پاسخ سوال ۱

قسمت اول

نادرست است، می‌تواند باعث افزایش یا کاهش دقت شود زیرا در این حالت ریشه‌یابی تعداد سندهایی که بازگردانده می‌شود را افزایش دهد بدون آنکه الزاماً تعداد سندهای مرتبط را افزایش دهد. یعنی ممکن است سندهایی را برگرداند که با پرسمان کاربر مرتبط نباشند ولی چون از ریشه‌یابی استفاده کرده‌ایم آنها نیز لحاظ شده باشند.

قسمت دوم

درست است، همانطور که در قسمت قبل نیز بیان کردیم، ریشه‌یابی فقط می‌تواند مجموعه برگردانده شده را افزایش دهد. پس در نتیجه برای recall دو حالت پیش می‌آید یا افزایش می‌یابد و یا تغییر نمی‌کند. پس گزاره گفته شده در صورت سوال درست است و مقدار recall کاهش پیدا نمی‌کند.

قسمت سوم

درست است، در نرمال‌سازی تمام کلمات به شکل استاندارد درمی‌آیند و شکل‌های مختلف یک کلمه ذخیره نمی‌شود. در نتیجه اندازه‌ی لغت‌نامه کاهش می‌یابد.

قسمت چهارم

نادرست است، برای اینکه از تطابق کاملاً مطمن باشیم، این کار باید در زمان پردازش کردن اسناد نیز انجام بگیرد.

پاسخ سوال ۲

در کل هدف از انجام این کار، یکسان کردن کاراکترها با جایگزین کردن کاراکترهای استاندارد در متن ورودی

است. در زیر چند نمونه از انجام این کارها در زبان فارسی آورده شده است.

۱. حذف فاصله‌ها و نیم فاصله‌های اضافی در متن (کلماتی مانند "کتاب خانه" و "کتابخانه" یکی شوند)
۲. حذف نشانه همزه از انتهای کلمات (کلماتی مانند "اشیاء" و "اشیا" یکی شوند)
۳. تبدیل "آ" به "ا " (کلماتی مانند "قرآن" و "قران" یکی شوند)
٤. تبدیل ارقام عربی و انگلیسی به معادل فارسی آنها (اعداد 95 و ۹۵ یکی شوند)
۵. یکسان‌سازی حروفی که چند شکل مختلف دارند.

پاسخ سوال ۳

ابتدا حاصل هر کدام از عبارت‌های داخل پرانتز را محاسبه می‌کنیم .پیچیدگی زمانی هر کدام به صورت زیر می‌شود:

۱ .اسکی or طلا $O(46653+316812)=O(363465)$
۲ . سالاد or جهان $O(107913 + 271658)=O(379571)$
۳ . بارگزاری or سهام $O(2113312 + 87009) = O(300321)$

می‌دانیم که پیچیدگی حاصل از postings list برابر با سایز لیست کوچک‌تر می‌باشد؛ به همین دلیل ابتدا لیست‌های کوچک‌تر را بررسی می کنیم. به این ترتیب ابتدا نتایج مراحل ۳ و ۱ با هم ادغام می‌شوند و نتیجه آن با نتیجه مرحله ۲ ادغام می‌شود.

پاسخ سوال ۴

ماتریس تنک یا اسپارس، ماتریسی است که اندازه‌ی آن به مقدار قابل توجهی بزرگ می‌باشد و همچنین اکثر درایه‌های آن صفر می‌باشد. ماتریس وقوع، نوعی از ماتریس اسپارس می‌باشد، ابتدا به بررسی خود ماتریس وقوع می‌پردازیم.

سطرهای ماتریس وقوع شامل تمام کلمات مختلف در مجموعه اسناد و ستون‌های ماتریس وقوع نیز بیانگر خود اسناد است. حال چون تمام لغات مجموعه اسناد بسیار بیشتر از مجموعه لغت‌های موجود در یک سند است پس

در نتیجه اکثر درایه‌ها در یک ستون صفر است. همچنین چون معمولاً مجموعه اسناد بسیار بزرگ و تعداد کلمات مختلف بسیار زیاد می‌باشد، پس در نتیجه این ماتریس معمولاً ماتریس بزرگی است.

پاسخ سوال ۵

قسمت اول

در شاخص معکوس مکانی، ما موقعیت هر کلمه در سند مربوط را نیز نگهداری می‌کنیم، پس به ازای هر کلمه در سند، یک لیست پیوندی داریم که در آن مکان‌هایی که کلمه ما قرار می‌گیرند، ذخیره می‌شوند. پس در کل ما یک لیست پیوندی داریم که در آن لغت‌ها را وارد می‌کنیم و به ازای هر کلمه متفاوت یک لیست پیوندی داریم و هر کدام از اجزای این لیست پیوندی یک شی می‌باشد که دارای شماره و یک لیست پیوندی است. شماره مشخص کننده سندی است که آن کلمه در آن قرار گرفته است و لیست پیوندی نیز مکانهایی را مشخص می‌کند که آن کلمه در سند تکرار شده است.

