



دانشکده مهندسی کامپیوتر

دانشگاه اصفهان

تکلیف دوم بازیابی اطلاعات

استاد: دکتر محمد مهدی رضاپور

مهر والسادات نوحی

۹۹۳۶۱۳۰۶۱

بهار ۱۴۰۳

صورت تمرین: 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 پاسخ دهی به کوئری های بولینی کاربر هیچ تفاوتی ایجاد نشود. کدها می‌بایست با زبان پایتون نوشته شود. برنامه می‌بایست قابل تست باشد و گرنه نمره‌ای به تکلیف تعلق نمی‌گیرد.

گام اول:

در تکلیف قبل ما تونستیم جدول را ایجاد کنیم و سرچ بزیم حال باز می‌ایم و شروع به خواندن جدول می‌کنیم و در یک دیکشنری می‌ریزیم که کلید آن کلمه یا توکن و مقدار آن لیست داکيومنت‌ها هست و حالا می‌خواهیم `front_coding` بزیم. فعلا در این گام فقط از فایل جدول معکوس خواندیم و دیکشنری `inverted_index` قرار دادیم و قرار است خروجی این الگوریتم `front_coding` را `output` می‌ریزیم.

```
def main():
    input_file = "inverted_index.txt"
    output_file = "compressed_inverted_index.txt"
    prefix_size = int(input("Enter the prefix size: "))
    inverted_index = read_inverted_index_from_file(input_file)
    compressed_index = compress_inverted_index(inverted_index, prefix_size)
    print_compressed_index_to_file(compressed_index, output_file)
    print("1. Search Inverted Index")
    print("2. Exit")
```

```
def read_inverted_index_from_file(input_file):
    inverted_index = {}
    try:
        with open(input_file, 'r', encoding="utf-8") as file:
            for line in file:
                parts = line.strip().split(':')
                word, docs_str = parts[0], parts[1]
                docs = docs_str.strip().strip('[]').split(', ')
                inverted_index[word] = docs
    except Exception as e:
        print(f"Error reading inverted index from file: {e}")
    return inverted_index
```

توضیحات:

ابتدا ببینیم الگوریتم فرانت کدینگ چجوری کار میکنه. الگوریتم برای فشرده کردن جدول معکوس از لحاظ سائز 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 می گیریم و به جای این کار می توانیم کلمه hell که به آن prefix گفته می شود را یکبار ذخیره کنیم و برای بقیه حروف باقیمانده که به اصطلاح suffix گفته می شود به صورت عادی ذخیره می کنیم 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 ۴ بار ذخیره شده است که اینجا اگر مثلا ۱۰۰۰۰ کلمه دارای lea باشد الکی حافظه گرفته میاییم و فقط lea را یکبار ذخیره می کنیم.

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 ۴ بار 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 داکيومنت‌ها 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.

```
def main():  
    input_file = "inverted_index.txt"  
    output_file = "compressed_inverted_index.txt"  
    prefix_size = int(input("Enter the prefix size: "))  
    inverted_index = read_inverted_index_from_file(input_file)  
    compressed_index = compress_inverted_index(inverted_index, prefix_size)  
    print_compressed_index_to_file(compressed_index, output_file)  
    print("1. Search Inverted Index")  
    print("2. Exit")
```

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 داکيومنت قرار می‌دهیم. و فشرده سازی انجام شده است و خروجی را **compress** می‌ریزیم.

```
def compress_inverted_index(inverted_index, prefix_size):
    common_prefixes = {}

    for word, document_ids in inverted_index.items():
        prefix_key = word[:prefix_size]
        if prefix_key not in common_prefixes:
            common_prefixes[prefix_key] = {}
        suffix = word[prefix_size:]
        if suffix not in common_prefixes[prefix_key]:
            common_prefixes[prefix_key][suffix] = document_ids

    return common_prefixes
```

گام سوم:

حالا این را در فایل خروجی می‌ریزیم.

```
def print_compressed_index_to_file(compressed_index, output_file):
    try:
        with open(output_file, "w", encoding="utf-8") as f:
            for prefix, suffixes in compressed_index.items():
                f.write(f"{prefix}{{")
                for suffix, docs in suffixes.items():
                    # Remove double quotes from document IDs
                    docs_str = ', '.join(doc.strip('"') for doc in docs)
                    f.write(f"'{suffix}':[{docs_str}],")
                f.write("}\n")
            print(f"Compressed inverted index saved to {output_file}")
    except Exception as e:
        print(f"Error printing compressed index to file: {e}")
```

