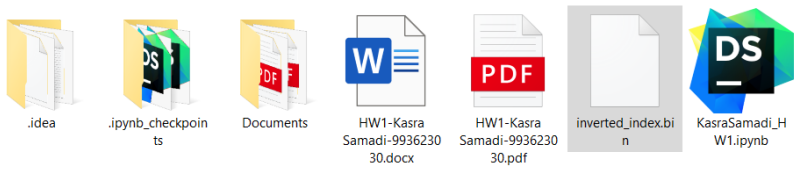
برای مقایسه مقدار فشردگی و کاهش حجم ایندکس در این تمرین با تمرین ۱، با استفاده از کتابخانه pickle ، دیکشنری compressed_inverted_index را به صورت باینری مطابق تصویر بالا در فایل compressed_inverted_index.bin ذخیره کرده و حجم آن را بدست می‌آوریم.

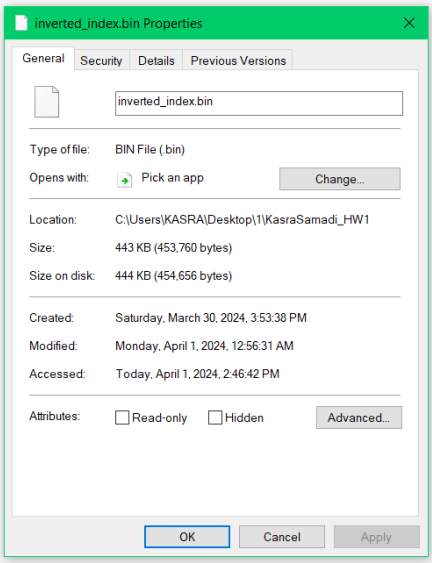




همانطور که مشاهده می‌کنید، حجم ایندکس‌های ذخیره شده در دیکشنری compressed_inverted_index برابر با 420 KB شده است.

مطابق با توضیحات داده شده، همین کار را با inverted_index تمرین شماره یک می‌کنیم و inverted_index را به صورت باینری در فایلی با نام inverted_index.bin ذخیره می‌کنیم و حجم ایندکس‌های ذخیره شده در دیکشنری inverted_index را بدست می‌آوریم.





همانطور که مشاهده می‌کنید، حجم ایندکس‌های ذخیره شده در دیکشنری inverted_index برابر با 443 KB شده است.

ایندکس‌های ذخیره شده در compressed_inverted_index با حجم 420 KB و ایندکس‌های ذخیره شده در inverted_index با حجم 443 KB هستند که در این تمرین با روش فشرده‌سازی، 23 KB از حجم ایندکس‌های ذخیره شده کاهش یافته است به طوریکه در پاسخ‌دهی به کوئری‌های بولینی کاربر هیچ تفاوتی ایجاد نشده است.

مقدار num_common_letters برابر با ۲ باعث ایجاد بیشترین فشردگی حجم ایندکس شده است و سایر مقادیر این متغیر، مقدار فضای ایندکس ذخیره شده‌ی بیشتر از 420 KB را ایجاد می‌کنند.