در کد زیر به ازای هر کلمه، یک آبجکت Term داریم. به ازای هر آبجکت Term، به تعداد متن‌هایی که این کلمات در آن‌ها ظاهر شده‌است آبجکت DocTerm داریم که هر آبجکت DocTerm شامل شماره متن positionها به صورت مرتب است.

```python
class DocTerm:
    # DocTerm = a list of positions
    def __init__(self, docId, termId):
        self.count = 0
        self.docId = docId
        self.termId = termId
        self.positions = []

    def insert(self, position):
        self.count += 1
        self.positions.append(position)
        self.positions = sorted(self.positions)


class Term:
    # Term = a list of DocTerms
    def __init__(self, id):
        self.count = 0
        self.id = id
        self.docTerms = dict()

    def insert(self, docId, position):
        self.count += 1
        if docId not in self.docTerms.keys():
            self.docTerms[docId] = DocTerm(docId, self.id)
        self.docTerms[docId].insert(position=position)


class Index:
    def __init__(self, docs):
        self = dict()
        for i in range(len(docs)):
            doc = docs[i]
            for j in range(len(doc)):
                word = doc[j]
                if word not in self.keys():
                    self[word] = Term(id=id(word))
                self[word].insert(docId=i, position=j)

```

قسمت دوم

برای پاسخگویی به پرسمان، ابتدا لغات موجود در پرسمان را پیدا می‌کند و سپس با توجه به نوع پرسمان عملیات مربوطه را انجام می‌دهد. ابتدا لیست پیوندی هر کلمه را پیدا کرده و سپس شروع به حرکت روی لیست پیوندی میکند ولی با این تفاوت که لیست پیوندی موقعیت هر کلمه در سند را هم در نظر می‌گیرد و با توجه به آن جلو می‌رود. مثلا اگر کلمه " to be " آمده باشد در اینصورت باید لیست پیوندی این دو کلمه را بدست آورده و سپس باید لیست پیوندی سندهایی که هر دو کلمه در آن آمده است را پیدا کنیم و سپس بررسی کنیم که آیا موقعیت این دو کلمه به همین شکل پشت سر هم می‌باشد یا نه.

```
5   def query(word1, word2, k):
6       ans = []
7       p1 = index[word1].docTerms
8       p2 = index[word2].docTerms
9       while p1 is not None and p2 is not None:
10          if p1.docId == p2.docId:
11              l = []
12              pp1 = p1.positions
13              pp2 = p2.positions
14              while pp1 is not None:
15                  while pp2 is not None:
16                      if abs(pp1.curPos - pp2.curPos) <= k:
17                          l.append(pp2.curPos)
18                      elif pp2.curPos > pp1.curPos:
19                          break
20                      pp2 = next(pp2)
21                  while l is not [] and abs(l[0] - pp1.curPos) > k:
22                      delete(l[0])
23                  for ps in l:
24                      ans.append([p1.docId, pp1.curPos, ps])
25                  pp1 = next(pp1)
26
27              p1 = next(p1)
28              p2 = next(p2)
29          elif p1.docId < p2.docId:
30              p1 = next(p1)
31          else:
32              p2 = next(p2)
33
```

پاسخ سوال ۶

می توانیم سه عملگر زیر را پیاده سازی کنیم:

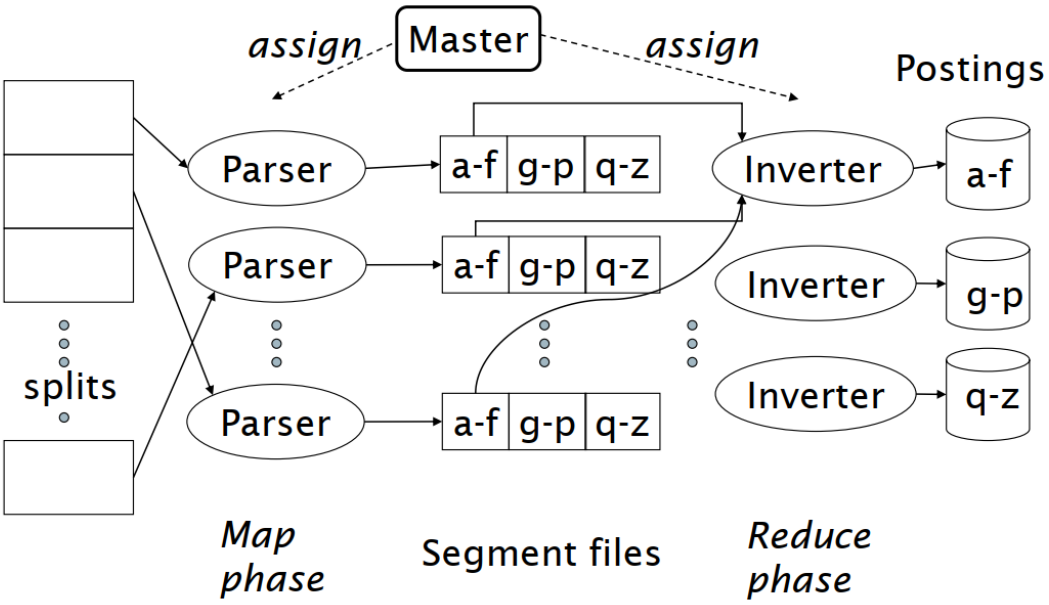
and: مانند حالت گفته شده در درس است. دو اشاره گر برای دو لیست در نظر گرفته می شود و با حرکت دو اشاره گر روی لیست‌ها، هر جا که اشتراکی بین دو لیست پیدا شود به لیست نتیجه اضافه می شود. پیچیدگی این عملیات برابر با جمع طول دو لیست ورودی است.