گام چهارم:

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

```

print_compressed_index_to_file(compressed_index, output_file)
print("1. Search Inverted Index")
print("2. Exit")
while True:
    choice = input("Enter your choice (1-2): ")
    if choice == "1":
        Query = input("Enter your Query: ")
        boolean_operators = ["and", "or", "not"]
        if any(op in Query.lower() for op in boolean_operators):
            results=process_boolean_query(Query, compressed_index,prefix_size)
        else:
            prefix = Query[:prefix_size]
            suffix = Query[prefix_size:]
            results = search_in_compressed_index(compressed_index, prefix, suffix)
        if results:
            for filename in results:
                print(filename)
        else:
            print("No documents found.")
    elif choice == "2":
        print("Exiting...")
        break

```

```

def process_boolean_query(query, compress_inverted_index, prefix_size):
    terms = query.lower().split()
    result = None
    i = 0
    while i < len(terms):
        term = terms[i]
        prefix = term[:prefix_size]
        suffix = term[prefix_size:]
        docs = search_in_compressed_index(compress_inverted_index, prefix, suffix)
        if result is None:
            result = set(docs)
        i += 1
    return result

```

```

def search_in_compressed_index(compressed_index, prefix, suffix):
    if prefix in compressed_index and suffix in compressed_index[prefix]:
        return compressed_index[prefix][suffix]
    else:
        return []

```




```

else:
    if i + 1 < len(terms):
        next_operator = terms[i + 1]
        next_term = terms[i + 2] if i + 2 < len(terms) else None
        if next_operator == "and":
            if next_term:
                result.intersection_update(search_in_compressed_index(compress_inverted_index,
                                next_term[:pr
            else:
                break
        elif next_operator == "or":
            if next_term:
                result.update(search_in_compressed_index(compress_inverted_index, next_term[:pr
            else:
                break
        elif next_operator == "not":
            if next_term:
                result.difference_update(search_in_compressed_index(compress_inverted_index, ne
            else:
                break
    i += 2




return result

```

الان `compressed_inverted_index` قبل و بعد اجرای برنامه ببینیم. قبل از اجرا فایل را پاک کردم.

 compressed_inverted_index.txt	4/4/2024 2:14 AM	Text Document	0 KB
 inverted_index.txt	4/3/2024 10:55 AM	Text Document	365 KB
 main.py	4/4/2024 1:17 AM	JetBrains PyCharm ...	5 KB

مشخص است که سایز از ۳۶۵ به ۲۸۹ کاهش یافت البته این به شرط این است که کاربر برای ورودی ۳ کارکتر مشترک در نظر گرفته باشد.

name	date modified	type	size
 compressed_inverted_index.txt	4/4/2024 2:16 AM	Text Document	289 KB
 inverted_index.txt	4/3/2024 10:55 AM	Text Document	365 KB
 main.py	4/4/2024 1:17 AM	JetBrains PyCharm ...	5 KB

تست برنامه:

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

نکته: الان کلمه `hello` فقط در داکيومنت ۲ است و کلمه `this` در هر ۲۰ داکيومنت موجود است حال کویری بولینی داشته باشیم و نباید در خروجی داکيومنت ۲ باشد اگر اپراتور مثلا `not` باشد.

```
Enter your choice (1-2): 1
Enter your Query: hello
'2'
Enter your choice (1-2): 1
Enter your Query: good
'1'
'2'
'3'
'4'
'9'
'10'
'16'
'20'
```

```
Enter your choice (1-2): 1
Enter your Query: this
'1'
'2'
'3'
'4'
'5'
'6'
'7'
'8'
'9'
'10'
'11'
'12'
'13'
'14'
'15'
'16'
'17'
'18'
'19'
'20'
```


مشخص است که داکيومنت ۲ نیست پس کویری بولینی نیز درست است.

```
Enter your Query: this not hello
'5'
'7'
'19'
'13'
'10'
'12'
'9'
'8'
'14'
'1'
'4'
'11'
'3'
'18'
'16'
'17'
'6'
'20'
'15'
```

حالا همین را با and برویم : خروجی مورد انتظار ما ۲ می باشد.

```
Enter your choice (1-2): 1
Enter your Query: this and hello
'2'
```

```
Enter your Query: finish
'4'
'17'
'19'
```

```
Enter your Query: ali
'1'
'2'
'12'
```