or: مانند حالت and است با این تفاوت که کلیه عناصر دو لیست به لیست نتیجه اضافه می شوند و حرکت اشاره گر ها در دو لیست مانند حالت قبل است. پیچیدگی این عملیات برابر با جمع طول دو لیست ورودی است.

and not: برای حالتی مثل X and not دو اشاره گر برای لیست های X و Y در نظر می گیریم. اشاره گر مربوط به لیست X را در ابتدای آن قرار می دهیم و اشاره گر Y را تا حدی جلو میدهیم که docid آن بیشتر یا مساوی docid مربوط به اشاره گر X شود. اگر در این حالت دو شناسه سند با هم برابر بودند که آن سند در لیست نتیجه درج نخواهد شد و هر دو اشاره گر یک واحد جلو می روند و اگر برابر نبودند شناسه سندی که اشاره گر لیست X به آن اشاره می کند به لیست نتایج اضافه می شود و اشاره گر لیست X یکی جلو می رود. مجددا به حرکت اشاره گر لیست Y به شکلی که گفته شد عملیات فوق را تکرار می کنیم اشاره گر لیست X به انتهای این لیست برسد. پیچیدگی این عملیات برابر با جمع طول دو لیست ورودی است.

پیاده سازی عملیاتی مانند not X یا X or not Y که در آن X و Y لیست های اسناد هستند می تواند دارای پیچیدگی از مرتبه کل اسناد مجموعه داده باشد.

پاسخ سوال ۷



بررسی واحد parser

همانطور که از شکل نیز قابل برداشت می‌باشد، ورودیهای این قسمت همان اسناد ما می‌باشند اما این اسناد ابتدا توسط واحد master به دسته‌هایی افراز شده‌اند و به هر کدام از این دسته‌ها یک split نامیده می شود. حال هر کدام از این دسته‌ها به یک parser محول می‌شوند و Parser مربوطه نیز token های موجود از اسناد را استخراج کرده ، پردازش های لازم را روی آنها انجام داده و آنها را به دسته‌هایی تقسیم می‌کند و توکن های مربوط به هر دسته را در محل خاصی که Master مشخص می کند ذخیره می کند.

بررسی واحد inverter

همانطور که گفته شد خروجی هر کدام از parser ها به قسمت‌هایی تقسیم شده بودند. مثلا قسمت a-f ، که tokenهای با این حروف شروع در آن قرار گرفته‌اند. حال قسمت اول خروجی‌های تمام parserها به عنوان ورودی به یک invertor داده می‌شود. مثلاً در شکل بالا قسمت a-f تمام خروجی های مرحله اول به عنوان ورودی به inverter شماره 1 داده می‌شود. سپس هر کدام از واحدهای invertor خروجی parser های مختلف را ادغام کرده و در محلی که master مشخص می کند ذخیره می کند.

پاسخ سوال ۸

ذخیره کردن هر postings list در یک فایل مجزا باعث می شود که در شاخص گذاری پویا به راحتی بتوان شاخص موجود را تغییر داد. در این حالت اضافه کردن یک سند به postings list یک کلمه تنها شامل append کردن یک docid به انتهای فایل مربوطه است و تغییری در سایر فایل ها ایجاد نمی کند. در این حالت ادغام شاخص ها نیز در صورتی که یکی از شاخص های خیلی کوچک باشد ممکن است با خواندن بخش کوچکی از شاخص بزرگتر انجام شود و نیازی به اعمال در بسیاری از بخش های شاخص بزرگتر نیز نباشد. اما در صورتی که از یک فایل بزرگ برای ذخیره سازی تمام postings listها استفاده شود، با هر تغییر کوچک در شاخص مجبور هستیم که این فایل بزرگ را تغییر دهیم که ممکن است مستلزم انجام تغییرات زیادی باشد. مثلا مجبور باشیم کل فایل را شیفت دهیم تا یک docid جدید در جایی از فایل درج شود.

از طرفی مدیریت فایل ها برای سیستم عامل سربار دارد و هر چه تعداد فایل ها بیشتر باشد این سربار نیز بیشتر خواهد و اگر تعداد فایل ها خیلی زیاد باشد ممکن است باعث افت شدید کارآیی در سیستم عامل شود.